TESIS DEFENDIDA POR

CARLOS ALBERTO CÓRDOVA FLORES

Y aprobada por el siguiente comité:

Dr. José Alberto Fernández Zepeda

*Director del Comité*

Dr. Andrei Tchernykh

*Miembro del Comité*

Dr. Carlos Alberto Brizuela Rodríguez

*Miembro del Comité*

Dr. J. Apolinar Reynoso Hernández

*Miembro del Comité*

Dr. Pedro Gilberto López Mariscal

*Coordinador del Programa de Posgrado en Ciencias de la Computación*

Dr. Edgar Gerardo Pavía López

*Director de Estudios de Posgrado*

13 de Abril de 2007

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA


DIVISIÓN DE FÍSICA APLICADA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

# SIMULACIÓN EN TIEMPO CONSTANTE DE UN MODELO R-MESH EN UN LR-MESH

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

CARLOS ALBERTO CÓRDOVA FLORES

Ensenada, Baja California, México. Abril de 2007.

**RESUMEN** de la tesis de **Carlos Alberto Córdova Flores**, presentada como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, Baja California, México. Abril de 2007.

# SIMULACIÓN EN TIEMPO CONSTANTE DE UN MODELO R-MESH EN UN LR-MESH

Resumen aprobado por:

Dr. José Alberto Fernández Zepeda
*Director de Tesis*

Se creía que el modelo LR-Mesh era computacionalmente menos poderoso que el modelo R-Mesh. Omer Reingold mostró un algoritmo para la máquina de Turing que resuelve USTCON requiriendo $O(\log N)$ unidades de espacio, implicando que las clases de complejidad $L$ y $SL$ son equivalentes. Los modelos LR-Mesh y R-Mesh son capaces de resolver los problemas de las clases $L$ y $SL$ en tiempo constante, respectivamente. Por lo tanto, los resultados de Reingold implican que los modelos LR-Mesh y R-Mesh son computacionalmente igual de poderosos.

En el presente trabajo se describe una simulación del modelo R-Mesh en el LR-Mesh en tiempo constante. La mejor simulación conocida requería de $O(\log N)$ unidades de tiempo. Se muestra cómo se puede convertir un grafo regular cualquiera en un expansor y como resolver el problema USTCON en un expansor en el modelo LR-Mesh siguiendo el algoritmo de Reingold. Con estos resultados se construye la smulación entre los modelos en tiempo constante. Finalmente se muestra un análisis de los recursos (procesadores) requeridos por dicha simulación.

**Palabras clave:** Modelos de Cómputo, Arquitecturas reconfigurables, Simulación.

**ABSTRACT** of the thesis presented by **Carlos Alberto Córdova Flores** as a partial requirement to obtain the MASTER IN SCIENCE degree in COMPUTER SCIENCE. Ensenada, Baja California, México. April 2007.

# CONSTANT TIME SIMULATION OF AN R-MESH MODEL ON AN LR-MESH MODEL

Abstract approved by:

Dr. José Alberto Fernández Zepeda
*Thesis Director*

There was the assumption that the LR-Mesh model was computationally less powerful than the R-Mesh. Omer Reingold showed an algorithm for the Turing machine that solves USTCON problem with $O(\log N)$ space units, implying that the complexity classes $L$ and $SL$ are equivalent. The LR-Mesh and R-Mesh models are capable of solving the problems that belong to the classes $L$ and $SL$ in constant time, respectively. Hence, Omer Reingold's results implies that the model LR-Mesh is computationally as powerful as the R-Mesh.

The present work decribes a simulation of the R-Mesh model on the LR-Mesh in constant time. The best simulation known required $O(\log N)$ time units. It shows how to transform a regular graph into an expander and how to solve USTCON in an expander graph on the LR-Mesh, following Reingold's algorithm. With these results the constant time simulation between the models is constructed. Finally it shows an analysis of the resources (processors) required for the stated simulation.

**Keywords:** Computing models, Reconfigurable architectures, Simulation.

# Dedicatoria

*A mi padre*

Guadalupe Córdova Rascón

## Agradecimientos

A mi madre Bertha Flores Huerta.
A mi hermana Ana Bertha Córdova Flores.
A la familia Flores, familia Córdova y en especial a
China, Diego, Gogo, Güera, Julio, Matias, Xtinita,
Galy, Mono, al próximo Matiín y los que estén por arrivar.

A mi asesor, Dr. José Alberto Fernández Zepeda.

A mi comité de tesis, Dr. Andrei Tchernykh,
Dr. Carlos Alberto Brizuela Rodríguez
Dr. J. Apolinar Reynoso Hernández

Al Centro de Investigación Científica y de Educación
Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología.

# Contenido

# Contenido (continuación)

Capítulo Página

# Contenido (continuación)

Capítulo Página

# Lista de Figuras

# Lista de Figuras (continuación)

# Lista de Figuras (continuación)

# Lista de Tablas

# Capítulo I

# Introducción

Debido a la creciente necesidad de procesar una gran cantidad de información en tiempos reducidos se han propuesto una gran cantidad de modelos de cómputo paralelo y distribuido, desde el tradicional modelo PRAM hasta modelos cuánticos.

Los modelos reconfigurables son otra propuesta dentro del cómputo paralelo. Estos modelos han mostrado un gran rendimiento para resolver cierto tipo de problemas de cómputo gracias a la habilidad de reconfiguración interna. Uno de los modelos más estudiados dentro de las arquitecturas reconfigurables es el llamado R-Mesh. Su gran versatilidad permite el diseño de algoritmos de forma sencilla pero es difícil su implementación en la vida real. Existe una versión restringida (en la cantidad de configuraciones internas) del modelo R-Mesh llamado LR-Mesh. Este modelo restringido es más factible de construirse que el modelo R-Mesh, aunque el diseño de algoritmos en él es más complicado.

Se conjeturaba que debido a las restricciones en su configuración interna, el modelo LR-Mesh era computacionalmente menos poderoso (dos modelos de cómputo paralelo son computacionalmente igual de poderosos si éstos pueden resolver los mismos problemas en el mismo tiempo permitiendo un aumento polinomial en el número de procesadores) que el modelo R-Mesh (Vaidyanathan y Trahan, 2004), sin embargo no se conocía una demostración formal al respecto.

Tanto el modelo R-Mesh como el modelo LR-Mesh tienen una relación con las clases de complejidad SL y L, respectivamente. La clase SL contiene a los problemas que puede resolver una *Máquina Simétrica de Turing* con un espacio logarítimico (en el

Capítulo III se desarrolla el tema con profundidad). Los problemas que puede resolver una Máquina Determinística de Turing con un espacio logarítmico pertenecen a la clase L.

Omer Reingold (Reingold, 2005) demostró que las clases de complejidad L y SL son las mismas. Esto implica que los modelos R-Mesh y LR-Mesh son igual de poderosos. Sin embargo hasta antes de este trabajo no se conocían los recursos requeridos para realizar dicha simulación. El presente trabajo muestra una simulación en tiempo constante del modelo R-Mesh sobre el modelo LR-Mesh.

A continuación se muestra una breve descripción del trabajo previo en el cual se sustenta parte del documento, los motivos por los cuales el presente trabajo resulta atractivo, se define formalmente el problema de la simulación, se muestra la metodología propuesta para resolver el problema y finalmente se da una descripción general del documento.

## I.1 Trabajo previo

A continuación se muestra una descripción general de la mejor simulación conocida del modelo R-Mesh en el modelo LR-Mesh.

Fernández *et al.* (Fernández-Zepeda *et al.*, 2002) diseñaron una simulación de un R-Mesh de $N \times N$ procesadores en un LR-Mesh de $N \times N$ procesadores. Esta simulación requiere sólo $O(\log N)$ unidades de tiempo y es la más rápida que se conoce. Esta simulación es óptima en cuanto al número de procesadores ya que el LR-Mesh (modelo simulador) requiere el mismo número de procesadores que el R-Mesh (modelo simulado). Esta simulación se basa en una técnica llamada *linealización de ductos*, cuyo objetivo es transformar cualquier ducto no lineal (aceptado por un R-Mesh) a un ducto lineal

equivalente (aceptado por un LR-Mesh). A continuación se describe en forma general cómo se realiza esta simulación.

Cualquier configuración del modelo R-Mesh se puede representar mediante un grafo $G$. Cada vértice del grafo $G$ representa a un conjunto de puertos fusionados. Sólo existe una arista en $G$ entre dos vértices $v_1$ y $v_2$, si existe una conexión en el modelo R-Mesh que una cualquiera de los puertos representados por $v_1$ y $v_2$.

Para llevar a cabo la simulación, se transforma en forma iterativa el grafo $G$ en un árbol de esparcimiento y este árbol en un pseudo-tour de Euler. Un árbol de esparcimiento de un grafo $G$ es un subgrafo de $G$ que contiene todo los vértices de $G$ y ningún ciclo. Cualquier árbol $T$ se puede transformar en un árbol $T'$ tal que los vértices de $T'$ son los mismos vértices de $T$ y cada arista $(u, v)$ en $T$ se cambia por dos aristas dirigidas $< u, v >$ y $< v, u >$. Esto es, se reemplaza cada arista de $T$ por dos arcos opuestos. Un tour de Euler de $T$ es un ciclo dirigido en $T'$ que recorre en forma cerrada cada arco de $T'$ una sola vez. Otra forma de describir un tour de Euler de un grafo $T$ es un ciclo que recorre cada arista de $T$ dos veces exactamente, una en cada dirección. Un pseudo-tour de Euler es un tour de Euler acíclico, esto es, se elimina una de las aristas que conforman el ciclo.

Al inicio de la simulación cada vértice del grafo $G$ decide cuál de los vértices vecinos será su padre dentro del árbol mediante un intercambio de identificadores. Hay que notar que en el proceso se pueden crear varios árboles para un mismo componente conectado. Después se crea un pseudo-tour de Euler para cada uno de los árboles cambiando cada arista por dos aristas. Es posible romper el ciclo creado si el nodo raíz de cada árbol elimina una de sus aristas.

En forma iterativa se fusionan los árboles que pertenecen al mismo componente conectado. Cada árbol determina si es posible injertarse a otro árbol vecino. Para

fusionar dos árboles $T_1$ y $T_2$ es necesario que exista un vértice $v_1 \in T_1$ y un vértice $v_2 \in T_2$ tal que exista la arista $(v1, v2)$ en el grafo original. Se rompe el pseudo-Tour de Euler en los vértices $v_1$ y $v_2$ y se incluye las aristas $< v1, v2 >$ y $< v2, v1 >$ para formar el nuevo pseudo-tour de Euler. Si todos los árboles participan en un proceso de injertado entonces la cantidad de árboles en cada iteración se reduce en el peor de los casos a la mitad. Si se repite el proceso $O(\log n)$ veces entonces se asegura que cada componente conectado ahora esté representado por un pseudo-tour de Euler.

Al final del proceso se tiene un grafo lineal acíclico que el modelo LR-Mesh puede simular sin problemas. Un ducto acíclico permite una elección de líder en tiempo constante, por lo tanto las escrituras en el ducto requieren de tiempo constante para simularlas. Todos los pasos de la simulación los puede efectuar un LR-Mesh de tamaño $N \times N$ en tiempo constante. El proceso de fusionar los árboles es el que requiere mayor tiempo para su ejecución, por lo tanto la simulación completa requiere $O(\log n)$ unidades de tiempo.

## I.2 Motivación

Antes se suponía que el modelo R-Mesh era más poderoso en comparación con el modelo LR-Mesh. Los Algoritmos que se ejecutaban en $O(1)$ en un R-Mesh requerían de $O(\log N)$ para ejecutarse en un LR-Mesh. La mejor simulación que se conocía requería de $O(\log N)$ pasos precisamente. Los resultados de Omer Reingold (Reingold, 2005) implican que esta simulación no es óptima y que debe de existir una simulación de $O(1)$.

Desde el punto de vista del diseño de algoritmos, el modelo R-Mesh presenta una ventaja significativa. Al ser un modelo menos restringido que el LR-Mesh, resulta más sencillo el diseño de algoritmos.

Ambos modelos teóricos presentan diversas dificultades para su realización. El modelo LR-Mesh, al ser una versión simplificada del modelo R-Mesh, es más sencillo de implementar. El modelo R-Mesh requiere de 5 tipos de conexiones internas más que el modelo LR-Mesh. Los modelos reconfigurables ópticos se han estudiado debido a que son capaces de manejar una alta tasa de comunicación en forma eficiente. Estos modelos, debido a propiedades ópticas, sólo permiten ductos lineales y acíclicos. Similar en funcionamiento al modelo LR-Mesh.

Contando con una simulación entre los modelos R-Mesh y LR-Mesh óptima se puede obtener los beneficios de ambos modelos. Se puede diseñar algoritmos bajo el modelo R-Mesh donde esto resulta más fácil, aunque sea difícil su :implementación, y se puede simular dicho algoritmo en un modelo LR-Mesh en tiempo constante. Dando con esto una gran importancia a la simulación en tiempo constante entre los modelos.

## I.3 Definición del problema

El modelo R-Mesh es capaz de resolver los problemas pertenecientes a la clase de complejidad SL en tiempo constante. De forma similar el modelo LR-Mesh puede resolver en tiempo constante los problemas pertenecientes a la clase L. En (Reingold, 2005) se establece la igualdad entere las clases L y SL. Debido a la unificación de estos resultados se infiere que el poder computacional que presentan los modelos R-Mesh y LR-Mesh es el mismo. Esto es, debe de existir una simulación de un modelo sobre otro en tiempo constante.

### I.3.1 Objetivos generales

Diseñar una simulación del modelo R-Mesh sobre el modelo LR-Mesh óptima en tiempo.

### I.3.2 Objetivos específicos

1. Diseñar un algoritmo para transformar un grafo a un expansor en el modelo LR-Mesh en tiempo constante.

2. Diseñar un algoritmo para resolver el problema USTCON en un expansor en el modelo LR-Mesh en tiempo constante.

3. Diseñar un algoritmo que simule cada uno de los subcíclos de máquina de un R-Mesh en el LR-Mesh.

4. Hacer un análisis de los recursos empleados por estos algoritmos.

## I.4 Metodología

Con el fin de lograr el trabajo de investigación se plantearon las siguientes etapas. El completar cada etapa brindó los recursos necesarios para el resto de la investigación.

Etapa 1. Estudio de las arquitecturas reconfigurables.

Como primera fase se estudiaron los modelos de arquitecturas reconfigurables con especial énfasis en los modelos R-Mesh y LR-Mesh. Con esto se obtuvieron los conocimientos necesarios para poder entender el funcionamiento de los modelos así como destreza en el análisis de algoritmos. Esto implicaba el conocimiento profundo de las diversas técnicas utilizadas en el desarrollo de algoritmos para modelos reconfigurables.

Etapa 2. Estudio de la Máquina de Turing y clases de complejidad.

En esta fase, se estudió la teoría relacionada con la máquina de Turing necesaría para comprender el algoritmo de USTCON que requiere $O(\log N)$ unidades de espacio. Tambien se buscó comprender dos de las clases de complejidad relacionadas con el espacio requerido y sus relaciones con las arquitecturas reconfigurables.

Etapa 3. Estudio del algoritmo para resolver el problema USTCON requiriendo un espacio $O(\log n)$.

Para poder desarrollar una simulación óptima fue necesario comprender el algoritmo presentado en (Reingold, 2005). Para comprender dicho algoritmo fue necesario el estudio formal de la teoría de expansores y el producto zig-zag entre grafos.

Etapa 4. Diseñó de una simulación del modelo R-Mesh en el LR-Mesh óptima.

Debido a la relación que existe entre R-Mesh y el LR-Mesh con las clases SL y L respectivamente, y aunado a los resultados de (Reingold, 2005), donde se establece la igualdad entre las clases L y SL, se infería que debía existir una simulación de un modelo al otro en tiempo constante. La elaboración de dicha simulación es el propósito de esta fase, destacando la cantidad de recursos necesarios para lograr la simulación.

Etapa 5. Diseño de un algoritmo que resuelva el problema USTCON en el modelo LR-Mesh en tiempo constante.

Se diseño un algoritmo que resuelve el problema de conectivdiad entre dos vértices en un expansor en tiempo constante. También se realizó un análisis de la cantidad de recursos requeridos por dicho algoritmo.

Etapa 6. Obtener la clausura transitiva de un grafo regular.

En función de los algoritmos de las etapas anteriores se obtuvo un algoritmo que calcula la clausura transitiva de un grafo regular en el modelo LR-Mesh. El algoritmo se ejecuta en tiempo constante. Al igual que en las etapas anteriores se analizó la cantidad de recursos requeridos por el algoritmo.

Etapa 7. Diseño de una simulación del modelo R-Mesh en el LR-Mesh óptima.

Debido a la relación que existe entre los modelos R-Mesh y LR-Mesh con las cases SL y L, respectivamente, y aunado a los resultados de Reingold, (2005), donde se

establece la igualdad entre las clases L y SL, se infería que debía existir una simulación de un modelo al otro en tiempo constante. Los algoritmos desarrollados en las etapas anteriores permitieron el diseño de la simulación en tiempo constante. La elaboración de dicha simulación fue el propósito de esta fase, destacando la cantidad de recursos necesarios para lograr la simulación.

## I.5 Organización del documento

- El trabajo realizado en la etapa 1 se plasmó en el Capítulo II. Se estudiaron varias rejillas reconfigurables así como las características generales de los modelos reconfigurables.

- El trabajo realizado en la etapa 2 corresponde al Capítulo III. Se muestran la descripción de diferentes máquinas de Turing, las clases de complejidad L y SL y la demostración de que el modelo R-Mesh resuelve los problemas de la clase SL en tiempo constante y el modelo LR-Mesh resuelve los problemas de la clase L en tiempo constante.

- La etapa 5 se describe en dos capítulos. En el Capítulo IV se muestra la definición formal de un grafo expansor y la definición del producto zig-zag. En el Capítulo V se describe en forma detallada el algoritmo para resolver el problema USTCON con un espacio de memoria logarítmico.

- La simulación y todo lo que ésta implica se dividió en cuatro capítulos. En el Capítulo VI se muestran una serie de algoritmos desarrollados para el modelo LR-Mesh. Todos los algoritmos se ejecutan en tiempo constante y se utilizan como bloques constructores para resolver problemas más complejos. El algoritmo para resolver el problema USTCON con espacio de memoria logarítmico se dividió

en dos subproblemas. El Capítulo VII trata con el primero de los subproblemas: transformar un grafo cualquiera a un expansor. El segundo subproblema, resolver el problema USTCON en un expansor, se muestra en el Capítulo VIII. Finalmente, la simulación del modelo R-Mesh en el modelo LR-Mesh en tiempo constante se muestra a detalle en el Capítulo IX.

- El documento termina con el Capítulo X donde se exponen las concluciones y algunas ideas para posibles investigaciones a futuro.

# Capítulo II

# Arquitecturas de ductos reconfigurables

Existe una gran cantidad de modelos para cómputo paralelo. Esto se debe a que el desempeño de un algoritmo paralelo depende en gran medida de un conjunto de factores. Los factores pueden ser la habilidad para calcular información en forma concurrente, la distribución de los procesadores, la calendarización de tareas, los procesos de sincronización, entre muchos otros.

El modelo PRAM (Parallel Random Access Machine) es uno de los modelos paralelos más estudiados. En (JaJa, 1992) se define el modelo PRAM como sigue: es un conjunto de procesadores numerados $1, 2, \cdots, p$; cada uno con su propia memoria local y pueden ejecutar algún programa en forma independiente, además son capaces de intercambiar información mediante una memoria compartida.

En otros modelos se incluye además la topología que conecta a estos procesadores: anillo, estrella, hipercubo, etc. Por supuesto el diseño de algoritmos bajo estos modelos tiene que considerar estas características. Por ello los algoritmos se tienen que adaptar a dicha topología.

Se han estudiado otros modelos que difieren de los anteriores precisamente en la forma en que se interconectan los elementos de cómputo. Algunos de ellos tienen habilidades de reconfiguración y mayor poder computacional.

Un conjunto de elementos de cómputo conectados mediante un medio reconfigurable de comunicación se denomina arquitectura reconfigurable dinámica. Esto es, una serie de procesadores que siguiendo alguna organización están interconectados pero no en

forma estática, como comúnmente se concibe a las topologías para cómputo paralelo. El sistema es capaz de cambiar las conexiones, de acuerdo a algún propósito, formando con esto diferentes configuraciones que permiten la realización de algún objetivo en forma eficiente. El término reconfiguración dinámica implica que la reconfiguración de la estructura de ductos (conexiones) es rápida y flexible, además el pago en tiempo por la reconfiguracón es tan bajo que es despreciable. La flexibilidad en la creación de ductos permite que se utilizen dichos ductos como una herramienta computacional.

Se han propuesto una gran cantidad de modelos de cómputo paralelo que presentan esta propiedad de reconfiguración dinámica, cada uno con sus propias características que lo distingue entre los demás. Algunos de estos modelos son: RN (Ben-Asher *et al.*, 1991), PARBS (Wang y Chen, 1990), RMBM (Trahan *et al.*, 1996), DMBC (Vaidyanathan y Trahan, 2004), etc.

El modelo red reconfigurable (RN) (Vaidyanathan y Trahan, 2004) consiste en una colección de procesadores interconectados, por medio de puertos, de acuerdo a la topología de algún grafo conectado. Cada vértice del grafo representa a un procesador y las aristas a las interconexiones entre procesadores. Cada procesador puede fusionar un subconjunto de sus puertos, creando con esto *ductos* de formas variadas. Cuando el grafo en que se basa el arreglo de procesadores es una rejilla, el modelo se denomina rejilla reconfigurable. A continuación se presenta una descripción más profunda de este caso especial del modelo RN.

## II.1 Rejilla reconfigurable

En esta sección se presenta la descripción general del modelo, el ciclo de máquina, el retardo de propagación en el ducto, el tamaño del ducto, el tipo de lecturas y escrituras

permitadas, así como diversas variantes de la rejilla reconfigurable (R-Mesh, LR-Mesh, HVR-Mesh, FR-Mesh, DR-Mesh).

### II.1.1 Ciclo de máquina

Cada procesador de una rejilla reconfigurable tiene su propia memoria local. No existe una memoria compartida entre los procesadores. Toda comunicación entre procesadores se realiza mediante el sistema de ductos reconfigurables. El sistema es síncrono, es decir en un instante dado todos los procesadores leen del ducto, escriben en el ducto, configuran sus puertos o realizan alguna operación (generalmente con diferentes datos). Un ciclo de máquina (Vaidyanathan y Trahan, 2004) consta de 3 etapas:

1. Partición. Cada procesador fusiona sus puertos en alguna de las configuraciones permitidas por el modelo (estas se discuten posteriormente). Es importante notar que el modelo permite que cada procesador configure sus puertos en forma independiente y dos procesadores cualesquiera pueden adoptar diferentes configuraciones en el mismo ciclo de máquina. Esta habilidad se denomina autonomía de conexión.

2. Comunicación. El procesador escribe y lee de sus puertos. Es posible que el procesador lea y escriba en algunos o incluso todos sus puertos al mismo tiempo. Cuando un procesador lee y escribe en algún puerto, el valor que lee es el valor final que se genera en el ducto en el presente ciclo de máquina.

3. Cálculo. El procesador efectúa una operación (lógica o aritmética) sobre alguna información local. Esta información puede estar contenida en la memoria local o puede ser un dato leído de algún ducto en la etapa de comunicación.

Se supone que cada una de estas fases se ejecutan en tiempo constante. Es decir el tiempo de ejecución del ciclo de máquina es independiente del número de procesadores utilizados por la rejilla reconfigurable. No es necesario que los procesadores participen en las tres etapas por ciclo de máquina. Es posible que no se requiera alguna de las etapas, de ser así el procesador permanece en espera por ese periodo de tiempo.

### II.1.2 Retardo de señal

Para que la etapa de comunicación ocurra en tiempo constante es imprescindible que la información escrita se propague por todo el ducto en tiempo constante, sin importar que tan grande sea dicho ducto. El modelo de retardo unitario (Vaidyanathan y Trahan, 2004) supone que el tiempo de propagación de la información en cualquier ducto, sin importar la cantidad de procesadores que atraviesa el ducto, es constante. Por supuesto la comunicación de información en un sistema real depende de la longitud del medio por el que se transmite entre otros factores, sin embargo para modelos con pocos procesadores la suposición de propagación constante en ductos es una buena aproximación de un sistema real.

El modelo de retardo logarítmico (Vaidyanathan y Trahan, 2004) contempla la longitud de los ductos dentro de la etapa de comunicación. La longitud de un ducto lineal (un ducto lineal es una cadena de puertos en el que cada uno de ellos tiene a lo más dos vecinos) $\alpha$ se define como la cantidad de puertos conectados al ducto $\alpha$. El modelo de retardo logarítmico establece un retardo de $\log L$ para un ducto de longitud $L$. La mayoría de los algoritmos que se ejecutan en una rejilla reconfigurable, con $N$ procesadores, utilizan por lo menos un ducto cuya longitud es polinomial en $N$. Por lo tanto el retardo establecido para cada etapa de comunicación bajo este modelo es generalmente $\Theta(\log N)$.

### II.1.3 Tamaño del ducto

Para que un procesador se identifique a si mismo y a los demás procesadores dentro de una rejilla de $N$ procesadores se requieren $\Omega(\log N)$ bits. Debido a que el identificador de un procesador se utiliza en muchas ocaciones dentro de un algoritmo, como información que se intercambia entre procesadores, se supone que los ductos son capaces de transmitir $\Omega(\log N)$ bits en forma simultanea. Esto es, el tamaño del ducto es de por lo menos $\log N$ bits. A este modelo se le denomina modelo de palabra.

El modelo de bit restringe el tamaño del ducto y de la memoria local a una constante. Por supuesto esto impacta en el diseño de algoritmos ya que bajo este modelo es imposible dirigirse a algún procesador en específico debido a la ausencia de identificadores.

En (Vaidyanathan y Trahan, 2004) se demuetra que una rejilla reconfigurable de $Nb \times Nb$, bajo el modelo de bit, puede simular cada paso de una rejilla reconfigurable de $N \times N$, bajo el modelo de palabra de $b$-bits, en tiempo constante.

### II.1.4 Acceso al ducto

Normalmente se supone que sólo uno de los procesadores conectados a cualquier ducto puede escribir en él en un instante dado, aunque todos los procesadores son capaces de leer la información. A este modelo se le denomina de lecturas concurrentes y escrituras exclusivas (CREW, por sus siglas en inglés), siguiendo la nomenclatura utilizada en el modelo PRAM. Otros modelos posibles son el de lecturas exclusivas y escrituras exclusivas (EREW), el de lecturas exclusivas y escrituras concurrentes (ERCW) y el de lecturas concurrentes y escrituras concurrentes (CRCW).

El hecho de que dos o más procesadores lean de un ducto al mismo tiempo no presenta ningún problema, sin embargo las escrituras concurrentes si requieren un manejo

más elaborado. Cuando varios procesadores intentan escribir al mismo tiempo en un mismo ducto se produce una escritura concurrente que se resuelve en función de alguna regla de escritura preestablecida. A continuación se definen las reglas de escritura más utilizadas:

1. Común. Todas los procesadores que requieran escribir en forma concurrente en un mismo ducto deben escribir el mismo valor.

2. Colisión. Cuando se presenta una escritura concurrente en un ducto, se produce en el ducto un símbolo único que informa a los procesadores de que ocurrió una colisión.

3. Colisión$^{+}$. Es la combinación de las reglas común y colisión. Cuando varios procesadores intentan escribir el mismo dato se logra la escritura. En el caso de que varios procesadores intenten escribir diferentes valores en el ducto se produce el símbolo de colisión.

4. Arbitrario. Del conjunto de procesadores que requieren escribir en el mismo ducto sólo uno de ellos tiene éxito. Es importante notar que el procesador que tiene exito es aleatorio y para dos ejecuciones diferentes el procesador que logra escribir en el ducto puede ser diferente. No importa el tipo de dato que intenten escribir los procesadores.

5. Prioridad. Bajo esta regla cada procesador tiene alguna prioridad predeterminada. Normalmente el índice del procesador determina su prioridad. Cuando ocurre una escritura concurrente el procesador con mayor prioridad es el que tiene éxito en la escritura.

### II.1.5 Rejilla reconfigurable (R-Mesh)

En (Vaidyanathan y Trahan, 2004) se define formalmente la rejilla reconfigurable como un arreglo lineal de procesadores numerados $0, 1, \cdots, N-1$ en el cual, para cada $0 \leq i < N$, el procesador $i$ está conectado al procesador $i \pm 1$, si es que existe. Cada procesador posee dos puertos denominados Este (E) y Oeste (O) que permiten la conexión con sus vecinos. Además cada procesador es capaz de fusionar sus dos puertos formando con esto un ducto que atraviesa al mismo procesador y conecta a sus vecinos como se ve en la Figura 1. A este arreglo en particular se le llama rejilla reconfigurable de una dimensión con $N$ procesadores.

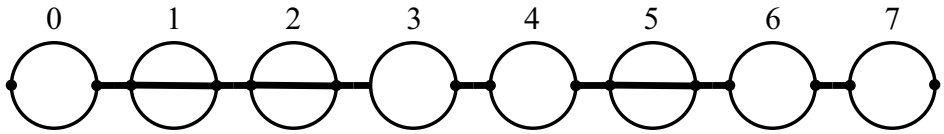



Figura 1: Rejilla reconfigurable de una dimensión. Note que los procesadores 1, 2 y 5 han fusionado sus respectivos puertos.

A partir de la estructura anterior se puede construir una rejilla reconfigurable de dos dimensiones. Es necesario cambiar el arreglo lineal de procesadores por una rejilla bidimensional con $R$ renglones y $C$ columnas. La también llamada $R \times C$ R-Mesh está compuesta por renglones denominados $0, 1, \cdots, R-1$ y columnas $0, 1, \cdots, C-1$, con el renglón 0 en la parte superior y la columna 0 en la parte izquierda de la rejilla como se muestraen la Figura 2.

La diferencia estriba en que cada procesador ahora posee cuatro puertos, siguiendo el mismo esquema, denominados Norte (N), Sur (S), Este (E) y Oeste (O) que conecta al procesador con sus respectivos vecinos, si es que existen. De igual forma al modelo de una dimensión, cada procesador es capaz de crear internamente conexiones entre

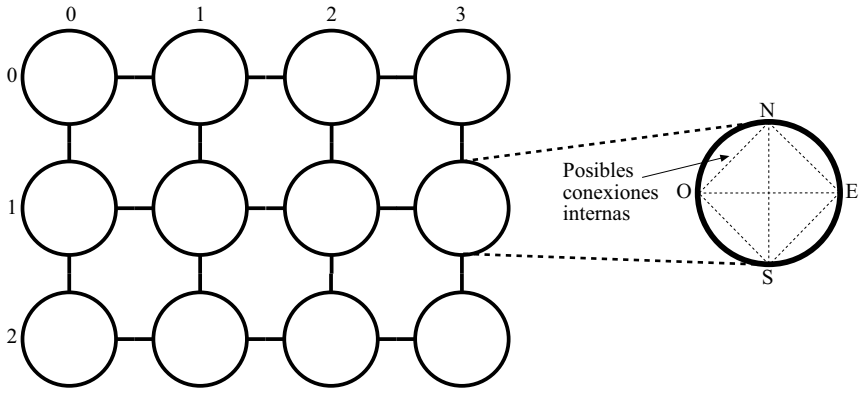



Figura 2: Estructura de un R-Mesh de $3 \times 4$.

sus puertos, que aunadas a las conexiones fijas (entre procesadores) se crean diferentes arreglos de ductos entre la estructura de procesadores. Cada una de las 15 configuraciones posibles se muestran en la Figura 3. La notación utilizada para representar las conexiones es la siguiente: dentro de un juego de llaves se colocan las iniciales de los cuatro puertos separados en grupos por comas. Si un grupo tiene 2 o más puertos indica que ellos están conectados entre si.

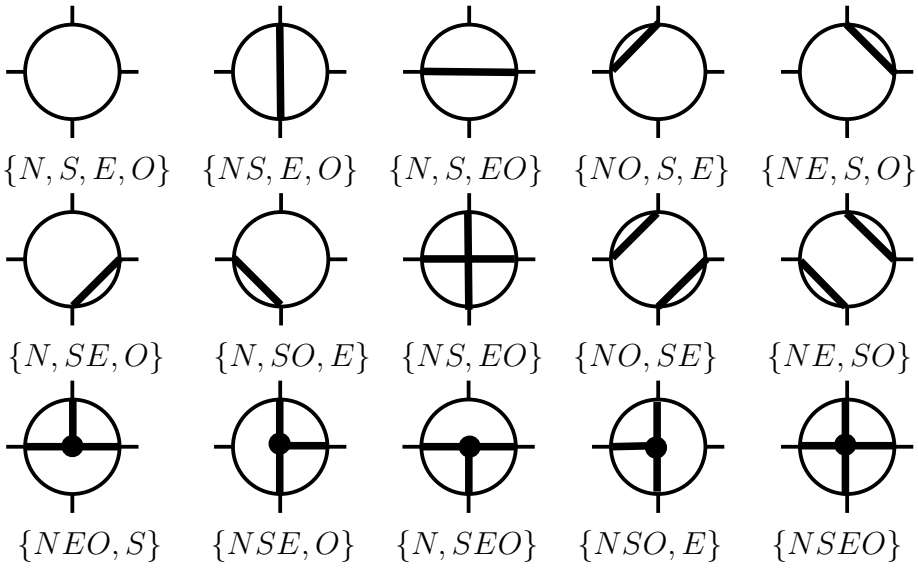



Figura 3: Posibles configuraciones internas para cada procesador de un R-Mesh

### II.1.6 Rejilla reconfigurable lineal (LR-Mesh)

La definición dada en (Vaidyanathan y Trahan, 2004) del modelo LR-Mesh indica que este modelo sólo permite que cada puerto de cada procesador se pueda conectar a lo más a otro puerto dentro del mismo procesador. Esto implica que un ducto no se puede bifurcar como en el caso del R-Mesh donde cada procesador puede fusionar 3 o 4 de sus puertos. Esta restricción en el modelo reduce la cantidad de conexiones internas a 10, las cuales se muestran en la Figura 4. Los ductos creados en el modelo R-Mesh se denominan *ductos lineales*.

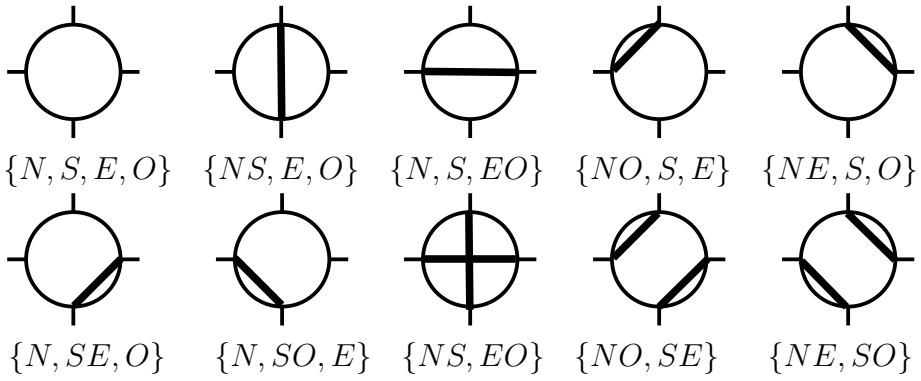



Figura 4: Posibles configuraciones internas para cada procesador de un LR-Mesh

### II.1.7 Rejilla Horizontal-Vertical (HVR-Mesh)

El modelo HVR-Mesh, al igual que el modelo LR-Mesh, es una versión restrictiva del modelo R-Mesh. Un HVR-Mesh sólo permite cuatro configuraciones internas del modelo R-Mesh: $\{N, S, E, O\}$, $\{NS, E, O\}$, $\{N, S, EO\}$ y $\{NS, EO\}$, ver Figura 5. Esto implica que cada ducto del modelo HVR-Mesh está restringido a operar dentro de un renglón o una columna del arreglo bidimensional. Es decir, un ducto es una línea vertical u horizontal.

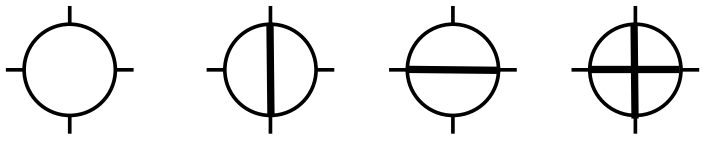



Figura 5: Configuraciones posibles de cada procesador en el modelo HVR-Mesh

### II.1.8 Rejilla de fusión (FR-Mesh)

Este modelo sólo permite dos configuraciones para cada procesador: $\{NS, EO\}$ y $\{NSEO\}$. Existen ductos verticales y horizontales que atraviesan todos los procesadores y cada procesador decide si fusiona el ducto vertical con el horizontal.

### II.1.9 Rejilla reconfigurable dirigida (DR-Mesh)

El modelo DR-Mesh es similar al modelo R-Mesh ya que permite las mismas configuraciones para sus puertos. La diferencia radica en los ductos. En el modelo R-Mesh cuando se realiza un proceso de escritura en algún ducto, la información se propaga en ambas direcciones. En el modelo DR-Mesh los ductos son dirigidos, es decir cada ducto propaga la información en una sóla dirección predeterminada. Cada procesador tiene 4 puertos de salida y 4 puertos de entrada, conectados a ductos de salida y ductos de entrada, respectivamente. Un ducto de entrada sólo permite introducir información al procesador y un ducto de salida sólo puede llevar información fuera del procesador.

### II.1.10 Poder computacional

El poder computacional de un modelo de computación es una característica importante. Para determinar el poder computacional de un modelo generalmente se compara con algún otro modelo ya estudiado. Formalmente el poder computacional se define como sigue (Delgado, 2003):

**Definición 1.** Sean $A$ y $B$ dos modelos distintos de cómputo paralelo, el modelo $A$ es computacionalmente igual de poderoso que el modelo $B$, si $A$ reproduce un ciclo de máquina de $B$ en un tiempo constante y viceversa, permitiendo un aumento polinomial en el número de procesadores. Asimismo, si $A$ simula a $B$ en tiempo constante pero no lo contrario, entonces se dice que $A$ es computacionalmente más poderoso que $B$.

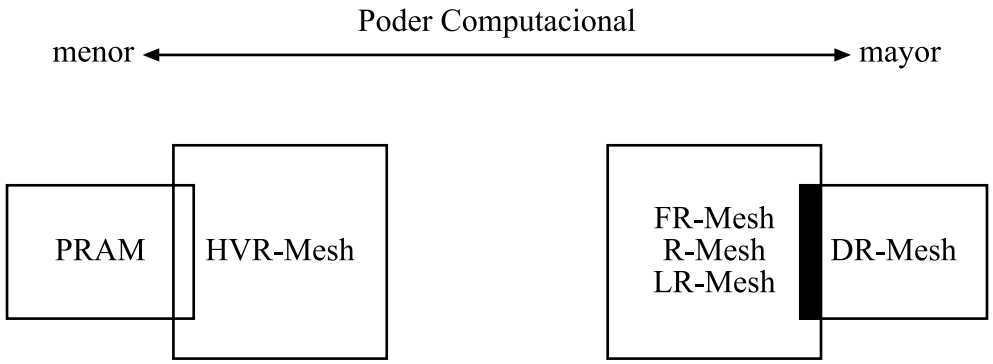



Figura 6: Comparación del poder computacional entre el modelo PRAM y algunos modelos reconfigurables.

En la Figura 6 se puede apreciar el poder computacional relativo entre los modelos descritos. EL modelo PRAM con escrituras exclusivas es menos poderoso que el modelo HVR-Mesh (Vaidyanathan y Trahan, 2004). El modelo PRAM con escrituras concurrentes es tan poderoso como el modelo HVR-Mesh. Cabe notar que la habilidad de permitir escrituras concurrentes no aumenta el poder computacional de los modelos reconfigurables, como ocurre en el modelo PRAM. El modelo HVR-Mesh es menos poderoso que el resto de los modelos reconfigurables.

La habilidad de fusionar ductos es aparentemente más poderosa que la habilidad de segmentar ductos. Si un modelo permite fusionar ductos entonces la habilidad de segmentar los ductos no agrega mayor poder computacional al modelo (Vaidyanathan y Trahan, 2004). Por lo tanto, los modelos R-Mesh y FR-Mesh son exactamente igual

de poderosos. El modelo LR-Mesh, se creía menos poderoso que el modelo R-Mesh aunque no hubiera una demostración formal al respecto. Los resultados obtenidos por Omer Reingold (Reingold, 2005) implican que los modelos R-Mesh y LR-Mesh son computacionalmente igual de poderosos. En el caso del modelo DR-Mesh, se cree que la dirección en los ductos aumenta el poder computacional de los modelos dirigidos, pero no se tiene la certeza. Los modelos dirigidos son al menos tan poderosos como sus contrapartes no dirigidas.

# Capítulo III

# Máquina de Turing

Alan Turing, matemático inglés, definió en 1936 una máquina abstracta que posteriormente se llamaría Máquina de Turing. El propósito de Turing no fue el de inventar una computadora. La idea era el de explorar los problemas que son posibles resolver mediante el uso de la lógica. La máquina de Turing determinística es uno de los modelos de computación secuencial clásicos y uno de los más estudiados.

Dentro de la Teoría de la Computación existe una vertiente llamada *Teoría de la Complejidad* que se encarga de determinar la cantidad de recursos computacionales que se requieren para resolver un problema dado. Los recursos que normalmente se contabilizan son *el tiempo*, esto es, cuántos pasos requiere un algoritmo para resolver un problema, y *el espacio*, que tanta memoria requiere un algoritmo para resolver algún problema.

Uno de los modelos más estudiados dentro de la Teoría de la Computación es la máquina de Turing. Se tiene un gran conocimiento de la cantidad de tiempo y espacio que requiere una máquina de Turing para resolver una gran variedad de problemas. Gracias a esto, el modelo como tal sirve de referencia para poner en contexto a otros modelos de computación. Se puede comparar la cantidad de tiempo (o espacio) que requiere un modelo $x$ para resolver un problema $\pi$ y la cantidad de tiempo (o espacio) que requiere una máquina de Turing para resolver el mismo problema. Con esto es posible determinar el poder computacional de algún modelo con respecto a la máquina de Turing o con algún otro.

Se describen a continuación unas variantes de la máquina de Turing así como dos

problemas en específico: CYCLE y USTCON. Se muestran los algoritmos que resuelven estos dos problemas en una máquina de Turing y en un modelo reconfigurable.

## III.1 Máquina de Turing determinística

Una máquina de Turing consiste de:

- Una cinta dividida en celdas consecutivas. Cada celda almacena un carácter de un alfabeto finito. El alfabeto contiene un carácter *blanco* y uno o más caracteres. La cinta es infinita en ambas direcciones. Las celdas en las cuales no se haya escrito algún carácter contienen el carácter blanco.

- Una cabeza que puede leer y escribir caracteres en la cinta y tiene la habilidad de efectuar un movimiento a la izquierda o a la derecha.

- Un registro de estado que almacena el estado actual de la máquina. El número total de estados es finito y siempre existe un estado especial de inicio.

- Una función de transición que indica a la máquina qué carácter escribir, el movimiento que se debe efectuar, y el siguiente estado dados el carácter y el estado actual. Si no se especifica en la función de transición la actual combinación de estado y carácter, entonces la máquina termina.

### III.1.1 Máquina de Turing de una cinta

Una Máquina de Turing de una cinta se define como $M = (Q, \Gamma, s, b, F, \delta)$ donde:
$Q$ es un conjunto finito de estados.
$\Gamma$ es un conjunto finito que contiene el alfabeto de la cinta.
$s \in Q$ es el estado inicial.

$b \in \Gamma$ es el carácter blanco.

$F \subseteq Q$ es el conjunto de estados finales.

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ es una función parcial llamada función de transición, donde $L$ y $R$ representan un movimiento de la cabeza de escritura a la izquierda y derecha respectivamente.

### III.1.2 Máquina de Turing de *k*-cintas

En forma similar se define una Máquina de Turing de $k$-cintas como $M = (Q, \Gamma, s, b, F, \delta)$ donde:

$Q, \Gamma, s, b, F$ se definen en forma similar al caso de una máquina de Turing de una cinta.

La función de transición ahora contempla las $k$ cintas.

$\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{L, R, S\})^k$ es una función parcial llamada función de transición, donde $L$ y $R$ representan un movimiento de la cabeza de escritura a la izquierda y derecha respectivamente. $S$ es no desplazamiento.

## III.2 Máquina de Turing no determinística

Formalmente la máquina de Turing no determinística se define como $M = (Q, \Gamma, s, b, F, \delta)$ donde:

$Q, \Gamma, s, b, F$ se definen en forma similar al caso de una máquina de Turing determinística.

$\delta : Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$ es una función parcial llamada función de transición, donde $L$ y $R$ representan un movimiento de la cabeza de escritura a la izquierda o a la derecha, respectivamente.

En una máquina de Turing determinística para cada combinación de estado y carácter se permite un nuevo estado. La máquina de Turing no determinística para

una combinación de estado y carácter dados la función de transición define un conjunto de cero o más estados. En cada estado la máquina de Turing no determinística ejecuta todos los posibles estados definidos en la función de transición. Se puede ver que una máquina de Turing no determinística funciona como un árbol computacional. Cada uno de los nodos del árbol representa una combinación de estado y carácter específico, y cada uno de los descendientes son los estados definidos por la función de transición. Si una de las ramas del árbol termina en un estado final de aceptación, entonces la máquina de Turing no determinística termina y acepta la entrada.

## III.3 Máquina de Turing Simétrica

Una Máquina de Turing Simétrica es una Máquina de Turing no Determinística que posee una particularidad adicional, cada movimiento de la máquina es "reversible". Para que esto se pueda realizar se permite a la máquina leer dos caracteres al mismo tiempo en cada una de las cintas. Esto es, cada movimiento hacia delante se puede revertir.

En (Lewis y Papadimitriou, 1982) se define formalmente la Máquina de Turing Simétrica como $M = (K, \Sigma, \Sigma_0, k, \Delta, s, F)$ donde:
$K, \Sigma, s, F$ corresponden a la definición de $Q, \Gamma, s, F$ de la máquina de Turing determinística respectivamente.
$\Sigma_0 \subseteq \Sigma$ es el alfabeto de la cinta de entrada.
$k > 0$ es la cantidad de cintas.
$\Delta$ es un conjunto finito de transiciones. Cada transición perteneciente al conjunto $\Delta$ es de la forma $(p, t_1, \cdots, t_k, q)$ donde $p$ y $q$ son estados, $k$ es la cantidad de cintas, y $t_1, \cdots, t_k$ son triadas de la cinta. Existen dos configuraciones posibles para cada triada: $(ab, D, cd)$ donde $a, b, c, d \in \Sigma$ y $D$ es +1 ó -1, y $(a, 0, b)$, donde $a, b \in \Sigma$.

Una transición de la forma $(p, (a, 0, b), q)$ implica que si $M$ se encuentra en el estado $p$ y lee el carácter $a$, podría rescribir el carácter $b$ en $a$ y cambiar al estado $q$ sin mover la cabeza de la cinta. Una transición de la forma $(p, (ab, +1, cd), q)$ significa que si $M$ se encuentra en el estado $p$, lee el carácter $a$ y la celda a la derecha contiene el carácter $b$, $M$ puede rescribir estas celdas con $c$ y $d$ respectivamente, mover la cabeza un espacio a la derecha y pasar al estado $q$. La forma $(p, (ab, -1, cd), q)$ significa que si $M$ se encuentra en el estado $p$, lee el carácter $b$ y la celda a la izquierda contiene el carácter $a$, $M$ puede rescribir estas celdas con $d$ y $c$ respectivamente, mover la cabeza un espacio a la izquierda y pasar al estado $q$.

## III.4 Máquinas de Turing y Arquitecturas reconfigurables

Como se explicó al inicio del capítulo, los recursos que normalmente se contabilizan son *el tiempo* y *el espacio*. Existen otros recursos que se utilizan para medir de alguna forma el desempeño de un algoritmo tal como la cantidad de procesadores requeridos para resolver un problema en paralelo.

A continuación se describen algunas clases de complejidad en función del espacio requerido por el algoritmo y la relación de estas clases con algunos modelos reconfigurables.

### III.4.1 Clases de complejidad

En función del espacio requerido existen tres clases de complejidad que tienen una relacion con tres modelos reconfigurables: *nondeterministic logspace* (NL), *logspace* (L)

y *Symetric logspace* (SL) y se definen como sigue:

NL es la clase de complejidad que alberga los problemas de decisión que puede resolver una máquina de Turing no determinística requiriendo únicamente una cantidad de espacio de memoria logarítmico en función de la entrada.

L es la clase de complejidad que contiene a los problemas de decisión que puede resolver una máquina de Turing determinística utilizando un espacio logarítmico de memoria en función de la entrada.

Para comprender la función de la clase SL es necesario ver el problema de conectividad entre dos vértices (USTCON). La definición formal del problema USTCON (Alvarez y Greenlaw, 1996) es la siguiente: dado un grafo no dirigido $G = (V, E)$, es decir con $|V| = N$ vértices y $|E|$ aristas, y dos vértices específicos $s, t \in V$. El problema es encontrar si $s$ y $t$ están conectados. Esto es, la solución de este problema responde a la pregunta ¿Existe una trayectoria que comunica $s$ y $t$?. Otra forma de ver el problema es la de decidir si los vértices $s$ y $t$ se encuentran en el mismo componente conectado.

La clase a la que se destinaba el problema USTCON antes de 1982 era NL aunque no se requería una máquina no determinística para resolverlo. (Lewis y Papadimitriou, 1982) definieron la clase SL en función del problema USTCON. Esta clase de complejidad computacional contiene todos los problemas que son reducibles logarítmicamente a USTCON. Por supuesto esto implica que el problema de conectividad es completo para la clase SL. Además demostraron que $L \subseteq SL \subseteq NL$.

### III.4.2 Relación entre modelos reconfigurables y clases de complejidad

En (Vaidyanathan y Trahan, 2004) se establece que los modelos reconfigurables LR-Mesh, R-Mesh y DR-Mesh pueden resolver cualquier problema que pertenece a las clases L, SL y NL, respectivamente, en tiempo constante.

Para demostrar que el modelo LR-Mesh puede resolver cualquier problema que pertenezca a la clase de complejidad L se toma un problema completo $\alpha$ para la clase L y se resuelve en el modelo LR-Mesh en tiempo constante. Cualquier otro problema $\beta$ que pertenezca a la clase L, se puede reducir al problema $\alpha$ y resolverlo en el modelo LR-Mesh.

La demostración del resto de las relaciones se efectúan en forma similar. Se toma un problema completo para la clase de complejidad deseada y se resuelve en el modelo reconfigurable correspondiente en tiempo constante.

A continuación se presentan las demostraciones de las relaciones entre el modelo LR-Mesh y la clase L, y R-Mesh y SL. Se describen dos problemas: CYCLE y USTCON. Estos problemas son completos para las clases de complejidad L y SL, respectivamente. También se muestra un algoritmo que resuelve cada problema en una máquina de Turing y en un modelo reconfigurable.

### III.4.3 Problema: CYCLE (completo para la clase L)

**Descripción**: Sea $G = (V, E)$ un grafo dirigido en donde el grado de entrada y de salida para cada vértice es igual a 1. Dado dos vértices $u, v \in V$, determinar si $u$ y $v$ se encuentran en el mismo ciclo.

Solución en el modelo LR-Mesh de tamaño $n \times n$.

**Entrada**: La matriz de adyacencia (ver el Capítulo III) de $n \times n$, $A_G = [a_{i,j}]$, del grafo dirigido $G = (V, E)$ (el procesador $P_{i,j}$ donde $0 \leq i, j < n$ del modelo almacena el elemento $a_{ij}$ de la matriz de adyacencia. Los vértices de $V$ están numerados $0, 1, \cdots, n-1$) y dos vértices $u, v \in V$.

**Salida**: $S$=1 si $u$ y $v$ se encuentran en el mismo ciclo, $S$=0 en caso contrario.

El algoritmo para cada procesador $P_{i,j}$, $0 \leq i, j < n$, es el siguiente:

BEGIN

1. a. if $i \neq j$ then

$P_{i,j}$ configura $\{NS, EO\}$

else

$P_{i,j}$ configura $\{N, S, E, O\}$

b. if $a_{i,j} = 1$ then

$P_{i,j}$ envía una señal $\alpha$ a través del puerto $N$ y una señal $\beta$ por el puerto $E$

if $i = j$ then

$P_{i,j}$ lee los puertos $N$ y $E$ y almacena el valor en *señalN(i)* y *señalE(i)* respectivamente

2. a. case $i = j$

if *señalN(i)*= $\alpha$

if *señalE(i)*= $\beta$

$P_{i,j}$ configura $\{NE, S, O\}$

else

$P_{i,j}$ configura $\{NO, S, E\}$

else

if *señalE(i)*= $\beta$

$P_{i,j}$ configura $\{N, SE, O\}$

else

$P_{i,j}$ configura $\{N, SO, E\}$

case $i > j$

if $a_{i,j} = 1$ then

$P_{i,j}$ configura $\{NE, S, O\}$

else

$P_{i,j}$ configura $\{NS, EO\}$

case $i < j$

if $a_{i,j} = 1$ then

$P_{i,j}$ configura $\{N, SO, E\}$

else

$P_{i,j}$ configura $\{NS, EO\}$

b. $P_{u,u}$ transmite una señal $\alpha$ por sus cuatro puertos.

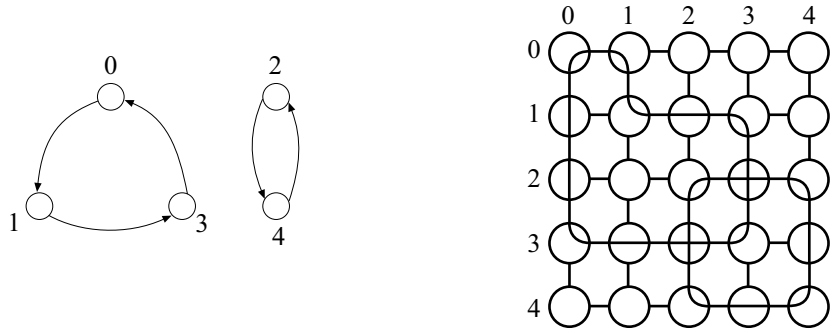
$P_{v,v}$ lee del ducto y almacena el alor en *senal*

3. a. if $senal = \alpha$ then $S = 1$

else $S = 0$

END

En la Figura 7a se muestra un grafo cíclico. La matriz de adyacencia de dicho grafo se observa en la Figura 7b. El algoritmo para el modelo LR-Mesh empotra el grafo de acuerdo a esta matriz de adyacencia. El procesador $P_{i,j}$ posee el elemento $a_{i,j}$ de la matriz de adyacencia. Si este valor es igual a 1 entonces se crea un ducto




$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

b)

Figura 7: a) Grafo dirigido $G$, el grado de entrada y salida de cada vértice es igual a 1, b) la matriz de adyacencia del grafo $G$ y c) El grafo $G$ empotrado en el modelo LR-Mesh.

entre dicho procesador y los procesadores de la diagonal principal en forma horizontal y vertical como se muestra en la Figura 7c. Cada ducto cíclico conecta únicamente los procesadores de un mismo componente conectado. El procesador $P_{u,u}$ transmite una señal predeterminada por el ducto. Un vértice $x$ está en el componente conectado de $u$ si y sólo si $P_{x,x}$ detecta dicha señal. Por tanto si $P_{v,v}$ detecta la señal se puede afirma que existe una trayectoria entre los vértices $u$ y $v$, en caso contrario $u$ y $v$ se encuentran en diferentes componentes conectados.

Solución en una Máquina de Turing Determinística.

**Entrada**: Los identificadores de los vértices $u$ y $v$, seguido del conjunto de aristas de un grafo dirigido $G = (V, E)$ de la forma $(u, v)((E_1)(E_2)\dots(E_{|E|}))$. $E_i$ es de la forma $v_a, v_b$ donde $v_a, v_b \in V$ para toda $0 < i \leq |E|$ y $0 < a, b < |V|$ . Ejemplo (4,2)((1,3)(5,2)(4,5)(3,1)(4,5))

**Salida**: La máquina termina en estado de aceptación si $u$ y $v$ se encuentran en el

mismo ciclo. De lo contrario la máquina termina en un estado de rechazo.

Estado inicial : $q_{00}$

Estado final aceptado : $q_{17}$

Estado final rechazo : $q_{18}$

Estados intermedios : $q_{01}, q_{02}, q_{03}, q_{04}, q_{05}, q_{06}, q_{07}, q_{08}, q_{09}, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}, q_{15}, q_{16}$

Alfabeto $\Sigma$ : $\{\mathbf{0}, \mathbf{1}, (, ), \mathbf{,}\} \cup \{b\}$

Tabla de Transición

Regla 00 $q_{00}$ [ $\boldsymbol{x}$, $\boldsymbol{x}$]→ $q_{01}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 01 $q_{01}$ [ $\boldsymbol{(}$, $\boldsymbol{x}$]→ $q_{01}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{(}$, $\boldsymbol{R}$ ) ]

Regla 02 $q_{01}$ [ $\boldsymbol{0}$, $\boldsymbol{x}$]→ $q_{01}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{0}$, $\boldsymbol{R}$ ) ]

Regla 03 $q_{01}$ [ $\boldsymbol{1}$, $\boldsymbol{x}$]→ $q_{01}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{1}$, $\boldsymbol{R}$ ) ]

Regla 04 $q_{01}$ [ $\boldsymbol{,}$, $\boldsymbol{x}$]→ $q_{01}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{,}$, $\boldsymbol{R}$ ) ]

Regla 05 $q_{01}$ [ $\boldsymbol{)}$, $\boldsymbol{x}$]→ $q_{02}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{)}$, $\boldsymbol{S}$ ) ]

Regla 06 $q_{02}$ [ $\boldsymbol{0}$, $\boldsymbol{x}$]→ $q_{02}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 07 $q_{02}$ [ $\boldsymbol{1}$, $\boldsymbol{x}$]→ $q_{02}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 08 $q_{02}$ [ $\boldsymbol{,}$, $\boldsymbol{x}$]→ $q_{02}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 09 $q_{02}$ [ $\boldsymbol{)}$, $\boldsymbol{x}$]→ $q_{02}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 10 $q_{02}$ [ $\boldsymbol{(}$, $\boldsymbol{x}$]→ $q_{03}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 11 $q_{03}$ [ $\boldsymbol{(}$, $\boldsymbol{x}$]→ $q_{04}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 12 $q_{04}$ [ $\boldsymbol{x}$, $\boldsymbol{0}$]→ $q_{04}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 13 $q_{04}$ [ $\boldsymbol{x}$, $\boldsymbol{1}$]→ $q_{04}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 14 $q_{04}$ [ $\boldsymbol{x}$, $\boldsymbol{,}$]→ $q_{04}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 15 $q_{04}$ [ $\boldsymbol{x}$, $\boldsymbol{)}$]→ $q_{04}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 16 $q_{04}$ [ $\boldsymbol{x}$, $\boldsymbol{(}$]→ $q_{05}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{R}$ ) ]

Regla 17 $q_{05}$ [ $\boldsymbol{0}$, $\boldsymbol{0}$]→ $q_{05}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{R}$ ) ]

Regla 18 $q_{05}$ [ $\boldsymbol{1}$, $\boldsymbol{1}$]→ $q_{05}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{R}$ ) ]

Regla 19 $q_{05}$ [ $\boldsymbol{,}$, $\boldsymbol{,}$]→ $q_{08}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{R}$ ) ]

Regla 20 $q_{05}$ [ $\boldsymbol{0}$, $\boldsymbol{1}$]→ $q_{06}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 21 $q_{05}$ [ $\boldsymbol{1}$, $\boldsymbol{0}$]→ $q_{06}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 22 $q_{06}$ [ $\boldsymbol{0}$, $\boldsymbol{x}$]→ $q_{06}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 23 $q_{06}$ [ $\boldsymbol{1}$, $\boldsymbol{x}$]→ $q_{06}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 24 $q_{06}$ [ $\boldsymbol{,}$, $\boldsymbol{x}$]→ $q_{06}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 25 $q_{06}$ [ $\boldsymbol{)}$, $\boldsymbol{x}$]→ $q_{06}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 26 $q_{06}$ [ $\boldsymbol{(}$, $\boldsymbol{x}$]→ $q_{07}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 27 $q_{07}$ [ $\boldsymbol{x}$, $\boldsymbol{0}$]→ $q_{07}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 28 $q_{07}$ [ $\boldsymbol{x}$, $\boldsymbol{1}$]→ $q_{07}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 29 $q_{07}$ [ $\boldsymbol{x}$, $\boldsymbol{(}$]→ $q_{05}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{R}$ ) ]

Regla 30 $q_{08}$ [ $\boldsymbol{0}$, $\boldsymbol{0}$]→ $q_{08}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{R}$ ) ]

Regla 31 $q_{08}$ [ $\boldsymbol{1}$, $\boldsymbol{1}$]→ $q_{08}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{R}$ ) ]

Regla 32 $q_{08}$ [ $\boldsymbol{,}$, $\boldsymbol{,}$]→ $q_{17}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 33 $q_{08}$ [ $\boldsymbol{0}$, $\boldsymbol{1}$]→ $q_{09}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 34 $q_{08}$ [ $\boldsymbol{1}$, $\boldsymbol{0}$]→ $q_{09}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 35 $q_{09}$ [ $\boldsymbol{0}$, $\boldsymbol{x}$]→ $q_{09}$ [ ( $\boldsymbol{x}$, $\boldsymbol{L}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 36 $q_{09}$ [ $\boldsymbol{1}$, $\boldsymbol{x}$]→ $q_{09}$ [ ( $\boldsymbol{x}$, $\boldsymbol{L}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 37 $q_{09}$ [ $\boldsymbol{,}$, $\boldsymbol{x}$]→ $q_{10}$ [ ( $\boldsymbol{x}$, $\boldsymbol{R}$ ), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Regla 38 $q_{10}$ [ $\boldsymbol{x}$, $\boldsymbol{0}$]→ $q_{10}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 39 $q_{10}$ [ $\boldsymbol{x}$, $\boldsymbol{1}$]→ $q_{10}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 40 $q_{10}$ [ $\boldsymbol{x}$, $\boldsymbol{,}$]→ $q_{10}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{L}$ ) ]

Regla 41 $q_{10}$ [ $\boldsymbol{x}$, $\boldsymbol{(}$]→ $q_{11}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$ ), ( $\boldsymbol{x}$, $\boldsymbol{R}$ ) ]

Regla 42 $q_{11}$ [ ***0***, ***x***]→ $q_{11}$ [ ( ***x***, ***R*** ), ( ***0***, ***R*** ) ]

Regla 43 $q_{11}$ [ ***1***, ***x***]→ $q_{11}$ [ ( ***x***, ***R*** ), ( ***1***, ***R*** ) ]

Regla 44 $q_{11}$ [ ***)***, ***x***]→ $q_{12}$ [ ( ***x***, ***S*** ), ( ***x***, ***S*** ) ]

Regla 45 $q_{12}$ [ ***x***, ***0***]→ $q_{12}$ [ ( ***x***, ***S*** ), ( ***x***, ***L*** ) ]

Regla 46 $q_{12}$ [ ***x***, ***1***]→ $q_{12}$ [ ( ***x***, ***S*** ), ( ***x***, ***L*** ) ]

Regla 47 $q_{12}$ [ ***x***, ***,***]→ $q_{12}$ [ ( ***x***, ***S*** ), ( ***x***, ***L*** ) ]

Regla 48 $q_{12}$ [ ***x***, ***(***]→ $q_{13}$ [ ( ***x***, ***S*** ), ( ***x***, ***R*** ) ]

Regla 49 $q_{13}$ [ ***0***, ***x***]→ $q_{13}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 50 $q_{13}$ [ ***1***, ***x***]→ $q_{13}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 51 $q_{13}$ [ ***,***, ***x***]→ $q_{13}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 52 $q_{13}$ [ ***)***, ***x***]→ $q_{13}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 53 $q_{13}$ [ ***(***, ***x***]→ $q_{14}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 54 $q_{14}$ [ ***0***, ***x***]→ $q_{13}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 55 $q_{14}$ [ ***1***, ***x***]→ $q_{13}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 56 $q_{14}$ [ ***)***, ***x***]→ $q_{13}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 57 $q_{14}$ [ ***,***, ***x***]→ $q_{13}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 58 $q_{14}$ [ ***(***, ***x***]→ $q_{15}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 59 $q_{15}$ [ ***0***, ***x***]→ $q_{15}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 60 $q_{15}$ [ ***1***, ***x***]→ $q_{15}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 61 $q_{15}$ [ ***,***, ***x***]→ $q_{15}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 62 $q_{15}$ [ ***)***, ***x***]→ $q_{15}$ [ ( ***x***, ***L*** ), ( ***x***, ***S*** ) ]

Regla 63 $q_{15}$ [ ***(***, ***x***]→ $q_{16}$ [ ( ***x***, ***R*** ), ( ***x***, ***S*** ) ]

Regla 64 $q_{16}$ [ ***0***, ***0***]→ $q_{16}$ [ ( ***x***, ***R*** ), ( ***x***, ***R*** ) ]

Regla 65 $q_{16}$ [ ***1***, ***1***]→ $q_{16}$ [ ( ***x***, ***R*** ), ( ***x***, ***R*** ) ]

Regla 66 $q_{16}$ [ ***,***, ***,***]→ $q_{18}$ [ ( ***x***, ***S*** ), ( ***x***, ***S*** ) ]

Regla 67 $q_{16}$ [ ***0***, ***1***]→ $q_{02}$ [ ( ***x***, ***S*** ), ( ***x***, ***S*** ) ]

Regla 68 $q_{16}$ [ $\boldsymbol{1}$, $\boldsymbol{0}$]$\rightarrow$ $q_{02}$ [ ( $\boldsymbol{x}$, $\boldsymbol{S}$), ( $\boldsymbol{x}$, $\boldsymbol{S}$ ) ]

Una regla de la forma $q_l[a, b] \rightarrow q_m[(x, W_1), (c, W_2)]$ donde $a, b, c, d \in \Sigma \cup \{x\}$ y $W_1, W_2 \in \{L, R, S\}$ se interpreta como sigue: Si la máquina de Turing se encuentra en el estado $q_l$, la cabeza de la cinta de entrada lee el carácter $a$ y la cabeza de la cinta de trabajo lee el carácter $b$, entonces la cabeza de la cinta de entrada realiza el movimiento especificado por $W_1$, la cabeza de la cinta de trabajo escribe el carácter $c$ y realiza el movimiento $W_2$ y la máquina de Turing pasa al estado $q_m$. El carácter $x$ implica en el caso de una lectura cualquier elemento del alfabeto $\Sigma$. Una escritura del tipo $(x, W)$ solo efectúa el movimiento $W$, la cinta no se altera.

A partir del sucesor del vértice $u$, se recorre el ciclo dentro del grafo. Dado que cada vértice del grafo tiene un grado de entrada y salida igual a 1, sólo se requiere mantener un registro con el identificador del vértice actual. El algoritmo compara el vértice actual con el vértice $v$. Si son iguales entonces el algoritmo termina, $u$ y $v$ se encuentran en el mismo ciclo. De lo contrario se compara el vértice actual con el vértice $u$, si son iguales el algoritmo termina ya que se recorrió el ciclo completo y no se encontró el vértice $v$. Si el vértice actual difiere de $u$ y $v$ entonces el sucesor del vértice actual se toma como el nuevo vértice actual y se repite el proceso. El algoritmo requiere $O(\log n)$ unidades de espacio.

### III.4.4 Problema USTCON. (Completo para la clase SL)

Solución en el modelo R-Mesh de tamaño $n \times n$. (Vaidyanathan y Trahan, 2004)

**Entrada**: La matriz de adyacencia $A_G = [a_{i,j}]$ del grafo no dirigido $G = (V, E)$, el procesador $P_{i,j}$ del modelo, donde $0 \leq i, j < n$, posee el elemento $a_{i,j}$ de la matriz de adyacencia. Los elementos de $V$ están numerados $0, 1, \cdots, n-1$. Dos vértices $s, t \in V$, el procesador $(s, s)$ y $(t, t)$ poseen esta información.

**Salida**: Procesador $(t, t)$ enciende una señal *conectado*$(t)$ si y sólo si en $G$ existe una trayectoria entre $s$ y $t$.

BEGIN

1. a. if $a(i, j) = 1$ OR $i = j$ then

$P_{i,j}$ configura $\{NSEO\}$

else

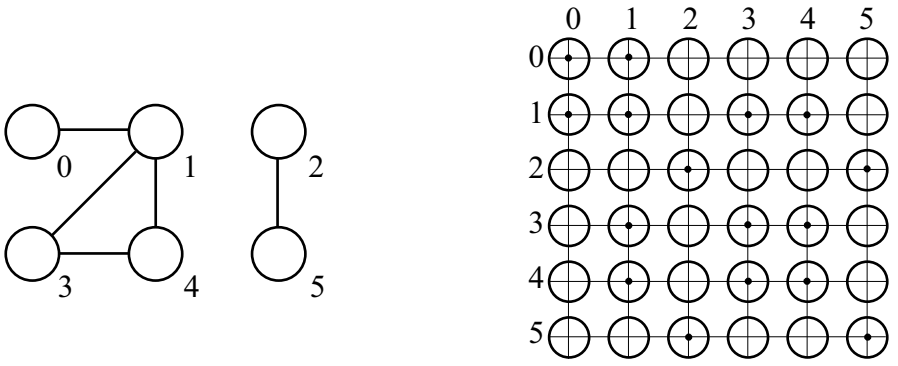
$P_{i,j}$ configura $\{NS, EO\}$

b. $P_{s,s}$ escribe un 1 en el puerto $N$

$P_{t,t}$ lee del puerto $N$ y almacena el resultado en *conectado*$(t)$

END

En la Figura 8a y Figura 8b se muestran un grafo $G$ y la matriz de adyacencia, respectivamente. Existen dos componentes conectados en el grafo $G$, uno compuesto por los vértices $\{0, 1, 3, 4\}$ y el otro por los vértices 2 y 5. En la Figura 8c se muestra la estructura de ductos necesaria para resolver el problema de conectividad en el modelo LR-Mesh dado el grafo $G$. Note la similitud de la matriz de adyacencia y la configuración interna de cada procesador. Si el elemento $a_{i,j}$ de la matriz de adyacencia es igual a 1 entonces el procesador $P_{i,j}$ adquiere una configuración $\{NSEO\}$. En caso contrario el procesador $P_{i,j}$ adquiere una configuracion interna $\{NS, EO\}$. Se puede observar que si




a) c)

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

b)

Figura 8: a) Grafo no dirigido $G$ de seis vértices b) la matriz de adyacencia del grafo $G$ y c) El grafo $G$ empotrado en el modelo R-Mesh.

el procesador $P_{0,0}$ transmite una señal predeterminada, esta señal es detectada por los procesadores $P_{1,1}, P_{3,3}$ y $P_{4,4}$. El procesador $P_{s,s}$ transmite una señal, si el procesador $P_{t,t}$ detecta dicha señal existe una trayectoria que une a los vértices $s$ y $t$ en $G$. En caso contrario $s$ y $t$ se encuentran en componentes conectados distintos.

Solución en una Máquina de Turing Simétrica. (Lewis y Papadimitriou, 1982).

**Entrada**: Lista de las etiquetas de cada vértice del grafo en binario, cada vértice separado por comas y la lista de todos los vértices de la forma $\{a, b\}$. El primer elemento de la lista de vértices representa a $s$ y el último elemento a $t$. La cinta de trabajo de $M$ contiene el vértice actual (vértice $s$ al inicio) y el vértice $t$.

**Salida**: La máquina termina en un estado de aceptación en el caso de que los vértices $s$ y $t$ estén en el mismo componente conectado. De lo contrario el proceso termina en un estado de rechazo.

La máquina $M$ puede realizar 3 acciones:

1. No analizar la arista actual y pasar a la arista que se encuentra en la derecha.
2. Regresar a la arista de la izquierda.
3. Recorrer la arista actual.
   (a) Revisa que la arista contiene al vértice actual
   (b) Reescribir el nodo actual por el nodo diferente que contiene la arista.
   (c) Si el nuevo nodo es $p$ terminar aceptando.
   (d) De otra forma, se reposiciona la cabeza para analizar la misma arista.

Si el vértice actual no se encuentra en la arista, si $M$ se mueve a la derecha de la última arista o a la izquierda de la primera o si se encuentra una inconsistencia en la cinta, el proceso termina.

La máquina sólo requiere mantener dos registros durante la ejecución, en uno se almacena el vértice actual y en el otro el vértice $t$. Cada uno de estos registros requiere un espacio de $\log n$ unidades (necesario para expresar en binario cualquier vértice de $G$), por lo tanto el espacio requerido por el algoritmo es de $O(\log n)$ unidades.

# Capítulo IV

# Expansores

Los grafos expansores son un tipo especial de grafos. En si mismos parecen una contradicción dentro de la teoría de grafos ya que poseen dos propiedades en aparencia antagónicas. Son grafos dispersos (tienen pocas aristas) y al mismo tiempo estan bien conectados (cualquier subconjunto de vértices posee un vecindario con una gran cantidad de vértices).

Existen una gran variedad de problemas disímiles en donde los expansores juegan un papel fundamental. Corrección de errores (Tanner, 1981), desaleatorización de algoritmos [(Karp *et al.*, 1985), (Naor y Naor, 1993), (Impagliazzo *et al.*, 1994), (Impagliazzo y Wigderson, 1997)], diseño de redes de comunicación (Pippenger, 1987), algoritmos de ordenamiento (Ajtai *et al.*, 1983), entre otros.

Debido a las múltiples aplicaciones de los expansores, la construcción explícita de este tipo de grafos se convirtió en un area interesante de estudio. Se han descrito familias de grafos que satisfacen la definición de expansor y se han propuesto métodos para la creación explícita de los mismos.

En el resto del capítulo se define formalmente los grafos expansores y algunas de sus propiedades (expansión de vértices y expansión espectral) y además se muestra un método para la construcción explícita de expansores.

## IV.1 Definición de expansores

Un grafo no dirigido $G = (V, E)$ tiene un conjunto de vértices $V$, donde $|V| = N$, y un conjunto de aristas $E$.

Para cada vértice $v \in V$, el vecindario de $v$ se denota como $\Gamma(v)$, y se define como el conjunto de nodos que están conectados a $v$ a través de una sola arista, más formalmente:

$$\Gamma(v) = \{u \in V | (u, v) \in E\}$$

El vecindario de un subconjunto $S$ de $V$ se define como la unión de los vecindarios de los vértices de $S$, esto es:

$$\Gamma(S) = \bigcup_{v \in S} \Gamma(v)$$

**Definición 2.** La *expansión de vértices* de un grafo $G$ se define como:

$$\min_{S \subset V, |S| \leq N/2} \frac{|\Gamma(S) \backslash S|}{|S|}$$

Se puede observar que la expansión de vértices de un grafo expresa la conectividad del grafo. Para cualquier subconjunto $S \subset V$ de a lo más $n/2$ vértices, el tamaño de su vecindario $\Gamma(S)$ es por lo menos $A$ veces el tamaño del subconjunto $S$. Un grafo completo posee la mejor expansión de vértices posible $\Theta(1)$, pero el grado de cada vértice es de $O(N)$. A un grafo se le llama expansor si el grado de cada vértice es constante y además tiene una buena expansión de vértices, $\Omega(1)$. Esto es la cantidad de vecinos de cualquier subconjunto $S$ es mayor o igual al tamaño del subconjunto.

### IV.1.1 Expansión espectral

El problema de determinar la expansión de vértices de un grafo está clasificado como co-NP hard (Blum *et al.*, 1981). Por lo tanto no se conoce un algoritmo que lo resuelva en tiempo polinomial. Pero existen otras formas de analizar ciertas propiedades de expansión de un grafo, es decir la conectividad que presenta un grafo. En particular, la expansión espectral de un grafo (dada por los valores propios de la matriz de adyacencia normalizada) es una buena medida para determinar una cota de la expansión de vértices del grafo.

**Definición 3.** La *matriz de adyacencia* $A$ de un grafo $G = (V, E)$, con vértices ordenados de $v_0$ a $v_{N-1}$, es una matriz de tamaño $N \times N$ cuyo elemento $a_{i,j}$ representa la cantidad de aristas que unen a los vértices $v_i$ y $v_j$.

La matriz de adyacencia $A$ define por completo al grafo $G$. Si el grafo $G$ es no dirigido entonces la matriz $A$ es una matriz simétrica. $G$ es un grafo sin autociclos si y sólo si $A_{ii} = 0$ para todo $v_i \in V$. Si $G$ es un multigrafo (más de una arista en algún par de vértices) el elemento correspondiente en la matriz de adyacencia es distinto a uno.

**Definición 4.** Sea $k \geq 2$ un entero. Se dice que el grafo $G$ es *k-regular* si, para cada $v_i \in V : \sum_{v_j \in V} A_{ij} = k$.

Esto es, un grafo se denomina $k$-regular si cada vértice del grafo tiene exactamente $k$ aristas. Por lo tanto, la suma de los elementos de la matriz de adyacencia por renglón o columna es igual a $k$.

**Definición 5.** La *matriz de adyacencia normalizada* (Reingold, 2005) $M$ de un grafo $G$ no dirigido y $D$-regular, es la matriz de adyacencia de $G$ dividida por $D$. $M_{ij} = \frac{A_{ij}}{D}$

**Valores Propios**

Los valores propios son un conjunto especial de escalares asociados a una matriz cuadrada. Normalmente la matriz representa a un sistema lineal de ecuaciones, aunque una matriz cuadrada, como ya se vio, puede representar un grafo cualquiera. El cálculo de los también llamados eigenvalores en un sistema lineal tiene una gran relevancia. Mediante los valores propios se puede efectuar la diagonalización de una matriz (es decir el convertir a una matriz en otra pero que compartan las mismas propiedades). Otra aplicación de los valores propios es el análisis de estabilidad de un sistema. Particularmente, el cálculo de los valores propios de una matriz de adyacencia de un grafo dado permite conocer la *expansión espectral* del grafo.

Sea $A$ una matriz cuadrada de tamaño $N \times N$. Si existe un vector $X \in \mathbb{R}^N \neq 0$ tal que

$$AX = \lambda X$$

para algun escalar $\lambda$, entonces $\lambda$ se llama el valor propio de $A$.

La ecuación, llamada ecuacion característica de $A$, permite determinar cada uno de los valores propios de $A$.

$$det(A - \lambda \cdot \mathbb{I}) = 0$$

donde $\mathbb{I}$ representa la matriz identidad.

Se resuelve el determinante de la matriz $A - \lambda \cdot \mathbb{I}$ y se obtiene una función $f(\lambda)$. Para calcular los valores propios de la matriz sólo es necesario entonces resolver la ecuación $f(\lambda) = 0$.

**Valores propios y expansión espectral**

Sea $G$ un grafo finito no dirigido de $N$ vértices y $D$-regular. Entonces la matriz de adyacencia normalizada $M$ de $G$ es una matriz simétrica de tamaño $N \times N$, por lo tanto $M$ posee $N$ valores propios reales, incluyendo multiplicidades (repeticiones), que se pueden escribir en orden decreciente:

$$|\lambda_0| \geq |\lambda_1| \geq \cdots \geq |\lambda_{N-1}|$$

Debido a la regularidad de $G$, $\lambda_0 = 1$ y el resto de los valores propios de $M$ tienen un valor absoluto menor igual que uno. Además $\lambda_0$ tiene multiplicidad (un valor propio $\lambda$ tiene una multiplicidad igual a $x$ si existen $x$ valores propios iguales a $\lambda$) 1 si y sólo si $G$ es un grafo conectado (Davidoff *et al.*, 2003).

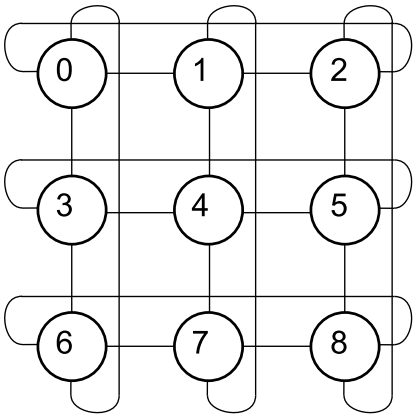
En (Tanner, 1984) y (Alon y Milman, 1985) se demuestra que el segundo valor propio mayor, $|\lambda_1|$, de una matriz que representa un grafo no dirigido $G$ y $D$-regular, tiene una relación directa con la noción de expansión de vértices de un grafo; entre más pequeño $|\lambda_1|$, el grafo tiene una mejor expansión de vértices. En [Alon, 1986] se demuestra que para un grafo no dirigido $G$, $D$-regular y una $|\lambda_1| < 1$ existe una $\epsilon > 0$ tal que para cualquier $S \subset V$ donde $|S| \leq |V|/2$, por lo menos $(1 + \epsilon) \cdot |S|$ vértices de $\{G - S\}$ están conectados por una arista a algún vértice de $S$.

**Definición 6.** Un grafo no dirigido, $D$-regular con $N$ vértices y cualquier $\lambda \geq |\lambda_1|$ se denomina grafo-($N$, $D$, $\lambda$)

A continuación se presentan dos ejemplos de grafos regulares con su respectiva expansión espectral. En cada uno se puede ver el grafo representado como vértices

y aristas, la matriz de adyacencia, la matriz de adyacencia normalizada y sus valores propios.

La Figura 9 a) muestra un torus de 9 vértices, note que cada vértice tiene 4 vecinos. La Figura 9 b) muestra la representación del grafo mediante la matriz de adyacencia, debido a que el grado de cada vértice es constante cada renglón y cada columna tiene exactamente 4 unos. La Figura 9 c) muestra la matriz de adyacencia normalizada, es decir la matriz de adyacencia dividida por 4.




a)

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \quad M = \begin{pmatrix} 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \end{pmatrix}$$

b) c)

Figura 9: Grafo-$(9, 4, \frac{1}{2})$. a) Representación visual del grafo, b) la matriz de adyacencia y c) la matriz de adyacencia normalizada.

Se puede apreciar en la tabla I que el valor propio mayor es igual a 1. El segundo y tercer valor propio tienen multiplicidad 4. $\lambda_1 = \frac{1}{2}$ por lo tanto y de acuerdo a la

| **Lambda** | **Valor** |
|---|---|
| $\|\lambda_0\|$ | 1 |
| $\|\lambda_1\|$ | $\frac{1}{2}$ |
| $\|\lambda_2\|$ | $\frac{1}{2}$ |
| $\|\lambda_3\|$ | $\frac{1}{2}$ |
| $\|\lambda_4\|$ | $\frac{1}{2}$ |
| $\|\lambda_5\|$ | $\frac{1}{4}$ |
| $\|\lambda_6\|$ | $\frac{1}{4}$ |
| $\|\lambda_7\|$ | $\frac{1}{4}$ |
| $\|\lambda_8\|$ | $\frac{1}{4}$ |

Tabla I: Valores propios del grafo-$(9, 4, \frac{1}{2})$, ordenados de mayor a menor.

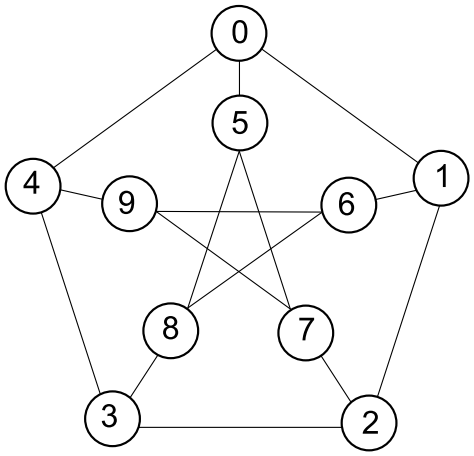
definición 6 el grafo es en efecto un grafo-$(9, 4, \frac{1}{2})$.

En la Figura 10 a) se muestra el grafo de Petersen, note que el grafo es un grafo 3-regular, cada vértice tiene exactamente tres vecinos. En la Figura 10 b) se muestra la matriz de adyacencia de dicho grafo y la Figura 10 c) muestra la matriz normalizada.

| **Lambda** | **Valor** |
|---|---|
| $\|\lambda_0\|$ | 1 |
| $\|\lambda_1\|$ | $\frac{2}{3}$ |
| $\|\lambda_2\|$ | $\frac{2}{3}$ |
| $\|\lambda_3\|$ | $\frac{2}{3}$ |
| $\|\lambda_4\|$ | $\frac{2}{3}$ |
| $\|\lambda_5\|$ | $\frac{1}{3}$ |
| $\|\lambda_6\|$ | $\frac{1}{3}$ |
| $\|\lambda_7\|$ | $\frac{1}{3}$ |
| $\|\lambda_8\|$ | $\frac{1}{3}$ |
| $\|\lambda_9\|$ | $\frac{1}{3}$ |

Tabla II: Valores propios del grafo-$(10, 3, \frac{2}{3})$, ordenados de mayor a menor.

En forma similar al grafo anterior, se puede apreciar en la tabla II que el valor propio mayor es igual a 1. El segundo valor propio es igual a $\frac{2}{3}$ y tiene un multiplicidad igual




a)

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

b)

$$M = \begin{pmatrix} 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 \end{pmatrix}$$

c)

Figura 10: Grafo-$(10, 3, \frac{2}{3})$. a) Representación visual del grafo, b) la matriz de adyacencia y c) la matriz de adyacencia normalizada.

a 4. El tercer valor propio tienen multiplicidad 5 y un valor absoluto igual a $\frac{1}{3}$. Debido a que el grafo tiene 10 vértices, es 3-regular y su $\lambda_1 = 2/3$ el grafo es un grafo-$(10, 3, \frac{2}{3})$.

## IV.2 Construcción de expansores

La construcción explícita de expansores es un problema interesante debido en gran parte a la gran cantidad de problemas donde los expansores se utilizan. Antes de crear explícitamente familias de expansores se demostró la existencia de los mismos. En (Pinsker, 1973) se demostró, por el método probabilístico, la existencia de grafos $D$-regulares que son expansores. Si se toma un grafo aleatorio $D$-regular para una $D > 2$ con una alta probabilidad el grafo es un expansor. Si la cantidad de vértices se aumenta, la probabilidad de que el grafo sea un expansor es mayor. Cuando la cantidad de vértices tiende a infinito, la probabilidad de que un grafo sea un expansor tiende a 1.

En la literatura se conocen varias familias de expansores de grado constante. Gregori Aleksandrovic Margulis (Margulis, 1973) definió por primera vez una familia de expansores de grado constante. En (Gabber y Galil, 1981), (Jimbo y Maruoka, 1987), (Morgenstern, 1994), entre otros, se describen familias de grafos $D$-regulares que son expansores.

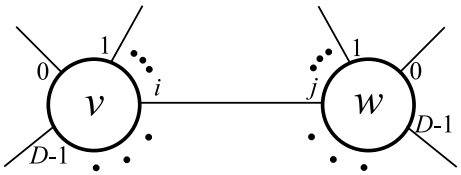
Reingold *et al.* (Reingold *et al.*, 2001) definieron un nuevo producto entre grafos llamado producto zig-zag. En función de dicho producto se logró la creación explícita de expansores de grado constante. A continuación se describe una representación de grafos $D$-regulares llamada "mapa rotacional", dos productos de grafos definidos en función del mapa rotacional (producto de reemplazo y producto zig-zag), y finalmente la utilización del producto zig-zag en la construcción de expansores de grado constante.

### IV.2.1 Mapa rotacional

Existen varias representaciones de grafos. La matriz de adyacencia es una de las formas más populares. El elemento $(u, v)$ de la matriz, como ya se expresó, indica la cantidad de aristas que conectan los vértices $u$ y $v$. Otra representación de un grafo consiste en listar todas sus aristas. Se tiene una lista de pares $(u, v)$ tal que entre los vértices $u$ y $v$ existe una arista que los conecte. El mapa rotacional es otra representación de grafos, particularmente para grafos $D$-regulares, que a continuación se describe.

Sea $G$ un grafo no dirigido, $D$-regular de $N$ vértices. Si cada vértice de $G$ está etiquetado en forma consistente $\{0, 1, \cdots, N-1\}$ y cada arista saliente de cada vértice está igualmente etiquetada en alguna forma arbitraria pero fija con $\{0, 1, \cdots, D-1\}$ entonces se puede referir a la $i$-ésima arista incidente al vértice $v$, esto es, la arista con etiqueta $i$ del vértice $v$, así como el $j$-ésimo vecino de $u$, es decir el vértice al que conecta la arista $j$ del vértice $u$. Bajo este marco es posible, a partir de un vértice, recorrer una arista y arribar a un nuevo vértice con la ventaja de tener información adicional. Se sabe por cuál arista se llegó al vértice actual y además se puede determinar cuál fue el vértice anterior. Se puede expresar la totalidad de un grafo regular mediante esta combinación de vértices y aristas, cada vértice con cada arista conducen a un determinado vértice. Esta representación de grafos regulares se denomina mapa rotacional.

**Definición 7.** Para un grafo $G$ no dirigido, $D$-Regular, el *mapa rotacional* (Reingold, 2005) $Rot_G : [N] \times [D] \rightarrow [N] \times [D]$ se define como sigue: $Rot_G(v, i) = (w, j)$ si la $i$-ésima arista incidente en $v$ conduce a $w$, y esta arista es la $j$-ésima arista incidente en $w$. Ver Figura 11.




$Rot_G(v, i) = (w, j))$

Figura 11: Representación gráfica de la definición del mapa rotacional de un grafo $D$-regular

### IV.2.2 Producto de reemplazo

Un producto entre dos grafos $G$ y $H$ es un grafo nuevo cuyo conjunto de vértices está formado por pares $(v, u)$, donde $v \in G$ y $u \in H$. El conjunto de aristas del nuevo grafo está definido por algún criterio en función de los grafos $G$ y $H$. Existen varios criterios diferentes que generan productos diferentes. Uno de estos criterios es el llamado producto de reemplazo. Si se tiene un grafo $G$ $D$-regular y un grafo $H$ $d$-regular, donde $d \ll D$, el producto de reemplazo produce un grafo $(d+1)$-regular que tiene la misma cantidad de componentes conectados que el grafo $G$. Es decir se reduce el grado del grafo $G$ mediante un aumento de la cantidad de vértices.

Sea un grafo $G$, $D_1$-regular con $N$ vértices, denominado $G(N, D_1)$ y un grafo $H$, $D_2$-regular con $D_1$ vértices, $H(D_1, D_2)$. Un ejemplo de $G$ y $H$ se muestra en la Figura 12. El producto de reemplazo entre $G$ y $H$ es un grafo con $D_1 \times N$ vértices y cada vértice tiene exactamente $D_2 + 1$ vecinos. La construcción del producto de reemplazo entre $G$ y $H$ es como sigue:

Cada vértice $v$ de $G$ se reemplaza por una copia del grafo $H$ como se ve en la Figura 13. Cada copia de $H$ mantiene sus aristas originales y cada una de estas copias se denomina nube de vértices $H_v$.

Al reemplazar cada vértice de $G$ por una copia de $H$, se produce un grafo con $N \times D_1$ vértices. Cada nube de vértices está aislada, es decir no existe una arista que

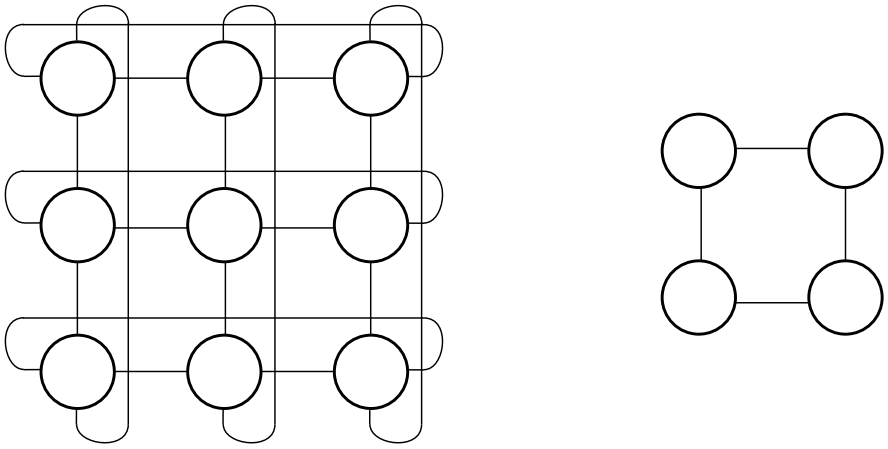



Figura 12: Ejemplo de un grafo $G(N, D_1)$ y un grafo $H(D_1, D_2)$

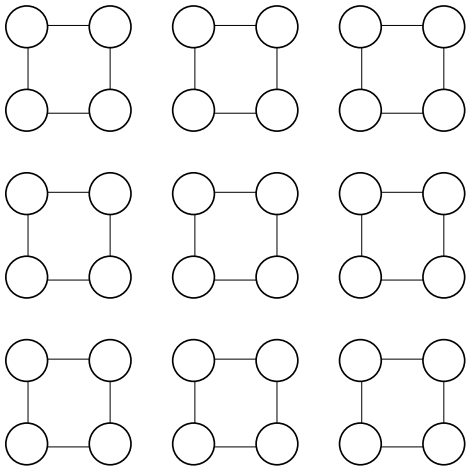


Figura 13: Cada vértice $G$ se reemplaza por una copia de $H$, a cada copia se le denomina nube de vértices

conecte vértices que pertenezcan a diferentes nubes. Note que cada vértice de $G$ tiene $D_1$ vecinos y cada vértice de $G$ está representado por $D_1$ vértices en el nuevo grafo. Es decir existe una correspondencia uno a uno entre la cantidad de vecinos de cada vértice de $G$ con la representación de sus vértices en el nuevo grafo. Si existe una arista entre los vértices $u$ y $v$ en $G$, se puede seleccionar uno de los vértices de $H_u$ y uno de los vértices de $H_v$ para crear esta arista en el nuevo grafo. Cada vértice en el nuevo grafo está unido a sus $D_2$ vecinos definidos por $H$ y a un vértice de una nube de vértices diferente, como se ve en la Figura 14.

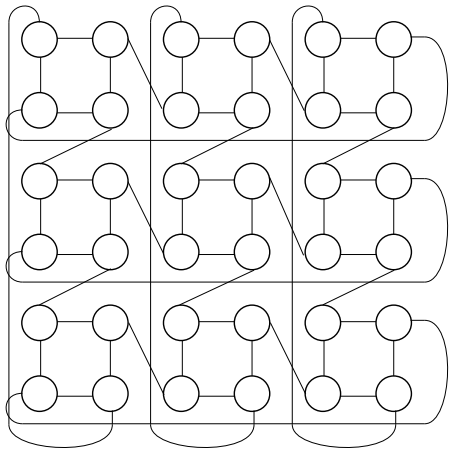


Figura 14: Grafo resultante del producto de reemplazo. El grado del grafo es $D_2 + 1$

El grafo resultante, es un grafo con $N \times D_1$ vértices y el grado de cada vértice se ha reducido de $D_1$ a $D_2 + 1$. El grafo tiene los mismos componentes conectados que $G$ ya que ninguna arista de $G$ se elimina.

Si se tiene el mapa rotacional del grafo $G$ y $H$, entonces cada vértice del producto de reemplazo de $G$ y $H$ está definido por pares $(v, a)$ donde $v \in G$ y $a \in H$. El rotacional del producto de reemplazo de $G$ y $H$ se define como sigue:

$Rot_{GH}(v, a, i) = (v, b, j)$ para toda $0 \leq i < D_2$ si $Rot_H(a, i) = (b, j)$

$Rot_{GH}(v, a, D2) = (w, b, D2)$ si $Rot_G(v, a) = (w, b)$

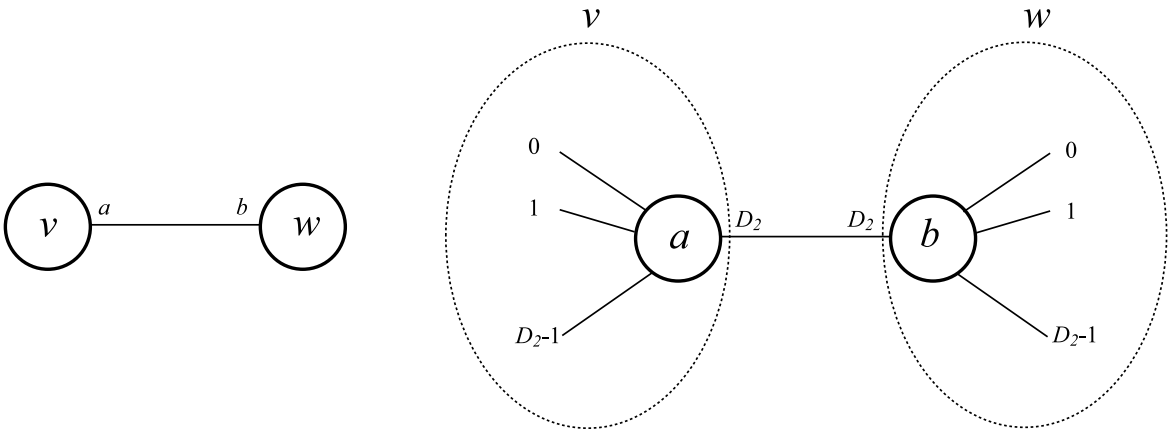



Figura 15: Si existe una arista entre los vértices $v$ y $w$, a través de las aristas $a$ y $b$ respectivamente, en $G$, entonces exisite una arista en el producto de reemplazo entre los vértices $(v,a)$ y $(w,b)$

Se puede observar que cada reemplazo de un vértice de $V$ por una copia de $H$, llamado nube $H_v$, se comporta internamente de acuerdo al mapa rotacional de $H$. Dos nubes diferentes, digamos $H_v$ y $H_w$, están conectadas en el producto de reemplazo si y sólo si existe una arista entre $v$ y $w$ en $G$. Ver Figura 15.

### IV.2.3 Producto zig-zag

El producto zig-zag entre dos grafos es una versión más elaborada del producto de reemplazo. Cada arista del producto zig-zag se puede ver como una trayectoria de tamaño 3 del producto de reemplazo entre los grafos. Una trayectoria debe estar compuesta de un *"paso corto"*, esto es, una arista que une a dos vértices de una misma nube, un *"paso largo"*, es decir, una arista entre dos nubes de vértices y finalmente otro *"paso corto"* en la nueva nube de vértices.

El producto zig-zag, formalmente definido en (Reingold *et al.*, 2001), se describe en función del mapa rotacional. Si $G$ es un grafo $D$-regular con $N$ vértices con mapa rotacional $Rot_G$ y $H$ es un grafo $d$-regular con $D$ vértices con mapa rotacional $Rot_H$,

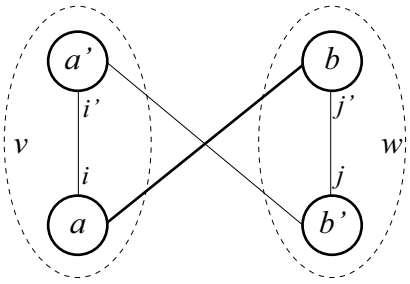
entonces el producto zig-zag entre $G$ y $H$, denotado como $G(z)H$, es un grafo $d^2$-regular con $N \times D$ vértices cuyo mapa rotacional $Rot_{G(z)H}$ se define como sigue:

$Rot_{G(z)H}((v,a)(i,j)) = ((w,b),(j',i'))$ si

1. $Rot_H(a,i) = (a',i')$
2. $Rot_G(v,a') = (w,b')$
3. $Rot_H(b',j) = (b,j')$

De la misma forma que en el producto de reemplazo cada vértice perteneciente a $G$ se sustituye por una nube de vértices, esto es una copia del grafo $H$. Cada vértice del producto zig-zag se puede expresar mediante una dupla $(v,a)$, donde $v \in G$ y $a \in H$. La expresión $Rot_{GH}((v,a),(i,j))$ se lee entonces como el rotacional del vértice $(v,a)$ a través del vecino $i$ de $a$ en $H$, que corresponde al primer *"paso corto"*. Esto es la primer condición, $Rot_H(a,i) = (a',i')$. La segunda condición, $Rot_G(v,a') = (w,b')$, corresponde al *"paso largo"*, esto es la arista entre dos nubes diferentes de vértices correspondiente al producto de reemplazo. Para completar la trayectoria de tamaño tres hace falta un último *"paso corto"*. La tercera condición se refiere a este paso, $Rot_H(b',j) = (b,j')$, se busca el $j$-ésimo vecino del vértice $b'$ en $H$. Con esto se llega al vértice $(w,b)$ formando la arista del producto zig-zag como se ilustra en la Figura 16.

Si se analiza lo que sucede con los vecinos de los vértices que participan en el producto de reemplazo se puede entender de forma más intuitiva el producto zig-zag. Como se explicó, si el $Rot_G(v,a) = (w,b)$ entonces en el producto de reemplazo existirá una arista entre los vértices $(v,a)$ y $(w,b)$. Cada uno de estos vértices tiene $d$ vecinos




$$Rot_{G(z)H}((v,a),(i,j)) = ((w,b),(j',i'))$$

Figura 16: Producto zig-zag. Se puede observar que la arisa entre los vértices $(v,a)$ y $(w,b)$ corresponde a una trayectoria de tamaño 3 en el producto de reemplazo.

definidos por el grafo $H$. En el producto zig-zag cada uno de los vecinos del vértice $(v,a)$ tendrá un arista con cada uno de los vecinos del vértice $(w,b)$ como se ve en la Figura 17. De esta forma, se puede entender por qué el grafo resultante es $d^2$-regular. Cada vértice del producto zig-zag tiene $d$ vecinos que a su vez están conectados mediante el producto de reemplazo a $d$ vértices de diferentes nubes de vértices. Cada uno de estos vértices tiene igualmente $d$ vecinos. Por lo tanto la cantidad de aristas para cada vértice será exactamente $d^2$.

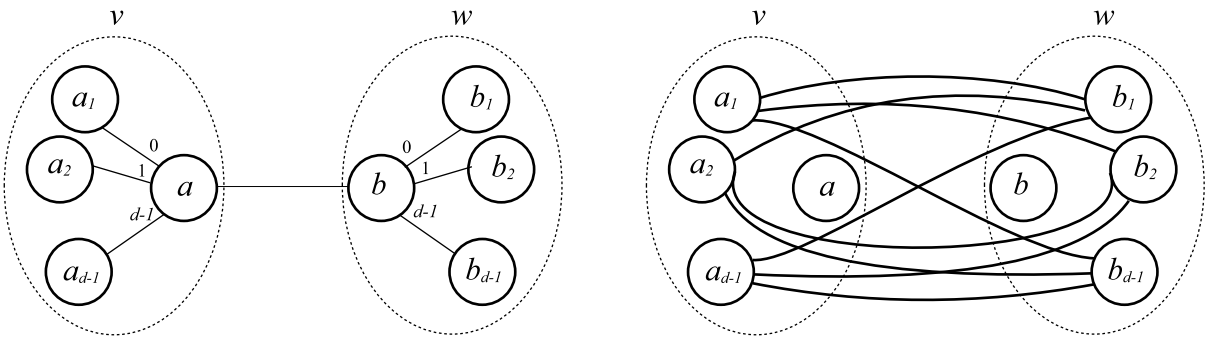



Figura 17: En la izquierda el producto de reemplazo entre $G$ y $H$. En la derecha se aprecia que cada vecino de $a$ está conectado a cada vecino de $b$ en el producto zig-zag

### IV.2.4 Construcción de expansores utilizando el producto zig-zag

Se sabe que si la matriz de adyacencia de un grafo cualquiera se eleva a una potencia, la conectividad del grafo aumenta al igual que la cantidad de aristas. Recordemos que al aumentar la conectividad de un grafo, el segundo valor propio más grande de la matriz de adyacencia del grafo disminuye. La idea de la construcción de grafos utilizando el producto zig-zag es sencilla: aumentar la conectividad del grafo elevando la matriz de adyacencia a alguna potencia y después aplicar el producto zig-zag reduciendo con esto la cantidad de aristas del grafo. Repitiendo este proceso se pueden crear explícitamente expansores del tamaño deseado.

Si se tiene un expansor $G$ $D$-regular con $N$ vértices, es decir un grafo-$(N, D, \lambda)$ y su matriz de adyacencia se eleva a la potencia $t$ el grafo resultante es un grafo-$(N, D^t, \lambda^t)$ (Reingold, 2005). Para un expansor se sabe que el valor de $\lambda$ es menor a 1, por lo tanto $\lambda^t < \lambda$ para cualquier $t > 1$.

Sea $H$ un expansor con $d^4$ vértices y $d$-regular con algún $\lambda$. Para cualquier $t \geq 1$ se define $G_{t+1} = G_t^2 ⓩ H$. Para $t = 1$, $G_1 = H^2$, es decir el grafo resultante de elevar al cuadrado la matriz de adyacencia de $H$. Entonces para cualquier $t$, $G_t$ es un expansor con $d^{4t}$ vértices y $d^2$-regular.

Si se eleva el expansor $H(d^4, d)$ al cuadrado se obtiene el grafo $G_1(d^4, d^2)$ con la misma cantidad de vértices y el cuadrado de la cantidad de aristas. $G_2$ entonces es $G_1^2 ⓩ H$, esto es $G(d^4, d^4) ⓩ H(d^4, d)$, por las propiedades del producto zig-zag se puede afirmar que el grafo resultante es un grafo $G_2(d^{2\cdot 4}, d^2)$. Se puede observar que en la siguiente iteración el grafo resultante es $G_3(d^{3\cdot 4}, d^2)$ y que en efecto para cualquier $t$ mayor que 1 el grafo resultante es $G_t(d^{4t}, d^2)$. Entonces elevar al cuadrado la matriz y

aplicar el producto zig-zag en forma iterativa es un método para crear expansores en forma explicita. Basta repetir el procedimiento las veces que sea necesario para obtener un expansor del tamaño deseado y siempre $d^2$ regular.

Reingold *et al.* (Reingold *et al.*, 2001) demostraron que las propiedades de expansión del grafo resultante del producto zig-zag está acotada por una función que depende de la expansión espectral de ambos grafos. En particular si tenemos dos expansores $G(N_1, d_1, \lambda_1)$ y $H(d_1, d_2, \lambda_2)$ entonces el producto zig-zag entre $G$ y $H$ es un expansor con $N_1 \times d_1$ vértices y $d_2^2$-regular y además la expansión espectral esta dada por $f(\lambda_1, \lambda_2)$ donde $f(\lambda_1, \lambda_2) \leq \lambda_1 + \lambda_2 + \lambda_2^2$. Implicando con esto que el grafo resultante tiene una expansión menor a los grafos originales.

# Capítulo V

# USTCON $\in$ L

El problema USTCON es un problema fundamental dentro de la teoría de grafos. El problema consiste en determinar si dos vértices dados se encuentran en el mismo componente conectado. El problema USTCON es completo para la clase de complejidad SL. Por lo tanto cualquier avance, en términos de espacio requerido para resolverlo, impacta a la relación que existe entre las clases L y SL.

Omer Reingold (Reingold, 2005) mostró un algoritmo determinístico que resuelve USTCON requiriendo sólo $O(\log N)$ unidades de espacio. Con este algoritmo el problema USTCON se situa en la clase de complejidad L. Debido a que el problema USTCON es completo para la clase SL se puede afirmar que las clases de complejidad L y SL son equivalentes.

El capítulo contiene un breve recuento de la historia del problema USTCON y del espacio requerido para resolverlo, y el algoritmo propuesto por Reingold que requiere $O(\log N)$ unidades de espacio.

## V.1 Trabajo previo en USTCON

El tiempo requerido para resolver USTCON con un algoritmo secuencial en forma óptima en el tiempo es bien conocido. Realizando una búsqueda a profundidad o anchura en un grafo, se puede determinar si dos vértices pertenecen al mismo componente conectado. Ambos algoritmos requieren tiempo y espacio de memoria lineal con respecto al número de nodos.

A continuación se presenta una breve recapitulación de los avances logrados, en función del espacio requerido para resolver el problema USTCON.

- El primer algoritmo con espacio sublineal para USTCON lo propuso Savitch en 1970. El algoritmo requiere de $O(\log^2 N)$ unidades de espacio (Savitch, 1970).
- En (Aleliunas *et al.*, 1979) se presentó un algoritmo probabilístico que se valía de una caminata aleatoria para resolver USTCON, requería un espacio de orden logarítmico y la probabilidad de error se reducía conforme se aumentaba la caminata aleatoria.
- En (Nisan *et al.*, 1992) se presentó el primer algoritmo determínistico que mejoraba el de Savitch. Sólo requería espacio $O(\log^{\frac{3}{2}} N)$.
- En (Armoni *et al.*, 1997) se mostró un algoritmo determinístico que requiere $O(\log^{\frac{4}{3}} N)$ de espacio.
- En (Reingold, 2005) se demostró la existencia de un algoritmo determinístico que requiere $O(\log N)$ unidades de espacio para resolver USTCON.

## V.2 Algoritmo de Reingold para resolver USTCON con espacio logarítmico

Los esfuerzos que precedieron a Reingold para resolver el problema de conectividad en un grafo, requiriendo un espacio sublineal, se basaban en la idea de reducir el grafo original. Si se transforma un grafo cualquiera $G$ con $N$ vértices y $E$ aristas a un grafo $G'$ con $n$ vértices, donde $n \ll N$, sin que se alteren las propiedades de conectividad del grafo original, se puede resolver el problema USTCON en el grafo $G'$ requiriendo una

menor cantidad de recursos de memoria. El algoritmo aplicado a $G'$ debe proporcionar la misma respuesta que aplicar el algoritmo al grafo original $G$, ya que ambos tienen los mismos componentes conectados.

El algoritmo de Reingold difiere de sus predecesores en la siguiente idea fundamental. En el algoritmo no se reduce el grafo original, todo lo contrario: se expande. Se transforma un grafo cualquiera a otro con los mismos componentes conectados pero de diámetro logarítmico, es decir se transforma un grafo cualquiera a un expansor.

El proceso de aumentar la cantidad de vértices del grafo original en forma considerable podría resultar contraproducente si lo que se desea es utilizar un espacio logarítmico de memoria. Se puede sortear esta dificultad gracias a que el algorítmo en un instante dado sólo requiere el mapa rotacional de una cantidad constante de vértices del expansor. Esto ocasiona que cada vez que se calcule el mapa rotacional de algún vértice del expansor se pierda la información de alguno de los vértices conocidos.

El algoritmo utiliza una secuencia de productos zig-zag y de elevaciones de matrices a una potencia, para convertir un grafo cualquiera en un expansor. La transformación requiere por supuesto $O(\log N)$ unidades de espacio. Determinar si dos vértices están en el mismo componente conectado en un expansor, con $O(\log N)$ unidades de espacio, es un problema relativamente sencillo ya que el diámetro de los expansores es logarítmico.

El algoritmo requiere que el grafo de entrada $G$ tenga $N$ vértices y sea $d^{16}$-regular (al final del capítulo se muestra un método para convertir cualquier grafo a un grafo $d^{16}$-regular) y un expansor $H$ con $d^{16}$ vértices, $d$-regular y con una $\lambda \leq \frac{1}{2}$. Este expansor se puede determinar por fuerza bruta o por medio del método mostrado para la construcción explícita de expansores.

### V.2.1 Algoritmo USTCON

A continuación se describe el algoritmo propuesto por Reingold para resolver el problema USTCON. El algoritmo específico que transforma un grafo cualquiera a un expansor con espacio de memoria logarítmico se analiza posteriormente.

**Entrada**: El mapa rotacional de un grafo $G$ con $|V| = N$ vértices, $d^{16}$-regular, dos vértices $s, t \in V$ y el mapa rotacional $H$ de un grafo-$(d^{16}, d, \lambda)$ con $\lambda \leq \frac{1}{2}$.

**Salida**: "conectado", si $s$ y $t$ están en el mismo componente conectado, "no conectado", en cualquier otro caso.

1. Sea $l$ el entero más pequeño tal que $\left(1 - \frac{1}{d^{16}n^2}\right)^{2^l} \leq \frac{1}{2}$
2. $G_0 \leftarrow G$
3. for $i = 1, \ldots, l$ do $G_i \leftarrow (G_{i-1} ⓩ H)^8$
4. Listar todas las posibles trayectorias de $O(\log N)$ desde el vértice $s$. Mostrar "conectado" si se encuetra $t$, mostrar "no conectado" en caso contrario.

Para cualquier grafo $d$-regular, el entero más pequeño que satisface la inecuación del paso 1 del algoritmo es $O(\log N)$ (Reingold, 2005). Por lo tanto $l$ siempre es $O(\log N)$. De acuerdo a las propiedades del producto zig-zag se puede apreciar que cada grafo $G_i$ es un grafo $d^{16}$-regular.

El proceso iterativo, como ya se mencionó, se compone de dos fases. Para cada etapa $i$, primero se obtiene el producto zig-zag entre $G_{i-1}$ y $H$. Con esto se aumenta la cantidad de vértices del grafo $G_{i-1}$ de $(d^{16})^{i-1}N$ a $(d^{16})^i N$ y al mismo tiempo se reduce el grado de cada vértice de $d^{16}$ a $d^2$, esto debido a las propiedades del producto

zig-zag. Posteriormente se eleva la matriz de adyacencia del grafo resultante a la octava potencia, lo cual no aumenta la cantidad de vértices del grafo $G_i$ pero si el grado de cada uno de sus vértices de $d^2$ a $d^{16}$. Después de $O(\log N)$ iteraciones se obtiene el mapa rotacional $G_l$.

En (Reingold, 2005) se demuestra que si $H$ tiene una expansión espectral menor o igual a $\frac{1}{2}$, entonces cada componente conectado del grafo $G_l$ tiene una expansión espectral menor ó igual a $\frac{1}{2}$. Por lo tanto cada componente conectado del grafo $G_l$ es en efecto un expansor con diámetro logarítmico.

Finalmente el determinar si dos vértices están conectados en un grafo con diámetro logarítmico es trivial. Sólo se requiere listar todas las trayectorias de longitud a lo más $O(\log N)$ originadas en el vértice $s$. Los vértices $s$ y $t$ se encuentran en el mismo componente conectado, si en por lo menos una de esas trayectorias se encuentra el vértice $t$.

**Transformación de un grafo $G$ a un expansor $G_l$**

A continuación se describe el algoritmo para la máquina de Turing que dados los grafos $G(N, d^{16})$ y $H(d^{16}, d, \lambda < \frac{1}{2})$ (en la cinta de sólo lectura) calcula el mapa rotacional de $G_l$.

Las dos operaciones fundamentales del algoritmo son el producto zig-zag entre grafos y la potenciación. La definición formal del producto zig-zag en función del mapa rotacional se muestra en el Capítulo V, sólo resta definir la potenciación. Si se eleva un grafo $G(N, D, \lambda)$ a una potencia $t$ se obtiene un grafo $G_t(N, D^t, \lambda^t)$. En función del mapa rotacional (Reingold, 2005) $Rot_{G^t}(v_0, (a_1, \ldots, a_t)) = (v_t, (b_t, \ldots, b_1))$ donde cada uno de estos valores se calcula mediante la regla $(v_i, b_i) = Rot_G(v_{i-1}, a_i)$. Es decir, son todas las posibles trayectorias de longitud $t$ en el grafo a partir de $v_0$.

Propiamente, dado un vértice $\overline{v}$ del expansor $G_l$ y una arista $\overline{a}$, el algoritmo calcula el $\mathrm{Rot}_{G_l}(\overline{v},\overline{a}) = (\overline{v'},\overline{a'})$. Cualquier vértice $\overline{v} \in G_l$ consiste de un vértice de $G$ y $l$ vértices de $H$. La arista $\overline{a}$ corresponde a un vértice de $H$. Debido a que $l = O(\log n)$ la longitud de cada uno de los pares $(\overline{v},\overline{a})$ es $O(\log n)$.

Primero el algoritmo establece las siguientes variables: $v$, tal que $0 \leq v < N$ (especificando un vértice de $G$), y $l+1$ variables llamadas $a_0, a_1 \ldots a_l$, tal que $0 \leq a_i < d^{16}$ (cada una refiriéndose a un vértice de $H$). Se requiere que cada uno de los $a_1 \ldots a_l$ se exprese como una secuencia de 16 números o dígitos en base $d$ (esto se puede ver como aristas de $H$). Esto es cada $a_i$ se expresa como $k_{i,1} \ldots k_{i,16}$. El algoritmo posteriormente copia los valores de entrada $(\overline{v},\overline{a})$ a las variables mencionadas: El valor de $\overline{v}$ se almacena en $v, a_0, \ldots, a_{l-1}$ y el valor de $\overline{a}$ se almacena en $a_l$. El algoritmo modifica gradualmente estos valores, de tal forma que al finalizar el mismo, las variables $v, a_0, \ldots, a_l$ almacenan el $\mathrm{Rot}_{G_l}(\overline{v},\overline{a})$.

El algoritmo opera en forma recursiva. En cada nivel de recursion el algoritmo evalua el $\mathrm{Rot}_{G_i}$ para alguna $i$. Para el caso base, $i = 0$, el $\mathrm{Rot}_{G_0}$ =$\mathrm{Rot}_G$ está escrito en la cinta de entrada de la máquina de Turing, por lo tanto es posible obtener dicha información requiriendo un espacio logarítmico, simplemente mediante la búsqueda en la cinta de la información deseada. Para $i > 0$, el cálculo del $\mathrm{Rot}_{G_i}$ se realiza de la siguiente manera:

Desde $j = 1$ hasta 16

- $a_{i-1}, k_{i,j} \leftarrow$$\mathrm{Rot}_H(a_{i-1}, k_{i,j})$.
- Si $j$ es impar calcular en forma recursiva
  $v, a_0 \ldots a_{i-1} \leftarrow \mathrm{Rot}_{G_{i-1}}((v, a_0 \ldots a_{i-2}), a_{i-1})$.

- Si $j = 16$, invertir el orden de las etiquetas individuales en $a_i$:
  $k_{i,1}, \ldots, k_{i,16} \leftarrow k_{i,16}, \ldots, k_{i,1}$.

Se puede observar que el algoritmo realiza el producto zig-zag y la potenciación de la matriz de adyacencia de forma simultanea. Para $j = 1$ el algoritmo determina el $\mathrm{Rot}_H(a_{i-1}, k_{i,1})$ y sustituye esta información en la cinta de trabajo. Esto corresponde al "paso corto" del producto zig-zag entre $G_{i-1}$ y $H$. Posteriormente el algoritmo calcula el $\mathrm{Rot}_{G_{i-1}}((v, a_0 \ldots a_{i-2}), a_{i-1})$, debido a que $j = 1$ es impar. Esta operacion corresponde al "paso largo" del producto zig-zag. Note que no se requiere espacio adicional en la cinta. En la siguiente iteración, para $j = 2$, se calcula $\mathrm{Rot}_H(a_{i-1}, k_{i,2})$ y se sobre escribe en la cinta. Con este segundo "paso corto" finalmente se obtiene una arista de la matriz de adyacencia del grafo $G_{i-1} ⓩ H$.

Para obtener una arista del grafo $(G_{i-1} ⓩ H)^8$ sólo se requiere obtener una trayectoria de tamaño 8 en el grafo $G_{i-1} ⓩ H$. Para ello se repite el proceso anterior exactamente 8 veces. Con $j = 3$ y $j = 4$ se obtiene la segunda arista del grafo $G_{i-1} ⓩ H$ y de forma similar se obtienen las seis restantes.

Finalmente cuando $j = 16$ se invierten las etiquetas individuales de $a_i$ de acuerdo a la definición del producto zig-zag.

Ninguna operación requiere un espacio adicional de memoria. Cuando se calcula el mapa rotacional de cualquier $(v, e) = (v', e')$, se pierde la información de $(v, e)$ ya que se cambia por $(v', e')$. Esto permite que el espacio requerido por el algoritmo sea únicamente el necesario para referirse a un vértice del expansor $G_l$ y una arista del mismo. La recursión tiene una profundidad logarítmica, por lo tanto sólo se requiere de una cantidad logarítmica de memoria para mantenerla. Entonces se puede concluir que el espacio requerido por el algoritmo es en efecto $O(\log n)$.

### V.2.2 Manejo de grafos no regulares y bipartitos

El algoritmo de Reingold sólo funciona para grafos que son $d$-regulares y no bipartitos. Un grafo es bipartito si existe una partición de sus vértices tal que cada arista del grafo conecta a dos vértices que pertenecen a los dos bloques de la partición. Cuando se quiere resolver el problema USTCON en un grafo no regular y/o bipartito utilizando el algoritmo de Reingold, es necesario transformar el grafo dado a un grafo que cumpla con las características requeridas por el algoritmo. Existen varias formas de efectuar dicha transformación. A continuación se muestra una de ellas.

La siguiente transformación toma un grafo $G$ cualquiera y lo convierte en un grafo $D^{16}$-regular y no bipartito, de acuerdo a lo requerido por el algoritmo de Reingold.

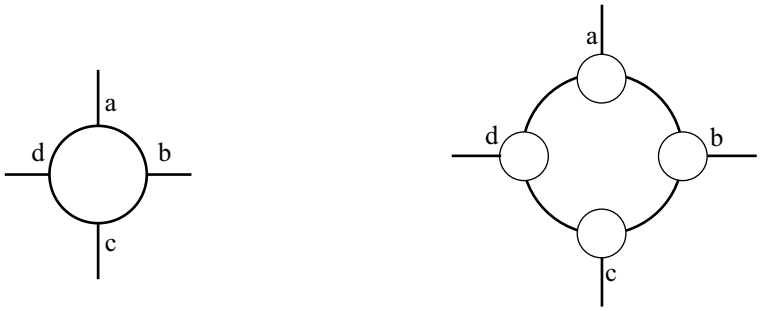



Figura 18: En la izquierda se puede ver un vértice de grado 4. En la derecha la transformación a un ciclo de 4 vértices

1. Se sustituye cada vértice $v$ de grado $d > 3$ por un ciclo de vértices de tamaño igual a $d$, como se ve en la Figura 18.

2. Cada uno de los $d$ vértices del anillo se conecta mediante una arista a uno de los $d$ vécinos de $v$. Con esto se reduce el grado de cada vértice a 3.

3. A cada vértice $v \in G$, de grado $d'$, se agrega $D^{16} - d'$ autociclos. Con esto se convierte el grafo $G$ a un grafo $D^{16}$-regular y no bipartito (ya que un grafo bipartito no puede contener autociclos).

# Capítulo VI

# Algoritmos Básicos

En este capítulo se muestran una serie de algoritmos básicos. Estos se utilizan para resolver los dos problemas principales para la simulación del modelo R-Mesh sobre el modelo LR-Mesh: convertir un grafo cualquiera a un expansor y determinar la conectividad entre dos vértices de un expansor. Estos dos problemas se resuelven en los capítulos siguientes.

Cabe señalar que todos los algoritmos mostrados en esta sección se diseñaron para el modelo LR-Mesh, el tiempo de ejecución de todos es $O(1)$ y la cantidad de recursos (procesadores) que requiere cada uno varía considerablemente.

A continuación se muestran cada uno de estos problemas básicos. En cada caso en específico se proporciona una breve descripción del problema, se muestra el algoritmo que lo resuelve en tiempo constante y se evalua número de procesadores requerido por el mismo.

## VI.1 Suma de prefijos

Un problema básico en el cómputo paralelo es el problema de la suma de prefijos. Si se tiene una secuencia $b_1, b_2, \ldots, b_k$ donde cada $b_i \in \{0, 1\}$, para toda $1 \leq i \leq k$, el problema de la suma de prefijos consiste en calcular $S_i = \sum_{j=1}^{i} b_j$. Esto es para cada $i$ se calcula la cantidad de bits que son igual a uno desde $b_1$ hasta $b_i$.

La idea del algoritmo es sencilla. Se utiliza un arreglo de $(N+1) \times N$ procesadores, para calcular la suma de prefijos de $N$ bits. Al inicio, el procesador $j$ del renglón 0

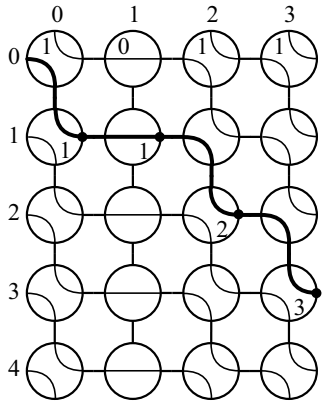



Figura 19: Suma de prefijos. Los procesadores del renglón 0 tienen la cadena de entrada 1 0 1 1. Se puede observar que los procesadores (1,0), (1,1), (2,2) y (3,3) determinan la suma de prefijos para su correspondiente elemento.

posee el bit $b_j$. Este bit se transmite a todos los procesadores de la columna $j$. En función de dicho bit, cada procesador adquiere una configuración interna particular. Si el bit $b_j$ es 0, los procesadores en la columna $j$ fusionan sus puertos $E$ y $O$, formando con esto un ducto horizontal que atraviesa dicha columna. En el caso contrario, si el bit $b_j$ es 1, los procesadores de la columna $j$ fusionan sus puertos $N$ y $E$ y los puertos $S$ y $O$. Con esto se crea un ducto en particular que recorre la malla de izquierda a derecha, ver Figura 19. El ducto principia en el puerto $E$ del procesador $(0, 0)$ y finaliza en el renglón $i$ de la columna $N - 1$. Si $b_j$ es igual a cero, el ducto en la columna $j$ sólo recorre dicha columna en forma horizontal. Esto corresponde con la columna 1 de la Figura 19. En cambio, si $b_j$ es igual a 1, el ducto en la columna $j$ y renglón $i$ se conecta al renglón $i + 1$ y columna $j + 1$, como en las columnas 0, 2 y 3 de la Figura 19. Al transimitir una señal predefinida $\alpha$ por el puerto $O$ del procesador $(0, 0)$, los procesadores de la columna $j$ pueden determinar la cantidad de unos que existen desde $b_0$ hasta $b_j$ mediante la detección de la señal $\alpha$. Si la señal se detecta en el renglón $i$, entonces la suma de prefijos para dicha columna es exactamente $i$. En el ejemplo de la Figura 19 se puede apreciar que en cada columna sólo un procesador detecta la señal $\alpha$ por el puerto este; el identificador del reglón de dicho procesador corresponde con la

suma de prefijos.

A continuación se presenta un pseudocódigo para resolver el problema de suma de prefijos en el modelo LR-Mesh.

**Entrada**: Para toda $j$, tal que $0 \leq j < N$, el procesador $P_{0,j}$ almacena el bit $b_j$.

**Salida**: Para toda $j$, tal que $0 \leq j < N$ y $b_j = 1$, el procesador $P_{0,j}$ almacena $\sum_{i=0}^{j} b_i$.

BEGIN

$senal = \beta$, $suma = 0$

Procesador $P_{0,j}$, para $0 \leq j < N$, transmite el bit $b_j$ a los procesadores de la columna $j$.

para $0 \leq i \leq N$ y $0 \leq j < N$ hacer

if $b_j = 1$ then

Procesador $P_{i,j}$ configura $\{NE, SO\}$

else

Procesador $P_{i,j}$ configura $\{N, S, EO\}$

Procesador $P_{0,0}$ transmite una señal $\alpha$ por el puerto $O$

para $0 \leq i \leq N$ y $0 \leq j < N$ hacer

Procesador $P_{i,j}$ lee del puerto $E$ y almacena el valor en *senal*

Procesador $P_{i,j}$ configura $\{NS, E, O\}$

if $senal = \alpha$ then

Procesador $P_{i,j}$ transmite $i$ por el puerto $N$

Procesador $P_{0,j}$, para $0 \leq j < N$, lee del puerto $S$ y almacena el valor en *suma*

END

**Lema 1.** *La suma de prefijos de $N$ bits se puede resolver en el modelo LR-Mesh de tamaño $(N+1) \times N$ en tiempo constante. Inicialmente cada procesador del renglón 0 tiene uno de los bits de entrada.*

Si se tienen $N$ bits y se conoce apriori la cantidad de bits iguales a 1, se puede reducir el tamaño de la malla requerida para resolver el problema. Si se tiene un conjunto de $N$ bits y $D$ de estos bits son iguales a 1, entonces la suma de prefijos se puede calcular en el modelo LR-Mesh de tamaño $(D+1) \times N$ en tiempo constante. Cada uno de los renglones de la malla representa la suma parcial de prefijos. Si se tiene como máximo $D$ bits iguales a 1 se requiere $D+1$ renglones para representar la suma total.

## VI.2 División y multiplicación

De forma similar al algoritmo de la suma de prefijos se puede crear una estructura que permita el cálculo de las operaciones aritméticas: división y multiplicación. El algoritmo requiere tener como entrada, para el caso de la división, el dividendo y el divisor. El dividendo lo requieren los procesadores de la primer columna de la malla. Este número se expresa en representación unaria (un número $B$ se representa por $B+1$ unos, esto es, para el número $B$ los procesadores $0, 1, \ldots, B$ poseen un 1, el resto de los procesadores 0). El divisor se transmite a la malla completa. En función del divisor se crea la estructura de ductos que realizan la operación de división. La idea es crear una estructura escalonada que transmita el posible 1 que posee el procesador $(i, 0)$ hasta el procesador $(i/divisor, N)$. Con esto se obtiene, en la última columna de la malla, la división en representación unaria.

A continuación se muestra un pseudocódigo del algoritmo que realiza la división en el modelo LR-Mesh en tiempo constante:

**Entrada**: El procesador $P_{0,0}$ tiene el dividendo $B$ y el divisor $D$.

**Salida**: La columna $N - \lfloor \frac{N}{D} \rfloor - 1$, tendrá en representación unaria $division = \lfloor \frac{B}{D} \rfloor$.

BEGIN

$senal = \alpha$, $division = 0$

Procesador $P_{0,0}$ transmite $B$ a la columna 0

Procesador $P_{0,0}$ transmite $D$ a la malla completa

para $0 \leq i \leq N$ y $0 \leq j < N - \lfloor \frac{N}{D} \rfloor$ hacer

if $j = i \times (D - 1)$ then

Procesador $P_{i,j}$ configura $\{N, S, E, O\}$

else

Procesador $P_{i,j}$ configura $\{N, S, EO\}$

if $j = i \times (D - 1)$ then

Procesador $P_{i,j}$ transmite:

por el puerto $E$ una señal $\beta$

por el puerto $O$ una señal $\alpha$

Procesador $P_{i,j}$ lee del puerto $E$ y almacena el valor en $senal$

if $senal = \alpha$

Procesador $P_{i,j}$ configura $\{NO, SE\}$

else

Procesador $P_{i,j}$ configura $\{N, S, EO\}$

para $0 \leq i \leq N$ hacer

if $j = 0$ AND $i \leq B$

Procesador $P_{i,0}$ transmite 1 por el puerto $E$

Procesador $P_{i,N-\lfloor \frac{N}{D} \rfloor - 1}$ lee del puerto $E$ y almacena el valor en $division$

END

**Lema 2.** *La división entre $B$ y $D$, para toda $0 \leq B \leq N$, se puede calcular en el modelo LR-Mesh de tamaño $(N+1) \times (N - \lfloor \frac{N}{D} \rfloor)$ en tiempo constante.*

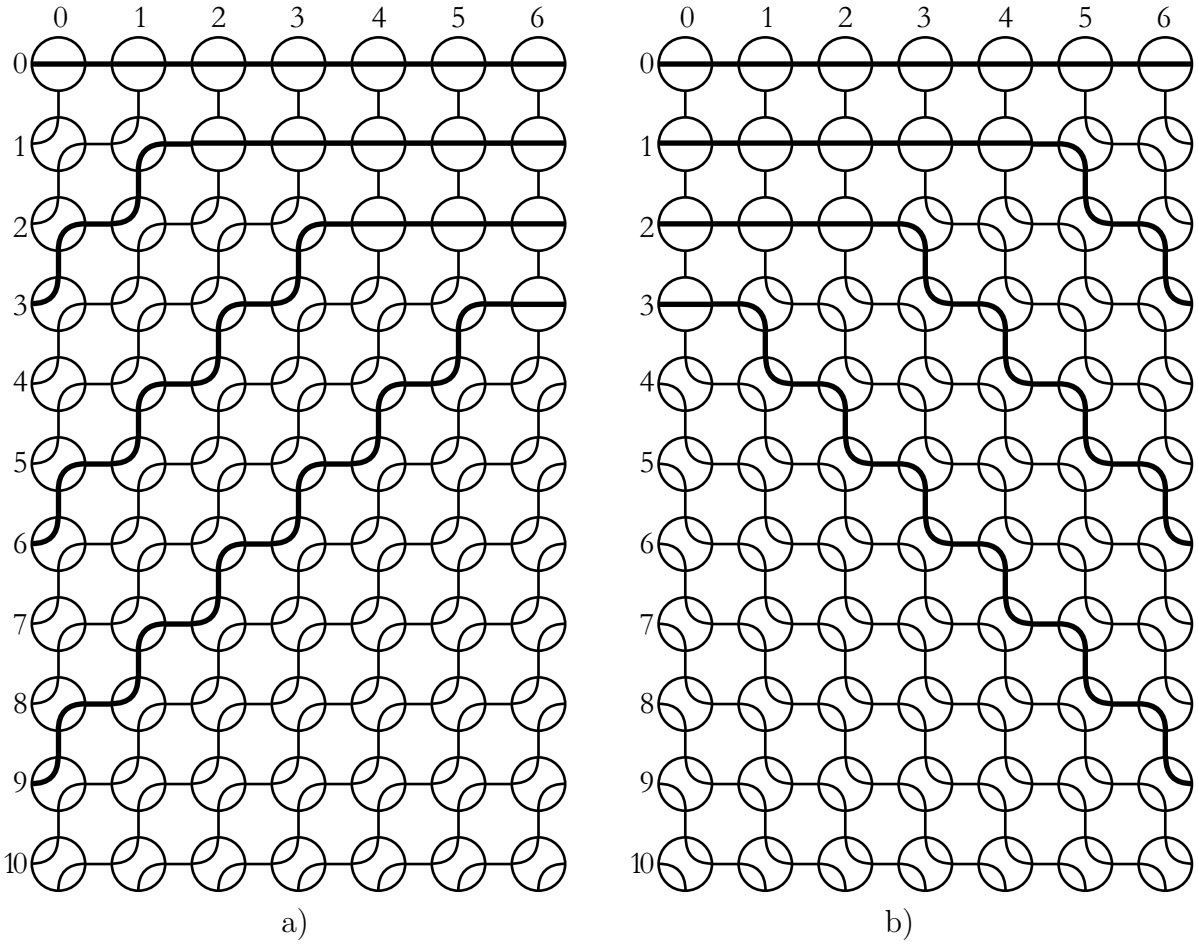



Figura 20: a) Malla de tamaño $11 \times 7$ que permite la división de un número $x$, para toda $0 \leq x \leq 10$, entre 3. b) Malla que permite la multiplicación de un número $x$, para toda $0 \leq x \leq 3$, por 3.

En la Figura 20a se aprecia un ejemplo de cómo el módelo LR-Mesh puede dividir un número cualquiera por una constante y se muestra la configuración que adquieren los procesadores para efectuar una división entre tres. Se puede observar que los ductos que recorren la malla desde la columna 0 hasta la columna 6 cumplen con las siguientes características: el extremo izquierdo (columna 0) del ducto se encuentra en un renglón $i$, cuyo identificador es múltiplo de tres y el ducto termina en la columna 6 y el renglón

$\frac{i}{3}$. Esto es el ducto cuyo extremo derecho se encuentra en el renglón 3, termina en el renglón 1, de igual forma los ductos que empiezan en los renglones 6 y 9 terminan en los renglones 2 y 3, respectivamente. Gracias a esta estructura escalonada se puede efectuar la división en la malla. Para dividir el número 7 entre 3 utilizando el algoritmo mostrado se requiere que los procesadores que se encuentren entre el renglón 0 y 7 envíen una señal a través del ducto correspondiente. En la columna 6 se tendrá en representación unaria el resultado.

Es fácil ver que si se transmite un número $B$, a través de la misma malla, pero por la última columna, en la primer columna se obtendrá (no propiamente en representación unaria) el número $B \times D$. Convertir el resultado a representación unaria es un proceso relativamente sencillo y se puede realizar en tiempo constante.

**Lema 3.** *La multiplicación de un número $B$ por un número $D$ se puede realizar en el modelo LR-Mesh de tamaño $(N+1) \times (N - \lfloor \frac{N}{D} \rfloor)$ para cualquier $N \geq (B \times D)$.*

En la Figura 20b se muestra una malla de tamaño $11 \times 7$ que permite la multiplicación de un número cualquiera por 3. Note que la configuración de procesadores es muy similar al caso de la división, la multiplicación es el proceso inverso de la división y viceversa. En este caso los ductos principian en el renglón $i$ y terminan en el renglón $i \times 3$.

## VI.3 División en cascada

Se puede apreciar que en la última columna de la malla (que divide un número $B$ entre $D$) se tiene el cociente de la operación $\frac{B}{D}$ en representación unaria. Este número se puede utilizar como entrada para otra malla idéntica. El resultado de la segunda malla

es el cociente de $\frac{\lfloor \frac{B}{D} \rfloor}{D}$. De la misma forma se puede acoplar otra malla idéntica más a la salida de la segunda malla y obtener el cociente del resultado de la malla anterior entre $D$.

A continuación se muestra el algoritmo para realizar la división en cascada:

**Entrada**: El procesador $P_{0,0}$ tiene el dividendo $B$ y el divisor $D$.

**Salida**: La columna $(l)(N-\lfloor \frac{N}{D} \rfloor)-1$, para toda $1 \leq l \leq k$, tendrá en representación unaria $division = \lfloor \frac{B}{D^l} \rfloor$.

BEGIN

$senal = \alpha$, $J = j \bmod (N - \lfloor \frac{N}{D} \rfloor)$, $division = 0$

Procesador $P_{0,0}$ transmite $B$ a la columna 0

Procesador $P_{0,0}$ transmite $D$ a la malla completa

para $0 \leq i \leq N$ y $0 \leq j < (k)(N - \lfloor \frac{N}{D} \rfloor)$ hacer

if $J = i \times (D-1)$ OR $J = 0$ then

Procesador $P_{i,j}$ configura $\{N, S, E, O\}$

else

Procesador $P_{i,j}$ configura $\{N, S, EO\}$

if $J = i \times (D-1)$ then

Procesador $P_{i,j}$ transmite:

por el puerto $E$ una señal $\beta$

por el puerto $O$ una señal $\alpha$

Procesador $P_{i,j}$ lee del puerto $E$ y almacena el valor en $senal$

if $senal = \alpha$

Procesador $P_{i,j}$ configura $\{NO, SE\}$

else

Procesador $P_{i,j}$ configura $\{N, S, EO\}$

para $0 \leq i \leq N$ hacer

if $j = 0$ AND $i \leq B$

Procesador $P_{i,0}$ transmite 1 por el puerto $E$

Procesador $P_{i,k\times(N-\lfloor\frac{N}{D}\rfloor)-1}$ lee del puerto $E$ y almacena el valor en *division*

END

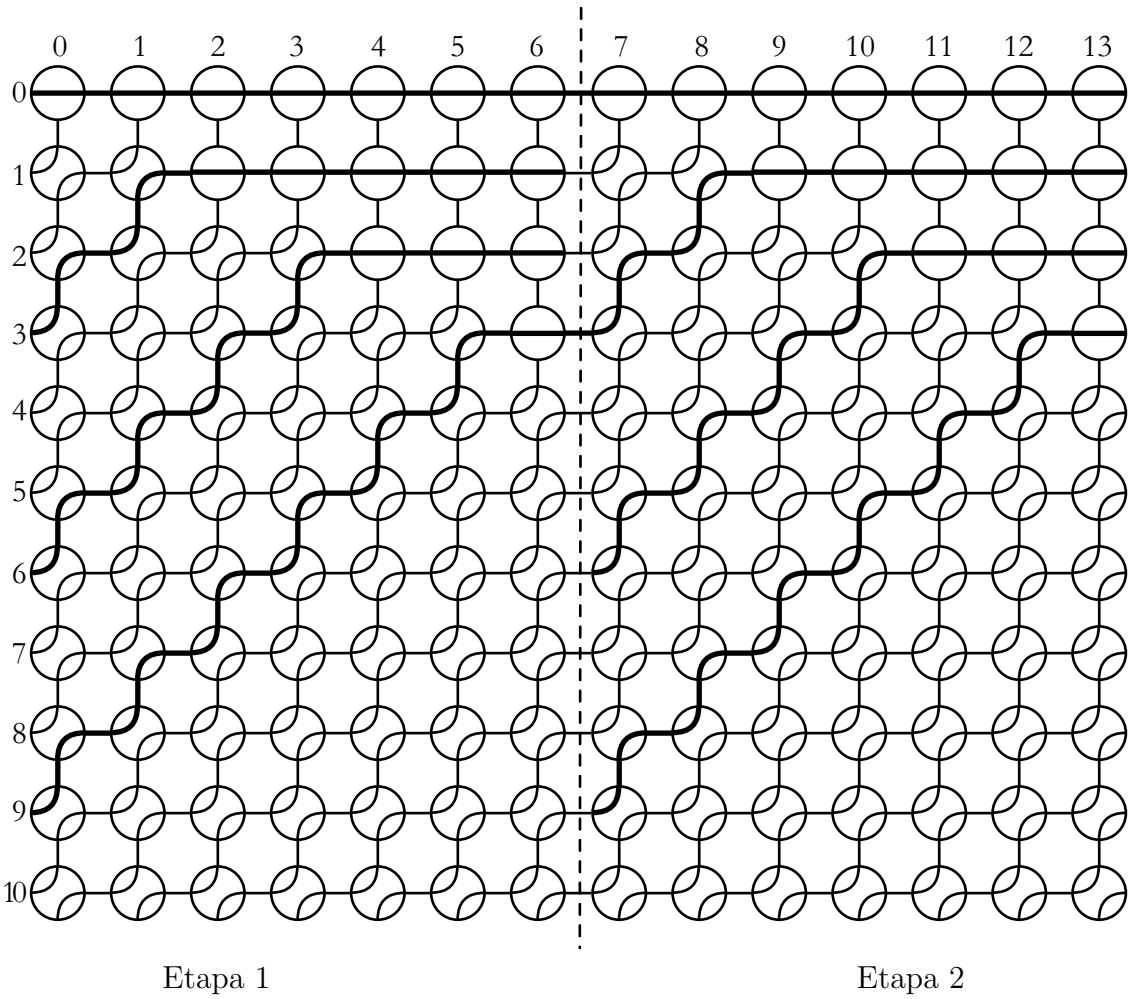



Figura 21: Divisor en cascada. Permite dividir un número $x$, para toda $0 \leq x \leq 10$, entre 3 y el resultado entre tres. Se requiere una malla de tamaño $11 \times 14$.

**Lema 4.** *La k-ésima división entre B y D, para toda* $0 \leq B \leq N$*, se puede calcular en el modelo LR-Mesh de tamaño* $(N+1) \times ((N - \lfloor\frac{N}{D}\rfloor)(k))$*, en tiempo constante.*

En la Figura 21 se muestra un ejemplo de cómo funciona el divisor en cascada. El arreglo mostrado permite dividir un número cualquiera entre 3 y el resultado se vuelve a dividir entre 3. Se puede observar que únicamente se requiere conectar tantos divisores, como el de la Figura 20a, para dividir tantas veces un número cualquiera entre 3 sucesivamente. Los procesadores de la columna cero transmiten en forma unaria el número que se quiere dividir y al final de la etapa 1 se obtiene el cociente de la división en representación unaria. Este cociente sirve de entrada para la etapa 2. Al final de la etapa 2 se obtiene la división del primer cociente entre 3. Se puede observar que si la entrada de la etapa 1 es el número 9, al final de la misma se tiene el numero 3, este a su vez sirve de entrada para la etapa dos y al final de la misma se obtiene un 1.

De forma similar se puede multiplicar en cascada un número $B$ por $D^k$. Sólo se crean $k$ mallas de la misma forma que el divisor. En la última columna se transmite el número $B$ y en la primer columna se tiene la multiplicación $B \times D^k$.

**Lema 5.** *La multiplicación en cascada de un número $B$ por $D^k$ se puede realizar en el modelo LR-Mesh de tamaño $(N+1) \times ((N - \lfloor \frac{N}{D} \rfloor)(k))$, para $(B \times D^k) \leq N$ en tiempo constante.*

## VI.4 Conversión de un número en base 10 a base D

Para convertir un número $B$ en base 10 a base $D$, se divide $B$ por $D$, el cociente se divide de nuevo por $B$ y así sucesivamente hasta que se obtenga un cociente de cero. Los residuos sucesivos de esta serie de divisiones son los dígitos que expresan $B$ en base $D$. El primer residuo que se obtiene es el dígito menos significativo.

El algoritmo para el modelo LR-Mesh sigue el mismo procedimiento. Se tiene el número $B$ en base 10 y la base $D$ a la que se desea convertir. Para obtener los primeros

$k$ dígitos en base $D$, se divide $k$ veces el número $B$ por $D$. Esto se puede realizar en tiempo constante gracias al Lema 4. El número original $B$ y cada una de las divisiones parciales $l_1, l_2, \ldots, l_k$ se transmiten al primer renglón de la malla. Cada uno de los procesadores que tiene uno de los resultados calcula la operación ($B$ mod $D$) o ($l_j$ mod $D$) según corresponda. Con esto se obtiene el número $B$ en base $D$. El procesador con menor identificador tiene el dígito menos significativo y el procesador con mayor identificador tiene el dígito más significativo.

A continuación se presenta el pseudocódigo para la conversión de un número en base 10 a base D:

**Entrada**: El procesador $P_{0,0}$ tiene el número $B$ en base 10, la base $D$ y el número $k$ de dígitos a obtener.

**Salida**: La columna $(l)(N - \lfloor \frac{N}{D} \rfloor) - 1$, para toda $0 \leq l < k$, tendrá el dígito $l$ del número $B$ en base $D$.

BEGIN

Algoritmo divisi'on en cascada, $B$ como dividendo y $D$ como divisor.

if $j = 0$ or $J = N - \lfloor \frac{N}{D} \rfloor - 1$ then

Procesador $P_{i,j}$, para $0 \leq i \leq N$ y $0 \leq j < (N - \lfloor \frac{N}{D} \rfloor)(k)$, configura $\{N, S, E, O\}$

if $division = 1$ then

para $0 \leq i \leq N$ y $0 \leq j < (N - \lfloor \frac{N}{D} \rfloor)(k)$ hacer

Procesador $P_{i,j}$ transmite una señal $\epsilon$ por el puerto $N$

Procesador $P_{i,j}$ lee del puerto $S$ y almacena el dato en $senal$.

Procesador $P_{i,j}$ configura $\{NS, E, O\}$

if $senal \neq \epsilon$ then

Procesador $P_{i,j}$ transmite $i$ por el puerto $S$

para $0 \leq j < (N - \lfloor \frac{N}{D} \rfloor)(k)$ hacer

if $i = 0$ then

Procesador $P_{0,j}$ lee del puerto $S$ y almacena el dato en *division*

Procesador $P_{0,j}$ calcula $l = division \bmod D$

Procesador $P_{0,0}$ calcula $l = B \bmod D$

END

**Lema 6.** *La obtención de k dígitos de un número B en base D se puede realizar en el modelo LR-Mesh de tamaño* $(N + 1) \times ((N - \lfloor \frac{N}{D} \rfloor)(k))$ *en tiempo constante.*

Para convertir un número cualquiera $B$ a una base $D$, el algoritmo hace uso de la división en cascada. Si $B = 7$ y la base $D = 3$, se obtiene un escenario parecido al mostrado en la Figura 21. Al final de la etapa 1 se tiene el resultado de la primer división: $\frac{7}{3} = 2$. El número dos sirve de entrada para la segunda etapa y al final de ésta se obtiene la segunda división: $\frac{2}{3} = 0$. Posteriormente el algoritmo determina el resultado de cada una de las divisiones y se envía el resultado al procesador en el renglón 0 de cada una de las etapas. Así los procesadores $P_{0,0}$, $P_{0,6}$ y $P_{0,13}$ obtienen los números 7, 2 y 0, respectivamente. Siguiendo el procedimiento para convertir de una base a otra se obtiene el residuo de cada una de las divisiones efectuadas. El procesador $P_{0,0}$ determina el residuo (mediante la operación mod) de la división $\frac{7}{3}$, de forma similar los procesadores $P_{0,6}$ y $P_{0,13}$ calculan el residuo de las divisiones $\frac{2}{3}$ y $\frac{0}{3}$, respectivamente. Obteniendo así el número 7 en base 3: 021. El procesador $P_{0,13}$ almacena la cifra más significativa (0) y el procesadore $P_{0,0}$ la cifra menos significativa (1).

## VI.5 Permutación

Sea $P = \{p_1, p_2, \ldots, p_N\}$ un conjunto de procesadores, cada $p_i$, para toda $0 < i \leq N$, posee un dato $d_i$. Sea $L = \{l_1, l_2, \ldots, l_N\}$ una permutación de $P$. El problema de permutación consiste en transmitir el dato $d_i$ que posee el procesador $p_i$ al procesador $l_i$.

En el modelo LR-Mesh se crea una estructura de ductos que permiten dicha permutación. La primer columna de la malla representa el conjunto de procesadores $P$. Cada $p_i$ posee el dato $d_i$ correspondiente y el elemento $l_i$ de la permutación. La última columna de la malla representa la permutacion $L$ deseada. Entre ambas columnas se crea una estructura de ductos que permite conectar al procesador $p_i$ de la primer columna con el procesador $l_i$ de la segunda columna. Para obtener la permutación el procesador $p_i$ transmite el dato $d_i$ por el ducto y el procesador $l_i$ lo recibe.

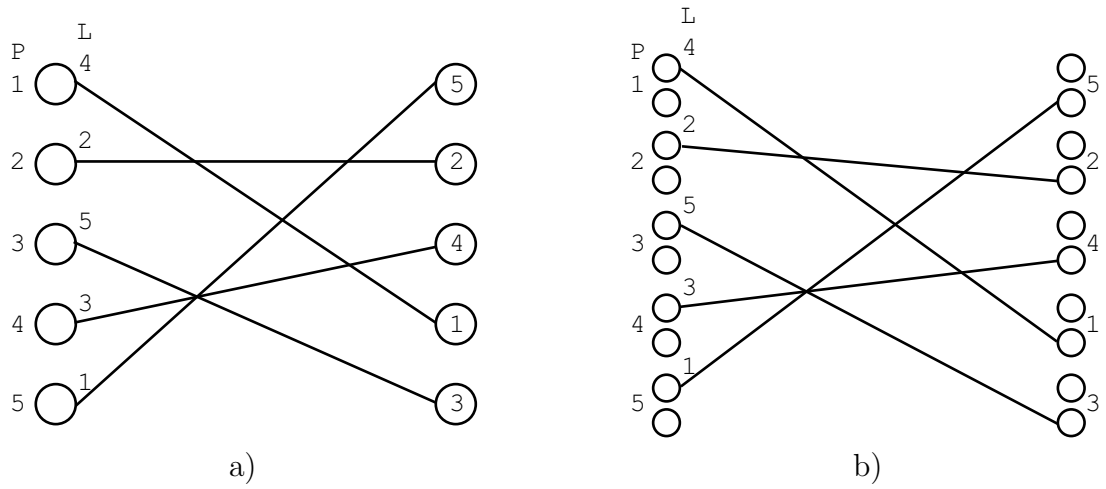



Figura 22: a) Ejemplo de una permutación donde $P = \{1, 2, 3, 4, 5\}$ y $L = \{4, 2, 5, 3, 1\}$. Las dos columnas de procesadores están conectadas mediante ductos en función de la permutación $L$. b) Representación en el modelo LR-Mesh de la permutación. Cada elemento de $P$ está representado por dos procesadores.

En la Figura 22a, se puede observar un ejemplo de una permutación del conjunto

$P = \{1, 2, 3, 4, 5\}$. El procesador 1 de la primer columna está conectado con el procesador 4 de acuerdo a la permutacion $L = \{4, 2, 5, 3, 1\}$. De forma similar el procesador 2 está conectado con el procesador 2, el 3 con el 5 y así sucesivamente.

El algoritmo para el modelo LR-Mesh requiere que cada elemento de $P$ se represente por dos procesadores en renglones consecutivos. Uno de los procesadores tiene identificador par y el otro impar. El procesador con identificador par tiene la información de entrada del algoritmo: el elemento $p_i \in P$, el elemento $l_i$ de la permutación $L$ de $P$ y el dato $d_i$. En la Figura 22b se puede observar un ejemplo de esta representación. Note que todos los ductos principian en un procesador con identificador par de la primer columna y terminan en un procesador impar de la última columna. Esto permite la creación en el modelo LR-Mesh de los ductos que comunican ambas columnas. La cantidad de columnas que se requieren para crear dicha estructura de ductos, para $|P| = N$, es $N$.

Por lo tanto el algoritmo requiere una malla con $2N$ renglones, ya que cada elemento de $P$ se representa con dos procesadores, y $N+2$ columnas, la primer columna representa al conjunto $P$, la última columna la permutación y $N$ columnas intermedias para la estructura de ductos.

La creación de la estructura de ductos se compone de tres etapas. Primero cada procesador par, que representa al elemento $p_i$, crea un ducto horizontal hasta el procesador de la columna $i$. En la Figura 23a se puede observar este proceso. El procesador par que representa al elemento 1 crea un ducto horizontal hasta el procesador de la columna 1. De forma similar el procesador que representa al elemento 2 crea un ducto hasta el procesador del mismo renglón que se encuentra en la columna 2 y así sucesivamente. Con el ducto formado se puede transmitir el dato $l_i$ y lo recibe el procesador de la columna $i$ correspondiente. Cada procesador que recibe un dato $l_i$ prolonga el

ducto, esta vez en forma vertical, hasta el procesador impar que representa al elemento $l_i$. En la Figura 23b se ejemplifica esta etapa. El procesador impar que representa el elemento 3 crea un ducto hasta la columna 3 en forma horizontal y prolonga el ducto en forma vertical hasta el procesador impar que representa al elemento 5. Finalmente, los $N$ ductos se prolongan en forma horizontal hasta alcanzar la última columna de la malla, como se muestra en la Figura 23c. Cada uno de los tres pasos se puede realizar en tiempo constante.

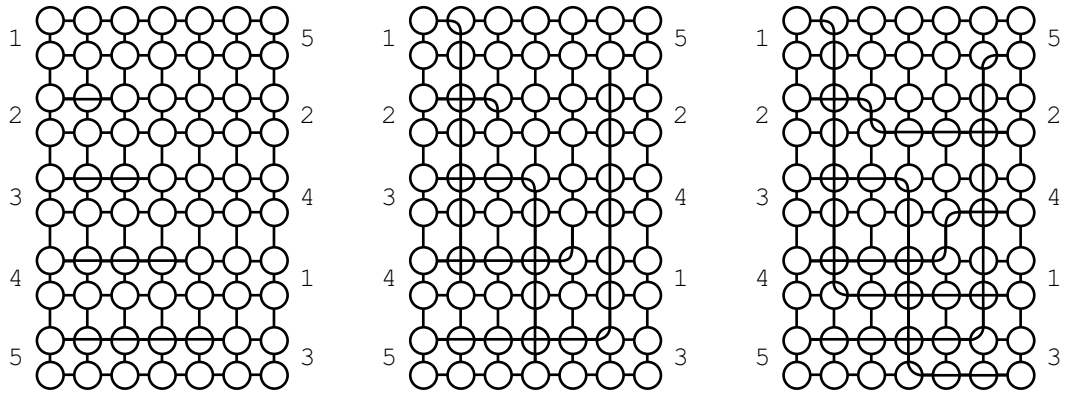



Figura 23: Ejemplo de las tres etapas para la creación de la estructura de ductos de acuerdo a la permutación $L = \{4, 2, 5, 3, 1\}$. a) Ductos en forma horizontal hasta la diagonal principal. b) Ductos verticales hasta el elemento deseado. c) Prolongación de cada ducto hasta la última columna.

Cuando se tiene la estructura de ductos formada es posible realizar la permutación deseada. Sólo se requiere que cada procesador par de la primer columna transmita por el ducto correspondiente el dato $d_i$. El dato se propaga por el ducto y el elemento $l_i$ de la última columna lo recibe.

A continuacón se muestra un pseudocódico del algoritmo de permutación.

**Entrada**: Cada procesador con identificador par tiene el elemento $p_i \in P$, un elemento $l_i \in L$ de la permutación de $P$ y el dato $d_i$.

**Salida**: El elemento $d_i$ correspondiente a la permutación $L$ en cada procesador con identificador par.

BEGIN

1. Los procesadores con identificador par de la primer columna, que representan a un elemento $p_i \in P$, transmiten mediante un ducto horizontal el elemento $l_i$ al procesador de la columna $i$.

2. Cada procesador que recibe el elemento $l_i$ prolonga el ducto en forma vertical al procesador con identificador impar que representa al elemento $l_i$.

3. Se prolongan los $N$ ductos en forma horizontal hasta la última columna.

4. Los procesadores con identificador impar de la primer columna transmiten por el ducto correspondiente el dato $d_i$.

5. Los procesadores con identificador par de la última columna leen del ducto y almacenan el dato en $d_i$.

END

**Lema 7.** *La permutación de los elementos $d_i$, dados un conjunto de procesadores $P = \{p_1, p_2, \ldots, p_N\}$ y una permutación de $P$, $L = \{l_1, l_2, \ldots, l_N\}$, se puede realizar en el modelo LR-Mesh de tamaño $2N \times N + 2$ en tiempo constante.*

## VI.6 Algoritmos

En este capítulo se muestran una serie de algoritmos básicos diseñados para el modelo LR-Mesh. Todos ellos requieren una cantidad constante de unidades de tiempo para su ejecución y fungen como bloques constructores para los problemas complejos que se

tratan en los capítulos siguientes. En la Tabla III se pueden observar en forma resumida dichos algoritmos así como la cantidad de recursos que requiere cada uno.

| **Algoritmo** | **Renglones×Columnas** |
|---|---|
| Suma de Prefijos | $(N+1) \times N$ |
| División | $(N+1) \times (N - \lfloor \frac{N}{D} \rfloor)$ |
| Multiplicación | $(N+1) \times (N - \lfloor \frac{N}{D} \rfloor)$ |
| División en cascada | $(N+1) \times ((N - \lfloor \frac{N}{D} \rfloor)(k))$ |
| Multiplicación en cascada | $(N+1) \times ((N - \lfloor \frac{N}{D} \rfloor)(k))$ |
| Conversión de un número a base D | $(N+1) \times ((N - \lfloor \frac{N}{D} \rfloor)(k))$ |
| Permutación | $2N \times N + 2$ |

Tabla III: Algoritmos fundamentales para resolver el problema de USTCON en el modelo LR-Mesh en tiempo constante.

# Capítulo VII

# Transformar un grafo $G$ a un expansor $G'$ en tiempo constante en el modelo LR-Mesh

El problema principal de la simulación del modelo R-Mesh en el modelo LR-Mesh es de conectividad. El problema USTCON es posible resolverlo en el modelo R-Mesh en tiempo constante y el algoritmo es bien conocido. Resolver el problema USTCON en el modelo LR-Mesh en tiempo constante es un problema complejo. Siguiendo el algoritmo de Reingold (Reingold, 2005) se dividió el problema USTCON en dos subproblemas: convertir un grafo cualquiera $G$ en un expansor $G'$ y resolver el problema de conectividad en un expansor. En este capítulo se muestra un algoritmo que resuelve el primero de los problemas. El capítulo siguiente se dedica a resolver el problema de conectividad en un expansor.

El algoritmo de Reingold se diseñó para la Máquina Determinística de Turing. La entrada del algoritmo, en la cinta de sólo lectura, es el mapa rotacional de un grafo $G(N, d^{16})$, un expansor $H(d^{16}, d)$, un vértice cualquiera $\overline{v}'$ del expansor $G'$ y una de sus aristas $a'$. Con esta información, el algoritmo calcula el rotacional en $G'$ del vértice $\overline{v}'$ a través de la arista $a'$. Dicho de otra forma se calcula sólo una arista del expansor $G'$.

El algoritmo para el modelo LR-Mesh construye una red multietapa que calcula el $\mathrm{Rot}_{G'}(\overline{v}', a')$ para todos los posibles pares $(\overline{v}', a')$ en paralelo. Es decir se calcula el mapa rotacional completo del expansor $G'$. Siguiendo el mismo procedimiento del algoritmo recursivo de Reingold, se realizan las operaciónes de producto *zig-zag* y la potenciación pero con una singular diferencia: el proceso es ahora concurrente. Todas las posibles operaciones se realizan en forma independiente y paralela mediante los ductos del modelo LR-Mesh.

A continuación se muestra el algoritmo que resuelve USTCON en un grafo $d^{16}$ regular en el modelo LR-Mesh en tiempo constante. Se describe cada uno de los pasos del algoritmo y se analiza la cantidad de recursos que se requieren. Finalmente se muestra un ejemplo pequeño del cálculo de un expansor.

## VII.1 Algoritmo $G \rightarrow G'$

El algoritmo de Reingold para la máquina de Turing transforma el grafo de entrada en un expansor mediante el producto zig-zag. Para realizar el producto zig-zag entre el grafo actual y el expansor $H$ se sustituye cada vértice del grafo actual por una copia del grafo $H$. En la Figura 24a se muestra la sustitucion de un vértice $v$ del grafo $G$ por una copia de $H$. Para referirse a un vértice de dicho grafo se requiere un vértice $v \in G$ y un vértice $a_0 \in H$. Cada vértice de $H$ se sustituye a su vez por otra copia de $H$. Se hace referencia a un vértice de este grafo mediante un vértice $v \in G$ y dos vértices $a_0, a_1 \in H$. Este proceso se realiza $l$ veces, donde $l$ es la profundidad máxima de la recursión.

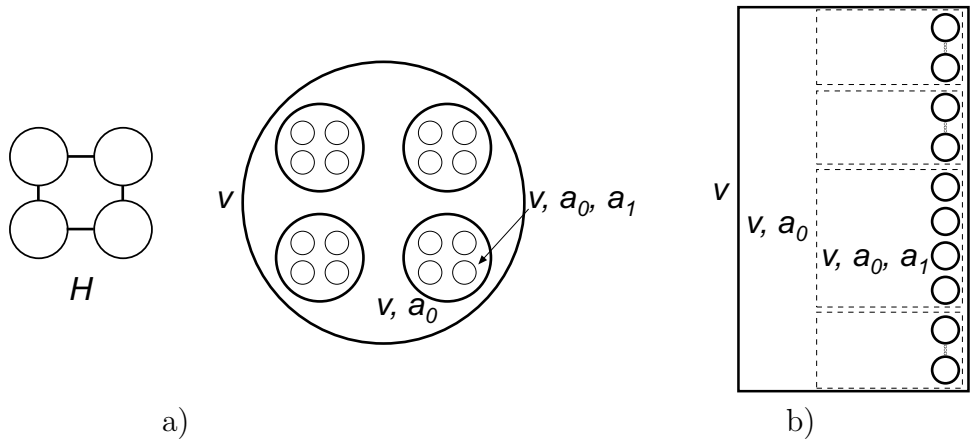



Figura 24: a)Expansor $G'$. Cada vértice de $V$ se sustituye por una copia de $H$ con cada operación zig-zag. b) Representación de cada vértice del expansor $G'$ en el modelo LR-Mesh.

En el algoritmo, para el modelo LR-Mesh, se crea un red múltietapa para obtener el mapa rotacional del grafo $G'$. Cada etapa se compone de $d^{16(O(\log N)+1)}$ procesadores, cada uno de ellos representa un vértice y una arista del grafo $G'$. Un subconjunto de procesadores representa ya sea un vértice de $G$ o alguna de las $O(\log N) + 1$ variables $a_i$. En la Figura 24b se observa la representación de un vértice cualquiera del grafo $G'$ en alguna etapa. Un procesador por si solo representa la etiqueta $a_1$. Un grupo de $d^{16}$ procesadores consecutivos representan la etiqueta $a_0$. Finalmente $d^{16} \times d^{16}$ representan a un vértice de $G$. Esto claro para un nivel de recursión $l = 2$. Cada etapa de la red se compone de los $N$ vértices del grafo $G$ expresados como en la Figura 24b.

Cada etapa de la red representa una operación. Una operación se refiere al cálculo del rotacional del grafo $H$ o $G$ según corresponda. La red multietapa requiere determinar para una etapa dada $k$, el nivel de recursión del algoritmo secuencial y el tipo de operación ($H$ o $G$) que éste realiza en el tiempo $k$. Dependiendo del nivel de recursión correspondiente, la operación afecta a un par de variables $a_i$ y $a_{i-1}$. En la Figura 25a se observa la $j$-ésima etapa $H$ del nivel de recursión $i$. Se puede ver la conexión del vértice $(a_{i-1}, a_i)$ con el vértice $(a'_{i-1}, a'_i)$ dado el $\mathrm{Rot}_H(a_{i-1}, k_{i,j}) = (a'_{i-1}, k_{i,j})$. Note que las variables $a_{i-2}, \ldots, a_0$ no se ven afectadas por el movimiento ya que éste ocurre dentro del vértice $a_{i-2}$. Tampoco se afectan las variables $a_{i+1}, \ldots, a_{O(\log N)}$ ya que la conexión entre los procesadores es uno a uno. De igual forma la etapa $H$ número 16 tiene que realizar la inversión de etiquetas de acuerdo al algoritmo para la máquina de Turing.

En el caso de una etapa $G$ cada vértice $(v, a_0)$ se conecta con el vértice $(v', a'_0)$ tal que $\mathrm{Rot}_G(v, a_0) = (v', a'_0)$. En la Figura 25b se observa cómo se realiza esta conexión. Se modifican las variables $v$ y $a_0$ por $v'$ y $a'_0$, respectivamente. Note que las variables $a_1, \ldots, a_{O(\log N)}$ no se alteran ya que las conexiones entre los procesadores son uno a uno, por lo tanto el índice relativo dentro del vértice $a_0$ permanece.

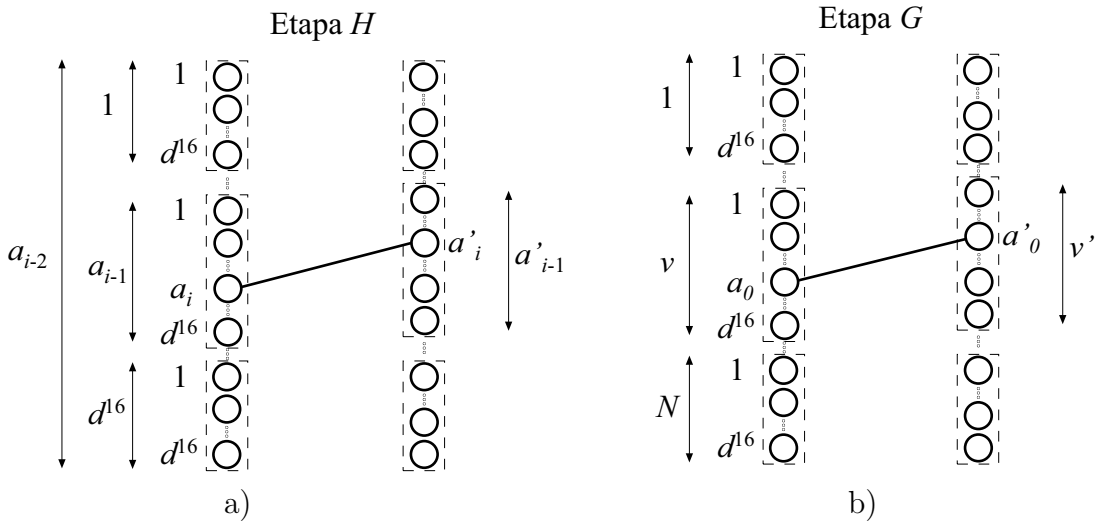



Figura 25: a)La $j$-ésima etapa $H$ de nivel de recursión $i$. El vértice $(a_{i-1}, a_i)$ se conecta al vértice $(a'_{i-1}, a'_i)$ tal que el $\mathrm{Rot}_H(a_{i-1}, k_{i,j}) = (a'_{i-1}, k_{i,j})$ b) Etapa $G$. Cada vértice $(v, a_0)$ se conecta al vértice dado por $\mathrm{Rot}_G(v, a_0)$.

Se puede decir que cada etapa del algoritmo en el modelo LR-Mesh realiza una operación equivalente a las operaciones realizadas por el algoritmo de Reingold en la máquina de Turing de Reingold. La diferencia estriba en que cada procesador de todas las etapas realiza la conexión entre él y el procesador correspondiente de la siguiente etapa en forma concurrente.

A continuación se muestra el algoritmo para transformar un grafo cualquiera $G(N, d^{16})$ a un expansor $G'(N \times d^{16O(\log N)}, d^{16}, \lambda \leq \frac{1}{2})$ dado un expansor $H(d^{16}, d, \lambda \leq \frac{1}{2})$.

**Entrada**: El mapa rotacional del grafo $G(N, d^{16})$. La columna 0 tiene esta información: Los primeros $d^{16(O(\log N)+1)}$ poseen el rotacional del vértice 0 a través de las $d^{16}$ aristas y así sucesivamente.

**Salida**: El mapa rotacional del expansor $G'(N \times N^c, d^{16}, \lambda \leq \frac{1}{2})$, donde $O(\log N) \leq a \log N$ y $c = 16a \log N$, en la última columna de la malla.

BEGIN

1. Generar $H$.
2. Calcular $l$.
3. Construir la red multietapa.
   (a) Calcular el número de etapas.
   (b) Calcular el tipo de operación y el nivel de recursión.
   (c) Calcular las variables $a_i$ y $v$ para cada procesador.
   (d) Realizar las conexiones.
4. Obtener el rotacional de $G'$

END

A continuación se describen cada uno de estos pasos a detalle, se muestra como se realiza cada paso en tiempo constante y la cantidad de recursos necesarios.

## VII.2 Generar $H$

De acuerdo al algoritmo de Reingold, se construye un grafo $H$ con $d^{16}$ vértices, $d$ regular y con una expansión espectral $\lambda \leq \frac{1}{2}$. Este proceso es trivial. Se puede construír el grafo expansor mediante la técnica mostrada en el Capítulo IV e incluso se puede utilizar fuerza bruta. Se tiene una cantidad constante de vértices por lo tanto se puede realizar una búsqueda exhaustiva hasta encontrar un grafo con las características deseadas.

Se transmite el mapa rotacional del grafo $H$ a toda la malla una vez calculado. Cada procesador de la malla entera puede tener una copia de $H$ en tiempo constante ya que la cantidad de vértices y por ende aristas del grafo es una constante.

Este paso sólo requiere de un procesador para ejecutarse.

## VII.3 Calcular $l$

Una vez construido el expansor $H$ con las características deseadas se requiere calcular el valor $l$ tal como se muestra en el algoritmo del Capítulo V. El valor de $l$ es el entero más pequeño que resuelve la inecuación:

$$\left(1 - \frac{1}{d^{16}n^2}\right)^{2^l} \leq \frac{1}{2} \tag{1}$$

$l = O(\log N)$ satisface 1, es decir una constante $a$ multiplicada por $\log N$. El problema radica entonces en encontrar el valor de la constante $a$. En un sólo procesador se puede obtener este resultado de la siguiente manera:

1. Se inicializa $a = 0$;
2. Se incrementa el valor de $a$ en uno.
3. Se evalua la expresión 1. Si no es verdadera ir al paso 2. En caso contrario terminar.

Se verifica si $a = 1$ es una solución. Si $a$ es una solución se termina el proceso. Si $a$ no es una solución se incrementa en uno y se vuelve a probar. Finalmente se transmite el valor de $a$ a toda la red.

El proceso es enteramente secuencial, sólo se requiere de un procesador. La cantidad de evaluaciones de la inecuación es exactamente $a$, por lo tanto el proceso requiere una cantidad constante de pasos.

## VII.4 Construir la red multietapa

En la máquina de Turing el algoritmo tiene en la cinta de trabajo un vértice del expansor, compuesto de un vértice de $G$, $O(\log N)$ vértices del expansor $H$ (denominados $a_0, \ldots, a_{O(\log N)-1}$), y una arista del expansor (un vértice de $H$ denominado $a_{O(\log N)}$). Durante el proceso recursivo se presentan dos tipos diferentes de operaciones básicas. La operación tipo $H$ ocurre cuando se requiere calcular el rotacional del expansor $H$. Para un nivel de recursión $i$, una operación $H$ consiste en efectuar $a_{i-1}, k_{i,j} \leftarrow \mathrm{Rot}_H(a_{i-1}, k_{i,j})$. Se puede observar que el algoritmo toma las variable $a_{i-1}$ y $k_{i,j}$ y las sustituye por el rotacional en $H$ (note que para un nivel de recursión dado sólo importan los valores de las dos variables $a$ y $k$, el resto de las variables no se requieren). Se puede entender esta operación como un translado del vértice $a_{i-1} \in H$ a través de la arista $k_{i,j}$ a un nuevo vértice mediante alguna arista. La operación tipo $G$ es el cálculo del rotacional del grafo $G$. El único nivel de recursión donde se requiere efectuar la operación $v, a_0 \leftarrow \mathrm{Rot}_G(v, a_0)$ es para $i = 1$. De igual forma se puede ver cómo un movimiento del vértice $v \in G$ por medio de la arista $a_0$ a un nuevo vértice $v' \in G$ por medio de la arista $a'_0$ (se puede observar, en forma similar a la operación $H$, que esta operación es independiente del resto de las variables $a_1, \ldots, a_{O(\log N)}$).

En la máquina de Turing se realizan las operaciones en forma secuencial. Para un instante dado $k$ la máquina de Turing realiza una operación $H$ o $G$. En el modelo LR-Mesh se requiere que la etapa $k$ realice la operación correspondiente de acuerdo al algoritmo para la máquina de Turing. En el caso base de la recursión se puede ver que las operaciones realizadas son las siguientes:

$$I_1 = HGH\ HGH\ HGH\ HGH\ HGH\ HGH\ HGH\ HGH$$

Se calcula ocho veces el producto *zig-zag* entre $G$ y $H$. Para calcular el producto zig-zag se requiere realizar una operación de tipo $H$ (correspondiente al primer paso corto), otra de tipo $G$ (correspondiente al paso largo) y finalmente una operación de tipo $H$ (correspondiente al segundo paso corto).

La red requiere tantas etapas como operaciones realizadas por la máquina de Turing para realizar el mismo trabajo en tiempo constante. En el caso base se requiere que cada una de las 24 etapas posea la información que indique el tipo de operación correspondiente. Debido a que el algoritmo, después de realizar 16 operaciones $H$, realiza una inversión de las variables $k_{i,1}, \ldots, k_{i,16}$, se requiere identificar la operación $H$ número 16.

Para el nivel de recursión $i = 2$ las operaciones realizadas son las siguientes:

$$I_2 = HI_1H\ HI_1H\ HI_1H\ HI_1H\ HI_1H\ HI_1H\ HI_1H\ HI_1H$$

Sólo se requiere agregar una etapa $H$ antes y después de una operación $I_1$ y repetirlo ocho veces. Para los niveles de recursion $i > 1$ se requiere además determinar para cada $H$ a que nivel de recursión pertenece, ya que de eso depende mediante cuales variables se calcula el rotacional del grafo $H$.

Para cualquier $i > 1$ las operaciones requeridas para calcular una arista de un expansor son:

$$I_i = HI_{i-1}H\ HI_{i-1}H\ HI_{i-1}H\ HI_{i-1}H\ HI_{i-1}H\ HI_{i-1}H\ HI_{i-1}H\ HI_{i-1}H$$

Cada una de los niveles de recursión se calcula de la misma manera y el caso base, $i = 1$, es el mostrado por $I_1$.

La construcción de la red multietapa consiste en calcular el número de etapas que constituyen la red, la operación que cada etapa realiza y el nivel de recursión al que

pertenecen. Se transmite esta información a cada etapa y en función de estos datos, cada procesador de la etapa $k$ crea una conexión con algún procesador de la etapa $k+1$ construyendo así la red multietapa. Finalmente se transmite cierta información a través de la red y se obtiene el rotacional. A continuación se explica a detalle cada uno de estos pasos.

### VII.4.1 Calcular el número de etapas

Se mostró la secuencia de etapas $H$ y $G$ para un nivel de recursión $i$. Esta secuencia está dada por la expresión $I_i$. El problema por resolver es determinar la cantidad de etapas que se requieren para convertir un grafo cualquiera a un expansor dada la profundidad $i$ de la recursión. Se sabe que para el caso base se requieren 24 etapas. Esto es debido a que cada etapa $G$ se acompaña de dos etapas $H$ y esta combinación se repite ocho veces.

Expresado en una función recursiva con el caso base $T(0) = 1$ la cantidad de etapas requeridas es:

$$T(1) = (T(0) + 2) \times 8 = (1 + 2) \times 8 = 24$$

$$T(2) = (T(1) + 2) \times 8 = (24 + 2) \times 8 = 208$$

Es fácil ver que para el caso general se tiene

$$T(k) = (T(k-1) + 2) \times 8$$

El objetivo es obtener todos los valores de $T(k)$, para toda $1 \leq k \leq O(\log N)$, para lo cual la función recursiva no es útil ya que no se puede paralelizar. Sin embargo es posible transformar la función a una función no recursiva.

La ecuación recursiva general es:

$$T(k) - 8T(k-1) - 16 = 0 \tag{2}$$

Sustituyendo $k$ por $k-1$ y $k-2$, respectivamente, en la Ecuación 2 se obtiene:

$$T(k-1) - 8T(k-2) - 16 = 0 \tag{3}$$

$$T(k-2) - 8T(k-3) - 16 = 0 \tag{4}$$

Múltiplicando por 8 la Ecuación 3 y sumándola a la Ecuación 2 se obtiene:

$$T(k) - 8^2T(k-2) - (2)(8^2) - (2)(8) = 0 \tag{5}$$

Múltiplicando por $8^2$ la Ecuación 4 y sumándola a la Ecuación 5 resulta:

$$T(k) - 8^3T(k-3) - (2)(8^3) - (2)(8^2) - (2)(8) = 0 \tag{6}$$

Se puede observar que si se repite $a$ veces este proceso se obtiene la siguiente Ecuación:

$$T(k) - 8^aT(k-a) - 2\sum_{i=1}^{a} 8^i = 0 \tag{7}$$

Sustituyendo $a$ por $k$ en la Ecuación 7 se obtiene:

$$T(k) = 8^kT(0) - 2\sum_{i=1}^{k} 8^i \tag{8}$$

Se sabe que $T(0) = 1$ y $\sum_{i=1}^{k} x^i = \frac{x^{k+1}-x}{x-1}$ por lo tanto se puede expresar la Ecuación 8 de la siguiente manera:

$$T(k) = 8^k + 2\frac{8^{k+1}-8}{7} = \frac{23}{7}8^k - \frac{16}{7} \tag{9}$$

Finalmente con la Ecuación 9 es posible calcular los valores $T(k)$, para toda $1 \leq k \leq O(\log N)$. Sólo se tiene que calcular todos los valores de $8^k$. Esto se puede lograr mediante el algoritmo de multiplicación en cascada, Lema 5.

La cantidad total de etapas requeridas por el algoritmo en el modelo LR-Mesh para un grafo que requiere un nivel de recursión $l$ es:

$$T(l) = \frac{23}{7}8^l - \frac{16}{7} \tag{10}$$

más una etapa extra que sirve para almacenar el rotacional del expansor.

Se sabe que el valor de $l = O(\log N) \leq a \log N$ por lo tanto:

$$T(l) = \frac{23}{7} 8^{a \log N} - \frac{16}{7} = \frac{23}{7} N^{a \log 8} - \frac{16}{7} = \frac{23}{7} N^{\alpha} - \frac{16}{7}$$

donde $\alpha = a \log 8$.

La red se compone de exactamente $\frac{23}{7} N^{\alpha} - \frac{9}{7}$ etapas.

### VII.4.2 Cálcular el tipo de operación y el nivel de recursión

Determinar para cada etapa el tipo de operación que se realizará y además el nivel de recursión es el problema medular de la transformación a un expansor de un grafo cualquiera. Se tienen $\frac{23}{7} N^{\alpha} - \frac{9}{7}$ etapas y se requiere determinar el tipo de operación ($H$ o $G$), si es una operación $H$ se determina la cantidad de operaciones $H$ que ocurrieron con anterioridad (únicamente las del mismo nivel de recursión) asignando un valor $1 \leq j \leq 16$ a cada etapa, además del nivel de recursión mismo.

El algoritmo se organiza en $O(\log N)$ fases (columnas). Cada una representa a un nivel de recursión. Cada renglón representa una de las $\frac{23}{7} N^{\alpha} - \frac{9}{7}$ etapas necesarias para transformar un grafo cualquera en un expansor. La primer columna representa al nivel de recursión $O(\log N)$ y la última etapa al nivel de recursión 1.

En cada nivel de recursión $i$ se puede determinar las etapas $H$ que acompañan a cada expresión $I_{i-1}$, de acuerdo a la expresión $I_i$. La forma más sencilla de determinar el nivel de recursión es calculando para cada fase $i$ el valor de $T(i)$. Se divide el identificador del procesador (renglón) entre $T(i)$. En todas las divisiones en que el residuo es cero se tiene una etapa $H$ correspondiente al nivel de recursión $i$ y el procesador inmediato

superior también corresponde a una etapa tipo $H$ del nivel de recursión $i$, al igual que el procesador del renglón 0.

Después en cada etapa $i$ se enumeran en forma consecutiva todos los vértices que corresponden a una etapa tipo $H$ (empezando de cero). Este índice se divide entre 16 y el residuo se almacena en la variable $j$. Después se enumeran los procesadores que no corresponden a un vértice $H$, asignándoles una etiqueta $l$ (de igual forma principiando de cero).

Cada procesador que no corresponde a una etapa $H$ establece un ducto al procesador $l$ de la etapa siguiente. De esta forma se crea una estructura de ductos que comunica a cada procesador de la columna cero con un procesador perteneciente a alguna de las fases $i$. En la fase 1 los procesadores que no se etiquetan como etapa tipo $H$ se consideran etapas $G$.

Finalmente cada procesador que se etiquetó como tipo $H$ o $G$ transmite por el ducto la fase $r$ (nivel de recursion) a la que pertenecen. Si es un procesador que representa una etapa tipo $H$ transmite también el índice $j$. Cada procesador de la columna 0, que representa a una etapa $k$ del algoritmo, recibe esta información y de esta forma determina que tipo de operación realiza la etapa $k$ y el índice $j$ si se requiere.

Se puede entender el algoritmo de la siguiente manera: cada fase $i$ determina las etapas $H$ correspondientes al nivel de recursión $i$. Las etapas $H$ y $G$ que no son parte del nivel de recursion $i$ se transmiten a la siguiente fase en donde se realiza el mismo proceso.

A continuación se muestra el algoritmo para determinar para cada etapa $k$, el tipo de operación que le corresponde y la cantidad de etapas $H$ anteriores (índice $j$).

Entrada: Se tienen $\frac{23}{7}N^{\alpha} - \frac{9}{7}$ renglones. Cada procesador de la primer columna representa a la etapa $k$ del algoritmo que convierte un grafo cualquiera en un expansor. Se tienen $O(\log N)$ etapas. Cada etapa representa un nivel de recursión. La etapa 1 representa al nivel de recursión $O(\log N)$ y la etapa $O(\log N)$ representa al caso base de la recursión.

Salida: La correspondiente operación para cada etapa $k$ y el índice $j$ para las etapas $H$.

1. Se calculan los $O(\log N)$ valores $T(i)$. La fase que representa al nivel de recursión $i$ almacena el dato $T(i)$. Este paso se puede realizar en tiempo constante resolviendo la Ecuación 9.

2. Cada etapa $i$ está coformada de acuerdo a la expresión $I_i$. En función de esto se determina (mediante el valor $T(i)$) los elementos $H$ que acompañan a la expresión $I_{i-1}$. Los procesadores de la fase 1 que no se etiquetan como $H$ se consideran etapas $G$.

3. Se enumeran en forma consecutiva (empezando de 0) cada procesador que representa a un elemento de la expresión $I_{i-1}$. Este índice permite a cada procesador conectarse a la siguiente etapa. Este paso se puede realizar en tiempo constante gracias al Lema 7.

4. Se enumeran en forma consecutiva (empezando de 0) cada procesador que no fue etiquetado en el paso anterior. La etiqueta se divide entre 16 y el residuo es el índice $j$.

5. Finalmente cada procesador ya etiquetado como $H$ o $G$ transmite por el ducto correspondiente el nivel de recursión $r$ al que pertenecen y al índice $j$ en su caso. Los procesadores de la columna uno reciben esta información y la almacenan.

La cantidad de renglones que requiere el algoritmo es exactamente $\frac{23}{7}N^{\alpha} - \frac{9}{7}$. Uno para cada etapa. La cantidad de fases que requiere el algoritmo son $O(\log N)$ pero se requieren $\frac{23}{7}N^{\alpha} - \frac{9}{7}$ columnas adicionales entre cada fase para permitir la conexión de la fase $i$ con la $i-1$.

**Lema 8.** *El cálculo del nivel de recursión y el tipo de operación para cada etapa del algoritmo que convierte un grafo cualquiera en un expansor se puede realizar en el modelo LR-Mesh en tiempo constante. Se requiere una malla con* $\frac{23}{7}N^{\alpha} - \frac{9}{7}$ *renglones y* $(\frac{23}{7}N^{\alpha} - \frac{9}{7}) \times O(\log N)$ *columnas, donde* $O(\log N) \leq a \log N$ *y* $\alpha = a \log 8$.

### VII.4.3 Calcular las variables $a_i$ y $v$ para cada procesador

Para convertir un grafo cualquiera, el algoritmo de Reingold mantiene una recursión de a lo más $O(\log N)$. Esto complica considerablemente el algoritmo en tiempo constante para el modelo LR-Mesh. Se mostró cómo determinar el tipo de operación que realiza cada etapa, el nivel de recursión al que pertenece cada etapa $H$ y el índice de cada una de ellas (un valor entre 1 y 16). Aunado a esto se requiere determinar la variable $v$, las $O(\log N)$ variables $a_i$ y la cantidad de procesadores que se requieren para representar cada una de estas variables. Estás variables corresponden a los renglones de la red multietapa.

A continuación se muestra cómo se puede calcular esta información en tiempo constante.

#### Cálculo de procesadores

El problema consiste en calcular cuántos procesadores se requieren para representar la cantidad total de vértices dado un nivel de recursión $i$.

Para el caso base ($i = 1$) se requieren $N \times d^{16} \times d^{16} = d^{(2)(16)} \times N$ procesadores. Para $i = 2$ se requiere $N \times d^{16} \times d^{16} \times d^{16} = d^{(3)(16)} \times N$. Es sencillo ver que para cualquier $i > 1$ la cantidad de procesadores requeridos son

$$d^{i+1} \times N$$

Se requieren $d^{(l)(16)}$ procesadores para representar las $l$ variables $a_i$, para toda $0 \leq i < l$. Debido a que cada uno de estos nodos se divide en $d^{16}$ procesadores, que representan las $d^{16}$ aristas del grafo expansor, se tienen $d^{(l+1)(16)}$ procesadores por cada uno de los $N$ vértices del grafo $G$.

El algoritmo requiere una malla que tenga al menos $d^{16(l+1)} \times N$ renglones ya que se requieren esa cantidad de procesadores para representar cada uno de los vértices del expansor y sus $d^{16}$ aristas. Debido a que la profundidad de la recursión del algoritmo es de $O(\log N)$ se requieren por lo menos $d^{16(O(\log N)+1)} \times N$ renglones, es decir $dN^{\beta+1}$ renglones, donde $O(\log N) \leq a \log N$ y $\beta = 16a \log d$.

**Cálculo de las variables $a_i$ y $v$**

Se puede calcular en el modelo LR-Mesh en tiempo constante las multiplicaciones parciales $d^k$, para toda $1 \leq k \leq l$, gracias al Lema 5. Cada etapa del algoritmo sin importar el nivel de recursión puede determinar en tiempo constante cada una de los resultados parciales. El resultado de $d^1$ se puede transmitir en forma horizontal mediante el renglón 1. El resultado de $d^2$ por el renglón 2 y así sucesivamente.

Suponiendo que la $k$-ésima etapa tipo $H$ conoce el nivel de recursión $i$ correspondiente se puede calcular los valores de $a_i$ y $a_{i+1}$ gracias a las multiplicaciones parciales con la fórmula:

$$a_i = \left( \frac{id}{d^{(l-i)(16)}} ) mod (d^{16} \right)$$

donde $a_i$ es la variable buscada, $id$ el renglón del procesador, $i$ el nivel de recursión correspondiente a la etapa $k$, $l$ la profundidad de la recursión (normalmente $O(\log N)$). El valor $d^{(l-i)(16)}$ se puede acceder mediante el procesador del renglón $l - i$.

En las etapas $G$ se puede obtener la variable $v$ simplemente dividiendo el identificador del procesador entre $d^{(l+1)(16)}$.

### VII.4.4 Realizar las conexiones

Cada procesador de una etapa $H$, perteneciente al nivel de recursión $i$, requiere calcular las variables $a_i$ y $a_{i+1}$. En función de dichas variables se calcula el rotacional en $H$ del vértice $(a_i, k_{i+1,j})$ y se obtiene un nuevo par $(a'_i, a'_{i+1})$. Con esta información cada etapa crea un ducto entre estos dos vértices. Para crear dicho ducto cada procesador (con identificador $id_s$) requiere conocer el identificador del procesador destino ($id_t$). Este identificador se puede obtener de la siguiente forma:

$$id_t = id_s + (a'_i - a_i) \times d^{(l-i)(16)} + (a'_{i+1} - a_{i+1}) \times d^{(l-i-1)(16)} \tag{11}$$

Para las etapas $G$ el proceso es muy similar:

$$id_t = id_s + (v' - v) \times d^{(l)(16)} + (a'_0 - a_0) \times d^{(l-1)(16)} \tag{12}$$

Después de calcular esta información finalmente se puede crear en tiempo constante la red multietapa que permite transformar un grafo $G$, $d^{16}$ regular a un expansor $G'$. Cada procesador conoce la etapa a la que pertenece, el nivel de recursión, el vértice del expansor que representa así como el identificador del procesador destino. Cada procesador de la etapa $k$ crea un ducto con el procesador con identificador $id_t$ de la etapa $k + 1$ de la siguiente manera:

- Para la $j$-ésima etapa $H$ de nivel de recursión $i$, para toda $0 < j < 16$, cada procesador con identificador $(a_{i-1}, k'_{i,1}, \ldots, k_{i,j}, \ldots, k_{i,16})$ de dicha etapa se conecta al procesador $(a'_{i-1}, k'_{i,1}, \ldots, k'_{i,j}, \ldots, k_{i,16})$ de la siguiente etapa, tal que $\mathrm{Rot}_H(a_{i-1}, k_{i,j}) = (a'_{i-1}, k'_{i,j})$. Para la etapa $H$ número 16 cada procesador $(a'_{i-1}, k'_{i,1}, \ldots, k'_{i,15}, k_{i,16})$ se conecta al procesador $(a'_{i-1}, k'_{i,16}, \ldots, k'_{i,2}, k'_{i,1})$, tal que $\mathrm{Rot}_H(a_{i-1}, k_{i,16}) = (a'_{i-1}, k'_{i,16})$.

- Para cada etapa $G$ se conectan los procesadores que representan al vértice $(v, a_0)$ con los procesadores que representan al vértice $(v', a'_0)$ de la siguiente etapa, tal que se cumpla que el $\mathrm{Rot}_G(v, a_0) = (v', a'_0)$.

- Cada procesador de cada una de las etapas fusiona sus puertos Este y Oeste creando con esto un ducto que recorre toda la malla. En cada procesador $(v, a_0, \ldots, a_{O(\log N)-1}, a_{O(\log N)})$ de la primer columna principia un ducto que recorre la malla realizando las operaciones $H$ y $G$ de tal forma que el ducto termina en el procesador $(v', a'_0, \ldots, a'_{O(\log N)-1}, a'_{O(\log N)})$ de la última columna y se cumple que $\mathrm{Rot}_{G'}((v, a_0, \ldots, a_{O(\log N)-1}), a_{O(\log N)}) = ((v', a'_0, \ldots, a'_{O(\log N)-1}), a'_{O(\log N)})$.

## VII.5 Obtener el rotacional de $G'$

Una vez creada la red multietapa cada vértice de la primer columna transmite su etiqueta a través del ducto y de esta forma en la última columna se obtiene el mapa rotacional del grafo $G'$.

Existe un inconveniente al realizar este paso. Cada vértice del expansor se expresa con $O(\log N)$ etiquetas. Por lo tanto, tratar de enviar estas $O(\log N)$ requiere una cantidad logarítmica de pasos. Una solución es crear $d^{O(\log N(16))} \times N$ mallas idénticas y en cada una de ellas transmitir una señal predefinida por un vértice predeterminado.

Este proceso requiere de una gran cantidad de recursos.

Existe una forma más eficiente y simple de realizar el proceso. Cada procesador convierte la etiqueta $(v, a_0, a_1, \ldots, a_{O(\log N)-1})$ a su valor decimal que coincidentemente es el identificador (renglón) del procesador. Por lo tanto no se requiere calcular nada. La información ya se conoce. De esta forma cada procesador sólo tiene que transmitir una cantidad constante de información por el ducto: su identificador de renglón, llamado $v''$, y la etiqueta $a_{O(\log N)}$.

El reetiquetado no afecta al grafo en si, las propiedades de conectividad se mantienen. Únicamente cambian las etiquetas de los vértices. Además cada vértice puede ser transformado, si se requiere, de su forma $(v, a_0, a_1, \ldots, a_{O(\log N)-1})$ a su forma decimal en tiempo constante gracias al Lema 6.

La cantidad de renglones que el algoritmo requiere es $dN^{\beta+1}$. Uno por cada par $(v \in G', a \in H)$. El algoritmo de permutación requerido para realizar las conexiones entre las etapas requiere el doble de renglones, Lema 7. Por lo tanto el algoritmo requiere $2dN^{\beta+1}$ renglones. La cantidad de etapas requeridas por el algoritmo es de $\frac{23}{7}N^{\alpha} - \frac{16}{7}$. Cada una de estas etapas requiere una cantidad igual a los renglones para poder realizar la estructura de ductos que las interconectan, por lo tanto la cantidad total de columnas que requiere el algoritmo es $(\frac{23}{7}N^{\alpha} - \frac{16}{7}) \times dN^{\beta+1}$.

**Lema 9.** *Transformar un grafo cualquiera $G(N, d^{16})$ a un expansor $G'(dN^{\beta+1}, d^{16}, \lambda \leq \frac{1}{2})$, donde $l = O(\log N) \leq a \log N$ y $\beta = 16a \log d$, se puede realizar en el modelo LR-Mesh en tiempo constante en una malla con $2dN^{\beta+1}$ renglones y $(\frac{23}{7}N^{\alpha} - \frac{16}{7}) \times dN^{\beta+1}$ columnas, donde $\alpha = a \log 8$.*

## VII.6 Caso base en el modelo LR-Mesh

Como ya se mencionó el algoritmo en el modelo LR-Mesh utiliza una red multietapa. Cada una de estas etapas representa una operación realizada por la máquina de Turing para obtener la arista del expansor. La etapa 1 corresponde a la primer operación efectuada por la máquina de Turing. La etapa $k$ corresponde a la $k$-ésima operación efectuada en el algoritmo secuencial.

Para un grafo que requiere sólo una iteración del algoritmo para convertirlo en expansor se realizan 24 operaciones. Por lo tanto se necesitan 24 etapas en el modelo LR-Mesh. Cada una de las etapas se conforma de $N$ vértices. A su vez cada uno de estos vértices se representa por $H$ vértices correspondientes al producto $zig - zag$. Finalmente cada uno de estos vértices se representa por $d^{16}$ procesadores. Con esto es posible representar los vértices del expansor ($D^{16}N$) y cada una de las $D^{16}$ aristas de cada vértice.

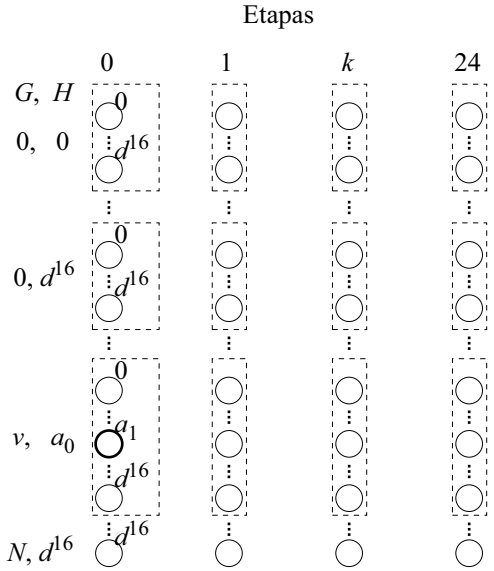



Figura 26: Representación en el modelo LR-Mesh del algoritmo para convertir un grafo a un expansor. Cada una de las primeras 24 etapas corresponden a una operación tipo $H$ o $G$. En la etapa 24 se almacena el resultado. Cada renglón representa un vértice y una arista de $G'$.

En la Figura 26 se observa la representación del algoritmo secuencial en el modelo LR-Mesh. Cada renglón de la malla representa un vértice y una arista del expansor $G'$. La cantidad total de renglones requeridos para resolver el problema es de $N \times d^{16} \times d^{16} = d^{32} \times N$. Se puede observar que cada vértice de $G$ está representado por los $d^{16}$ vértices de $H$ y cada uno de estos vértices está representado por otra copia de $H$. Cada columna está asociada con una de las dos operaciones del algoritmo secuencial. La columna 0 corresponde a una operación $H$, la columna 1 a una operación $G$ y así sucesivamente. La columna 24 se utiliza para almacenar el resultado del algoritmo.

En cada una de las columnas se tienen todos los posibles estados de las variables $((v, a_0), a_1)$. Cada procesador puede determinar, gracias a su identificador, dichas variables. Además no se requiere calcular todas las variables. Para los procesadores que pertenecen a una etapa $H$ sólo se necesita calcular los valores de $a_0$ y $a_1$. No se requiere conocer a qué vértice $v$ pertenece. Los procesadores de las etapas $G$ sólo requieren determinar las variables $v$ y $a_0$.

El algoritmo de Reingold además de utilizar las variables $a_i$ requiere que cada una de estas variables se represente por una secuencia de 16 aristas de $H$. Expresar cada $a_i \in \{0, \ldots, d^{16} - 1\}$ como $k_{i,1} \ldots k_{i,16}$, donde cada $k_{i,j} \in \{0, \ldots, d - 1\}$, es un proceso trivial ya que sólo se requieren realizar una cantidad constante de operaciones.

Cada grupo de procesadores representa entonces una variable de la máquina de Turing, ya sea un vértice de $G$ o uno de los $O(\log N)$ vértices de $H$. En cada grupo de procesadores, que pertenece a la $j$-ésima etapa $H$, se determina las variables $a_0$ y $k_{1,1}, \ldots, k_{1,16}$. Con esta información cada grupo de procesadores puede calcular el $\mathrm{Rot}_H(a_0, k_{i,j}) = (a'_0, k'_{i,j})$. Debido a que cada grupo de procesadores representa en si mismos un estado del total de las variables se puede crear un ducto entre cada grupo $(a_0, k_{i,j})$ de una etapa $H$ y el grupo que representa al par $(a'_0, k'_{i,j})$ de la siguiente etapa.

Dicho de otra forma se puede crear una permutación entre cada elemento $(a_0, k_{i,j})$ y el rotacional en $H$. Sólo se tiene que demostrar que el conjunto de pares $(a_0, k_{i,j})$ y su rotacional en $H$ son en efecto una permutación. La demostración se desprende directamente de la definición del rotacional de un grafo. Para obtener el rotacional de un grafo regular cada vértice y cada arista se etiqueta en forma única y definitiva, por lo tanto el conjunto de pares $(a_0, k_{i,j})$ con el rotacional correspondiente constituyen en efecto una permutación. No es posible que dos o más vértices se conecten a algún vértice $\alpha$ por medio de la misma arista (etiquetada en $\alpha$) ya que esto contradice la definición del rotacional de un grafo.

Para las etapas con operación tipo $G$ ocurre algo similar. Cada grupo de procesadores determinan las variables $(v, a_0)$. El conjunto de pares $(v, a_0)$ y sus respectivos rotacionales en $G$ conforman una permutación. Cada vértice se comunica con sus $d^{16}$ vértices vecinos por una arista etiquetada con un identificador único. Por lo tanto cada grupo de procesadores $(v, a_0)$ de la etapa $k$ puede crear un ducto con el grupo de procesadores $(v', a'_0)$ de la etapa $k+1$ en función de $\mathrm{Rot}_G(v, a_0) = (v', a'_0)$.

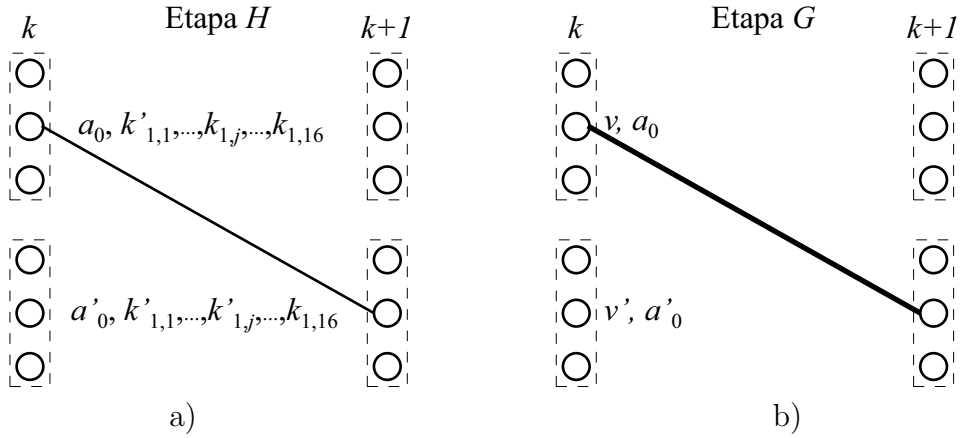



Figura 27: a) Conexión de la $j$-ésima etapa $H$. Cada vértice $(a_0, k'_{1,1}, \ldots, k_{1,j}, \ldots, k_{1,16})$ se conecta con el rotacional en $H$ del mismo. b) Conexión de una etapa $G$. Cada grupo que representa al vértice $(v, a_0)$ se conecta al grupo que representa al rotacional en $G$ del mismo.

Debido al arreglo de los procesadores en el modelo LR-Mesh se pueden efectuar las operaciones tipo $H$ y tipo $G$ afectando sólo a las variables deseadas, siempre y cuando las conexiones entre procesadore sean uno a uno. En la Figura 27 se muestra cómo se realizan estas conexiones. En el caso de la $j$-ésima etapa $H$ cada uno de los pares $(a_0, a_1)$ se representa por un único procesador. La conexión entonces se da entre los procesadores $(a_0, k'_{1,1}, \ldots, k_{1,j}, \ldots, k_{1,16})$ y $(a_0, k'_{1,1}, \ldots, k'_{1,j}, \ldots, k_{1,16})$ dentro de un vértice cualquiera $v$. Sólo se efectua un cambio de las variables $a_0$ y $a_1$, el vértice $v$ no se ve afectado; como se ve en la Figura 27a. El caso de una etapa $G$ es ligeramente diferente. Cada par $(v, a_0)$ se representa por una copia de $H$ ($d^{16}$ procesadores). Las conexiones entre el grupo de procesadores que represanta al vértice $(v, a_0)$ y $(v', a'_0)$ se realizan uno a uno. Ambos vértices están representados por la misma cantidad de procesadores. El primer procesador del grupo $(v, a_0)$ se conecta al primer vértice del grupo $(v', a'_0)$. El segundo con el segundo y así sucesivamente. Con esto se logra actualizar las variables $v$ y $a_0$ pero se mantienen intacta la variable $a_1$.

En la etapa $H$ número 16 se requiere de un proceso especial. Si fuera una etapa normal se conectaría cada procesador $(a_0, k'_{1,1}, \ldots, k'_{1,15}, k_{1,16})$ con el procesador $(a'_0, k'_{1,1}, \ldots, k'_{1,15}, k'_{1,16})$ de la etapa final, tal que el $\mathrm{Rot}_H(a_0, k_{1,16}) = (a'_0, k'_{1,16})$. Pero haría falta un proceso final. El algoritmo secuencial después de realizar las 16 etapas $H$ realiza una inversión de las etiquetas individuales $k_{1,i}$. Es decir la variable $a_1$ representada por $k_{1,1}, \ldots, k_{1,16}$ se cambia por $k_{1,16}, \ldots, k_{1,1}$. Esto se puede realizar en el modelo LR-Mesh realizando la conexión de cada par $(a_0, a_1)$ de la etapa 24 con el vértice $(a'_0, a'_1)$ de la siguiente manera: se conecta el vértice $(a'_0, k'_{1,1}, \ldots, k'_{1,15}, k_{1,16})$ con el vértice $(a'_0, k'_{1,16}, \ldots, k'_{1,2}, k'_{1,1})$ en función de $\mathrm{Rot}_H(a_0, k_{1,16}) = (a'_0, k'_{1,16})$.

Se crea la red multietapa de la forma descrita. En cada procesador $((v, a_0), a_1)$ de la primer columna principia un ducto que recorre la malla realizando las operaciones $H$ y

$G$ de tal forma que el ducto termina en el procesador $((v', a'_0, a'_1))$ de la última columna y además se cumple que $\mathrm{Rot}_{G'}((v, a_0), a_1) = ((v', a'_0), a'_1)$. Cada procesador $((v, a_0), a_1)$ de la primer columna transmite el identificador del vértice $v, a_0$ y la arista $a_1$ por la red multietapa. El procesador $((v', a'_0, a'_1))$ recibe esta información y construye con esto el rotacional de $G'$.

# Capítulo VIII

# USTCON en un expansor en tiempo constante en el modelo LR-Mesh

Esta sección presenta un algoritmo que resuelve el problema de conectividad entre dos vértices en un expansor en el modelo LR-Mesh en tiempo constante. Se tiene un expansor $G$ de $N$ vértices, $D$ regular, de diámetro logarítmico (esto es, la mayor de las distancias más cortas entre cualquier par de vértices, si están en el mismo componente conectado, es de $O(\log N)$), y dos vértices $u$ y $v$. El problema consiste en determinar si existe una trayectoria entre los vértices $u$ y $v$ en $G$.

El algoritmo que resuelve dicho problema en forma secuencial es sencillo. Sólo es necesario etiquetar en forma aleatoria, pero definitiva, cada una de las aristas del grafo, obteniendo con esto el mapa rotacional. Después se recorren todas las *trayectorias* de tamaño $O(\log N)$ partiendo del vértice $u$. Si en alguna de dichas trayectorias se encuentra el vértice $v$, entonces $u$ y $v$ están en el mismo componente conectado. En caso contrario, $u$ y $v$ no están en el mismo componente conectado.

Se puede ver que es factible, si se tiene más de un procesador, recorrer todas las trayectorias al mismo tiempo. La idea del algoritmo para el modelo LR-Mesh se basa en esta posibilidad. El resto del capítulo está organizado como sigue: primero se muestra un algoritmo que recorre una trayectoria cualquiera, a partir de un vértice $u \in G$, en un expansor en tiempo constante, después se muestra un algoritmo que resuelve el problema de conectividad en un expansor en tiempo constante.

## VIII.1 Recorrer una trayectoria de tamaño $O(\log N)$ en un expansor en tiempo constante

Una trayectoria se define como $\{d_0, d_1, d_2, \ldots, d_{O(\log N)-1}\}$, donde $d_i \in \{0, 1, 2, \ldots, D-1\}$ para toda $0 \leq i < O(\log N)$. Recorrer una trayectoria dada $\{d_0, d_1, d_2, \ldots, d_{O(\log N)-1}\}$ a partir del vértice $u = u_0$ implica un movimiento en $G$ del vértice $u_0$, por medio de la arista con etiqueta $d_0$, al vértice $u_1$, después, otro movimiento del vértice $u_1$, por medio de la arista $d_1$, al vértice $u_2$, y así sucesivamente.

El algoritmo aplica ciertas modificiaciones al grafo original de tal suerte que resulta sencillo empotrar el grafo final en el modelo LR-Mesh. Una vez realizado este proceso se puede determinar el conjunto de vértices $L \subseteq G$ que se *visitan* al recorrer una trayectoria $T$ cualquiera a partir de un vértice $u$. Posteriormente, en tiempo constante se puede determinar si el vértice $t$ se encuentra en el subconjunto $L$. De ser así, se puede concluir que existe una trayectoria entre los vértices $u$ y $t$ (*i. e.* $u$ y $t$ están en el mismo componente conectado). El caso contrario no es determinante ya que puede exisitir otra trayectoria $T' \neq T$ que describa un camino entre los vértices $u$ y $t$ en $G$.

El algoritmo primero obtiene una versión expandida del grafo original llamado grafo multietapa. Después sustituye cada vértice del nuevo grafo por $D + 1$ vértices lógicos. Finalmente se crea un tour de Euler en el grafo resultante. A continuación se explica cada uno de estos procesos y se analiza la cantidad de recursos que se requieren.

### VIII.1.1 Grafo multietapa

El fin del algoritmo es representar una trayectoria cualquiera $T$ de tamaño $O(\log N)$. Para poder representar dicha trayectoria en el modelo LR-Mesh, se requiere obtener una

representación multietapa del grafo original. Esto es se realizan $O(\log N)+1$ copias del grafo original.

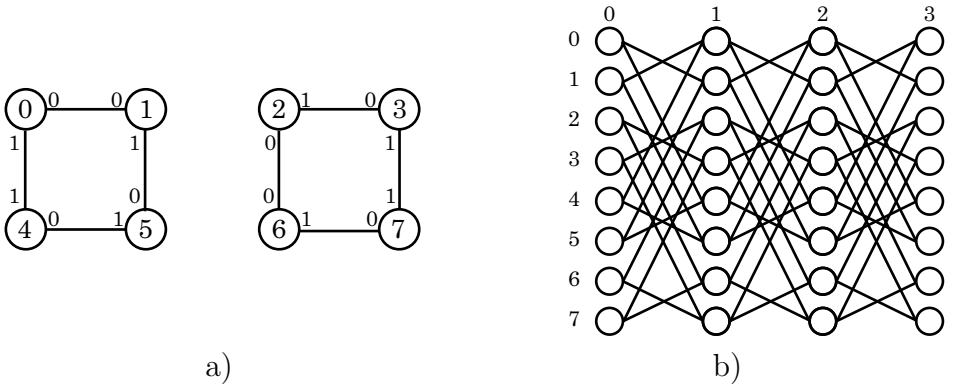



Figura 28: a)El mapa rotacional de un grafo G(8,2). b) La representación multietapa del grafo G(8,2).

El grafo multietapa tiene las siguientes características:

- La columna $j$ del grafo multietapa representa una copia completa del grafo $G$. A cada una de estas columnas se le denomina etapa $j$.
- El renglon $i$ del grafo multietapa representa al vértice $v_i \in G$. Es decir se tienen $O(\log N)+1$ copias del mismo vértice, cada una correspondiente a una etapa del grafo multietapa. El vértice $v_{i,j}$ hace referencia al vértice $v_i$ de la etapa $j$.
- Una arista cualquiera (del grafo original) de la forma $(v_a, v_b)$, donde $v_a, v_b \in G$, corresponde en el grafo multietapa al conjunto de aristas $(v_{a,j}, v_{b,j+1}) \cup (v_{b,j}, v_{a,j+1})$ para toda $0 \leq j \leq a \log N \leq O(\log N)$. Es decir existe una arista entre el vértice $v_a$ de la etapa $j$ y el vértice $v_b$ de la etapa $j+1$ si existe una arista $(v_a, v_b)$ en el grafo original.

En la Figura 28 se puede observar el mapa rotacional de un grafo (no es propiamente un expansor pero sirve para ilustrar el proceso) $G(8,2)$ y su representación multietapa. Note que el renglón $i$ representa al vértice $i \in G$ y cada columna $j$ es una copia completa

de $G$. Además las aristas entre dos etapas corresponden al mapa rotacional del grafo $G$.

Es importante señalar que esta transformación no modifica la cantidad de componentes conectados del grafo original. Sólo se tiene un arreglo bidimensional y expandido del mismo grafo ya que una arista entre dos vértices del grafo expandido existe si y sólo si existe la correspondiente arista en el grafo original.

Se puede ver el grafo expandido como una representación en el tiempo de todas las posibles trayectorias a partir de todos y cada uno de los vértices del grafo original. El eje $x$ corresponde al tiempo y el eje $y$ la posición del viajero en el grafo. Sin embargo, el algoritmo únicamente requiere determinar una trayectoria $T = \{d_0, \ldots, d_{O(\log N)-1}\}$ dada. Por ello es posible reducir la cantidad de aristas del grafo multietapa. Esta transformación implica que los vértices que pertenecen a la etapa $j$ eliminen todas las aristas que lo conectan a la etapa $j+1$ exceptuando la arista indicada por $d_j$. Es decir cada uno de los vértices de la etapa $j$, para toda $0 \leq j < O(\log N)$, se conecta a la etapa $j+1$ sólamente por la arista $d_j$.

El resultado de esta transformación es un grafo acíclico (un bosque) que describe la trayectoria $T$ a partir de todos los vértices del grafo $G$. Para la etapa 0 todos los vértices se conectan a la etapa uno por la arista indicada por $d_0$. Los vértices de la etapa 1 se conectan a la etapa 2 mediante la arista $d_1$ y así sucesivamente. Note que cada vértice está conectado con sólo un vértice de la etapa siguiente y puede estar conectado hasta con $D$ vértices de la etapa anterior. Esta característica es precisamente la que convierte al grafo expandido en un bosque.

En la Figura 29 se ilustra esta transformación. En el grafo de la izquierda se ve el grafo multietapa en función de una trayectoria $T$ dada. Si se recorre el grafo a partir

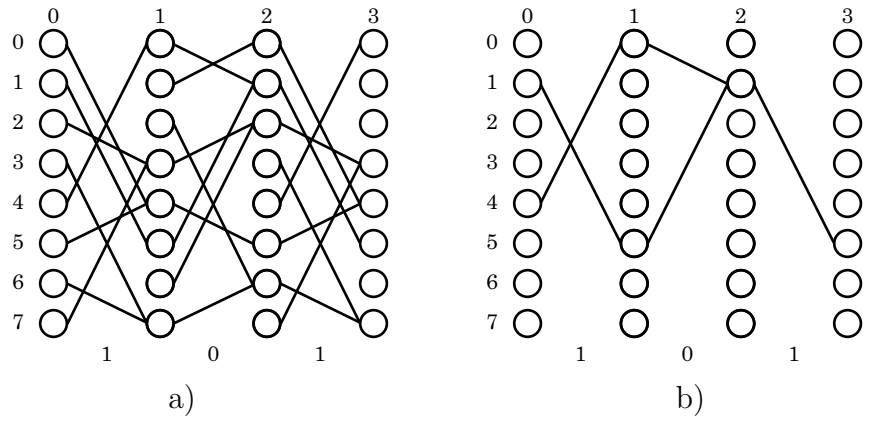



Figura 29: a)La trayectoria $T = \{1, 0, 1\}$ a partir de todos los vértices del grafo $G(8, 2)$. b) Uno de los árboles correspondiente a la misma trayectoria $T$. Note que para el vértice 1 y 4 el árbol es el mismo.

del vértice 4 siguiendo la trayectoria $\{1, 0, 1\}$ se visitan los vértices $\{4, 0, 1, 5\}$ en ese orden como se muestra en el grafo de la derecha.

Si cada procesador en el modelo LR-Mesh representa un vértice del grafo multietapa, entonces se requieren al menos $N$ renglones para representar a los $N$ vértices y $O(\log N+1)$ columnas para representar una trayectoria de tamaño $O(\log N)$. Sin tomar en cuenta, por supuesto, los procesadores que se requieren para realizar las conexiones correspondientes.

Como se ve en la Figura 29a, el grafo resultante no se puede empotrar directamente en el modelo LR-Mesh ya que los ductos no son lineales. Se puede observar que el vértice $v_{1,2}$ tiene dos antecesores que impiden transladar directamente este grafo al modelo LR-Mesh (para un expansor cualquiera, en el peor de los casos, se pueden tener hasta $D$ antecesores). Por ello todavía se requiere efectuar las siguientes transformaciones.

### VIII.1.2 Sustitución de un vértice del grafo expandido por $D+1$ vértices lógicos

Esta transformación es relativamente sencilla. Únicamente se requiere sustituir cada vértice del grafo multietapa por una colección de $D+1$ nodos. Esto con el fin de reducir el grado de cada vértice (que potencialmente puede ser $D+1$, por los $D$ posibles antecesores y la arista a la siguiente etapa) a máximo tres.

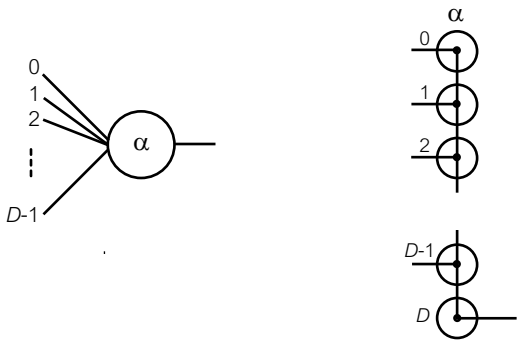



Figura 30: Transformación de un vértice cualquiera del grafo extendido a $D+1$ nodos.

Como se ve en la Figura 30, los procesadores $0, 1, \ldots, D-1$ fungen como las $D$ aristas que conectan al grafo con sus vecinos (antecesores), si es que existen. El procesador $D$ se encarga de conectar al vértice $v_a$ de la etapa $j$ con el nodo $k$ del vértice $v_b$ de la etapa $j+1$ de acuerdo al $\mathrm{Rot}_G(v_a, d_j) = (v_b, k)$. Debido a que cada arista está etiquetada con un identificador único, es posible utilizar dicho identificador para determinar a cual de los $D$ nodos se conecta un vértice cualquiera a la siguiente etapa.

Esta transformación no altera la cantidad de columnas que se requiere para representar el grafo en el modelo LR-Mesh. La cantidad de renglones, por otro lado, si se ve afectada. Se requieren $(D+1)N$ renglones para empotrar este grafo en el modelo LR-Mesh.

Las conexiones entre dos etapas consecutivas se pueden ver como una permutación. Se puede realizar la permutación de $N$ en tiempo constante gracias al Lema 7. Si en cada una de las etapas el algoritmo requiere permutar exactamente $N$ ductos, entonces se requieren $N$ columnas entre cada etapa que permitan esta conexión.

Después de esta transformación el grafo resultante se puede representar en el modelo LR-Mesh en un arreglo bidimensional de $(D+1)N$ renglones (ya que cada vértice se representa por $D+1$ nodos) y $(N+1)O(\log N)+1$ columnas (ya que cada etapa requiere de $N$ columnas para representar las aristas correspondientes), como se muestra en la Figura 31.

### VIII.1.3 Tour de Euler del expansor

El grafo que se obtiene después de estas transformaciones es un conjunto de árboles que describen los vértices que se visitan a partir de los $N$ vértices del expansor al seguir una trayectoria $T$. Además, cada vértice tiene a lo más 3 vécinos. A partir de este grafo es relativamente sencillo construír un Tour de Euler y finalmente empotrar el grafo en el modelo LR-Mesh.

Para lograrlo, se sustituye cada arista en el grafo por dos aristas. Para ello se requiere duplicar la cantidad de renglones y columnas de la malla. Las aristas que conectan a dos vértices no representan mayor problema. La transformación de cada uno de los $D+1$ nodos que conforman un vértice de tal forma que el resultado sea un tour se muestra en la Figura 32.

Al realizar esta ultima transformación el grafo resultante se puede trasladar directamente al modelo LR-Mesh ya que todos los ductos son cíclicos. Además un tour que principia en el vértice $v_i$ recorre, por lo menos, los vértices dados por la trayectoria $T$

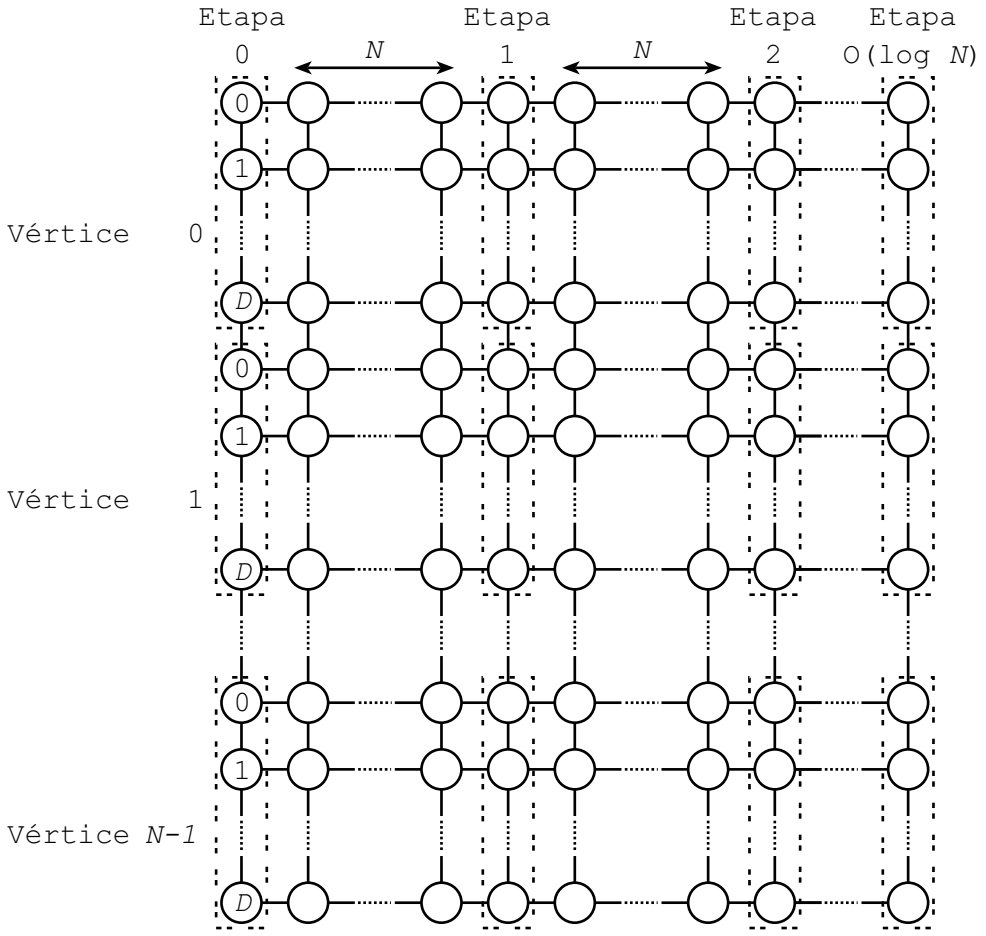



Figura 31: Expansor en el modelo LR-Mesh. Se requieren $O(\log N) + 1$ etapas. Cada una de estas etapas está separada por $N$ procesadores que permiten la conexión de la etapa $j$ a la $j + 1$. Cada una de las etapas se compone de los $N$ vértices del expansor y cada vértice se representa por $D + 1$ procesadores

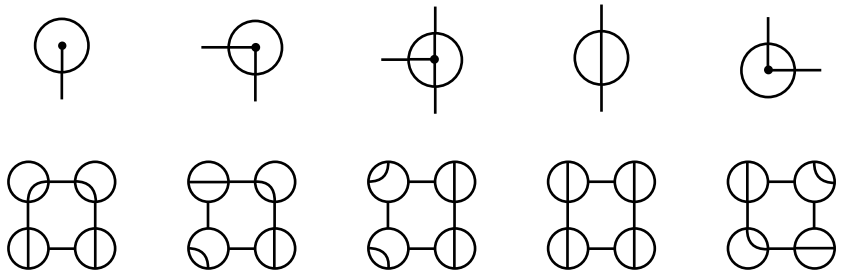


Figura 32: Representación en el modelo LR-Mesh de los vértices del Tour de Euler del grafo multietapa.

a partir de $v_i$. Por lo tanto el vértice $v_i$ puede enviar una señal predefinida por dicho ducto y todos los vértices que se visitan a partir del vértice $v_i$ siguiendo la trayectoria $T$ detectan la señal enviada en tiempo constante.

**Lema 10.** *Recorrer una trayectoria de tamaño $O(\log N)$ en un expansor se puede realizar en el modelo LR-Mesh de tamaño $2(D+1)N \times 2((N+1)O(\log N)+1)$ en tiempo constante.*

Gracias al Lema 10 se puede determinar los vértices que se recorren dada un trayectoria $T$ a partir del vértice $u$. Si al menos uno de los vértices que representan al vértice $t$ detecta la señal en el ducto, entonces se puede concluir que $s$ y $t$ están en el mismo componente conectado.

**Lema 11.** *Determinar si dos vértices $u$ y $t$ están conectados dada una trayectoria $T$ de tamaño $O(\log N)$ en un expansor se puede realizar en el modelo LR-Mesh de tamaño $2(D+1)N \times 2((N+1)O(\log N)+1)$ en tiempo constante.*

## VIII.2 USTCON en un expansor en tiempo constante

Se mostró que se puede recorrer un expansor a partir de un vértice $u$ dada una trayectoria en tiempo constante. El algoritmo para resolver USTCON en un expansor recorre los vértices en función de dichas trayectorias, desde la trayectoria $\{0_1, 0_2, 0_3, \ldots, 0_{O(\log N)}\}$ hasta la trayectoria $\{(D-1)_1, (D-1)_2, (D-1)_3, \ldots, (D-1)_{O(\log N)}\}$ y todas las intermedias. La cantidad total de trayectorias es $D^{O(\log N)}$. Expresado de otra forma, la cantidad de trayectorias es $N^{O(\log D)}$, es decir existe una cantidad polinomial de trayectorias. Por lo tanto se pueden evaluar todas las trayectorias posibles al mismo tiempo.

Los vértices $u$ y $t$ estan conectados si en al menos una de dichas trayectorias se visita el vértice $t$ a partir del vértice $u$.

**Problema USTCON en un expansor**

**Entrada**: El procesador $P_{0,0}$ tiene el identificador del vértice $u$ y $v$, la constante $D$ y el diámetro del grafo $G$ (siempre es $a \log N$), además de la trayectoria $T \in \{0, 1, \ldots, N^{O(\log D)} - 1\}$. El procesador $P_{i,j}$, para toda $0 \leq i, j < N$, posee el elemento $a_{i,j}$ de la matriz de adyacencia del expansor $G$.

**Salida**: El procesador $P_{0,0}$ tiene una variable *conectado* $= 1$ si $u$ y $v$ están en el mismo componente conectado; *conectado* $= 0$ en cualquier otro caso.

BEGIN

1. Calcular el mapa rotacional de $G$.
2. Crear $N^{O(\log D)}$ mallas cada una con un identificador distinto $T'$.
3. Transmitir el mapa rotacional a cada una de las mallas.
4. Transformar $T'$ a base D, llamado $T$, en cada malla.
5. En cada malla $T'$, determinar si $u$ y $v$ están conectados en $G$ dada la trayectoria $T$.
6. Determinar si en al menos una de las trayectorias están conectados $u$ y $t$.

END

Realizar el primer paso del algoritmo sólo implica etiquetar cada uno de las aristas del grafo de acuerdo a la definición de mapa rotacional. Se puede construir el mapa

rotacional de un grafo efectuando una suma de prefijos por renglón y otra por columna en la matriz de adyacencia del grafo. Cada arista se etiqueta de esta forma con un número consecutivo. El primer 1 (de cada renglón o columna de la matriz de adyacencia según corresponda) se convierte en la arista 0, el segundo en 1 y el último en $D-1$. Este paso se puede realizar en el modelo LR-Mesh en tiempo constante gracias al Lema 1. Se conoce a priori que la cantidad total de unos es exactamente $D$, por lo tanto se puede realizar la suma de prefijos de cada renglón y cada columna de la matriz de adyacencia en el modelo LR-Mesh de tamaño $(D+1)N \times (D+1)N$.

El paso dos requiere determinar para cada una de las $N^{O(\log D)}$ mallas el identificador único $T'$. Cada una de las mallas requiere al menos ser de tamaño $2(D+1)N \times 2((N+1)O(\log N)+1)$ (lo necesario para determinar si $u$ y $v$ están conectados dada una trayectoria $T$). Este paso sólo requiere que cada procesador divida su identificador de renglón entre $2(D+1)N$ y el cociente es en efecto el identificador $T$ correspondiente.

El paso tres es un proceso trivial. Transmitir el mapa rotacional a cada una de las mallas sólo implica transmitir los $D$ datos (el rotacional del renglón $x$ a través de las aristas $\{0,\ldots,D-1\}$) correspondientes a cada renglón a la diagonal principal. Después se envian en forma vértical los datos a cada renglón correspondiente de acuerdo a la representación del grafo multietapa. Finalmente se transmiten en forma horizonal los datos a cada una de las etapas de la malla.

La operación que más recursos necesita es la comprendida en el paso cuatro. Cada malla debe transformar a base $D$ el identificador $T'$ y asignar el resultado a $T$ (cada etapa de la malla posee uno de los elementos de $T$). Esto se puede realizar en tiempo constante gracias al Lema 1. La cantidad de recursos necesarios, si el número mayor que se convierte a base $D$ es $N^{O(\log D)}$, es a lo más $(N^{O(\log D)}+1) \times N^{O(\log D)}O(\log N)$.

El paso cinco es el proceso más complejo del algoritmo. Debido al Lema 11 se

puede determinar si $u$ y $v$ se encuentran en el mismo componente conectado dada una trayectoria $T$. Como ya se explicó primero se transforma el grafo a un grafo multietapa, se conectan cada etapa gracias a la trayectoria $T$, cada vértice del grafo expandido se sustituye por $D+1$ vértices lógicos, finalmente se crea un tour de Euler del árbol resultante y se determina si $u$ y $v$ están conectados. El primer procesador de la primer columna de cada malla almacena un 1 en la variable *conectado* si $u$ y $v$ se encuentran en el mismo componente conectado; en caso contrario, se realiza la operación $conectado = 0$. Este paso requiere menos recursos que el paso anterior por lo tanto cada una de las $N^{O(\log D)}$ mallas deben ser de tamanño $(N^{O(\log D)}+1) \times N^{O(\log D)}O(\log N)$.

El último paso consiste en realizar una operación lógica OR. Cada malla tiene una variable *conectado* con un resultado 0 ó 1. Si en al menos una de las mallas se determina que $u$ y $v$ se encuentran en el mismo componente conectado entonces existe una trayectoria que une a los vértices $u$ y $v$. Si en todas y cada una de las mallas se determina que no existe un trayectoria entre $u$ y $v$ entonces los vértices no están conectados. El resultado del algoritmo es el resultado de la operacion lógica OR de las variables *conectado* correspondientes a cada malla. Esta operación se puede realizar en el modelo LR-Mesh en tiempo constante.

Cada uno de los pasos del algoritmo se pueden realizar en el modelo LR-Mesh en tiempo constante. El paso 4 es el proceso que requiere una mayor cantidad de recursos por lo tanto se concluye que:

**Lema 12.** *Se puede resolver el problema de conectivdad en un expansor* $G(N, D, \lambda \leq \frac{1}{2})$ *en el modelo LR-Mesh de tamaño* $(N^{O(\log D)}+1)N^{O(\log D)} \times N^{O(\log D)}O(\log N)$ *en tiempo constante.*

# Capítulo IX

# Simulación del Modelo R-Mesh en el modelo LR-Mesh en tiempo constante

En este capitulo se muestra una simulación en el modelo LR-Mesh del modelo R-Mesh en tiempo constante. En el Capítulo II se mostró el ciclo de máquina del modelo de rejillas reconfigurables. Cada ciclo de máquina se compone de tres pasos: partición, comunicación y cálculo. En la etapa de partición cada procesador adquiere una de las configuraciones internas permitidas por el modelo. En la etapa de comunicación los procesadores pueden escribir en uno o varios de sus puertos y leer de los mismos. La etapa de cálculo se refiere a alguna operación aritmética o lógica.

Para simular el modelo R-Mesh en el modelo LR-Mesh se muestra cómo el modelo LR-Mesh puede realizar un paso arbitrario del ciclo de máquina del modelo R-Mesh. La etapa de cálculo es trivial simularla en el modelo LR-Mesh ya que ambos modelos pueden realizar las mismas operaciones lógicas y aritméticas. Debido a las restricciones del modelo LR-Mesh, la etapa de partición y por ende la de comunicación son las que no se pueden simular directamente.

En el resto del capítulo se muestran algoritmos para resolver unos problemas para grafos que se utilizan para realizar la simulación entre los modelos. Finalmente se muestra y se describe la simulación del modelo R-Mesh en el modelo LR-Mesh en tiempo constante.

# IX.1 Algoritmos para grafos $G(N, d^{16})$ en tiempo constante

A continuación se muestra cómo resolver el problema USTCON, cómo obtener la *clausura transitiva* y finalmente como resolver el problema de componentes conectados. Este último problema se utiliza para realizar la simulación del modelo R-Mesh en el modelo LR-Mesh. Todos los algoritmos se diseñaron para el modelo LR-Mesh y se ejecutan en tiempo constante.

## IX.1.1 USTCON en tiempo constante en el modelo LR-Mesh

En los capítulos anteriores se muestran los algoritmos para transformar un grafo cualquiera en un expansor y para resolver el problema de conectividad en un expansor. Ambos algoritmos requieren una cantidad constante de tiempo. Es posible utilizar estos algoritmos para resolver el problema USTCON.

Algortimo para resolver USTCON en el modelo LR-Mesh en tiempo constante.

**Entrada**: El mapa rotacional del grafo $G(N, d^{16})$. La columna 0 tiene esta información: Los primeros $d^{16(O(\log N)+1)}$ poseen el rotacional del vértice 0 a traves de las $d^{16}$ aristas y así sucesivamente. Cada procesador de la malla tiene el mapa rotacional del grafo $H(d^{16}, d, \lambda \leq \frac{1}{2})$. El valor de $i = O(\log N)$. Dos vértices $s, t \in G$.

**Salida**: El procesador $P_{0,0}$ tiene una variable *conectado* $= 1$ si $u$ y $v$ están en el mismo componente conectado; *conectado* $= 0$ en cualquier otro caso.

BEGIN

1. Se transforma el grafo no dirigido $G$ a un expansor $G'$. Por medio del Lema 9 se puede tranformar un grafo $G$ a un expansor $G'$ en tiempo constante.

2. Se resuelve el problema USTCON en el expansor $G'$ en tiempo constante el resultado se almacena en *conectado*, Lema 12.

END

Cada vértice del grafo $G$ se representa en el expansor $G'$ por $d^{16O(\log N)}$ vértices. Los vértices $s, t \in G$ corresponden a $d^{16O(\log N)}$ vértices en $G'$. Para resolver el problema de conectividad del grafo $G$ en el expansor $G'$ se selecciona uno de los vértices que representan al vértice $s, t \in G$ en el expansor $G'$. Debido a que los componentes conectados no se alteran en el proceso de creación del expander se puede resolver el problema USTCON en $G'$ entre los vértices $s, a_0, \ldots, a_{O(\log N)-1}$ y $t, a'_0, \ldots, a'_{O(\log N)-1}$. Los vértices $s, t \in G$ se encuentran en el mismo componente conectado si y sólo si existe una trayectoria entre los vértices $(s, a_0, \ldots, a_{O(\log N)-1}), (t, a'_0, \ldots, a'_{O(\log N)-1}) \in G'$, sin importar los valores de $a_i, a'_j \in H$.

El algoritmo para convertir un grafo $G(N, D^{16})$ a un expansor $G'(dN^{\beta+1}, d^{16}, \lambda \leq \frac{1}{2})$ requiere una malla con $2dN^{\beta+1}$ renglones y $(\frac{23}{7}N^{\alpha} - \frac{16}{7}) \times dN^{\beta+1}$ columnas, donde el diámetro del expansor es $l = O(\log N) = a \log N$, $\alpha = a \log 8$ y $\beta = 16a \log d$. Simplificando lo anterior se puede decir que la transformación requiere una malla con $O(N^{\beta+1})$ renglones y $O(N^{\alpha+\beta+1})$ columnas.

Resolver el problema USTCON en un expansor $G(N, D)$ y diámetro $l = O(\log N) = a \log N$ en el modelo LR-Mesh en tiempo constante requiere una malla con $(N^{a \log D} + 1)N^{a \log D}$ renglones y $N^{a \log D} a \log N$ columnas. Simplificando lo anterior se puede decir que USTCON se puede resolver en un expansor en el modelo LR-Mesh en una malla con $O(N^{2a \log D})$ renglones y $O(N^{a \log D} \log N)$ columnas.

Resolver el problema USTCON en el expansor $G'$ requiere entonces una malla de

tamaño

$$O((dN^{\beta+1})^{2a\log d^{16}}) \times O((dN^{\beta+1})^{2a\log d^{16}} \log dN^{\beta+1})$$

Se sabe que $2a\log d^{16} = 32a\log d = \beta$ y $O(\log dN^{\beta+1}) = O(\log N)$, por lo tanto la expresión queda como

$$O(N^{(\beta+1)\beta}) \times O(N^{(\beta+1)\beta}\log N)$$

Se puede ver que en el paso 2 se requiere una mayor cantidad de recursos.

**Lema 13.** *Se puede resolver el problema USTCON en un grafo $G(N, d^{16})$ en el modelo LR-Mesh en tiempo constante. Se requiere una malla con $O(N^{(\beta+1)\beta})$ renglones y $O(N^{(\beta+1)\beta}\log N)$ columnas, donde a es una constante y $\beta = 16a\log d$.*

### IX.1.2 Clausura Transitiva

La *"clausura transitiva"* de un grafo $G$, denotada $A^*$, es una matriz que se define como sigue (Wang y Chen, 1990): el elemento $a^*_{i,j} = 1$ si y sólo si existe una trayectoria entre el vértice $i$ y $j$ en $G$, cero en caso contrario.

Se mostró que se puede resolver el problema USTCON en el modelo LR-Mesh en tiempo constate. Determinar la clausura transitiva a partir de dicho algoritmo es sencillo. Se compara cada par de vértices $i, j \in G$ y se determinan si existe una trayectoria que los conecta. Si $i$ y $j$ se encuentran en el mismo componente conectado entonces el elemento $a^*_{i,j} = a^*_{j,i} = 1$, cero en caso contrario. Cabe señalar que $A^*$ es una matriz simétrica ya que $G$ es un grafo no dirigido.

A continuación se muestra un algoritmo para determinar la clausura transitiva de un grafo.

Entrada: La matriz de adyacencia $A$ de un grafo no dirigido $G(N, d^{16})$. Cada elemento $a_{i,j}$ se encuentra en el procesador $P_{iN,j}$.

Salida: La clausura transitiva $A^*$ del grafo $G$. Cada elemento $a^*_{i,j}$ se encuentra en el procesador $P_{iN,j}$.

BEGIN

1. Para cada par de vértices $i, j \in G$ hacer

   (a) Resolver USTCON con los vértices $i, j \in G$.

   (b) Si *conectado* $= 1$ entonces $a^*_{i,j} = a^*_{j,i} = 1$ else $a^*_{i,j} = a^*_{j,i} = 0$

END

Se requiere determinar todas las posibles combinaciones de pares de vértices en paralelo. La cantidad de pares de vértices es $O(N^2)$ por lo tanto se requieren $O(N^2)$ mallas y en cada una se resuelve USTCON en tiempo constante, Lema 13.

**Lema 14.** *Se puede obtener la clausura transitiva de un grafo $G$ en el modelo LR-Mesh en tiempo constante. El algoritmo requiere una malla con $O(N^{(\beta+1)\beta+2})$ renglones y $O(N^{(\beta+1)\beta} \log N)$ columnas.*

### IX.1.3 Componentes conectados

El identificador de cada componente conectado en $G$ es el índice menor de todos los vértices que le pertenecen. El problema de componentes conectados consiste en etiquetar cada vértice con su correspondiente identificador de componente conectado.

Para resolver el problema de componentes conectados en el modelo LR-Mesh se obtiene la clausura transitiva del grafo $G$. El renglón $i$ de la clausura transitiva muestra

todos los vértices que están en el mismo componente conectado de $i$. Se puede determinar cuál es el índice (etiqueta del vértice) menor en tiempo constante. Los procesadores cuyo elemento $a^*_{i,j}$ sea igual a cero fusionan sus puertos este y oeste, en caso contrario no fusionan ninguno de sus puertos. De esta forma cada ducto comunica a dos procesadores consecutivos cuyo elemento $a^*_{i,j}$ es igual a uno. Cada uno de estos procesadores transmite una señal predefinida por el puerto este. Con esto el procesador $j$ informa al procesador con índice $j' > j$ que existe otro procesador $j$ con un índice menor. El único procesador que no recibe esta señal es el procesador cuyo índice es el menor de todos.

A continuación se muestra el algoritmo para resolver el problema de componentes conectados.

Entrada: Un grafo no dirigido $G(N, D^{16})$. El procesador $P_{iN,j}$ posee el elemento $a_{i,j}$ de la matriz de adyacencia del grafo $G$.

Salida: El procesador $P_{iN,i}$ posee el identificador del componente conectado $ID$ para el vértice $i \in G$.

BEGIN

1. Calcular la clausura transitiva $A^*$ del grafo $G$.

2. Determinar en cada renglón el índice menor.

3. Transmitir el índice al procesador $P_{iN,i}$

END

Cada paso del algoritmo se puede realizar en el modelo LR-Mesh en tiempo constante. El paso uno es el que requiere una mayor cantidad de recursos, Lema 14.

**Lema 15.** *El problema de componentes conectados en un grafo* $G(N, d^{16})$ *se puede resolver en el modelo LR-Mesh en tiempo constante. El algoritmo requiere una malla con* $O(N^{(\beta+1)\beta+2})$ *renglones y* $O(N^{(\beta+1)\beta} \log N)$ *columnas.*

## IX.2 Simulación en tiempo constante del modelo R-Mesh en el modelo LR-Mesh

El objetivo de la simulación es realizar un paso arbitrario $i$ del ciclo de máquina del modelo R-Mesh en el modelo LR-Mesh. Cada procesador $P_{i,j}$ en el modelo LR-Mesh debe de tener por lo tanto la configuración interna que adquiere el procesador $P_{i,j}$ en el modelo R-Mesh en el paso $i$, los datos $d_t$ que el procesador $P_{i,j}$ va a transmitir a través del puerto $t$ y el conjunto de puertos $s \subseteq N, S, E, O$ (resulta conveniente representar a los puertos numéricamente $N = 0$, $S = 1$, $E = 2$ y $O = 3$) de los cuales se requiere leer y la operción $f(a)$ que el modelo R-Mesh realiza en el paso $i$, donde $a$ representa las variables locales envueltas en la operación.

A continuación se muestra cómo se puede realizar cada paso del ciclo de máquina del modelo R-Mesh en el modelo LR-Mesh en tiempo constante y se discuten la cantidad de recursos necesarios para el modelo LR-Mesh.

### IX.2.1 Partición

El modelo R-Mesh en esta etapa puede crear ductos no lineales los cuales el modelo LR-Mesh no puede representar. El objetivo de esta fase es distribuir los procesadores que están conectados a un mismo ducto de tal forma que todos se situen en una misma columna del modelo LR-Mesh. Es obvio que no se pueden *mover* los procesadores pero

sí la información contenida en ellos. De esta forma se puede crear en el modelo LR-Mesh un ducto lineal que comunica a todos los procesadores de un mismo ducto del modelo R-Mesh.

**Definición 8.** El grafo $G$ de una configuración particular del modelo R-Mesh es una representación gráfica de las conexiones entre sus puertos. Cada vértice del grafo representa a un puerto $t$ del procesador $P_{i,j}$. Una arista del tipo $(t_1, t_2)$ existe en $G$ si y s'olo si existe un ducto que comunica al puerto $t_1$ con el puerto $t_2$ en el modelo R-Mesh.

El grafo $G$ es similar al definido en (Fernández-Zepeda *et al.*, 2002), la única diferencia es que en la definición anterior cada puerto del modelo R-Mesh se representa por un vértice en $G$ sin importar que un puerto este fusionado internamente con otro puerto. Estas conexiones internas se representan por las aristas del grafo $G$. Se puede ver un ejemplo de un grafo $G$ en la Figura 33.

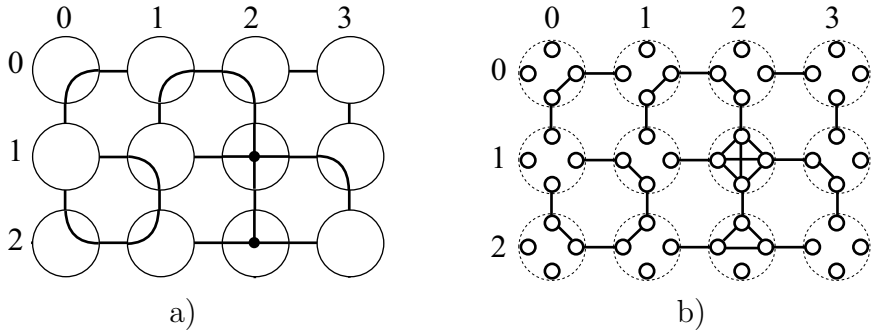



Figura 33: a) Configuración interna de un modelo R-Mesh. b) Grafo $G$ de dicha configuración.

Cada procesador del modelo LR-Mesh conoce su identificador $(i, j)$ y la configuración interna del modelo R-Mesh. Se puede obtener el mapa rotacional del grafo $G$ en tiempo constante. Cada procesador tiene una cantidad constante (4) de puertos y se puede determinar para cada vértice de $G$ las aristas correspondientes. Además se puede

observar que cada vértice del grafo $G$ tiene a lo más 4 aristas. Se puede transformar este grafo $G(N^2)$ a un grafo con $N^2$ vértices y $d^{16}$ regular agregando tantos autociclos a cada vértice como sea necesario.

Se obtiene el grafo $G$ y se resuelve el problema de componentes conectados en tiempo constante gracias al Lema 15. Cada vértice de $G$ tiene un identificador $ID$ del componente conectado al que pertenecen. Se mueve cada uno de los cuatro vértices que representan al procesador $P_{i,j}$ al renglón $i$ y columna $ID$. Esto se puede hacer en tiempo constante enviado la información a la diagonal principal y posteriormente a la columna correspondiente. Finalmente todos los procesadores fusionan sus puertos $N$ y $S$ y se crean ductos verticales que recorren toda la rejilla. Cada puerto perteneciente al componente conectado $ID$ se encuentran en un renglón diferente pero todos en la misma columna $ID$.

### IX.2.2 Comunicación

Una vez reordenados los vértices del grafo $G$ es sencillo el proceso de comunicación. Cada vértice del grafo $G$ transmite, si es que se requiere, el dato $d_t$ por el ducto vertical. Si existen escrituras concurrentes el ducto resuelve el problema de acuerdo a la regla de escritura utilizada. Todos los vértices de $G$, que requieren leer, leen del ducto y almacenan el resultado en una variable local. Finalmente se transmite esta información (cabe señalar que esta información es una cantidad constante de datos) al procesador $P_{i,j}$ original.

### IX.2.3 Cálculo

Cómo se explicó éste es un paso trivial. Cada procesador tiene toda la información localmente por lo tanto éste puede realizar la operación lógica o aritmética en tiempo

constante.

Se tienen $N^2$ procesadores para una R-Mesh de $N \times N$ por lo tanto la cantidad de posibles componentes conectados es de $O(N^2)$. Se requiere una LR-Mesh de tamaño $O(N^2) \times O(N^2)$ para poder situar cada componente conectado en una columna diferente y transmitir la información pertinente.

El proceso de resolver el problema de componentes conectados es el que requiere una mayor cantidad de procesadores. Para resolver el problema de componentes conectados en un grafo $G(N, d^{16})$ se requiere una rejilla con $O(N^{(\beta+1)\beta+2})$ renglones y $O(N^{(\beta+1)\beta} \log N)$ columnas. Para resolver el mismo problema en un grafo $G(N^2, d^{16})$ se requiere una malla con $O(N^{2((\beta+1)\beta+2)})$ renglones y $O(N^{2(\beta+1)\beta} \log N)$ columnas.

**Teorema 1.** *El modelo R-Mesh de tamaño $N \times N$ se puede simular en el modelo LR-Mesh de tamaño $O(N^{2((\beta+1)\beta+2)}) \times O(N^{2(\beta+1)\beta} \log N)$ en tiempo constante, donde $\beta$ es una constante.*

# Capítulo X

# Conclusiones y Trabajo Futuro

## X.1 Sumario

Se mostró que es posible transformar un grafo regular a un expansor en el modelo LR-Mesh en tiempo constante, ver Capítulo VII. El algoritmo es una versión paralela del algoritmo propuesto por Reingold (Reingold, 2005) para resolver USTCON con espacio de memoria logarítmico. Se construye una red multietapa que realiza las operaciones necesarias en forma concurrente y se transmite cierta información a atrevés de la red y se obtiene el mapa rotacional del expansor.

En el Capítulo VIII se muestra un algoritmo que permite resolver el problema USTCON en un expansor en el modelo LR-Mesh en tiempo constante. En forma general el algoritmo recorre en tiempo constante el expansor a través de un vértice $s$ y se verifica si se encuentra el vértice $t$. Se recorren en paralelo todas las posibles trayectorias de tamaño logarítmico. Los vértices $u$ y $v$ están conectados si en al menos una de ellas se encuentra el vértice $t$ a partir de $s$.

En el Capítulo IX se muestra que es posible resolver el problema USTCON para un grafo regular no dirigido. Después se muestra un algoritmo para resolver el problema de componentes conectados.

La mayor aportación del presente trabajo es la simulación del modelo R-Mesh en el modelo LR-Mesh en tiempo constante. Se muestra en el Capítulo IX que se puede obtener dicha simulación resolviendo el problema de componentes conectados. Esta es la simulación más rápida conocida. Con esto se cumple el objetivo general propuesto al inicio de la investigación.

Para cumplir los objetivos específicos se estudió el diseño de arquitecturas reconfigurables, el resultado se puede ver en el Capítulo II. En el capítulo I se pueden ver las simulaciones existentes del modelo R-Mesh sobre el modelo LR-Mesh. Se estudio la teoría de expansores y el producto zig-zag entre grafos, ver Capítulo IV. En el Capítulo V se describe el algoritmo que resuelve el problema USTCON con espacio logarítmico.

## X.2 Conclusiones

Se creía que el modelo LR-Mesh era menos poderoso que el modelo R-Mesh (Vaidyanathan y Trahan, 2004). En el presente trabajo se muestra de forma explícita una simulación entre los modelos en tiempo constante, confirmando con esto que los modelos son igual de poderosos, tal como se infería del trabajo de Reingold.

La simulación presentada requiere una gran cantidad de recursos. Si el modelo R-Mesh de tamaño $N \times N$ puede resolver un problema $\pi$ en $t$ pasos, el modelo LR-Mesh puede resolver el mismo problema $\pi$ en $t$ pasos pero requiere una rejilla de tamaño $O(N^{2((\beta+1)\beta+2)}) \times O(N^{2(\beta+1)\beta} \log N)$, donde $\beta > 0$ es una constante.

## X.3 Trabajo a futuro

La simulación mostrada requiere una gran cantidad de recursos, lo cual es una gran desventaja. El primer problema por resolver es la optimización de la simulación. Se puede optimizar cada paso de la simulación y reducir la cantidad de recursos requeridos.

Se puede flexibilizar el tiempo de la simulación para ahorar procesadores. En el Capítulo V se muestran algoritmos que requieren espacio sublineal para resolver el problema USTCON. Cada uno de dichos algoritmos realiza una cantidad distinta de

operaciones y requiere una cantidad distinta de espacio. Se puede paralelizar cada uno de ellos y obtener varias simulaciones. Se puede explorar la posibilidad de obtener una simulación no en tiempo constante pero si mejor que $O(\log N)$ requiriendo una menor cantidad de recursos que la simulación mostrada en el presente trabajo. Es interesante determinar la cantidad de recursos que requieren dichas simulaciones y las ventajas y desventajas que presentan.

# Bibliografía

Ajtai, M., Komlós, J., y Szemerédi, E. 1983. "Sorting in c log n parallel sets.". Combinatorica. 3(1):1-19 p.

Aleliunas, R., Karp, R. M., Lipton, R. J., Lovász, L., y Rackoff, C. 1979. "Random walks, universal traversal sequences, and the complexity of maze problems". En: "Annual IEEE Symposium on Foundations of Computer Science". 218-223 p.

Alon, N. y Milman, V. D. 1985. "$\lambda_1$ isoperimetric inequalities for graphs, and superconcentrators". Journal of Combinatorial Theory. Series B. 32(1):271-284 p.

Alvarez, C. y Greenlaw, R. 1996. "A compendium of problems complete for symmetric logarithmic space". Technical Report TR96-039, Electronic Colloquium on Computational Complexity.

Armoni, R., Ta-Shma, A., Wigderson, A., y Zhou, S. 1997. "SL <= $L^{4/3}$.". En: "STOC". 230-239 p.

Ben-Asher, Y., Peleg, D., Ramaswami, R., y Schuster, A. 1991. "The power of reconfiguration.". J. Parallel Distrib. Comput. 13(2):139-153 p.

Blum, M., Karp, R. M., Vornberger, O., Papadimitriou, C. H., y Yannakakis, M. 1981. "The complexity of testing whether a graph is a superconcentrator.". Inf. Process. Lett. 13(4/5):164-167 p.

Davidoff, G., Sarnak, P., y Valette, A. 2003. "Elementary number theory, group theory and ramanujan graphs". London Mathematical Society Student Texts. Cambridge University Press. Primera edición. Cambridge, United Kingdom.

Delgado, D. 2003. "Análisis probabílistico de algoritmos para contracción de grafos". Tesis de Maestría, Centro e Investigación Científica y de Educación Superior de Ensenada.

Fernández-Zepeda, J. A., Vaidyanathan, R., y Trahan, J. L. 2002. "Using bus linearization to scale the reconfigurable mesh". Journal of Parallel and Distributed Computing. 62(4):495-515 p.

Gabber, O. y Galil, Z. 1981. "Explicit constructions of linear-sized superconcentrators". Journal Comp. Syst. Sci. 22:407-420 p.

Impagliazzo, R., Nisan, N., y Wigderson, A. 1994. "Pseudorandomness for network algorithms.". En: "STOC". 356-364 p.

Impagliazzo, R. y Wigderson, A. 1997. "$P = BPP$ if $E$ requires exponential circuits: Derandomizing the xor lemma.". En: "STOC". 220-229 p.

JaJa, J. 1992. "Introduction to parallel algorithms". Addison Wesley.

Jimbo, S. y Maruoka, A. 1987. "Expanders obtained from affine transformations". Combinatorica. 7(4):343-355 p.

Karp, R., Pippenger, N., y Sipser, M. 1985. "A time-randomness tradeoff". En: "AMS conference on Probabilistic Computation Complexity".

Lewis, H. R. y Papadimitriou, C. H. 1982. "Symmetric space-bounded computation.". Theor. Comput. Sci. 19:161-187 p.

Margulis, G. A. 1973. "Explicit construction of expanders". Problemy Peredači Informacii. 9(4):71-80 p.

Morgenstern, M. 1994. "Existence and explicit constructions of $q+1$ regular ramanujan graphs for every prime power $q$". Journal of Combinatorial Theory. Series B. 62(1):619-623 p.

Naor, J. y Naor, M. 1993. "Small-bias probability spaces: Efficient constructions and applications.". SIAM J. Comput. 22(4):838-856 p.

Nisan, N., Szemerédi, E., y Wigderson, A. 1992. "Undirected connectivity in $O(\log^{1.5} n)$ Space". En: "Annual IEEE Symposium on Foundations of Computer Science". 24-29 p.

Pinsker, M. S. 1973. “On the complexity of a concentrator”. En: “7th Annual Teletraffic Conference”. 318/1-318/4 p.

Pippenger, N. 1987. “Sorting and selecting in rounds.”. SIAM J. Comput. 16(6):1032-1038 p.

Reingold, O. 2005. “Undirected st-connectivity in log-space.”. En: Gabow, H. N. y Fagin, R. (eds.), “STOC”. ACM, 376-385 p.

Reingold, O., Vadhan, S. P., y Wigderson, A. 2001. “Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors”. Electronic Colloquium on Computational Complexity (ECCC). 8(18).

Savitch, W. J. 1970. “Relationships between nondeterministic and deterministic tape complexities.”. J. Comput. Syst. Sci. 4(2):177-192 p.

Tanner, M. R. 1984. “explicit concentrators from generalized n-gons”. SIAM Journal on Algebraic Discrete Methods. 5(3):287-293 p.

Tanner, R. M. 1981. “A recursive approach to low complexity codes.”. IEEE Transactions on Information Theory. 27(5):533-547 p.

Trahan, J. L., Vaidyanathan, R., y Thiruchelvan, R. K. 1996. “On the power of segmenting and fusing buses.”. J. Parallel Distrib. Comput. 34(1):82-94 p.

Vaidyanathan, R. y Trahan, J. 2004. “Dynamic reconfiguration: Architetures and algorithms”. Series in Computer Science. Kluwer Academic/Plenum Publishers. Primera edición. New York.

Wang, B.-F. y Chen, G.-H. 1990. “Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems.”. IEEE Trans. Parallel Distrib. Syst. 1(4):500-507 p.