UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"ALGORITMO PRIMAL PARA RESOLVER EL PROBLEMA DE FLUJO A COSTO MÍNIMO"

T E S I S

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA:
ANTONIO MAYORQUÍN GUTIÉRREZ

DIRECTOR DE TESIS:
M.C. MARCELA RIVERA MARTÍNEZ

HUAJUAPAN DE LEÓN, OAXACA; NOVIEMBRE DE 2004

Mi tesis la dedico con todo mi amor y cariño

A mi madre:
Por su gran amor y comprensión,
por haber estado conmigo en todo momento,
por darme una carrera para mi futuro y por confiar en mi.

A mi hermana Desiree:
Gracias por estar conmigo y apoyarme.

A mis tíos:
Tino, Deme y Polo, gracias por la confianza y
apoyo brindados durante todo este tiempo.

A mis abuelos: Ezequiel y Braulia (Q.E.P.D.).
Por todas sus enseñanzas.

A todos mis amigos
Gracias por estar conmigo durante esta etapa de mi vida
y gracias por ser mi amigos.

Agradecimientos

A mi asesora, la profesora M.C. Marcela Rivera Martínez por su paciencia y apoyo, por todo el tiempo dedicado en la realización de esta tesis y por sus valiosos consejos.

A los profesores Laura C. Torres Araujo, Raúl Cruz Barbosa y Luís René Marcial Castillo por su apoyo en la revisión de esta tesis y por el tiempo invertido en ella.

A todos los profesores que influyeron en mi formación profesional.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

La optimización es una potente técnica de modelado usada en el proceso de toma de decisiones. Cuando se trata de resolver un problema de este tipo, la primera etapa consiste en identificar las posibles decisiones que pueden tomarse; esto lleva a identificar las variables del problema concreto. Normalmente, las variables son de carácter cuantitativo y se buscan los valores que optimizan el objetivo. La segunda etapa supone determinar qué decisiones resultan admisibles; esto conduce a un conjunto de restricciones que se determinan teniendo presente la naturaleza del problema en cuestión. En la tercera etapa, se calcula el coste/beneficio asociado a cada decisión admisible; esto supone determinar una función objetivo que asigna, a cada conjunto posible de valores para las variables que determinan una decisión, un valor de coste/beneficio. El conjunto de todos estos elementos define el problema de optimización[CA02].

Problemas en todas las áreas de matemáticas, ciencias aplicadas, ingeniería, economía, medicina, estadística, etc. pueden ser modelados en términos de optimización.

Una división de la optimización es la que se presenta a continuación: optimización continua y optimización discreta[NE96]. De acuerdo a dicha clasificación, la optimización en redes se encuentra dentro de la optimización continua con restricciones (Figura 1.1).

La programación en redes estudia problemas de optimización que se pueden estructurar en forma de una red, es decir, de un grafo.

Las redes tienen una aplicación extensa en diversos campos como son: planeación, administración, ingeniería, química, educación, etc; en estos campos, innumerables situaciones pueden formularse como modelos matemáticos de redes. Algunos ejemplos son: sistemas de producción-

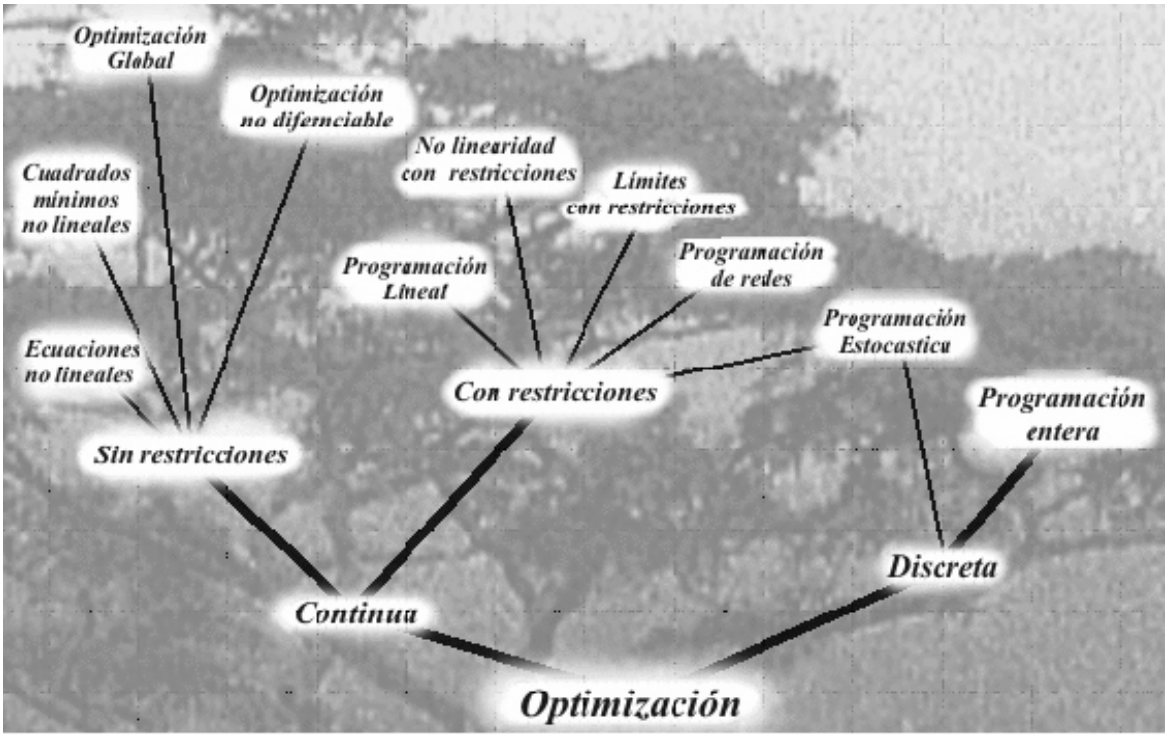


Figura 1.1: Árbol de optimización

distribución, tráfico urbano, transporte colectivo, comunicación, redes eléctricas, reemplazo de equipo, inventarios, presas, flujo de dinero, tuberías, oleoductos, asignación de recursos, etc.

En la teoría de redes existe un conjunto de problemas básicos como: ruta más corta, flujo máximo, flujo a costo mínimo, entre otros. Los variados algoritmos de solución desarrollados para ellos se han desprendido de dos ramas desarrolladas paralelamente: la programación matemática y la teoría de redes . En los últimos años, se han hecho intentos por unificar dichas ramas con la teoría de coloraciones en gráficas, enriqueciendo de este modo la teoría de redes.

Por lo anterior, el objetivo principal de este trabajo, es desarrollar un software freeware[1] que resuelva el problema de flujo a costo mínimo utilizando el algoritmo Primal, como base; dado que dentro de el Primal se involucran los algoritmos de Ford y Fulkerson y Floyd-Warshall, estos se estudiarán detalladamente para realizarles algunas modificaciones que permitan lograr el fin deseado.

[1] http://www.gnu.org/philosophy/categories.es.html

Como un objetivo particular, se mostrará en forma gráfica la solución de los problemas con el fin de facilitar el entendimiento de los algoritmos.

Cabe mencionar que existe software comercial que resuelve este tipo de problemas como: CPLEX[2], X-PRESS MP[3], SAS System[4], NETFLOW[5], TORA entre otros; sin embargo, este software tiene un costo muy elevado, por lo cual resulta muy díficil adquirirlo, es por eso, que se desarrollará un software freeware, que podrá ser utilizado por cualquier persona con previo conocimiento sobre teoría de redes.

El problema de flujo a costo mínimo se plantea de la siguiente manera: Sea $R = [X, A, q, c]$ una red con una función q de capacidad y una función c de costos por unidad de flujo asociadas a sus arcos. Se define como el costo de un flujo factible f a la cantidad $\sum_{(i,j)\epsilon A} c_{ij} f_{ij}$, donde c_{ij} es el costo unitario del flujo a través del arco (i, j) y f_{ij} es la cantidad de flujo a través de (i, j). Supóngase que se tiene interés en determinar un flujo factible de valor v, entre el origen y el destino, incurriendo en el menor costo posible; al flujo de menor costo se le llama flujo a costo mínimo de valor v.

Claramente, si la cantidad v de flujo requerido es mayor que el valor del flujo máximo el problema no tiene solución [AY97].

Existen varias maneras para resolver el problema de flujo a costo mínimo en una red $R = [X, A, q, c]$. Una de ellas, involucra la determinación de circuitos negativos en la red marginal, este procedimiento será llamado algoritmo basado en la eliminación de circuitos negativos y algunas veces se hace referencia a él como algoritmo Primal, puesto que empieza a aplicarse a partir de un flujo factible de valor v requerido y en cada iteración el costo de este flujo se mejora, sin modificar su valor, hasta alcanzar la optimalidad.

El **algoritmo Primal** propuesto en este trabajo, primeramente determina un flujo factible del valor v requerido, para ello se utiliza el algoritmo de Ford y Fulkerson con una modificación en el criterio de paro; una vez determinado el flujo de valor v se verifica si se cumple con el criterio de optimalidad, esto es, verificar si en la red marginal existen circuitos negativos o no; si no existen, el flujo es de costo mínimo; si existe alguno, éste será "eliminado". Este

[2] http://www.cplex.com
[3] http://www.dashoptimization.com
[4] http://www.sas.com
[5] http://www.cisco.com/

procedimiento se aplicará hasta que se hayan eliminado todos los circuitos negativos. En este caso el flujo de valor v actual es de costo mínimo. Nótese que, durante este procedimiento es necesario determinar circuitos negativos, para determinarlos se hace uso del algoritmo de Floyd-Warshall.

Este trabajo se divide en 5 capítulos, en el primer capítulo se define el problema a resolver.

Los conceptos correspondientes a la teoría de redes, que es una de las áreas que se involucran fuertemente en este trabajo, se presentan en el capítulo 2.

El capítulo 3, presenta los fundamentos matemáticos que sustentan el problema de flujo a costo mínimo y los problemas que se abordan en este trabajo.

En el capítulo 4 se presentan y explican los diversos algoritmos que se instrumentan en este tema de tesis.

Las pruebas realizadas y los resultados obtenidos por el software desarrollado, se presentan en el capítulo 5.

Finalmente, se muestran las conclusiones a las que se llegaron al terminar esta tesis.

En el apéndice A se presenta el manual de usuario correspondiente al software desarrollado, en el cual se explica su funcionamiento y utilización.

Ejemplos en los que se explican detallamente los algoritmos utilizados son expuestos en el apéndice B.

Ejemplos prácticos que se pueden resolver con los algoritmos instrumentados, se muestran en el apéndice C.

Por último, se muestra la bibliografía utilizada en el desarrollo de este trabajo de tesis.

Capítulo 2

Teoría de redes

En la sección 2.1 se presentan los conceptos básicos de teoría de grafos que son utilizados durante la exposición de los problemas de redes tratados en esta tesis [AY97]. En la sección 2.2 se ilustran las representaciones de los grafos algebraicamente.

2.1 Conceptos básicos

Definición 2.1 . *Un grafo es una pareja de conjuntos* $[X, A]$*, donde* X *es un conjunto de puntos llamados vértices o nodos y* A *es un conjunto de líneas que unen todos o algunos de los vértices llamados arcos o aristas. Se denota con* $G = [X, A]$*. Ejemplo:*

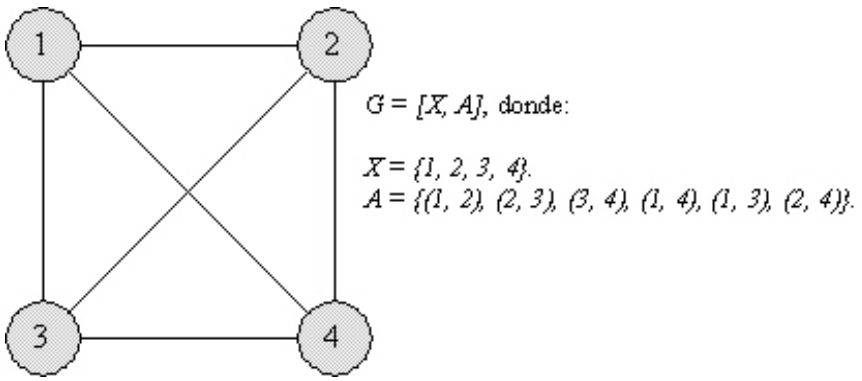


Figura 2.1: Ejemplo de un grafo

Definición 2.2 . *Un grafo dirigido o digrafo es una pareja de conjuntos* $[X, A]$*, donde* X *es un conjunto de puntos llamados vértices o nodos y* A *es un subconjunto de pares ordenados de vértices. Ejemplo:*

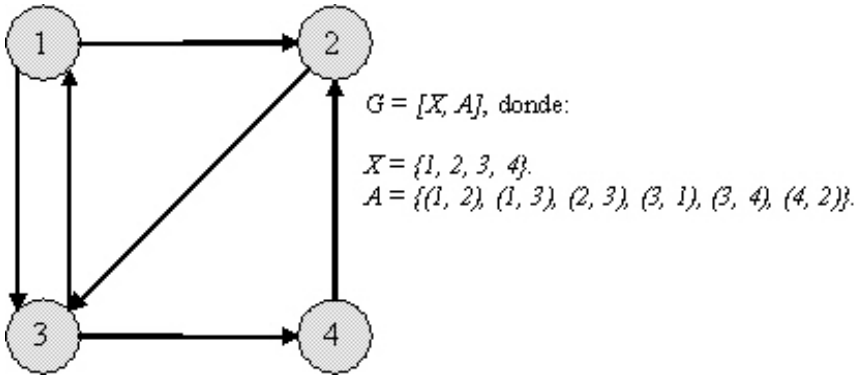


Figura 2.2: Ejemplo de un grafo dirigido

Un arco puede representarse como la pareja (i, j), donde $i, j \in X$ son los nodos que unen dicho arco. Si $a = (i, j) \in A$, i es el nodo inicial de a y j es el nodo final de a.

Para las definiciones 2.3, 2.4 y 2.5 sea $G = [X, A]$ un grafo dirigido y sea $i \in X$.

Definición 2.3 . *Se llama sucesor de* i *a todo nodo* $j \in X$ *tal que existe* (i, j)*.*

Definición 2.4 . *Se llama predecesor de* i *a todo nodo* $j \in X$ *tal que existe* (j, i)*.*

Definición 2.5 . $\Gamma^+, \Gamma^- : X \longrightarrow \wp(X)$ *(conjunto de potencia de X), donde, para* $i \in X$*:*

$\Gamma^+(i) =$ *{sucesores de i}* $= \{j \in X \mid (i, j) \in A\}$ *y*

$\Gamma^-(i) =$ *{predecesores de i}* $= \{j \in X \mid (j, i) \in A\}$*.*

Como ejemplo considérese el grafo dirigido de la Figura 2.3.

Puede verificarse fácilmente por ejemplo, que: $\Gamma^+(1) = \{2, 5\}$, $\Gamma^+(2) = \{1, 3\}$, $\Gamma^+(4) = \phi$, $\Gamma^-(1) = \{2, 3\}$, $\Gamma^-(5) = \{1\}$.

Nuevamente para las definiciones 2.6, 2.7 y 2.8 considérese un grafo $G = [X, A]$ dirigido.

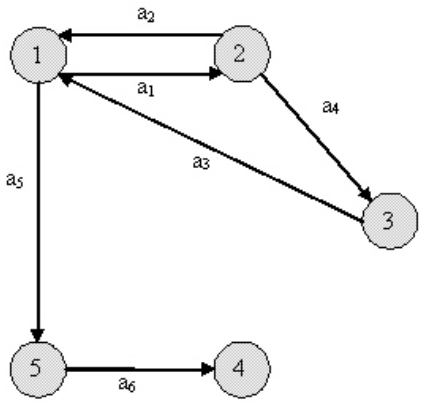


Figura 2.3: Grafo de ejemplo para la definición 2.5

Definición 2.6 . *Un camino es una secuencia de arcos en el cual el nodo final de uno es el nodo inicial del que sigue en la secuencia. Un camino puede representarse con la secuencia de los arcos que lo forman o por la secuencia de nodos extremos de estos arcos (cuando no haya confusión) o por la secuencia alternada de nodos y arcos. Si* $a_1 = (i_1, i_2)$ *es el primer arco del camino y* $a_q = (i_q, i_{q+1})$ *es el último se dice que el camino, es un camino de* i_1 *a* i_{q+1}.

Definición 2.7 . *Un camino simple es un camino* $i_1, a_1, i_2, a_2, ..., a_{q-1}, i_q$ *tal que* $a_i \neq a_j$ *si* $i \neq j$.

Definición 2.8 . *Un camino elemental es un camino* $i_1, a_1, i_2, a_2, ..., a_{q-1}, i_q$ *tal que* $i_j \neq i_k$ *si* $j \neq k$.

Dicho en otras palabras, un camino simple es uno en el cual no se repiten arcos y uno elemental es uno en el que no se repiten nodos. Obsérvese que todo camino elemental es también simple. Como ejemplo considérese el grafo de la Figura 2.4.

La secuencia a_5, a_9, a_8, a_5 es un camino de 5 a 4. También puede representarse: 5, a_5, 4, a_9, 3, a_8, 5, a_5, 4 ó 5, 4, 3, 5, 4. Este camino no es ni simple ni elemental, El camino a_6, a_5, a_9, a_8, a_4 es camino simple y no elemental. El camino a_6, a_5, a_9 es camino simple y elemental.

Definición 2.9 . *Un circuito es un camino* $i_1, a_1, i_2, a_2, ..., a_{q-1}, i_q$ *donde el nodo final de* a_{q-1} *y el inicial de* a_i *coinciden, es decir, un circuito es un camino cerrado.*

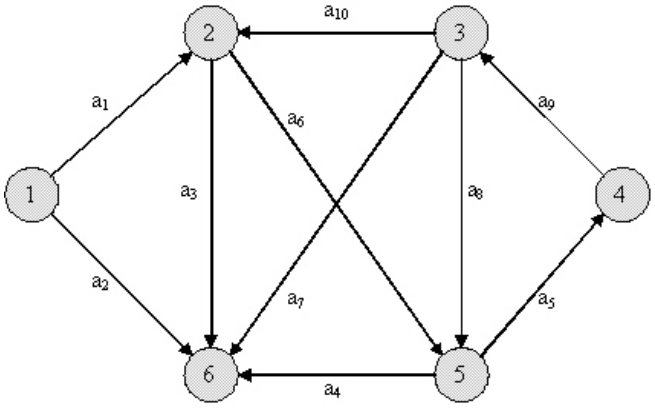


Figura 2.4: Grafo de ejemplo para las definiciones 2.6, 2.7, 2.8, 2.9, 2.10, 2.11 y 2.12.

En el grafo de la Figura 2.4 a_5, a_9, a_8 es un circuito.

Definición 2.10 . *Una cadena es una secuencia de aristas (o arcos)* $a_1, a_2, ..., a_q$ *donde toda* a_i *está conectada con* a_{i-1} *por un extremo y con* a_{i+1} *por el otro.*

Una cadena puede denotarse con la secuencia de nodos y arcos como en el caso de un camino aunque conviene señalar que la sola secuencia de nodos puede resultar insuficiente para describirla puesto que entre dos nodos puede existir más de un arco.

En el grafo de la Figura 2.4 a_{10}, a_9, a_5 (ó $2, 3, 4, 5$) es una cadena que une los nodos 2 y 5.

Definición 2.11 . *Sea* $G = [X, A]$ *un grafo dirigido, y sea* $i \epsilon X$. *El grado exterior de* i *es el número de arcos que tienen a* i *como vértice inicial, se denota* $g^+(i)$. *El grado interior de* i *es el número de arcos que tienen a* i *como extremo final, se denota* $g^-(i)$.

Obsérvese que $g^+(i)$ es igual a la cardinalidad de $\Gamma^+(i)$ y $g^-(i)$ es igual a la cardinalidad de $\Gamma^-(i)$. Además, es fácil verificar que, si m es el número de arcos y n es es número de nodos de G, entonces:

$$\sum_{i=1}^{n} g^+(i) = \sum_{i=1}^{n} g^-(i) = m.$$

Definición 2.12 . *El grado de* $i \epsilon X$, *es el número de arcos que tienen a* i *como uno de sus extremos. Se denota* $g(i)$.

En la Figura 2.4 se tiene que $g^+(6) = 0, g^-(6) = 4, g(6) = 4$.

2.2 Representación algebraica de grafos

Los grafos pueden ser representados algebraicamente por medio de matrices. Sea $G = [X, A]$ un grafo dirigido, la **matriz de adyacencia** de G es una matriz B cuadrada de n x n, donde n es el número de nodos de G, y se forma mediante:

$$B_{ij} = \begin{cases} 1, & \text{si } (i,j) \in A \\ 0, & \text{si } (i,j) \notin A. \end{cases}$$

La matriz de incidencia de G es una matriz M de dimensión n x m, donde n y m son el número de nodos y arcos de G, respectivamente, cuyos elementos son:

$$M_{ij} = \begin{cases} 1, & \text{si } i \text{ es extremo inicial del arco } a_j \\ -1, & \text{si } i \text{ es extremo final del arco } a_j \\ 0, & \text{en otro caso.} \end{cases}$$

Si se considera el grafo de la Figura 2.5, sus matrices de adyacencia e incidencia son B y M respectivamente.

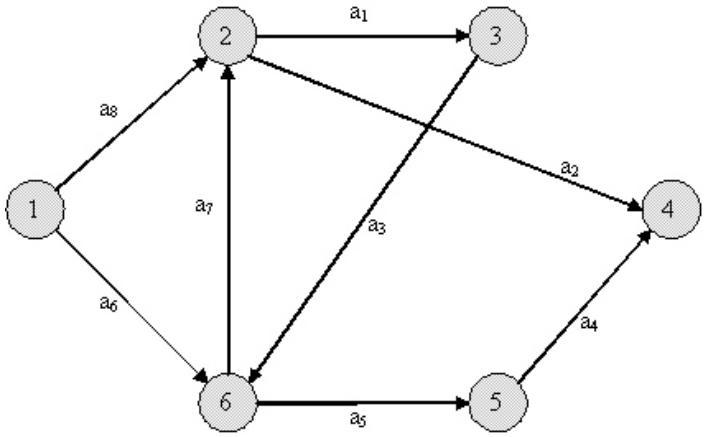


Figura 2.5: Grafo de ejemplo para las matrices de adyacencia e incidencia

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

y

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 & 1 & 0 \end{bmatrix}.$$

Capítulo 3

Fundamentos matemáticos

En este capítulo, se presentan conceptos necesarios para la solución del problema de Flujo a Costo Mínimo. En la sección 3.1 se presentan los conceptos básicos para resolver el problema de Flujo Máximo[AY97]. En la sección 3.2 se presentan conceptos básicos para el problema de rutas más cortas[CO01]. En la sección 3.3 se exponen conceptos básicos para el problema de Flujo a Costo Mínimo[AY97].

Primeramente damos una definición formal de Red, que es un concepto que será utilizado de aquí en adelante.

Definición 3.1 . *Una red $R = [X, A]$ es un grafo dirigido cuyos arcos y/o nodos tienen asociados números enteros, estos números típicamente representan costos, capacidades o flujos en el caso de arcos y ofertas o demandas en el caso de nodos.*

3.1 Conceptos básicos para el problema de flujo máximo

En general, en una red $R = [X, A, q]$ al numero $q(i, j)$ asociado a cada arco se le llama capacidad del arco (i, j) y a la cantidad enviada a través de él se le llama flujo a través del arco (i, j), denotado por f.

Definición 3.2 . *Un flujo factible en una red $R = [X, A, q]$ es una función $f : A \longrightarrow \Re$ tal que:*

1. $\sum_{j\epsilon\Gamma^+(i)} f(i,j) - \sum_{k\epsilon\Gamma^-(i)} f(k,i) = \begin{cases} v, & si \quad i = s \\ 0, & si \quad i \neq s,t \\ -v, & si \quad i = t \end{cases}$

2. $0 \leq f(i,j) \leq q(i,j)$, *para todo* $(i,j)\ \epsilon\ A$.

Al número v se le llama valor del flujo f y a las ecuaciones 1 y 2 se les conoce como "ecuaciones de conservación de flujo"; a los vértices s y t (único con oferta y demanda respectivamente) se les llama origen y destino respectivamente.

Se dice que un flujo f es máximo, si genera el mayor valor posible de v.

Sea $R = [X, A, q]$ una red y sea f un flujo factible definido en ella. Sea $C : \{s = i_1, a_1, i_2, a_2, ..., i_k, a_k, i_{k+1} = t\}$ una cadena de s a t y sean C^+ y C^- dos subconjuntos de arcos de C tales que:

$a_j\ \epsilon\ C^+ \Longleftrightarrow a_j = (i_j, i_{j+1})$

$a_j\ \epsilon\ C^- \Longleftrightarrow a_j = (i_{j+1}, i_j)$.

Es decir, los arcos de C^+ son aquellos arcos de C que tienen el sentido de s a t; los arcos de C^- son aquellos arcos de C que tienen el sentido inverso.

Definición 3.3 . *Una cadena de s a t es aumentante si* $f_{ij} < q_{ij}$ *para todo* $(i,j)\ \epsilon\ C^+$ *y* $f_{ij} > 0$ *para todo* $(i,j)\ \epsilon\ C^-$.

Recibe el nombre de cadena aumentante ya que a través de ella puede enviarse flujo de s a t construyéndose, de este modo, un flujo factible de mayor valor. Las cadenas que se muestran en las Figuras 3.1, 3.2 y 3.3 son aumentantes; la cadena de la Figura 3.4 no es aumentante; en todas ellas la pareja de números asociada a cada arco es (f_{ij}, q_{ij}).

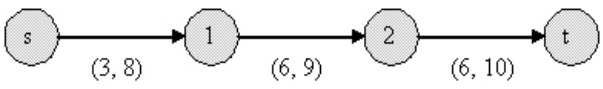


Figura 3.1: Cadena aumentante 1

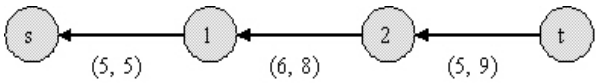


Figura 3.2: Cadena aumentante 2

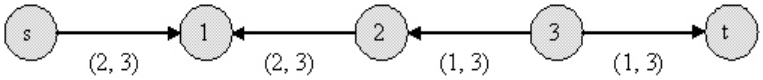


Figura 3.3: Cadena aumentante 3

Puede verificarse fácilmente que se construye un flujo factible f' de mayor valor que el flujo factible f definido en las primeras tres cadenas si se procede como sigue:

$$\begin{aligned} f_{ij} &= f_{ij} + z, \qquad \text{para todo}(i,j)\ \epsilon\ C^{+}. \\ f_{ij} &= f_{ij} - z, \qquad \text{para todo}(i,j)\ \epsilon\ C^{-}, \end{aligned}$$

donde z es una cantidad tal que $f_{ij} + z \leq q_{ij}$, para todo $(i,j)\ \epsilon\ C^{+}$ y $f_{ij} - z \geq 0$ para todo (i,j) $\epsilon\ C^{-}$. Nótese que si v es el valor de f entonces el valor de f' es $v + z$. Puesto que se desea el flujo máximo, es importante calcular el mayor valor posible de z; esta cantidad se conoce como capacidad incremental de la cadena.

Definición 3.4 . *La capacidad incremental de una cadena aumentante C es la máxima cantidad de flujo que puede enviarse aún a través de ella de s a t; se denota con $q(C)$. En base a la manera de incrementar el flujo expuesto anteriormente, $q(C)$ se calcula:*

$$q(C) = Min\{\underset{(i,j)\epsilon C^{+}}{Min}\{q_{ij} - f_{ij}\},\ \underset{(i,j)\epsilon C^{-}}{Min}\{f_{ij}\}\}.$$

Las capacidades incrementales de las cadenas de las Figuras 3.1, 3.2 y 3.3, son 3, 5 y 1 respectivamente.

En vista de los conceptos de cadena aumentante y capacidad incremental puede concluirse que un procedimiento natural para determinar el flujo máximo en una red es encontrar ca-

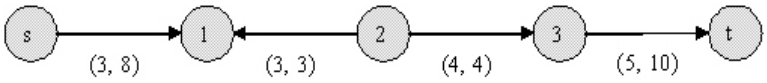


Figura 3.4: Cadena no aumentante

denas aumentantes en ésta e incrementar el flujo a través de ellas lo más que se pueda. Sin embargo, se requiere una herramienta que indique el momento en que se cumple con el criterio de optimalidad.

Sean $R = [X, A, q]$ una red y $N \subset X$. Sea $\overline{N} = X - N$. Se denota con $(N, \overline{N})$ al conjunto de arcos que tienen un extremo en N y el otro fuera de N. Se denota con $(N, \overline{N})^+$ y $(N, \overline{N})^-$ al conjunto de arcos (i, j) con $i \in N(\in \overline{N})$ y $j \in \overline{N}(\in N)$.

Definición 3.5 . *El conjunto de arcos $(N, \overline{N})$ es un corte o cortadura, que separa s de t, de R si $s \in N$ y $t \in \overline{N}$, donde s y t son el origen y destino de R. Obsérvese que si se remueve este conjunto de arcos de R ya no existe cadena alguna de s a t.*

Definición 3.6 . *La capacidad de una cortadura $(N, \overline{N})$ es la suma de las capacidades de los arcos de $(N, \overline{N})^+$. Una cortadura mínima es aquella con mínima capacidad. Se denota $q(N, \overline{N})$.*

Existen ciertas relaciones entre cortaduras y flujos en una red. Una de ellas se demuestra en la siguiente proposición:

Proposición 3.1 . *Sea $R = [X, A, q]$ una red. Sea f un flujo factible de valor v y sea $(N, \overline{N})$ una cortadura de R, entonces:*

$$v \leq q(N, \overline{N}).$$

Demostración. Sumando las ecuaciones de conservación de flujo para $i \in N$ se tiene:

$$\begin{aligned} v &= \sum_{i \in N} \sum_{j \in \Gamma^+(i)} f_{ij} - \sum_{i \in N} \sum_{k \in \Gamma^-(i)} f_{ki} \\ &= \sum_{i,j \in N} f_{ij} + \sum_{i \in N, j \in \overline{N}} f_{ij} - \sum_{i,k \in N} f_{ki} - \sum_{i \in N, k \in \overline{N}} f_{ki} \\ &= \sum_{i \in N, j \in \overline{N}} f_{ij} - \sum_{i \in N, k \in \overline{N}} f_{ki}. \end{aligned}$$

Por otro lado, se sabe que $0 \leq f_{ij} \leq q_{ij}$ para todo $(i,j)\ \epsilon\ A$, entonces:

$$v = \sum_{i\epsilon N, j\epsilon \overline{N}} f_{ij} - \sum_{i\epsilon N, k\epsilon \overline{N}} f_{ki} \leq \sum_{i\epsilon N, j\epsilon \overline{N}} f_{ij} \leq \sum_{j\epsilon N, j\epsilon \overline{N}} q_{ij}$$

como $\sum_{j\epsilon N, j\epsilon \overline{N}} q_{ij} = q(N, \overline{N})$, entonces

$v \leq q(N, \overline{N})$. ■

Otra de las relaciones importantes entre flujos y cortaduras se establecen en el siguiente teorema que proporciona una herramienta de optimalidad para el problema de flujo máximo.

Teorema 3.1 . *(Flujo Máximo-cortadura mínima). En una red $R = [X, A, q]$ el valor del flujo máximo es igual a la capacidad de la cortadura mínima.*

Demostración. En vista de la Proposición 3.1, basta demostrar que existe un flujo factible en R con valor igual a la capacidad de una cortadura de R. Para ello, considérese cualquier flujo factible f en R y constrúyase una cortadura aplicando el siguiente procedimiento:

1. Sea $N = \{s\}$.

2. Si $i\ \epsilon\ N$ y $f_{ij} < q_{ij}$ ó $f_{ij} > 0$, agréguese j a N.

Repítase 2 hasta que no pueda agregarse vértice alguno a N. Pueden presentarse entonces dos casos: o bien $t\ \epsilon\ N$, o bien $t \notin N$.

Caso 1 . *Si $t\ \epsilon\ N$ entonces, dada la construcción de N, existe una cadena C de s a t tal que $f_{ij} < q_{ij}$, para todo $(i,j)\ \epsilon\ C^+$ y $f_{ij} > 0$, para todo $(i,j)\ \epsilon\ C^-$; entonces esta cadena es aumentante y por lo tanto puede construirse un flujo mejor. En efecto, sea $q(C)$ la capacidad incremental de C y redefínase f y v como sigue:*

$$f_{ij} = \begin{cases} f_{ij} + q(C), & \textit{para todo} \quad (i,j)\ \epsilon\ C^+ \\ f_{ij} - q(C), & \textit{para todo} \quad (i,j)\ \epsilon\ C^- \\ f_{ij}, & \textit{en otro caso} \end{cases}$$

$$v = v + q(c).$$

Obsérverse que f es un flujo factible de valor v.

Constrúyase de nuevo el conjunto N con el procedimiento 1, 2 utilizando este flujo de mayor valor.

Caso 2 . *Si $t \notin N$ entonces $(N, \overline{N})$ es una cortadura de R. Por construcción se tiene que $f_{ij} = q_{ij}$, para todo $(i,j) \epsilon (N,\overline{N})^+$, y $f_{ij} = 0$, para todo $(i,j) \epsilon (N,\overline{N})^-$; entonces:*

$$v = \sum_{i\epsilon N, j\epsilon \overline{N}} f_{ij} - \sum_{i\epsilon N, k\epsilon \overline{N}} f_{ki} = \sum_{i\epsilon N, j\epsilon \overline{N}} q_{ij} = q(N,\overline{N}).$$

Bajo el supuesto de que q_{ij} es entero para todo $(i,j) \epsilon A$, en el caso 1 se incrementa el flujo en al menos una unidad; por esta razón el flujo máximo se obtiene en un número finito de pasos.

■

3.2 Conceptos básicos para el problema de rutas más cortas

En general, en una red $R = [X, A, d]$, al número $d(a)$ asociado a cada arco se le llama longitud o costo de a. Por otro lado se define la longitud de una ruta o camino como la suma de las longitudes de los arcos que la forman; aquella ruta tal que su longitud sea mínima se le llama ruta más corta o camino más corto.

Lema 3.1 *Dada una red $R = (X, A, d)$ sea $x_1, x_2, ..., x_k$ la ruta más corta entre los nodos 1 y k, y para cada i, j de tal manera que $1 \leq i \leq j \leq k$, sea $p_{ij} = x_i, x_{i+1}, ..., x_j$ el subcamino p del nodo i al nodo j. Entonces p_{ij} es la ruta más corta de i a j.*

Demostración. *Si descomponemos el camino p en subcaminos p_{1i}, p_{ij} y p_{ik}, entonces se tiene $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$. Ahora, se asume que hay un camino p'_{ij} de i a j con pesos $w(p\prime_{ij}) < w(p_{ij})$. Entonces, x_1, x_i, x_j, x_k es un camino de 1 a k el cual tiene un peso menor que $w(p)$, lo cual contradice que p es la rutas más corta de 1 a k.* ■

3.3 Conceptos básicos para el problema de flujo a costo mínimo

Definición 3.7 . *Sea $R = [X, A, q, c]$ una red y sea f un flujo factible definido en R. La red marginal o incremental de R, con respecto al flujo, es la red $R'(f) = [X, A_1 \cup A_2, q', c']$, donde:*

- $A_1 = \{(i,j) \in A \mid f_{ij} < q_{ij}\}$.
- $A_2 = \{(i,j) \mid (j,i) \in A \text{ y } f_{ij} > 0\}$.
- *q' describe la capacidad de los arcos de $R'(f)$ de la siguiente manera:*

$$q'_{ij} = q_{ij} - f_{ij}, \quad \text{para todo } (i,j) \in A_1$$

$$q'_{ij} = f_{ij}, \quad \text{para todo } (i,j) \in A_2$$

c' describe el costo unitario del flujo a través de los arcos de $R'(f)$ de la siguiente manera:

$$c'_{ij} = c_{ij}, \quad \text{para todo } (i,j) \in A_1$$

$$c'_{ij} = -c_{ij}, \quad \text{para todo } (i,j) \in A_2.$$

Considérese la red R de la Figura 3.5, donde los flujos asociados a cada arco son (capacidad, costo).

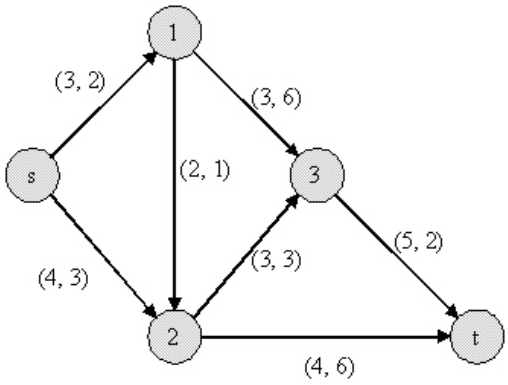


Figura 3.5: Red de ejemplo para mostrar la red marginal

Y supóngase que se requiere un flujo de valor $v = 5$. En la Figura 3.6 se observa la red de valor $v = 5$, donde el valor asociado a cada arco es el flujo.

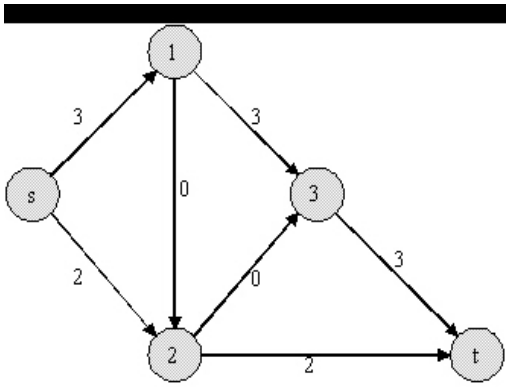


Figura 3.6: Red con un flujo de valor 5 y costo 48

En la Figura 3.7 se tiene la red marginal $R'(f)$ de la red R, con respecto al flujo definido en la Figura 3-6, los números asociados a cada arco son, la capacidad y el costo unitario a través de él.

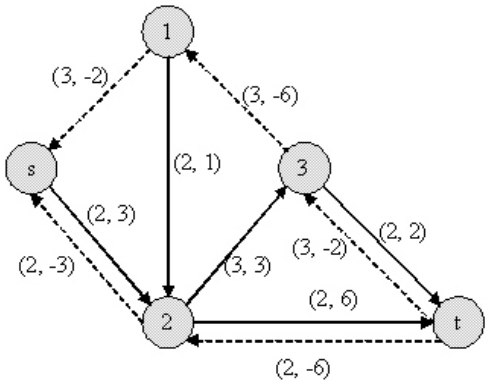


Figura 3.7: Red marginal

En base al concepto de red marginal (*Definición 3.7*) se enuncia el siguiente teorema en el cual se plasman los resultados discutidos anteriormente. Con él se propone, un algoritmo para determinar el flujo a costo mínimo en una red.

Teorema 3.2 *.Sea f un flujo factible de valor v en la red $R = [X, A, q, c]$. Entonces, el flujo f es de costo mínimo si y sólo si no existen circuitos negativos en la red marginal $R'(f)$.*

Demostración. Sean $c(f)$ el costo del flujo f y $c[C|R'(f)]$ el costo de un circuito C en $R'(f)$.

Primeramente se demostrará, por contradicción, que si f es de costo mínimo entonces no existen circuitos negativos en $R'(f)$. Para ello supóngase que existe algún circuito C en $R'(f)$ tal que $c[C|R'(f)] < 0$. Se denotará con $f + 1 \circ (C)$ al flujo definido en R de la siguiente manera, donde K es el ciclo correspondiente a C.

$$fij = \begin{cases} f_{ij}, & \text{si } (i,j) \notin K \\ f_{ij} + 1, & \text{si } (i,j) \in A_1 \cap C \\ f_{ij} - 1, & \text{si } (j,i) \in A_2 \cap C. \end{cases}$$

Es sencillo verificar que $f + 1 \circ (C)$ es de valor v. El costo de este nuevo flujo es:

$$c(f + 1 \circ (C)) = c(f) + c[C|R'(f)] < c(f),$$

lo cual contradice la hipótesis.

Para demostrar la suficiencia de la condición se utilizará un resultado que no se prueba ya que para ello se requiere de conceptos no expuestos en este texto. Este resultado establece que un flujo f de valor v puede descomponerse como: $f = 1 \circ (P_1) + 1 \circ (P_2) + ... + 1 \circ (P_{v-1}) + 1 \circ (P_v) + 1 \circ (C_1) + 1 \circ (C_2) + ... + 1 \circ (C_k)$ donde P_i es un camino elemental de s a t en $R (j = 1, ..., k)$ y $1 \circ (S)$ denota un flujo igual a una unidad en los arcos de S y a cero unidades en los arcos fuera de S.

Supóngase que $c[C|R'(f)] \geq 0$, para todo circuito C en $R'(f)$, y que $f^* (f^* \neq F)$ es el flujo a costo mínimo de valor v en R. Denótese con $f^* - f$ el flujo que describe $f^*_{ij} - f_{ij}$ unidades de flujo a través del arco $(i,j) \in A$. En base al resultado anteriormente mencionado, $f^* - f$ puede descomponerse como:

$$f^* - f = 1 \circ (C_1) + 1 \circ (C_2) + ... + 1 \circ (C_k),$$

donde C_j es un circuito en R, para $j = 1, ..., k$. Entonces,

$$f^* = f + 1 \circ (C_1) + 1 \circ (C_2) + ... + 1 \circ (C_k).$$

Puesto que este último flujo es factible, se tiene que cualquier suma $f + 1 \circ (C_1) + ... + 1 \circ (C_r)$ es factible para $1 \leq r \leq k$.

Considérese el flujo $f + 1 \circ (C_1)$, entonces:

$$c(f + 1 \circ (C_1)) = c(f) + c[C_1|R'(f)] \geq c(f).$$

Por otro lado puede verificarse que:

$c[C_r|R'(f + 1 \circ (C_1))] \geq c[C_r|R'(f)]$ para $r = 2, ...k$.

Luego, para el flujo $f + 1 \circ (C_1) + 1 \circ (C_2)$, se tiene:

$c(f + 1 \circ (C_1) + 1 \circ (C_2))$

$= c(f + 1 \circ (C_1)) + c[C_2|R'(f + 1 \circ (C_1))]$

$\geq c(f + 1 \circ (C_1)) + c[C_2|R'(f)]$

$\geq c(f + 1 \circ (C_1)) \geq c(f)$.

Continuando con este razonamiento se llega a $c(f^*) \geq c(f)$ de donde f es un flujo a costo mínimo de valor v. ■

Capítulo 4

Algoritmos y métodos

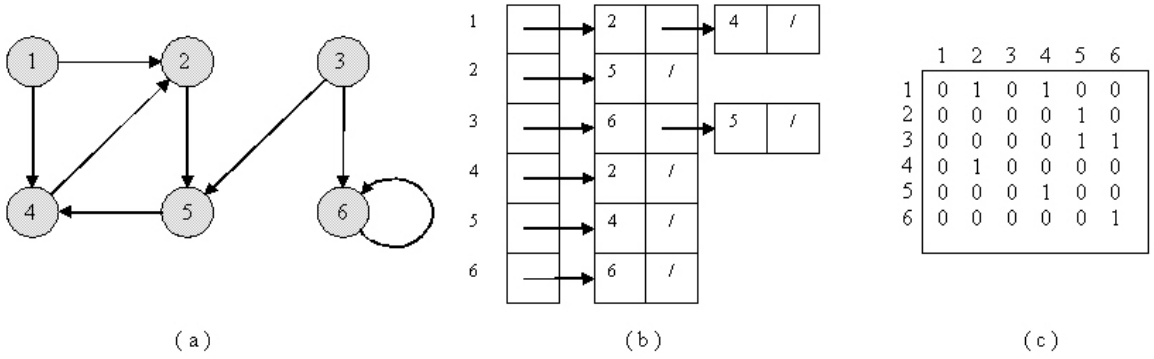
En este capítulo se presentan los dos métodos más comunes para representar grafos computacionalmente y los algoritmos utilizados para resolver el problema de Flujo a Costo Mínimo. En la sección 4.1 se muestran las dos representaciones más utilizadas de los grafos; por otra parte, el algoritmo de Ford y Fulkerson es presentado en la sección 4.2, este algoritmo es utilizado para determinar un flujo factible de valor v en la red, dicho algoritmo originalmente resuelve el problema de Flujo Máximo en una red, por lo que es necesario hacer una pequeña modificación en el criterio de paro de dicho algoritmo. La sección 4.3 presenta el algoritmo de Floyd-Warshall que se utiliza para determinar la existencia de circuitos negativos en la red marginal, el algoritmo de Floyd-Warshall se utiliza para determinar rutas más cortas entre todo par de nodos, si existen, o se determina algún circuito negativo concluyéndose, en este caso, que no hay solución al problema. En la sección 4.4 se muestra el algoritmo Primal para resolver el problema de Flujo a Costo Mínimo que está basado en la eliminación de circuitos negativos. Por último, en la sección 4.5 se ilustra un algoritmo para representar de manera gráfica las redes.

4.1 Representación de grafos

Las dos formas más comunes de representar un grafo $G = [X, A]$ computacionalmente son: colección de listas de adyacencia y matrices de adyacencia. La representación de listas de adyacencia se prefiere generalmente, porque proporciona una manera compacta de representar grafos en los cuales m es mucho menor que n^2, donde n es el número de nodos y m el número

de arcos del grafo. Una representación de matriz de adyacencia puede ser preferida cuando el grafo es denso - m es cercano a n^2 - o cuando necesitamos saber rápidamente si existe un arco que conecte dos nodos dados.

La representación mediante listas de adyacencias de un grafo $G(X, A)$ consiste de un arreglo Adj de n listas, uno para cado nodo en X. Para cada nodo $u \epsilon X$, la lista de adyacencia $Adj[u]$ contiene todos los nodos v como si fuera un arco $(u, v) \epsilon A$. Esto es, $Adj[u]$ está compuesta de todos los nodos adyacentes a u en G (alternativamente, podría contener apuntadores a todos esos nodos.) Los nodos en cada lista de adyacencia son normalmente almacenados en orden arbitrario. La Figura 4.1(b) es una representación de listas de adyacencia del grafo de la Figura 4.1(a) [CO01].



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Figura 4.1: Representación de un grafo. (a) Grafo dirigido. (b) Representación de listas de adyacencia. (c) Representación de matriz de adyacencia

Las listas de adyacencia se pueden adaptar fácilmente para representar una red, esto se hace de la siguiente manera: los nodos del grafo se representan por una lista ligada de *nodos cabecera*. Cada nodo cabecera contiene tres campos: *info*, *nextnode* y *arcptr*. Si p apunta a un nodo cabecera que representa un nodo $a \epsilon X$, *info(p)* contiene cualquier información asociada al nodo del grafo, a. *nextnode(p)* es un apuntador al nodo cabecera que representa el siguiente nodo del grafo, si lo hay. Cada nodo cabecera está a la cabeza de una lista de nodos de un segundo tipo llamados *nodos de lista*. Esta lista se llama *lista de adyacencia*. Cada nodo en una lista de adyacencia representa un arco del grafo, *arcptr(p)* apunta a la lista de adyacencia de nodos que representan los arcos que salen del nodo a [TE93].

Cada nodo de la lista de adyacencia contiene tres campos: *info*, *ndptr*, *nextarc*. Si q apunta a un nodo de la lista que representa un arco (i, j), *ndptr(q)* es un apuntador al nodo cabecera que representa el nodo j, del grafo. *Nextarc(q)* apunta a un nodo de la lista que representa el siguiente arco que sale del nodo i del grafo, si lo hay. Cada nodo de lista está contenido en una sola lista de adyacencia, que representa todos los arcos que salen de un nodo dado del grafo.

Adviértase que los nodos cabecera y los nodos de lista tienen formato diferente y tienen que ser representados por estructuras diferentes.

Para la representación con matrices de adyacencia de un grafo $G(X, A)$, se asume que los nodos son numerados $1, 2, ..., n$ (ver sección 2.1).

La Figura 4.1(c) muestra la matriz de adyacencia del grafo de la Figura 4.1(a).

Al igual que en la representación de listas de adyacencia, en la matriz de adyacencia también se pueden representar las redes. Por ejemplo, si $G = [X, A, q]$ es un grafo con una función de capacidad q, la capacidad del arco (i, j), $q(i, j)$ es simplemente asignado en el renglón i y la columna j de la matriz de adyacencia. Si un arco no existe, un valor nulo es almacenado en la posición correspondiente de la matriz.

La matriz de adyacencia, es con frecuencia una representación inadecuada de un grafo porque se requiere el previo conocimiento del número de nodos. Si un grafo tiene que ser construido en el transcurso de la resolución del problema, o si tiene que ser actualizado de manera dinámica cuando el programa procede, tiene que ser creada una nueva matriz para cada adición o eliminación de un nodo. Esto es demasiado ineficiente, en especial en una situación real donde un grafo puede tener 100 ó más nodos. Además, aun si un grafo tiene pocos arcos de manera que la matriz de adyacencia sea casi nula, tiene que reservarse espacio para todo arco posible entre nodos, ya sea que exista o no un arco. Si el grafo tiene n nodos, se tienen que usar n^2 localidades de memoria.

Para la implementación del Algoritmo Primal para resolver el problema de Flujo a Costo Mínimo se utiliza la representación de listas de adyacencia.

4.2 Flujo máximo

4.2.1 Descripción del problema

Un problema típico de flujo máximo entre origen y destino es el siguiente: supóngase que en un poblado s se dispone de cierto producto. Este producto es demandado en un poblado t. Se requiere obtener la cantidad máxima posible del producto en t considerando que al transportarse, éste puede pasar por otros poblados que no tienen oferta ni demanda. Supóngase además, que hay una capacidad máxima de transporte entre cada par de poblados. A este problema puede asociarse la red $R = [X, A, q]$ donde:

- X representa al conjunto de poblados,
- si $i, j \epsilon X$, $(i, j) \epsilon A \Longleftrightarrow$ es posible transportar el producto del poblado i al poblado j.
- $q : A \longrightarrow \Re$, donde para $(i, j) \epsilon A$, $q(i, j) =$ capacidad máxima de transporte del poblado i al poblado j.

Se desea entonces determinar la cantidad del producto a transportar del poblado i al poblado j de tal manera que en t se obtenga la máxima cantidad posible de tal producto. Debe observarse que la cantidad total que entre a un poblado i, distinto de s y t, debe ser igual a la cantidad total que sale de él puesto que no tiene oferta ni demanda; por otro lado, la cantidad total que se obtenga en t debe ser igual a la cantidad total que sale de s puesto que no se genera ni se pierde el producto a través de la red; por último la cantidad a enviar de i a j, para $(i, j) \epsilon A$, debe ser menor o igual que $q(i, j)$, que no es otra cosa que las leyes de conservación de flujo (Definición 3.2).

El problema de flujo máximo ha sido estudiado por mas de 40 años. El primer método clásico para resolver este problema fue el método de Ford y Fulkerson, este algoritmo está basado en el hecho de que un flujo es máximo si y solo si no existen cadenas aumentantes. Otros métodos clásicos son el método de etiquetado de Edmonds y Karp, el método de bloqueo de flujo de Dinic, el método preflujo de Goldberg, y una mejora al método preflujo desarrollado por Goldberg y Tarjan. En la siguiente sección se describe el método de Ford y Fulkerson, que es el método utilizado en esta tesis para determinar el flujo máximo en una red, se ha decidido utilizar este método por su simplicidad y por su facilidad de implementación.

4.2.2 Método de Ford y Fulkerson

El método de Ford-Fulkerson resuelve el problema de flujo máximo. Le llamamos método y no un algoritmo, dado que permite varias implementaciones con diferentes tiempos de ejecución [CO01]. El método Ford-Fulkerson es iterativo, puesto que se empieza con un flujo $f(i,j) = 0$ para todo $i, j \epsilon X$ y en cada iteración, se incrementa el valor del flujo hasta que logre el mayor flujo posible sin violar las restricciones de capacidad. Para lograr ir aumentando la cantidad de flujo transportado, en cada paso se utilizan cadenas aumentantes, las cuales pueden pensarse como un camino desde la fuente s hasta el destino t por el que se puede enviar todavía flujo. El método de Ford-Fulkerson consiste en utilizar tantas cadenas aumentantes como existan. El método de Ford-Fulkerson se resume de la siguiente forma [CO01]:

Método Ford-Fulkerson

Inicie el flujo f en 0.
Mientras exista una cadena aumentante C
    Aumentar el flujo a lo largo de C
Regresar f.

Dicho en otras palabras, la idea es encontrar conductos subutilizados entre la fuente y el destino, que pueden aumentarse hasta que una restricción de capacidad detenga el aumento.

La implementación del método anterior, mostrado en la Figura 4.2 calcula un flujo de valor v en una Red $R = [X, A, q]$, donde:

v : Es el valor de flujo factible.

v' : Es el valor del flujo actual.

$q(C)$: Es la capacidad incremental de la cadena aumentante C.

s : Nodo origen.

t : Nodo destino.

Este algoritmo hace uso de la rutina $CapacidadIncremental()$, que determina la capacidad incremental de la cadena aumentante, en caso de no encontrar una cadena aumentante, devuelve un valor 0, por lo tanto se ha encontrado un flujo máximo. Como se mencionó anteriormente, se hace una pequeña modificación en este algoritmo, esto lo observamos en el paso 3 del diagrama

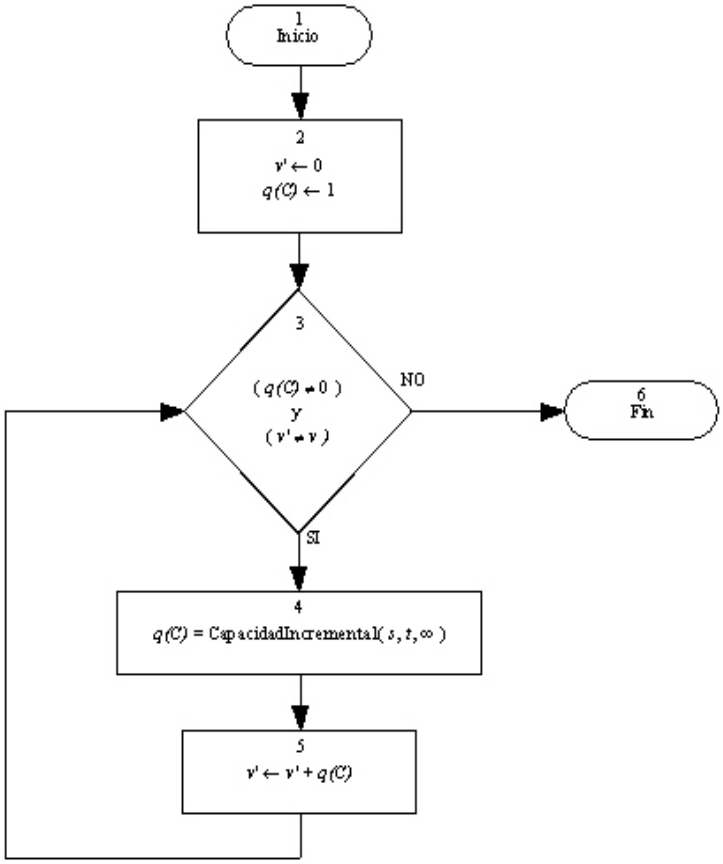


Figura 4.2: Algoritmo de Ford y Fulkerson

de flujo de la Figura 4.2, se añade una nueva condición de paro si el flujo actual v' es igual al flujo factible v, entonces se termina el algoritmo.

La rutina $CapacidadIncremental(\ s, t,\ f\ )$ de la Figura 4.3, recibe tres argumentos: el nodo origen s, el nodo destino t y el flujo máximo que se puede enviar a través del arco $(s, j)\ \epsilon\ A$.

En esta rutina también se hace una modificación en el paso 4 del diagrama de flujo de la Figura 4.3, después de que se ha llegado al criterio de paro, es decir, cuando se ha encontrado una cadena aumentante. Esta modificación consiste en lo siguiente: si el valor del flujo actual v' más el valor del mayor flujo que se puede transportar de s a t denotado por f es mayor o igual que el flujo factible v, entonces se modifica el flujo en la cantidad $v - v'$.

La rutina $EncontrarArco1(u)$ utilizada dentro de la rutina $CapacidadIncremental(s, t, f)$,

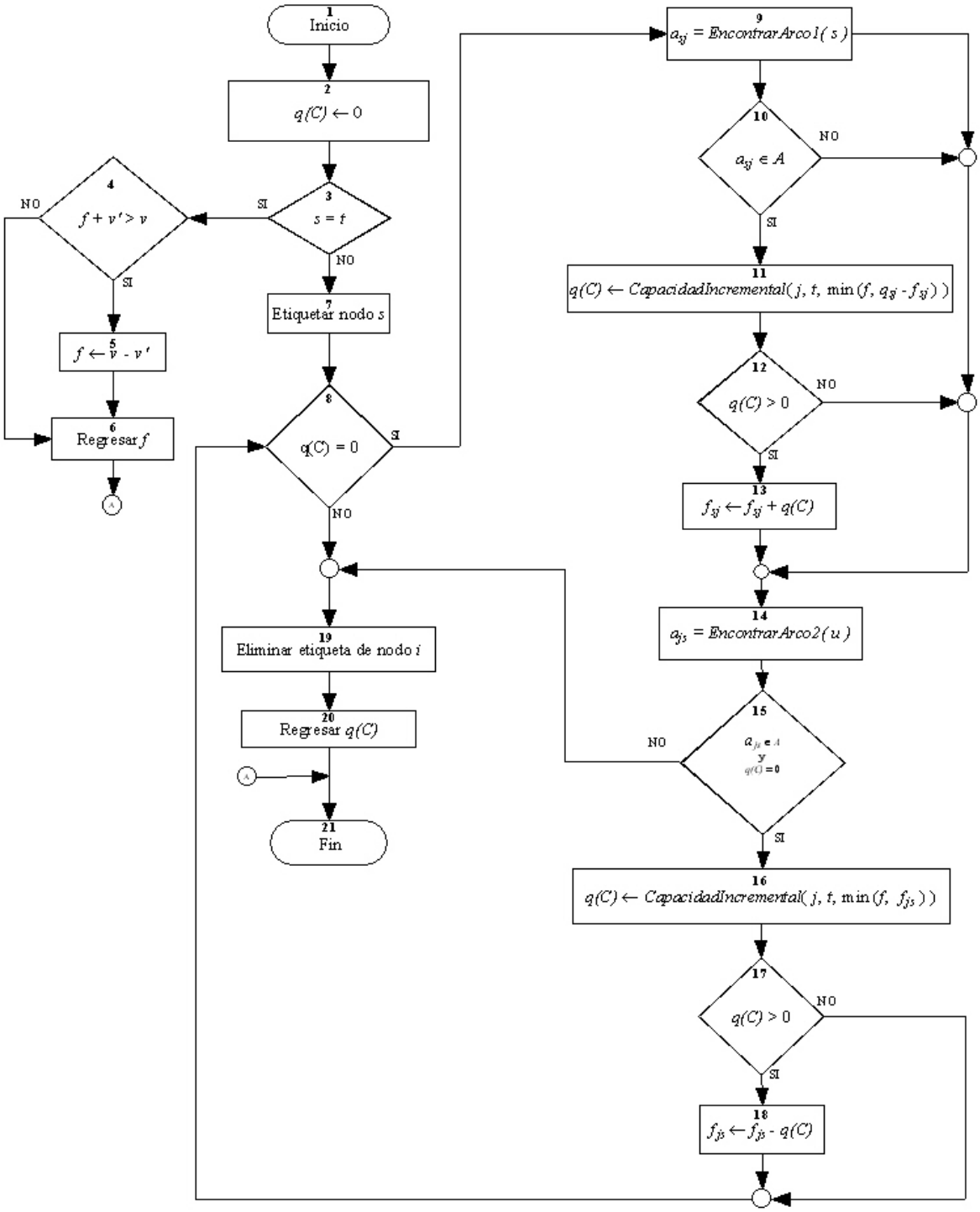


Figura 4.3: Implementación de la rutina *CapacidadIncremental*

busca de entre todos los arcos que salen de el nodo u el de menor capacidad, de tal manera, que el flujo a través de él sea menor que la capacidad. La rutina $EncontrarArco2(u)$ busca un arco el cual su nodo destino sea el nodo u, y el flujo a través de él sea mayor que cero (ver Apéndice B/Ejemplo B.1).

4.3 Ruta más corta entre todo par de nodos

4.3.1 Descripción del problema

Para ejemplificar considérese el siguiente problema:

En una terminal de autobuses para pasajeros se tiene interés en encontrar las rutas más cortas para todos los camiones que prestan servicio entre cada par de ciudades. A este problema se le puede asociar una red $R = [X, A, d]$ donde:

- $X = \{$Ciudades a la cuales ofrece servicio$\}$.
- $A = \{$Tramos de carretera entre las ciudades$\}$.
- $d : A \longrightarrow \Re$ donde, para todo $a \epsilon A$, $d(a) =$ longitud o distancia del tramo de carretera a.

Se deben encontrar, entonces, las rutas más cortas entre todo par de nodos en la red R.

Para que exista una solución en cualquier red $R = [X, A, d]$, deberá cumplirse lo siguiente:

1. Existe, al menos, un camino entre todo par de nodos.
2. No existen circuitos negativos en la red R.

Se puede resolver el problema de rutas más cortas entre todo par de nodos con un algoritmo de rutas más cortas entre dos nodos, corriéndolo n veces, una para cada nodo fuente. Si los pesos asociados a cada arcos no son negativos se puede utilizar el algoritmo de Dijkstra. Si existen arcos con pesos negativos el algoritmo de Dijkstra no es aplicable. Sin embargo, se puede utilizar el algoritmo de Ford-Bellman corriéndolo una vez para cada nodo. Pero existen algoritmos más eficientes para resolver el problema de rutas más cortas entre todo par de nodos, como el algoritmo de Floyd-Warshall o el algoritmo de Johnson para grafos densos. En la siguiente sección se explica detalladamente el algoritmo de Floyd-Warshall.

4.3.2 Algoritmo de Floyd-Warshall

Este algoritmo fue desarrollado por R.W. Floyd basándose en un teorema de Warshall (1962) y es aplicable a redes que admiten cualquier costo en sus arcos. En dicho algoritmo se supondrá una numeración de nodos de la red $1, 2, ..., n$.

En el algoritmo de Floyd-Warshall en la k-ésima iteración se calcula la longitud de la ruta más corta entre i y j que puede admitir a los primero k nodos, o a alguno de ellos, como nodos intermedios, este número se le asigna al arco a_{ij}. Las longitudes de las rutas más cortas, entre todo par de nodos i y j, que no contengan ningún nodo como nodo intermedio están determinadas por el costo del arco (i, j).

Al principio de la k-ésima iteración, el costo del arco (i, j) es igual a la longitud de la ruta más corta entre i y j, que contiene a los primeros $k-1$ nodos, o a alguno de ellos, como nodos intermedios. Durante esta iteración se compara la longitud de esta ruta con la de aquella formada por la unión de las rutas más cortas que contienen a los primeros $k-1$ nodos como nodos intermedios, entre i y k, y k y j; de esta manera se obtiene la ruta más corta que contiene a los primeros k nodos, o a alguno de ellos, como nodos intermedios. Procediendo de este modo se tendrá que, al final de la n-ésima iteración, el valor del arco (i, j) es la longitud más corta, entre i y j, que contiene a los primeros n nodos como nodos intermedios o a alguno de ellos; es decir, se habrá calculado la longitud de la ruta más corta entre i y j.

Si algún arco (i, i) es menor que cero en alguna iteración, se habrá encontrado una ruta de i a i de longitud negativa (es decir, un circuito negativo). Luego, en este caso, el problema no tiene solución. Por esta razón se utiliza este algoritmo.

El algoritmo de Floyd-Warshall termina en exactamente n iteraciones, donde n es el número de nodos de la red, a menos que se encuentre algún circuito negativo en la red; en este último caso, se terminará en a los más n iteraciones. A continuación se describe el algoritmo de Floyd-Warshall [AY97].

Algoritmo Floyd-Warshall (c)

Para cada $k \epsilon X$.

Para cada $i \epsilon X$.

Para cada $j \epsilon X$.

Si $i \neq k$ *y* $k \neq j$.

Si $c_{ik} \epsilon A$ *y* $c_{kj} \epsilon A$,

$$c_{ij} = \min\{c_{ij}, c_{ik} + c_{kj}\}.$$

Regresar Ruta más corta.

La implementación de este algoritmo se ilustra en la Figura 4.4.

Para este caso específico utilizamos el algoritmo de Floyd-Warshall para determinar la existencia de circuitos negativos en la red marginal, por lo tanto en esta implementación si se encuentra algún circuito negativo en la red, el algoritmo devuelve el nodo en el cual está el circuito negativo, en caso de no encontrar circuitos negativos el algoritmo devuelve un valor nulo (ver Apéndice B/Ejemplo.B.2).

4.4 Flujo a costo mínimo

4.4.1 Descripción del problema

Se considera el problema de elegir el mejor flujo, bajo el criterio de su costo. En general, puede tenerse interés en elegir el mejor flujo, entre origen y destino, de cierto valor que no es necesariamente el máximo. Sea $R = [X, A, q, c]$ una red con una función q de capacidad y una función c de costos por unidad de flujo asociadas a sus arcos. Supóngase que se tiene interés en determinar un flujo factible de valor v, entre el origen y el destino, incurriendo en el mínimo costo posible; al flujo de menor costo se le llama flujo a costo mínimo de valor v.

Si la cantidad v de flujo requerido es mayor que el valor del flujo máximo, el problema no tiene solución.

Existen diferentes algoritmos para resolver el problema de flujo a costo mínimo como son: el algoritmo Primal, el algoritmo Dual y el algoritmo Primal-dual, entre otros. En la siguiente sección se describe detalladamente el algoritmo Primal.

4.4.2 Algoritmo Primal

El algoritmo Primal es llamado así, puesto que empieza a aplicarse a partir de un flujo factible de valor v requerido y, en cada iteración, el costo del flujo se mejora, sin modificar su valor, hasta alcanzar la optimalidad.

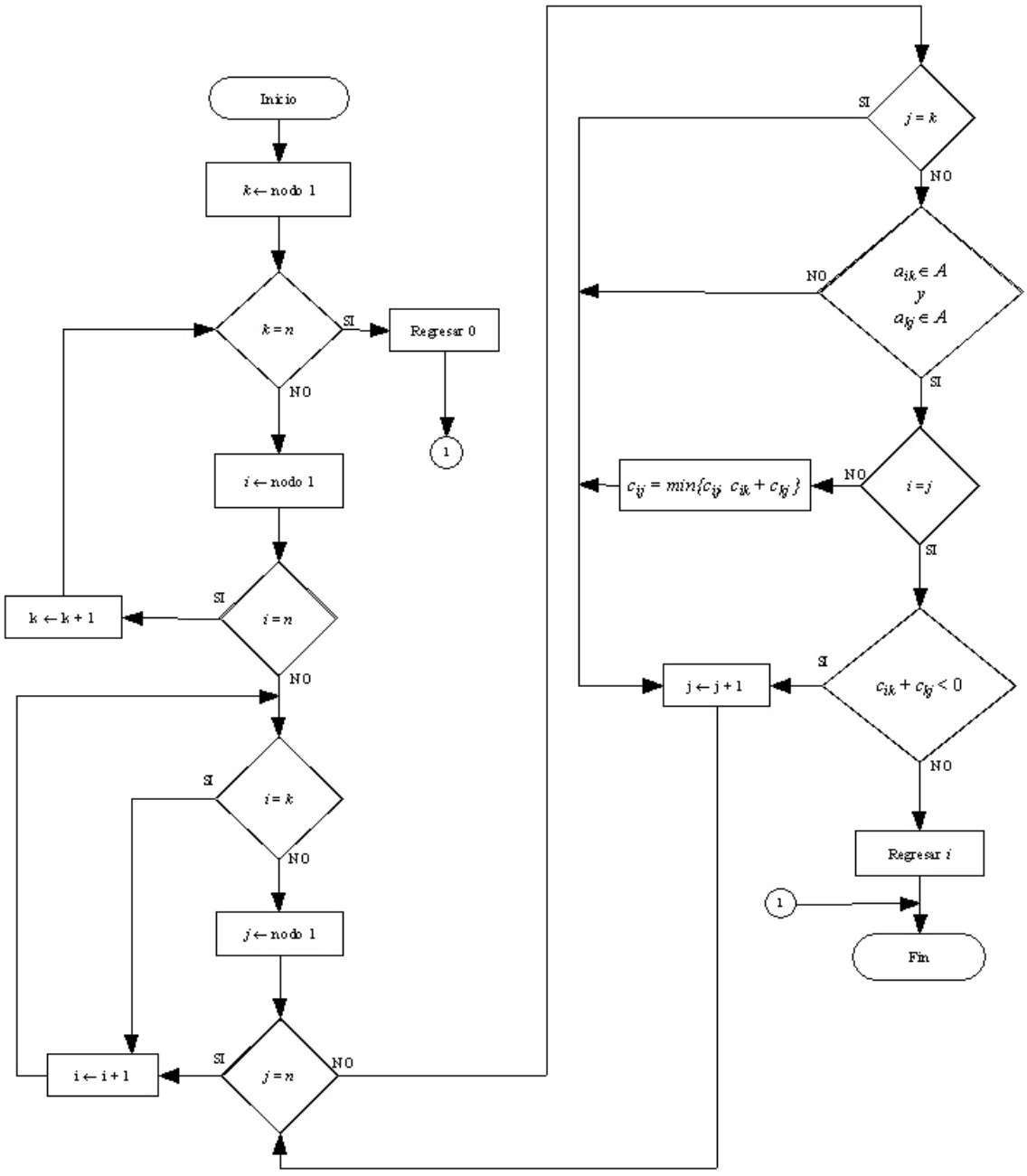


Figura 4.4: Algoritmo de Floyd

La red marginal nos permite determinar si un flujo factible dado es de costo mínimo o no y, en caso de que no lo sea, nos proporciona una manera de construir un flujo de menor costo a partir de aquél con que se cuenta.

Un circuito en la red marginal corresponde a un ciclo en la red R con la característica de que es posible modificar el flujo a través de él. En efecto, cada arco $(i,j)\ \epsilon\ A_1$ corresponde a uno de A con $f_{ij} < q_{ij}$ por lo cual el flujo puede aumentarse en este arco; cada arco $(i,j)\ \epsilon\ A_2$ corresponde a un arco $(j,i)\ \epsilon\ A$ con $f_{ji} > 0$ por lo cual puede decrementarse el flujo a través de él.

Por otro lado, se obtiene un nuevo flujo factible del mismo valor v, si se modifica en cierta cantidad d el flujo a través del ciclo de la siguiente forma:

$$f_{ij} = \begin{cases} f_{ij}, & \text{si } (i,j) \quad \text{no pertenece al ciclo} \\ f_{ij} + d, & \text{si } (i,j) \in A_1 \quad \text{y pertenece al ciclo} \\ f_{ij} - d, & \text{si } (j,i) \in A_2 \quad \text{y el arco } (i,j) \text{ pertenece al ciclo.} \end{cases}$$

Es fácil verificar que el flujo f' así definido es factible de valor v. Para ello se demostrará que:

$$\sum_j f'_{ij} - \sum_k f'_{ki} = \begin{cases} v, & \text{si } i = s \\ 0, & \text{si } i \neq s,t \\ -v, & \text{si } i = t. \end{cases}$$

Esto es claro si el nodo i no está en el ciclo. Ahora, si i es un nodo del ciclo, entonces tiene dos vecinos m y n en dicho ciclo. Si en el circuito correspondiente en $R'(f)$ están los arcos (m,i) e (i,n) existen varios casos a considerar:

(a) $(m,i),(i,n)\ \epsilon\ A_1$. Esto implicaría que m es predecesor de i y n es sucesor de i en el ciclo (Figura 4.5).

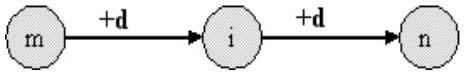


Figura 4.5: Caso (a)

(b) $(m,i),(i,n) \epsilon A_2$. Entonces m es sucesor de i y n es predecesor de i en el ciclo (Figura 4.6).

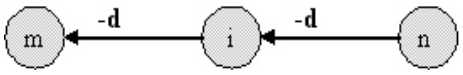


Figura 4.6: Caso (b)

(c) $(m,i) \epsilon A_1, (i,n) \epsilon A_2$. En este caso m y n son predecesores de i en el ciclo (Figura 4.7).

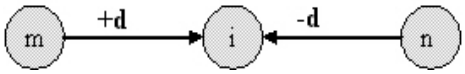


Figura 4.7: Caso (c)

(d) $(m,i) \epsilon A_2, (i,n) \epsilon A_1$. Por tanto m y n son sucesores de i en el ciclo (Figura 4.8).

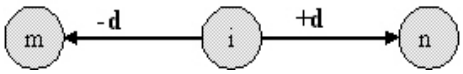


Figura 4.8: Caso (d)

En el caso (a) se tiene:

$$\sum_j f'_{ij} - \sum_k f'_{ki} = \sum_{j \neq n} f'_{ij} + f'_{in} - \sum_{k \neq m} f'_{ki} - f'_{mi} =$$

$$\sum_{j \neq n} f_{ij} + (f_{in} + d) - \sum_{k \neq m} f_{ki} - (f_{mi} + d) =$$

$$\sum_j f_{ij} - \sum_k f_{ki} = \begin{cases} v, & \text{si } i = s \\ 0, & \text{si } i \neq s,t \\ -v, & \text{si } i = t. \end{cases}$$

Análogamente se prueban los otros tres casos.

La red marginal indica entonces la manera de obtener otros flujos factibles del mismo valor, más aún, indica el cambio en el costo al modificar dicho flujo. Puede entonces concluirse que si existe algún circuito negativo en $R'(f)$ conviene modificar el flujo a través del ciclo, el cambio total en el costo está dado por el costo del circuito correspondiente mejorando así el mismo. Es deseable determinar la máxima cantidad, d, en que puede modificarse el flujo a través del ciclo. Claramente, d debe ser tal que no rebase la capacidad de los arcos del circuito en $R'(f)$ ya que estas capacidades representan el cambio posible en los arcos de R. Por lo tanto, $d = Min\{q'_{ij}$ con (i,j) en el circuito$\}$ [AY97].

Algoritmo Primal

Determinar un flujo factible f de valor v mediante el algoritmo de Ford y Fulkerson.

Construir la red marginal, con respecto a f, $R'(f)$.

Mientras exista algún circuito negativo C

$d = \min_{(i,j)\epsilon C} \{q'_{ij}\}$.

Actualizar el flujo en una cantidad d en la red.

Construir la red marginal, con respecto a f, $R'(f)$.

Regresar v.

En la Figura 4.9 se ilustra la implementación de este algoritmo.

(ver Apéndice B/Ejemplo B.3).

4.5 Representación gráfica de redes

4.5.1 Descripción del problema

La representación gráfica de redes, estudia cómo pueden ser presentadas las redes en un plano, para proveer una representación visual del grafo de forma más comprensible.

Recientemente la representación gráfica automatizada de grafos ha creado un intenso interés debido a sus extensas aplicaciones y, como consecuencia, ha surgido una gran variedad en el tipo de representaciones de grafos y sus correspondientes algoritmos. Generalmente, se representan grafos en el plano y otras superficies asociando puntos a sus vértices y arcos de curvas a sus aristas.

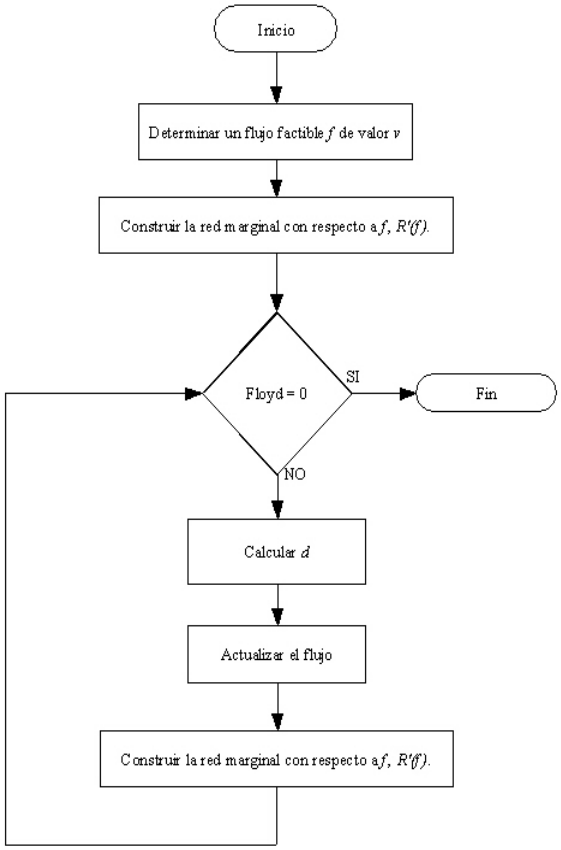


Figura 4.9: Algoritmo Primal

Como es de imaginarse, no existe solo una forma de representar gráficamente los grafos, ni existe solo un algoritmo que dibuje los grafos de la mejor manera siempre [BA99].

4.5.2 Algoritmo Representación Gráfica

Como parte de esta tesis se ha implementado un algoritmo que hace una representación gráfica de redes, este algoritmo funciona para grafos pequeños (entre 2 y 15 nodos) dependiendo de la complejidad del mismo.

Definición 4.1 *Sea G un grafo dirigido sin ciclos (o conjunto parcialmente ordenado), una ordenación topológica T de G es una ordenación lineal de los nodos de G que preserva la ordenación parcial de G original, o sea, que si $u < v$ en G (si existe un arco de u a v en G), entonces u va delante de v en la ordenación lineal T .*

El algoritmo que se ha implementado consta de tres fases: en la primera fase se hace un preprocesamiento del grafo que consiste en eliminar todos los ciclos del grafo, si es que existen; en la segunda fase se hace una ordenación topológica del grafo; como última fase del algoritmo se realiza la asignación de las coordenadas x y y para cada nodo [BA99].

Algoritmo Representación Gráfica

Eliminar ciclos de G.

Realizar una ordenación topológica T *de* G.

Para cada nodo $u \epsilon T$

    Asignar coordenadas x *y* y.

Capítulo 5

Pruebas y resultados

En este capítulo se presentan las pruebas realizadas con el software desarrollado, así como los resultados obtenidos. Las pruebas fueron realizadas en una PC con un procesador Intel® Pentium® 4 a 1.5 GHz, 384 MB de memoria RAM, con sistema operativo Linux Mandrake 9.1. y ambiente gráfico KDE.[6]

Para probar la eficiencia del algoritmo Primal para resolver el problema de flujo a costo mínimo, se clasificaron los grafos de la siguiente manera:

Grafos de complejidad simple (Figura 5.1), estos grafos son aquellos que no contienen ciclos y que no contienen ningún arco de regreso. Grafos de complejidad media (Figura 5.2), son aquellos grafos que contienen algún ciclo o algún arco de regreso. Y por último, grafos de complejidad alta, son aquellos que contienen arcos de regreso y ciclos (Figura 5.3).

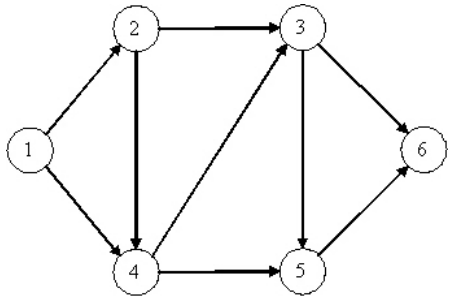


Figura 5.1: Grafo de complejidad simple

[6]K Desktop Environment. http://www.kde.org

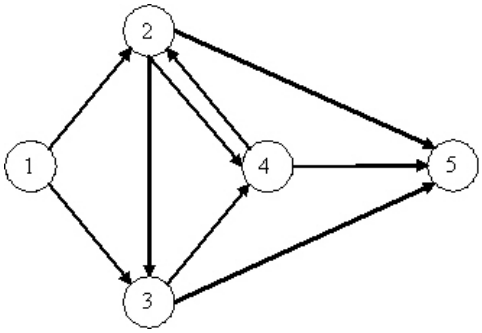


Figura 5.2: Grafo de complejidad media con un ciclo entre los nodos 2 y 4

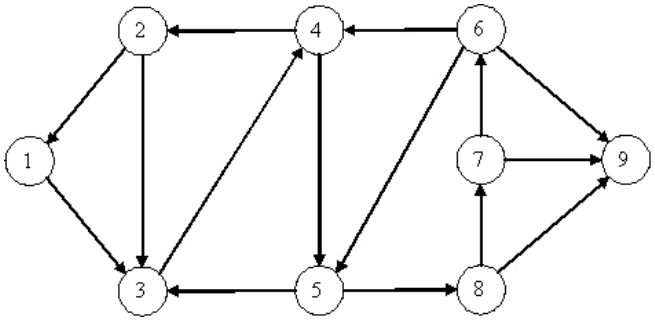


Figura 5.3: Grafo de complejidad alta con ciclos y con arcos de regreso

Para las pruebas se tomaron como medidas: el tiempo de ejecución, el número de iteraciones del algoritmo Primal, el número de iteraciones realizadas por el algoritmo de Ford y Fulkerson, y el número de iteraciones totales[7] realizadas por el algoritmo de Floyd-Warshall. En la siguiente sección se muestran los resultados obtenidos de los grafos seleccionados. Para cada grafo se muestra una tabla, la cual contiene en la primera columna el flujo desde 1 hasta el flujo máximo de la red, en la segunda columna se ha colocado el número de iteraciones del algoritmo Primal; en la tercera, el número de iteraciones del algoritmo de Ford y Fulkerson utilizado para calcular el flujo deseado; la cuarta columna contiene el número de iteraciones del algoritmo de Floyd-Warshall; la quinta columna muestra el tiempo de ejecución en microsegundos y en algunos casos en milisegundos. En la sexta columna se muestra el costo mínimo encontrado.

[7]Son todas las iteraciones realizadas en cada ciclo del algoritmo Primal

5.1 Resultados

5.1.1 Grafos de complejidad simple

El primer grafo que se presenta fue desarrollado como ejemplo dentro de las clases de la materia Investigación de Operaciones, impartida por la profra M.C. Marcela Rivera Martínez.

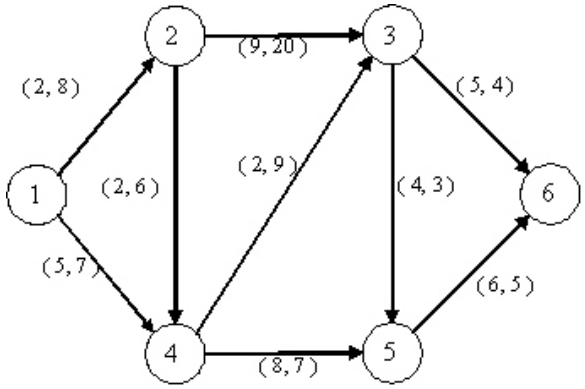


Figura 5.4: Grafo 1

A continuación se presenta la tabla con los resultados obtenidos al correr el programa.

v	Primal	F-F	F-W	t (μs)	Costo
1	3	1	13	121	19
2	3	1	13	123	38
3	3	2	13	126	57
4	3	2	13	128	76
5	3	2	13	123	95
6	3	2	13	125	121
7	2	2	11	95	148

Tabla 5.1: Resultados del Grafo 1

Después de haber corrido el algoritmo, los resultados fueron comparados con los obtenidos en el curso de Investigación de Operaciones y, para todos los casos los resultados obtenidos fueron los óptimos. En la gráfica (Figura 5.5) se muestra el comportamiento del algoritmo Primal respecto al tiempo y valor del flujo, como se puede observar en este caso el algoritmo alcanza su tiempo máximo cuando se corre para un flujo de valor 4 y a partir de ahí, ese tiempo

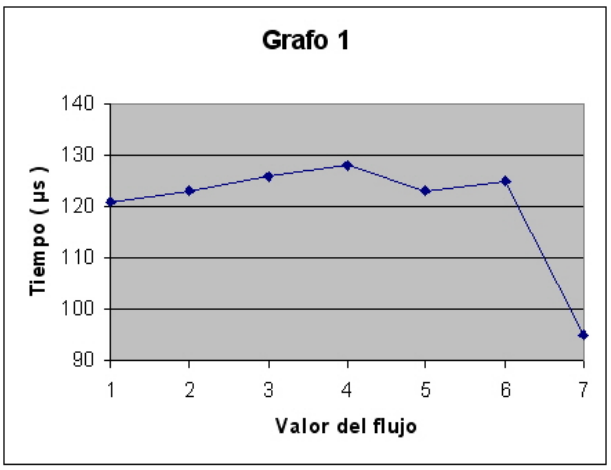


Figura 5.5: Gráfica para el Grafo 1

se empieza a decrementar y se mantiene el decremento hasta llegar a un flujo igual al flujo máximo.

El siguiente grafo fue tomado del libro Introducción a la Teoría de Redes [AY97].

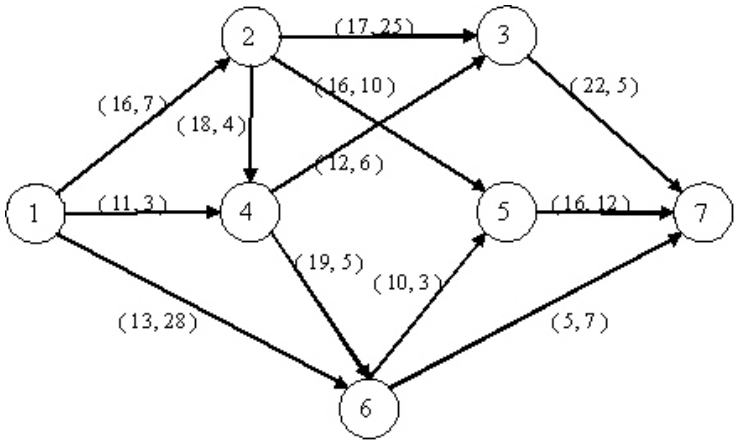


Figura 5.6: Grafo 2

En la tabla 5.2 se muestran los resultados obtenidos al correr el programa.

Los resultados mostrados en la tabla 5.2 son los óptimos y fueron comprobados con los resultados presentados en el libro. Para este ejemplo, en la gráfica (Figura 5.7) observamos

v	Primal	F-F	F-W	Tiempo (μs)	Costo
1	2	1	9	104	22
2	2	1	9	105	44
3	2	1	9	116	66
4	2	1	9	102	88
5	2	1	9	101	110
6	2	1	9	102	132
7	2	1	9	103	154
8	2	1	9	103	176
9	2	1	9	100	198
10	2	1	9	102	220
11	2	1	9	101	242
12	4	2	19	214	264
13	4	2	19	218	287
14	4	2	19	231	310
15	4	2	19	226	333
16	4	2	19	231	356
17	6	3	24	370	381
18	6	3	24	329	412
19	6	3	24	338	443
20	6	3	24	376	474
21	6	3	24	339	505
22	6	3	24	346	536
23	6	3	24	365	567
24	6	3	24	357	598
25	6	4	24	365	629
26	6	4	24	339	660
27	6	4	27	351	691
28	6	4	27	355	732
29	5	4	23	282	773
30	5	4	23	293	824
31	5	4	23	294	855
32	5	4	23	277	896
33	5	5	21	265	939
34	5	5	21	282	990
35	5	5	21	27	1041
36	5	5	21	272	1092
37	5	5	21	257	1143
38	5	5	21	281	1194
39	5	5	21	266	1245
40	2	5	10	130	1296

Tabla 5.2: Resultados del Grafo 2

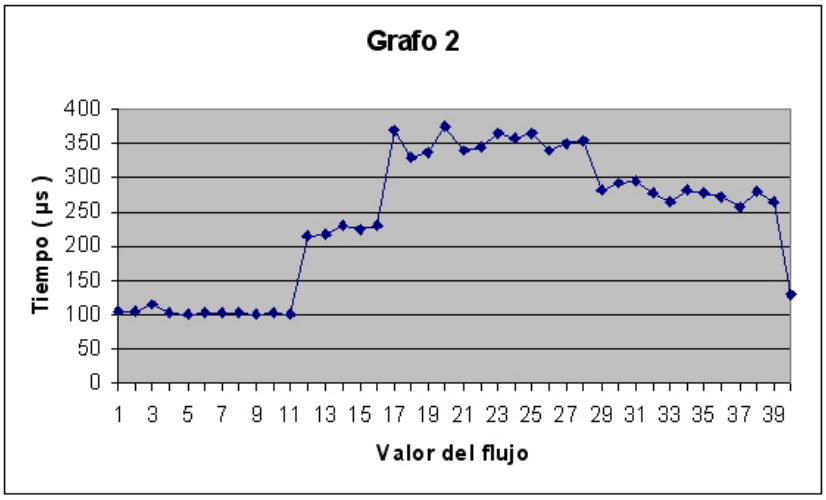


Figura 5.7: Gráfica para el Grafo 2

que el algoritmo realiza un mayor número de iteraciones y por lo tanto de tiempo cuando se corre con un flujo de valor 20, y a partir de ese flujo el número de iteraciones y el tiempo empiezan a disminuir.

Por último se presenta un grafo de complejidad simple que fue tomado del ejemplo del programa NETFLOW[8] para problemas de flujo a costo mínimo, el problema es el siguiente:

Piñas enteras se sirven en un restaurante en Londres. Para asegurar la frescura, las piñas se compran en Hawaii y son transportados por aire de Honolulu a Heathrow en Londres. El costo de transportar las piñas es mostrado en la siguiente red (Figura 5.8, donde el valor asociado a cada arco es el costo por unidad de piñas). No se pueden transportar más de 350 piñas semanalmente, el restaurante utiliza 500 piñas a la semana.

[8] http://www.csc.fi/cschelp/sovellukset/stat/sas/sasdoc/sashtml/ormp/chap4/sect63.htm

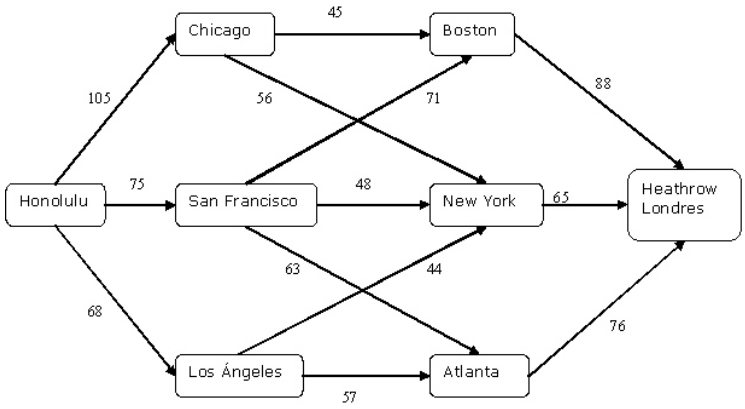


Figura 5.8: Ejemplo del programa NETFLOW.

Entonces, el problema es formulado de la siguiente manera:

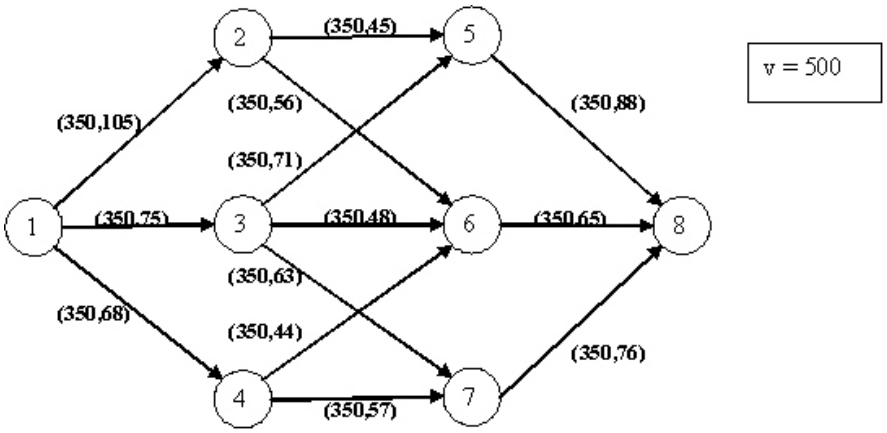


Figura 5.9: Grafo 3

Para este grafo sólo se hizo la prueba con un flujo de 500 por ser una restrcción el problema. Al correr el programa con esta red, obtenemos el siguiente resultado

el costo para el flujo de valor 500 es óptimo y se comprobó con el resultado obtenido por el programa NETFLOW.

v	Primal	F-F	F-W	Tiempo (μs)	Costo
500	3	2	21	275	93750

Tabla 5.3: Resultados del Grafo 3

5.1.2 Grafos de complejidad media

El siguiente grafo fue elaborado especialmente para realizar las pruebas.

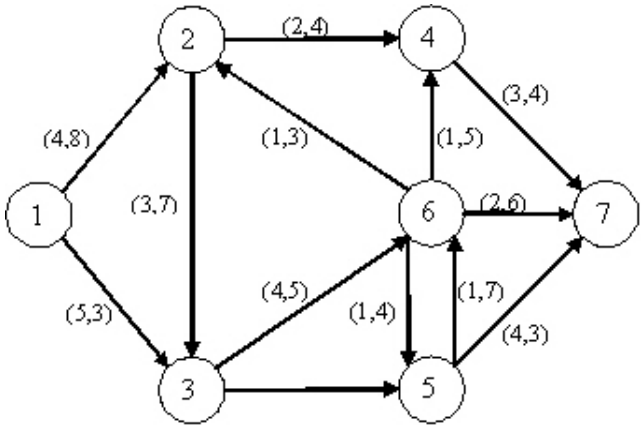


Figura 5.10: Grafo 4

En la siguiente tabla se muestran los resultados obtenidos.

v	Primal	F-F	F-W	Tiempo (μs)	Costo
1	1	1	6	58	5
2	1	1	6	36	10
3	2	2	11	110	17
4	2	2	11	108	24
5	2	2	11	104	32
6	2	2	11	108	40
7	3	3	13	145	48
8	2	3	8	76	56

Tabla 5.4: Resultados del Grafo 4

Los resultados obtenidos fueron comprobados con los que se obtuvieron al resolver el grafo manualmente y en todos los casos se llegó al óptimo. En la gráfica (Figura 5.11) se observa el comportamiento del algoritmo, cuando se corre el algoritmo para un flujo de valor 7 el algoritmo

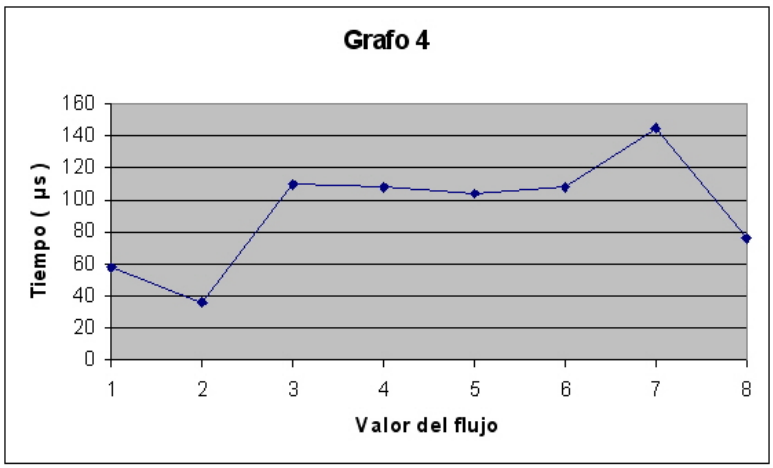


Figura 5.11: Gráfica para el Grafo 4

realiza el mayor número de iteraciones, lo cual se ve reflejado en el tiempo de ejecución, después de este flujo el número de iteraciones disminuye.

v	Primal	F-F	F-W	Tiempo (μs)	Costo
1	2	1	12	132	8
2	2	1	12	136	16
3	4	2	20	227	24
4	4	2	20	225	32
5	5	3	25	286	46
6	7	4	34	396	62
7	5	5	26	306	78
8	4	5	20	231	104
9	2	5	12	120	133

Tabla 5.5: Resultados del Grafo 5

El siguiente ejemplo fue tomado del libro Introducción a la teoría de redes[AY97].

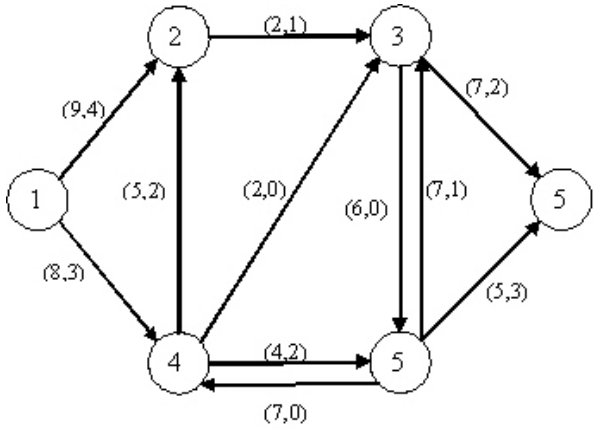


Figura 5.12: Grafo 5

En la Tabla 5.5 observamos los resultados obtenidos.

Estos resultados fueron comparados con los resultados presentados en el libro y en todos los casos se llegó al óptimo. En la gráfica (Figura 5.13) podemos observar que el algoritmo realiza un mayor número de iteraciones cuando se corre para un flujo de valor 6, y por lo tanto el tiempo es mayor, a partir de este flujo el número de iteraciones se empieza a decrementar y por lo tanto el tiempo es menor.

El último grafo de complejidad media que se utilizó para realizar pruebas es un grafo de 38 nodos, éste es el grafo más grande con que se cuenta (Figura 5.14).

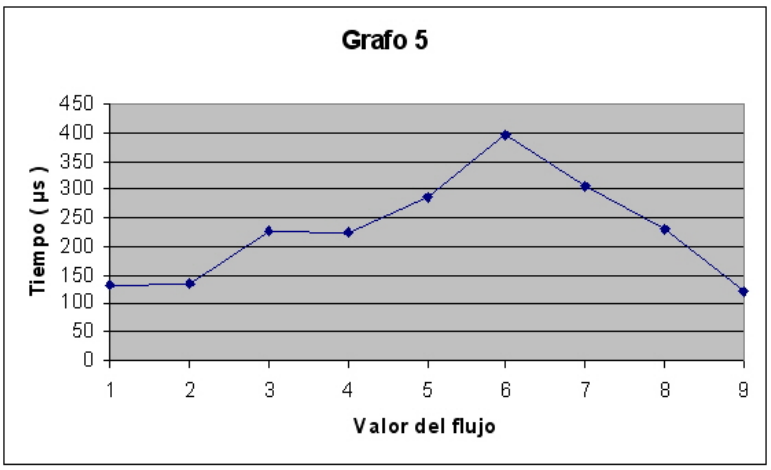


Figura 5.13: Gráfica para el Grafo 5

Los resultados obtenidos se muestran en la Tabla 5.6. Estos resultados obtenidos por el programa fueron comprobados con los resultados obtenidos al resolver el problema manualmente y en ambos casos se llegó al óptimo. Para este grafo se observa que el mayor número de iteraciones y por lo tanto de tiempo, se obtienen al correr el algoritmo para un flujo de valor 8 (Figura 5.15), a partir de este flujo, el número de iteraciones se empieza a decrementar.

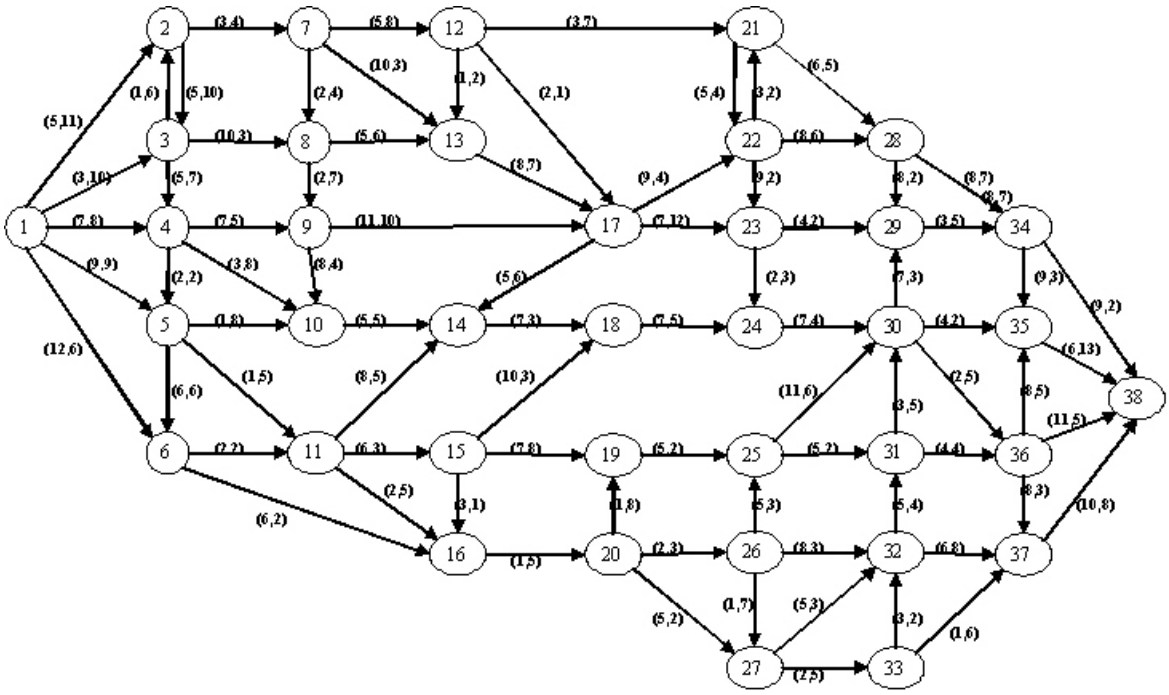


Figura 5.14: Grafo 6

5.1.3 Grafos de complejidad difícil

El grafo que se muestra a continuación fue hecho especialmente para realizar las pruebas[9].

En la tabla 5.7 se muestran los resultados obtenidos.

Los resultados que se obtuvieron fueron comparados con los obtenidos al resolver el problema manualmente y en ambos casos se obtuvieron los óptimos. En la gráfica (Figura 5.17) observamos que el mayor número de iteraciones las realiza el algoritmo con un flujo de valor 4, y a partir de este flujo el tiempo de ejecución empieza a decrementarse.

Finalmente, observamos dos gráficas, una para grafos simples (Figura 5.18) y otra para grafos medios (Figura 5.19), en estas gráficas se puede observar el comportamiento del algoritmo respecto al tiempo y valor del flujo, en la segunda gráfica se observa claramente, que el tiempo de ejecución esta relacionado con el número de nodos del grafo y la complejidad del mismo, ya que en esta gráfica los tiempos son mayores que para los de la primera gráfica.

[9]Elaborado por la Profra M.C. Marcela Rivera Martínez.

v	Primal	F-F	F-W	Tiempo (ms)	Costo
1	8	1	211	19	30
2	16	2	250	41	62
3	19	3	318	92	94
4	21	4	364	101	132
5	25	5	438	103	170
6	25	6	438	135	208
7	27	7	497	150	246
8	29	8	580	180	288
9	29	9	553	167	330
10	24	10	406	108	372
11	26	11	431	139	414
12	26	11	497	158	457
13	25	12	469	149	500
14	23	13	366	119	544
15	22	14	373	102	589
16	23	15	403	105	640
17	20	16	381	110	691
18	18	17	380	115	743
19	17	18	337	102	808
20	13	18	246	61	879

Tabla 5.6: Resultados del Grafo 6

Durante la fase de pruebas y resultados se ha notado un comportamiento uniforme en todas las redes sin importar el tamaño de estas; esto es, el tiempo de ejecución y el número de iteraciones del algoritmo Primal empiezan a incrementarse a partir de un flujo de valor 1, pero llega un punto en el cual llegan a un máximo y a partir de este punto el número de iteraciones y el tiempo de ejecución empiezan a decrementarse.

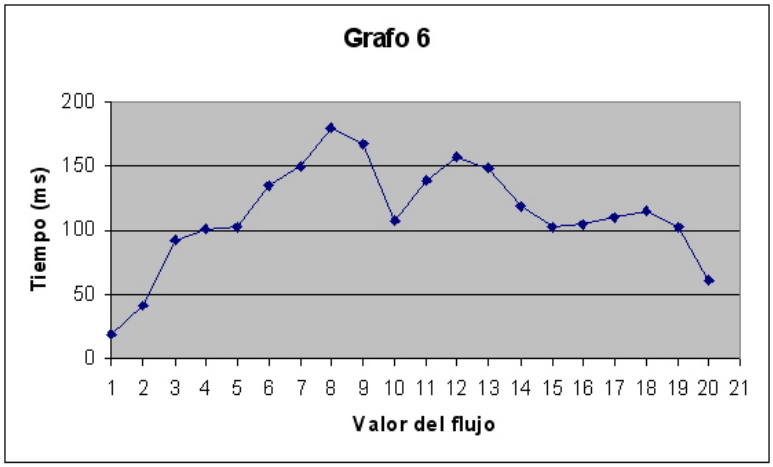


Figura 5.15: Gráfica para el Grafo 6

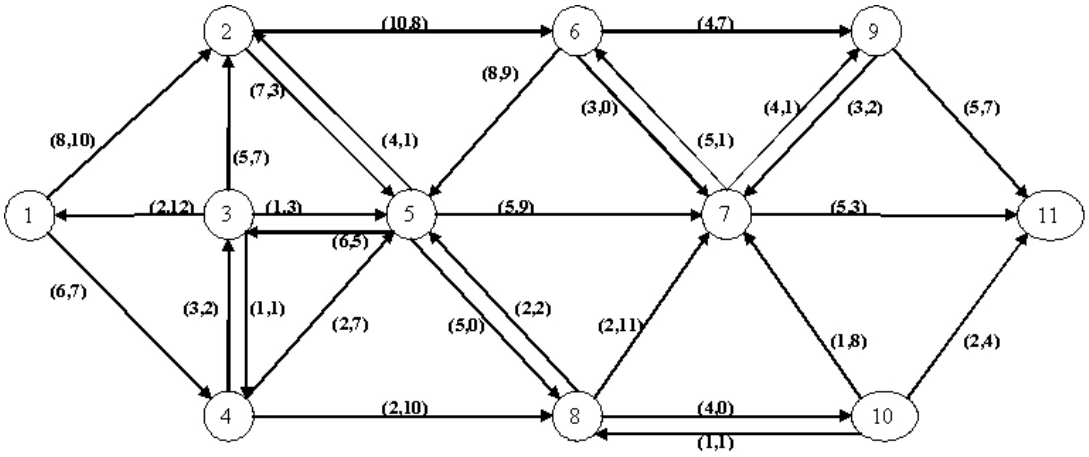


Figura 5.16: Grafo 7

v	Primal	F-F	F-W	Tiempo (ms)	Costo
1	4	1	33	1.42	16
2	7	1	55	1.22	33
3	8	2	59	1.32	54
4	9	2	67	1.57	75
5	9	3	64	1.39	96
6	8	4	50	1.13	120
7	8	5	44	0.962	145
8	6	6	32	0.749	175
9	6	7	32	0.756	205
10	7	7	37	0.895	236
11	6	7	32	0.741	267
12	7	8	38	0.864	303

Tabla 5.7: Resultados del Grafo 7

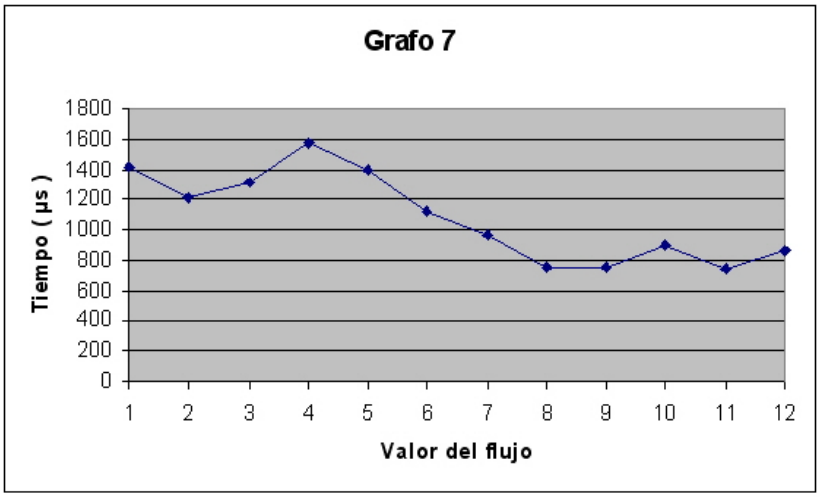


Figura 5.17: Gráfica para el Grafo 7

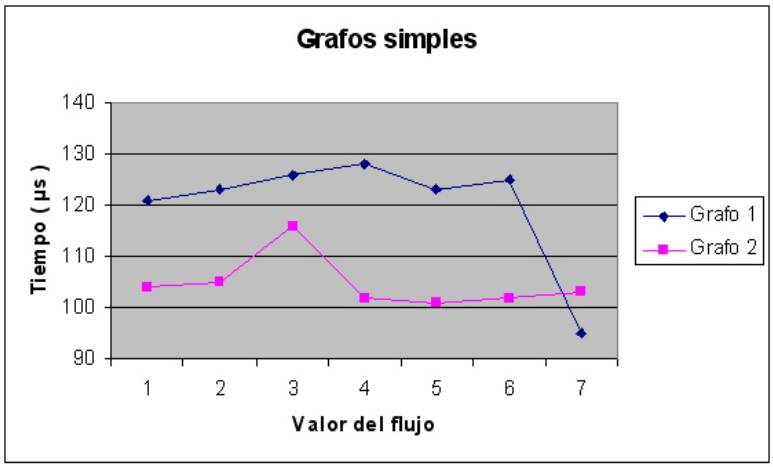


Figura 5.18: Gráfica para grafos simples

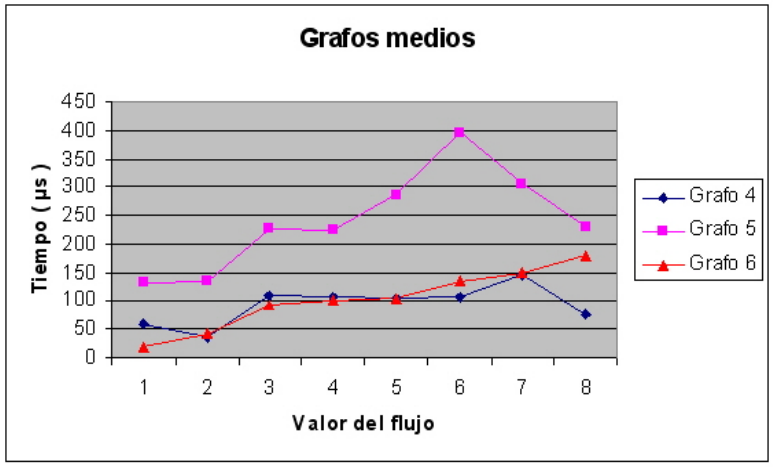


Figura 5.19: Gráfica para grafos de complejidad media

Conclusiones

Después de haber concluido el desarrollo de esta tesis se han cumplido los objetivos planteados al inicio de la misma.

Se desarrolló un programa en plataforma linux que resuelve el problema de flujo a costo mínimo, el programa llamado Primal. Una de las partes importantes del programa es que muestra la solución de los problemas en forma gráfica lo que hace más fácil el entendimiento del problema, y es útil en los casos en los cuales se quiere explicar los pasos del algoritmo detalladamente. Como es común, el algoritmo tal y como se presenta en [AY97] funciona de manera correcta para los casos generales, sin embargo, se tuvieron que hacer modificaciones en los algoritmos para que el programa funcionara correctamente, como por ejemplo para grafos con ciclos.

En el algoritmo de Ford y Fulkerson, la primera modificación que se hizo fue cambiar el criterio de paro con el fin de utilizar dicho algoritmo para obtener un flujo requerido y no el flujo máximo, esta modificación es mencionada en la sección 4.2.2. El algortimo también se modificó para que funcionará para grafos con ciclos y finalmente se hizo una modificación para determinar cuál arista elegir en el caso de tener múltiples de ellas en una decisión, en este caso siempre se toma la arista de menor capacidad. Por lo que se refiere a el algoritmo Floyd-Warshall, se le hizo una modificación ya que su función principal en este trabajo, es el detectar circuitos negativos y con ello determinar si se cumple o no con el criterio de optimalidad.

Durante la etapa de pruebas se ha observado el comportamiento del algoritmo, midiendo el tiempo de ejecución y el número de iteraciones realizadas, y como se pudo observar todas las ejecuciones terminan en un número finito de iteraciones y tiempo.

Como se mencionó en el capítulo anterior el algoritmo muestra un comportamiento en el cual el número de iteraciones sube y luego baja, esto se debe a que conforme el valor del flujo se va acercando al flujo máximo de la red, el flujo en los arcos se va saturando, y el algoritmo de Floyd-Warshall, que es el encargado de detectar los circuitos negativos en la red realiza menos iteraciones ya que existen menos circuitos negativos en la red. El número de iteraciones del algoritmo Primal depende directamente del número de iteraciones del algoritmo de Floyd-Warshall por lo tanto mientras mayor sea el número de iteraciones en el algoritmo de Floyd-Warshall, mayor será el número de iteraciones del algoritmo Primal y mayor será el tiempo de ejecución.

Durante la fase de pruebas no hubo ningún problema con los grafos ya que para todos los ejemplos el programa siempre llegó al óptimo.

El problema de flujo a costo mínimo es un problema muy común en el mundo real y se puede aplicar en muchas áreas, el programa no se particularizó para ningún problema en específico por lo cual, haciendo un debido planteamiento del problema se puede utilizar para cualquier caso, como por ejemplo los problemas mostrados en el apéndice C, uno de ellos pertenece al conjunto de problemas del concurso internacional de programación de la ACM[10].

El programa también resuelve el problema de flujo máximo y el de rutas más cortas, siendo estos problemas parte del algoritmo principal para resolver el problema de flujo a costo mínimo.

El número máximo de nodos que soporta el programa no se pudo determinar, debido a que no se encontraron grafos suficientemente grandes para hacer las pruebas, el grafo más grande con el que se probó el programa fue la red de 38 nodos mostrada en la Figura 5.14.

El algoritmo Primal es solo uno de los diversos algoritmos existentes para resolver el problema de flujo a costo mínimo, en esta tesis se reprodujo dicho algoritmo para que pueda ser usado en trabajos futuros.

[10] Association for Computing Machinery. http://icpc.baylor.edu/icpc/

Apéndice A

Manual de usuario

En este apéndice se presenta el manual de usuario correspondiente al programa desarrollado. El programa fue desarrollado bajo el sistema operativo Linux Mandrake 9.1 en el ambiente gráfico KDE y se utilizó el ambiente visual Borland® Kylix™ 3 Open Edition en lenguaje C++. Como requerimientos mínimos para el sistema operativo Linux Mandrake 9.1 se requiere un procesador tipo x586 o superior; esto incluye Intel Pentium I/II/III/IV, AMD K6/II/III, AMD Duron, AMD Athlon/XP/MP, requeridos al menos 64 MB de memoria en RAM, 128 MB recomendados; se requiere al menos 100 MB de espacio libre en disco duro, 1 GB recomendado.

El programa fue desarrollado y probado en una PC con un procesador Intel® Pentium® 4 a 1.5 GHz, 384 MB de memoria RAM, con sistema operativo Linux Mandrake 9.1.

La principal razón por la que se decidió trabajar en ambiente linux es porque existen herramientas y software libre. Podemos observar que Kylix es una plataforma de desarrollo libre ya que en todas las aplicaciones que se desarrollan en Kylix muestran la siguiente pantalla inicial:

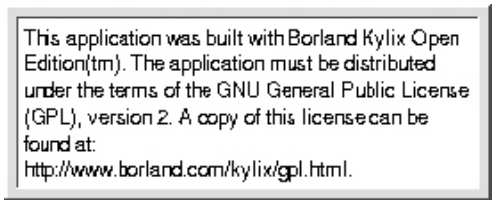


Figura A.1: Pantalla Inicial al ejecutar el programa

Es decir, muestra la herramienta utilizada y que la aplicación está bajo la licencia GPL.

El software desarrollado nos permite resolver problemas de flujo máximo, rutas más cortas, y flujo a costo mínimo, cabe hacer la aclaración de que el usuario debe de tener conocimientos en el área de teoría de redes.

Para instalar el programa se debe ejecutar el archivo Setup-Primal en modo root, esto instalará el programa Primal en la carpeta usr/local/Primal. Para iniciar el programa en el menú principal elija Flujo a Costo Mínimo y seleccionar Primal. Al iniciar el programa podemos observar la siguiente pantalla:

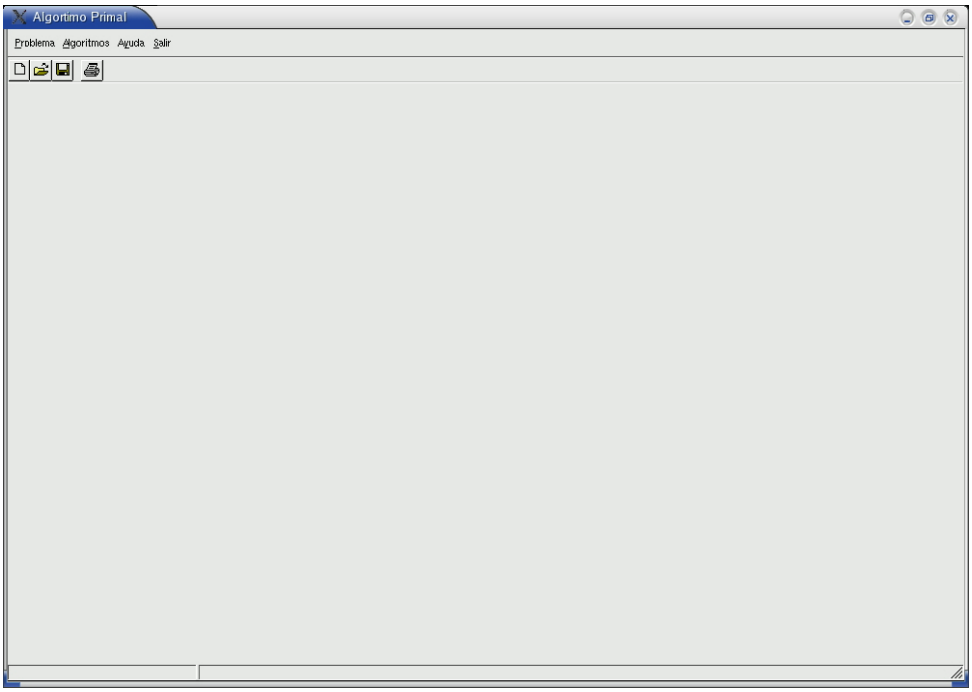


Figura A.2: Pantalla principal

A.1 Cargar problema

Para cargar un problema en memoria se puede hacer de dos maneras: introduciendo los datos mediante teclado o abriendo un archivo previamente guardado.

A.1.1 Introducir un nuevo grafo por teclado

Para crear un nuevo grafo se selecciona la opción Nuevo en el menú Problema, y aparecerá un cuadro de diálogo (Figura A.3) en el que hay que introducir el número total de nodos que conforman el problema representado por una red (o grafo).

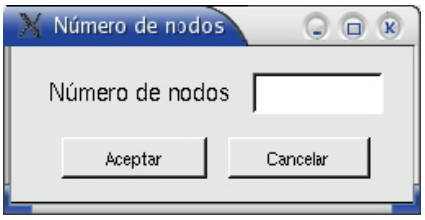


Figura A.3: Cuadro de diálogo para introducir el número de nodos de la red

Se presiona el botón Aceptar y aparecerá una tabla (Figura A.4) en la cual se introducen los datos (capacidades, costos) del grafo, o bien solo una de las anteriores.

	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

Figura A.4: Tabla para introducir los valores de los arcos de la red

Si se desean introducir capacidades y costos estos valores deben ser introducidos separados por una coma (,); por ejemplo para un arco con capacidad 3 y costo 2 la introducción en la tabla sería 3,2. También se pueden introducir sólo capacidades o costos.

Hay que tomar en cuenta que si sólo se introducen capacidades o costos, el programa sólo podrá utilizar los algoritmos de Ford y Fulkerson o de Floyd-Warshall, respectivamente. Las capacidades deberán ser siempre números positivos mayores que cero, y los costos deberán ser números mayores o iguales que cero en el caso de problemas de flujo a costo mínimo. Cuando se introduce un solo valor asociado a cada arco, se pueden introducir números enteros positivos

o negativos, teniendo en cuenta que si se introducen números negativos sólo se podrán resolver problemas de rutas más cortas aplicando el algoritmo de Floyd-Warshall.

Una vez que se han terminado de introducir los datos, en la parte inferior de la pantalla se presiona el botón:

Figura A.5: Botón Aceptar

El programa determinará si el grafo se puede presentar en modo gráfico o en modo texto, dependiendo del números de nodos y de la resolución de la pantalla.

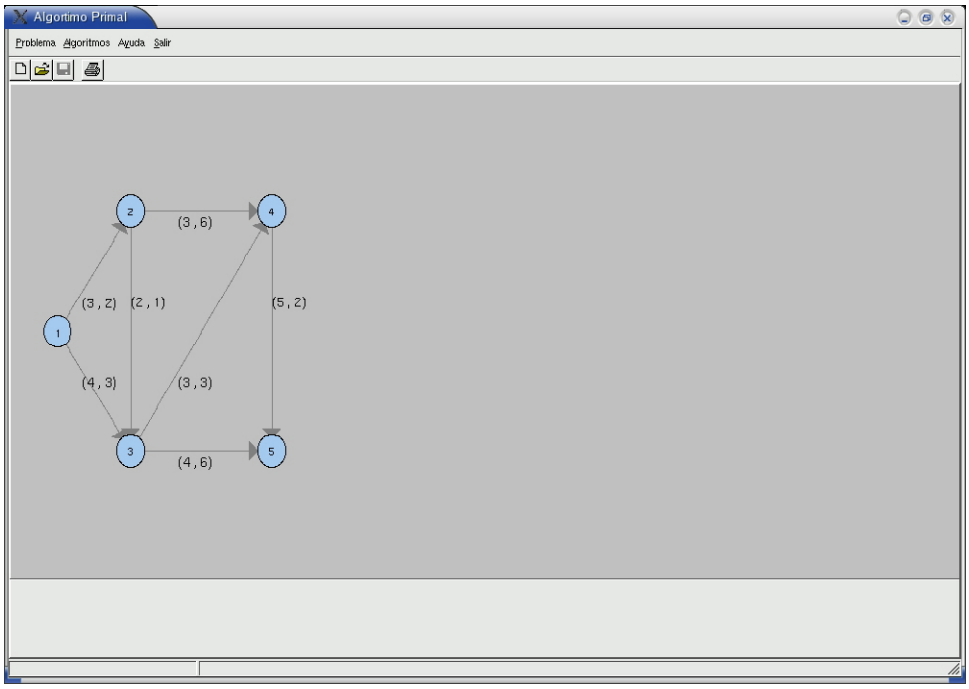


Figura A.6: Visualización de la red

Una vez que se ha cargado el problema (Figura A.6) se habilita el menú algoritmos y los algoritmos correspondientes, dependiendo de los valores en los arcos del grafo.

A.1.2 Cargar un nuevo problema por archivo

Si se desea cargar un problema desde archivo para poder realizar su análisis o imprimirlo se selecciona la opción Abrir del menú Problema. Esta acción abre un cuadro de diálogo en el cual se selecciona el archivo de tipo *.txt que se desea abrir, y una vez que se presiona el botón Abrir es presentado en pantalla lo siguiente:

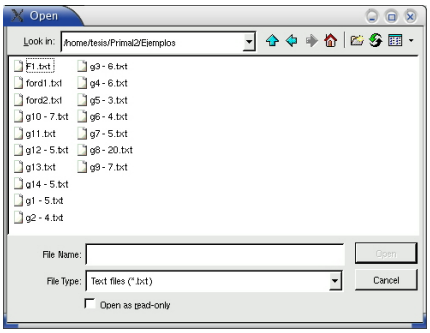


Figura A.7: Cuadro de diálogo Abrir

Al presionar Open, se carga el problema y se muestra la siguiente pantalla:
El formato del archivo que se va a abrir debe ser el siguiente:

Grafo1.txt

2				<—— Indica que cada arco tiene dos valores asociados.
1	2	2	3	<—— El primer número indica el nodo origen, el segundo
1	3	4	3	el nodo destino el tercero la capacidad y el cuarto
2	3	2	1	el costo.
2	4	3	6	
3	4	3	3	
3	5	4	6	
4	5	5	2	

ó

Grafo2.txt

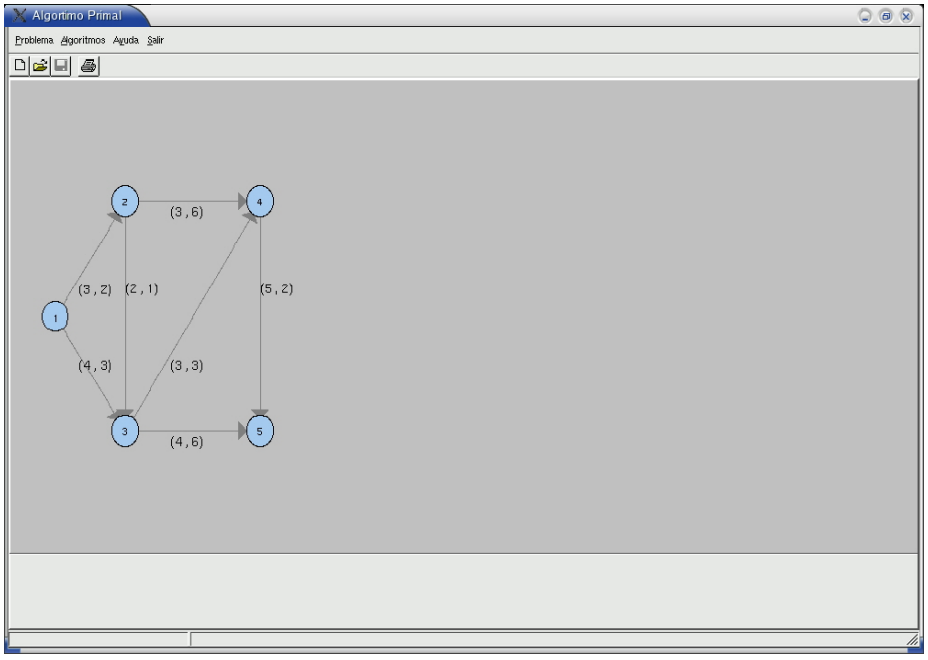


Figura A.8: Visualización de la red

```
1                <—— Indica que cada arco tiene asociado un valor.
1   2   2    <—— El primer número indica el nodo origen, el segundo el nodo
1   3   4        destino y el tercero la capacidad o el costo.
2   3   2
2   4   3
3   4   3
3   5   4
4   5   5
```

A.2 Guardar problema

Cuando se desee guardar un grafo capturado para después hacer su análisis es necesario seleccionar la opción Guardar del menú Grafo (Figura A.9).

De esta manera se guarda un archivo de tipo *.txt que se podrá cargar en cualquier momento desde este programa. Los archivos son guardados con el mismo formato mostrado anteriormente.

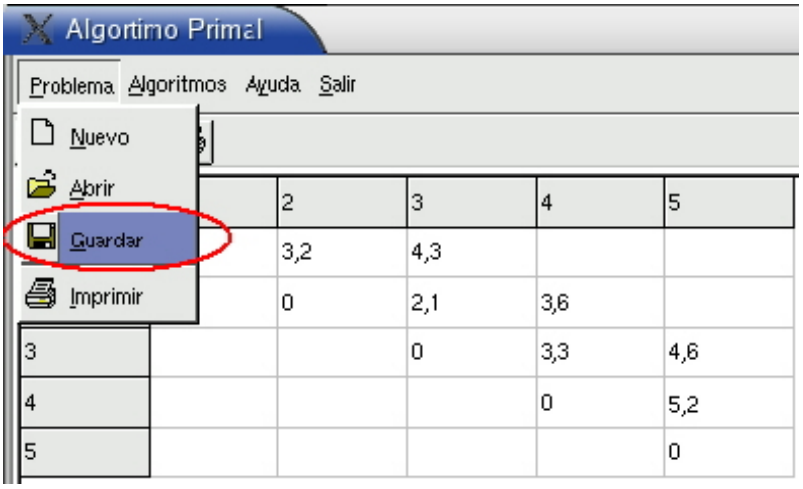


Figura A.9: Guardar Problema

A.3 Algoritmos

En el menú Algoritmos (Figura A.11), se muestran los algoritmos que han sido implementados en este programa. Los algoritmos estarán disponibles si el problema que se desea resolver cumple con las reestricciones siguientes: el algoritmo de Ford y Fulkerson sólo está disponible para grafos que tengan arcos con capacidades mayores que cero. El algoritmo de Floyd se puede aplicar a grafos con arcos cuyos costos sean positivos o negativos. Y el algoritmo de Flujo a costo mínimo acepta capacidades mayores que cero y costos mayores o iguales que cero. En caso de que el tamaño del problema sea muy grande y no sea posible su visualización en pantalla, se mandará a un archivo donde se concentrará toda la información correspondiente a la solución del problema.

Figura A.10: Menú Algoritmos

A.3.1 Ford y Fulkerson

El algoritmo de Ford y Fulkerson se puede aplicar a grafos que tengan arcos con una capacidad positiva. Este algoritmo resuelve el problema de flujo máximo en una red. Para aplicarle el algoritmo de Ford y Fulkerson a la red previamente cargada, se selecciona la opción Ford y Fulkerson del menú Algoritmos (Figura A.11).

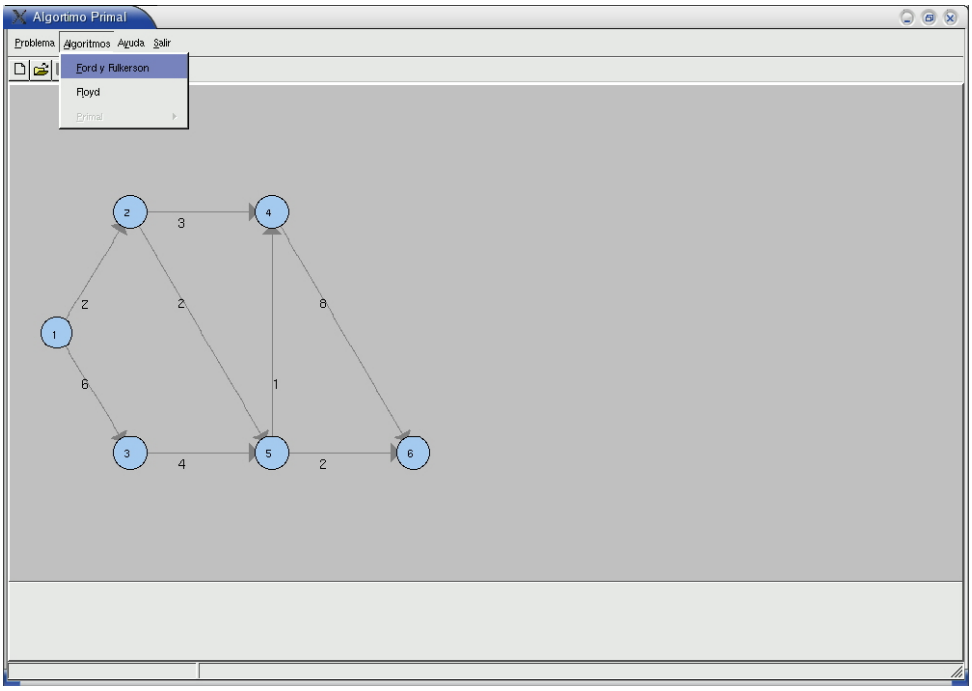


Figura A.11: Aplicar algoritmo de Ford y Fulkerson

Después de aplicar el algoritmo obtenemos la salida de la Figura A.12

Los arcos en color naranja nos indican por dónde es que pasa el flujo, y en la parte inferior de la pantalla se muestra el flujo máximo.

A.3.2 Floyd-Warshall

El algoritmo de Floyd-Warshall resuelve el problema de rutas más cortas, para el caso del algortimo Primal se utiliza para determinar la existencia de circuitos negativos en la red y se aplica a redes con valores positivos y negativos en sus arcos, al aplicar este algoritmo de forma directa como resultado se obtienen las rutas más cortas entre cada par de nodos y en caso de

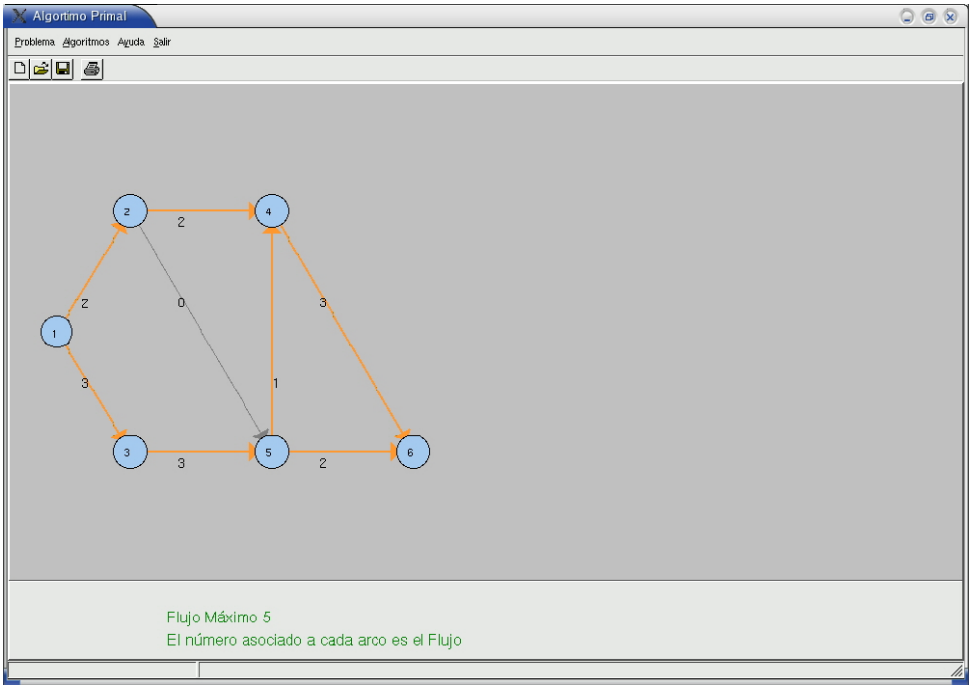


Figura A.12: Resultado obtenido al aplicar el algoritmo de Ford y Fulkerson

que el problema no tenga solución se muestra el circuito negativo. Este algoritmo se aplica seleccionando la opción Floyd del menú Algoritmos (Figura A.13).

y se obtiene como resultado:

Para determinar la ruta más corta entre algún par de nodos es necesario seleccionar en la parte inferior de la pantalla entre que nodos se desea saber la ruta más corta y presionar el botón Aceptar. En la pantalla se mostrará cúal es la ruta y en la parte inferior la longitud de dicha ruta.

En caso de haber introducido un grafo que no tenga solucion la salida será la siguiente:

Los arcos en color rojo muestran el circuito negativo que se encontró, y en el panel inferior de la pantalla se muestra la ruta de dicho circuito negativo..

A.3.3 Primal

El algoritmo Primal resuelve el problema de flujo a costo mínimo y se aplica a grafos en los que sus arcos tengan asociados dos valores, capacidad y costo. Las capacidades deben ser siempre

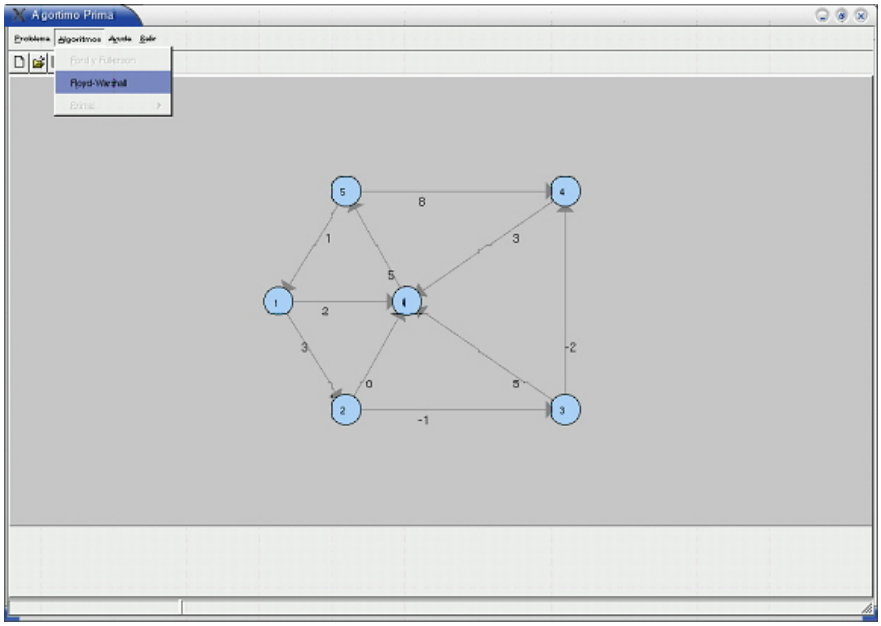


Figura A.13: Aplicar el algoritmo de Floyd-Warshall

mayores que cero, y los costos deben ser mayores o iguales que cero. Hay dos formas de correr este algoritmo: Por pasos y de forma directa.

a) Por pasos

Al correr el algoritmo de Flujo a Costo Mínimo nos pide el flujo deseado:

Al presionar el botón Aceptar, se calcula un flujo factible de este valor mediante el algoritmo de Ford y Fulkerson y se muestra la siguiente pantalla:

Para continuar presionamos el botón:que va a construir la red marginal asociada a esta red.

Nótese que en la parte inferior de la pantalla se despliega el número de iteración correspondiente, además del valor del flujo y su costo.

Al presionar el botón se aplica el algoritmo de Floyd-Warshall para detectar si existen circuitos negativos.

Nota: Al posicionar el mouse sobre un arco, se mostrará la informacion respectiva del arco (Figura A.23).

Para este ejemplo, observamos que se detectó un circuito negativo en el nodo 4 y dicho

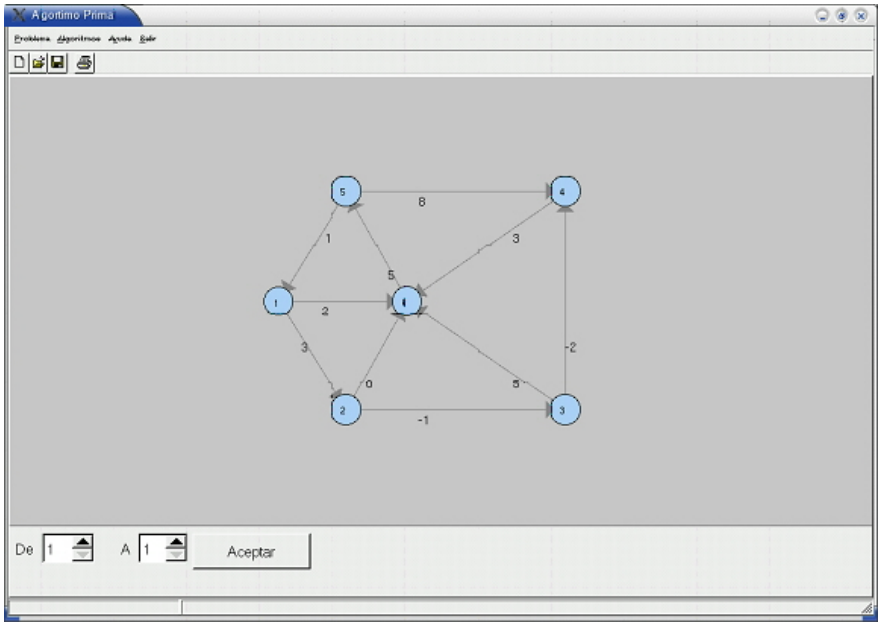


Figura A.14: Resultado obtenido después de aplicar el algoritmo de Floyd-Warshall

circuito es 4-3-1-2-4 que el programa nos lo menciona en el panel inferior con letras verdes. Este circuito lo podemos observar en la red con colores más resaltados.

Como en este caso existieron circuitos negativos, al presionar el botón
pasaremos a una nueva iteración. De esta forma se continúa hasta que ya no existan circuitos negativos. Y se obtiene el siguiente resultado:

Se observa en la parte inferior izquierda de la pantalla que para este ejemplo se necesitaron dos iteraciones para obtener el Flujo a Costo Mínimo.

b) Directo

Al correr el algoritmo de forma directa, obtenemos como resultado la pantalla de la Figura A.25.

A.4 Guardar resultados

Si se desean guardar los resultados obtenidos después de aplicar un algoritmo, se selecciona la opción Guardar del submenú Problema una vez que se aplicó el algoritmo. Los resultados son

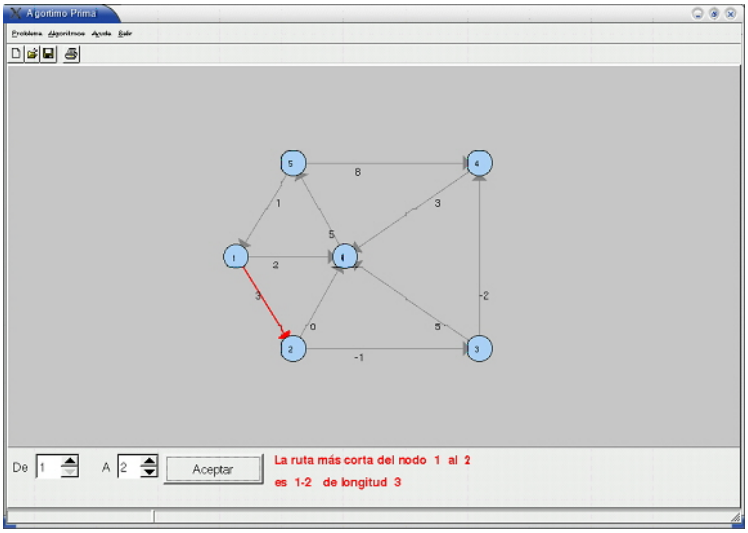


Figura A.15: Determinar la ruta más corta entre un par de nodos

guardados en un archivo de tipo *.txt de la siguiente forma:

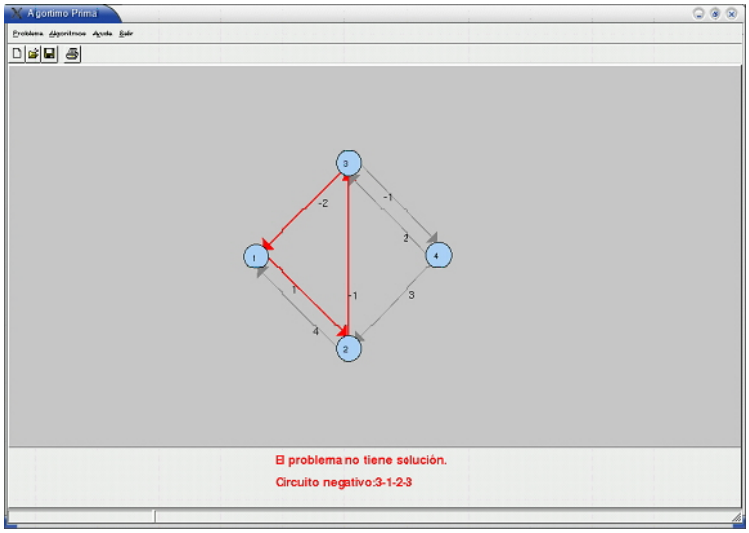


Figura A.16: El problema de rutas más cortas no tiene solución

Salida.txt

Grafo

Origen	Destino	Capacidad	Costo
1	2	3	2
1	3	4	3
2	3	2	1
2	4	3	6
3	4	3	3
3	5	4	6
4	5	5	2

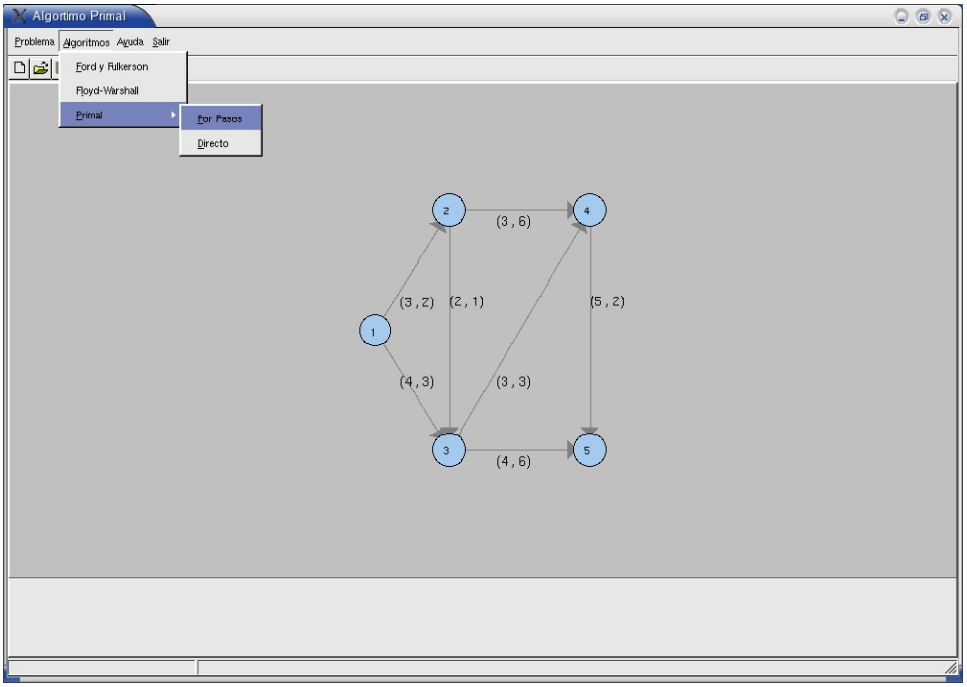


Figura A.17: Aplicar el algoritmo Primal por pasos

Origen	Destino	Flujo
1	2	2
1	3	3
2	3	2
2	4	0
3	4	3
3	5	2
4	5	3

El Flujo 5 a costo mínimo es 42

A.5 Imprimir pantalla

Se pueden imprimir las pantallas seleccionando la opción imprimir.

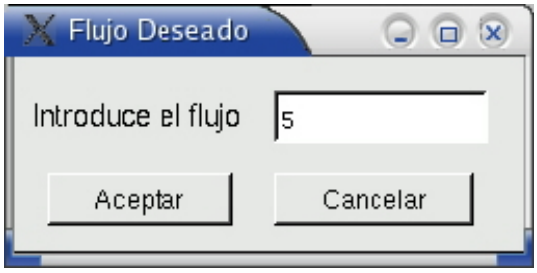


Figura A.18: Cuadro de diálogo para introducir el flujo deseado

A.6 Ayuda

La opción Ayuda nos muestra una ayuda sobre cómo utilizar este programa.

Y la opción Acerca despliega un cuadro de diálogo en el cual se muestra información acerca de los creadores de dicha aplicación.

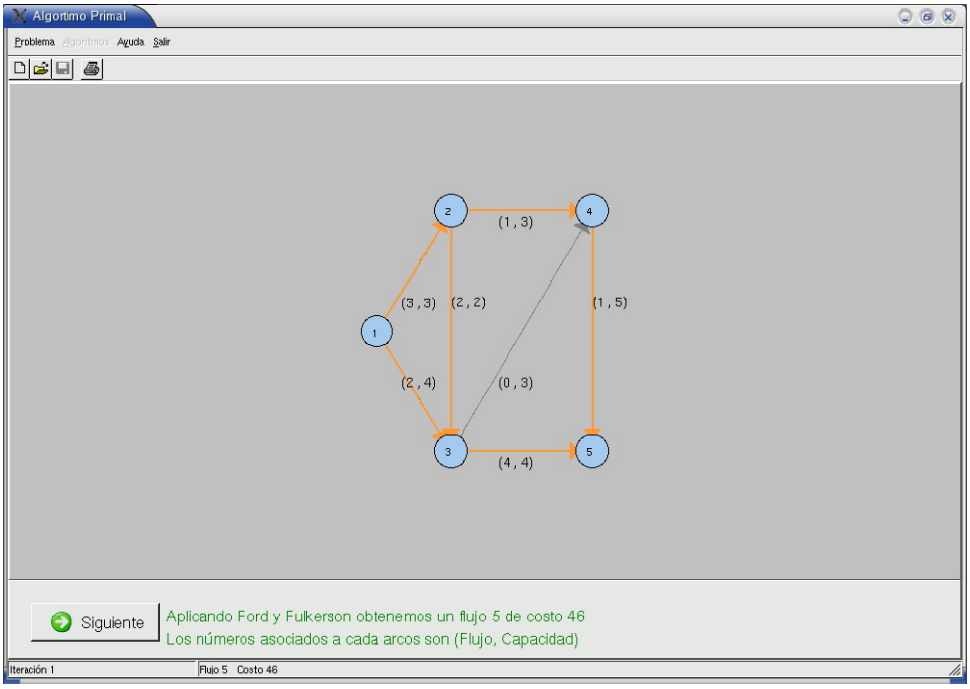


Figura A.19: Flujo factible para la red de la Figura A.15

Figura A.20: Botón Siguiente

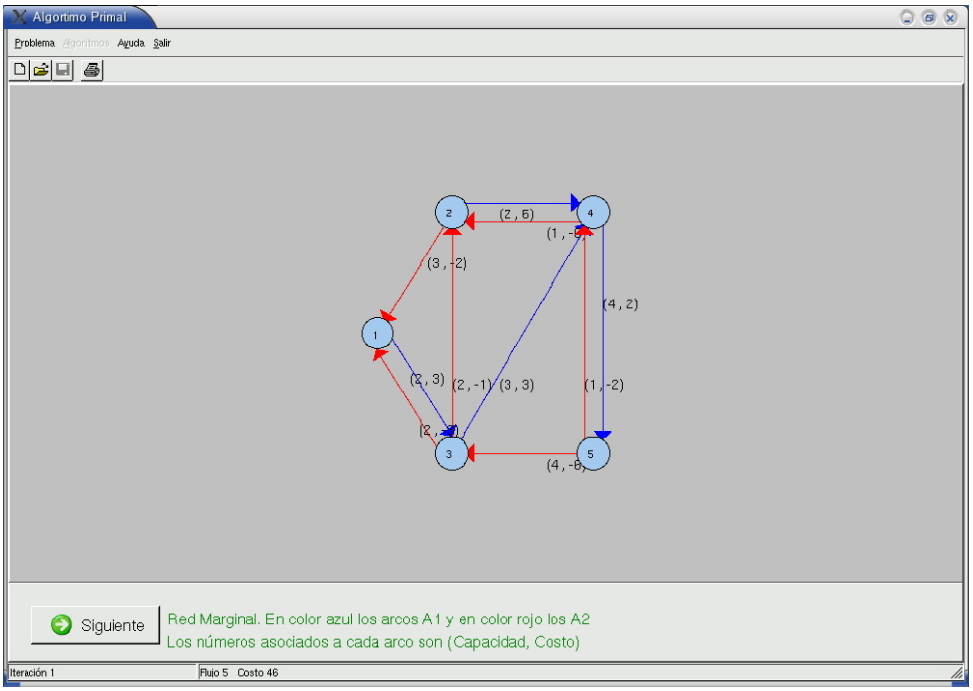


Figura A.21: Red marginal asociada a la red de la figura A.17

Figura A.22: Botón Siguiente

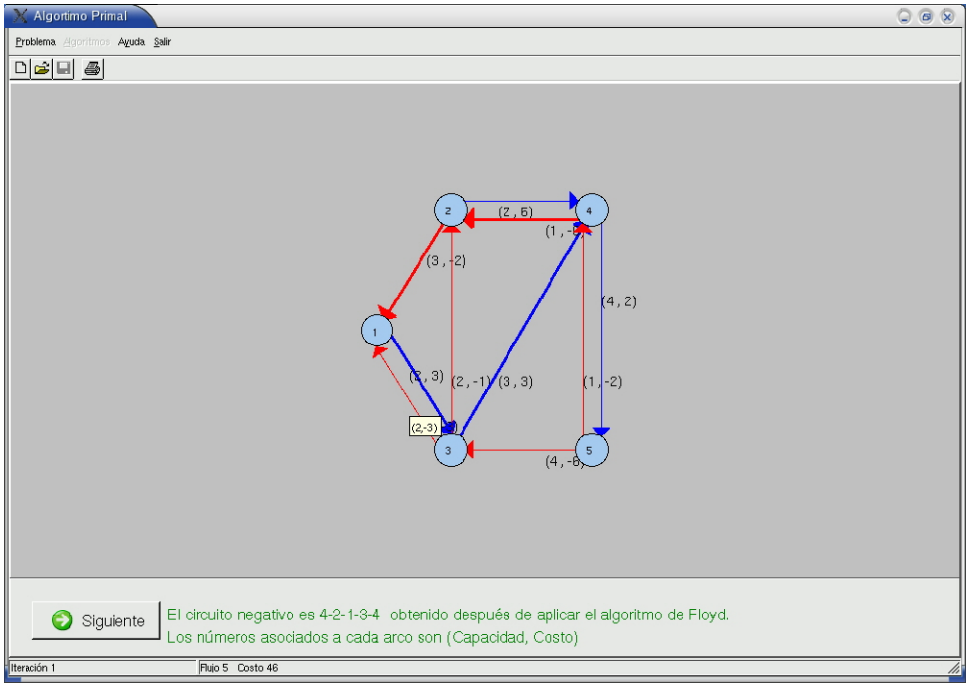


Figura A.23: Circuito negativo detectado en la red marginal de la Figura A.19

Figura A.24: Botón Siguiente

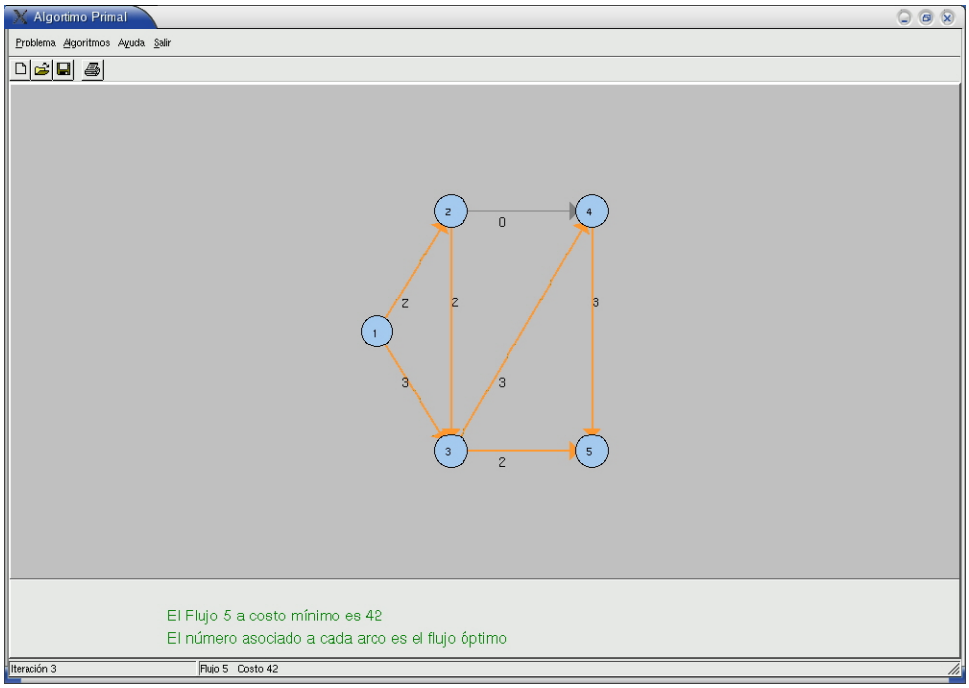


Figura A.25: Nuevo flujo factible

Figura A.26: Menú Ayuda

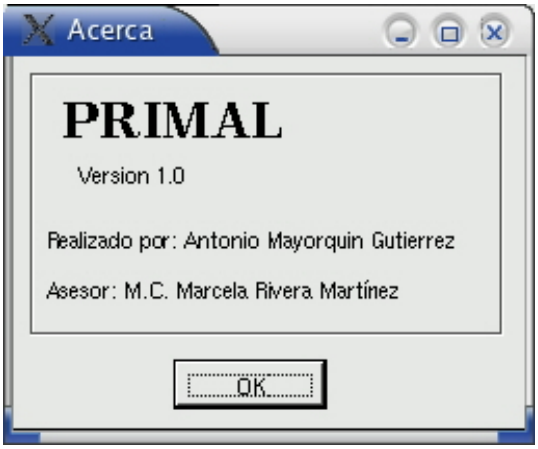


Figura A.27: Ventana Acerca

Apéndice B

Ejemplos

En este apéndice se muestran los ejemplos de cada algortimo instrumentado en esta tesis.

Primeramente se presenta un ejemplo del algoritmo de Ford y Fulkerson.

Ejemplo B.1 *Determínese el flujo máximo de s a t en la red de la Figura B.1 mediante el algoritmo de Ford y Fulkerson. El número asociado a cado arco representa su capacidad.*

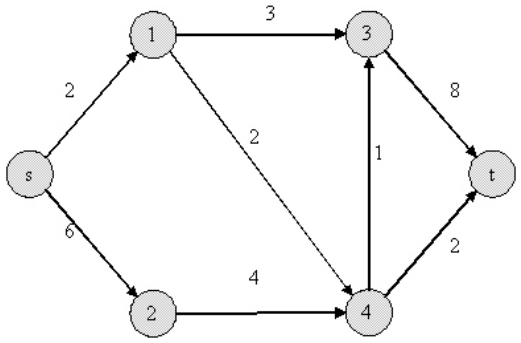


Figura B.1: Red para ejemplo de flujo máximo

Iteración 1. El la Figura B.2 se muestra el flujo obtenido en la primera iteración. Los números asociados a cada arco son el flujo a través de él y su capacidad.

Primeramente se etiquetó el nodo s, a partir de este nodo se buscan los sucesores i de s no etiquetados tales que $f_{si} < q_{si}$, estos sucesores son 1 y 2, ahora de entre los sucesores se

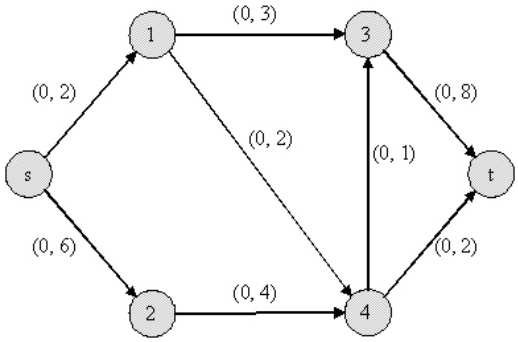


Figura B.2: Flujo inicial

busca el arco de menor capacidad o el arco que lleve al nodo destino. En este caso el de menor capacidad es el arco $(s, 1)$, por lo que se etiqueta el nodo 1.

Los sucesores i de 1 no etiquetados tales que $f_{si} < q_{si}$ son 3 y 4. El arco de menor capacidad es $(1, 4)$, por lo tanto se etiqueta el nodo 4.

Ahora estamos en el nodo 4, y los sucesores i de 4 no etiquetados tales que $f_{si} < q_{si}$ son 3 y t, como en este caso hay un arco que nos lleva al nodo destino se toma el arco $(4, t)$. Hemos terminado ya que se ha llegado al nodo destino t, ahora se actualiza el flujo de valor 2 a través de la cadena aumentante. Con este nuevo flujo se realiza otra iteración.

Iteración 2. En esta iteración se empieza con el flujo que se muestra en la Figura B.3, en ella se encuentra la cadena aumentante $s - 2 - 4 - 3 - t$ con una capacidad incremental de valor 1. Se actualizan los flujos a través de la cadena y se realiza otra iteración.

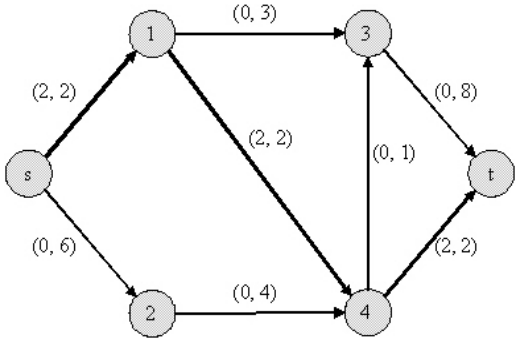


Figura B.3: Flujo obtenido después de la primera iteración

Iteración 3. Ahora empezamos esta iteración con el flujo mostrado en la Figura B.4, en esta iteración se encuentra la cadena $s-2-4-1-3-t$ con una capacidad incremental de valor 2. Se actualizan los flujos a través de los arcos y se realiza otra iteración.

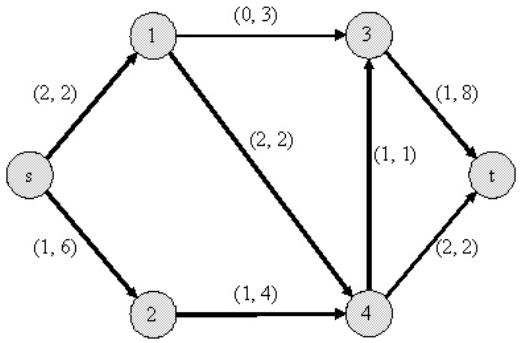


Figura B.4: Flujo obtenido después de la segunda iteración

Iteración 4. El flujo inicial para esta iteración se muestra en la Figura B.5, en esta iteración no se encuentra una cadena aumentante, por lo tanto se ha determinado un flujo máximo en la red.

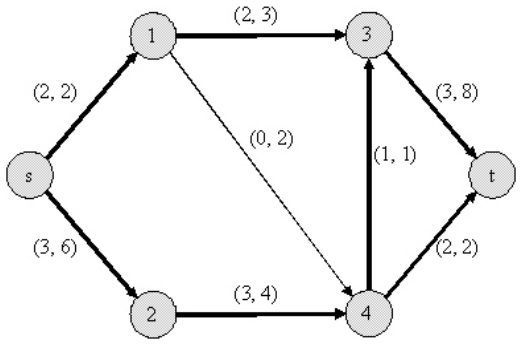


Figura B.5: Flujo obtenido después de la tercera iteración (óptimo)

A continuación se muestra un ejemplo para ilustrar el algoritmo de Floyd-Warshall.

Ejemplo B.2 *Determínese las rutas más cortas, entre todo par de nodos, en la red de la Figura B.6.*

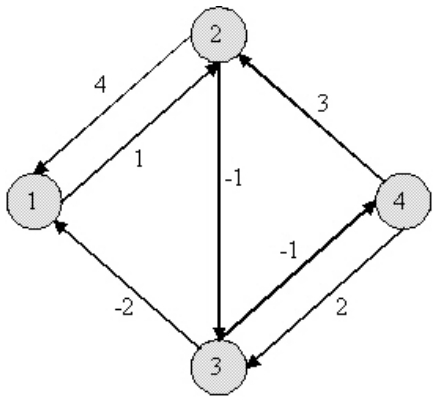


Figura B.6: Red para ejemplo de rutas más cortas entre todo par de nodos

Iteración 1.Se asigna $k = 1$ y se actualizan los elementos

$a_{22} = \min\{a_{22}, a_{21} + a_{12}\} = \min\{0, 4 + 1\} = 0.$

$a_{32} = \min\{a_{32}, a_{31} + a_{12}\} = \min\{\infty, -2 + 1\} = -1,$

como a_{32}, no existe, se agrega al grafo.

Iteración 2.Se asigna $k = 2$ y se actualizan los elementos

$a_{11} = \min\{a_{11}, a_{12} + a_{21}\} = \min\{0, 1 + 4\} = 0.$

$a_{13} = \min\{a_{13}, a_{12} + a_{23}\} = \min\{\infty, 1 - 1\} = 0,$

como a_{13}, no existe, se agrega al grafo.

$a_{31} = \min\{a_{31}, a_{32} + a_{21}\} = \min\{-2, -1 + 4\} = -2.$

$a_{33} = \min\{a_{33}, a_{32} + a_{23}\} = \min\{0, -1 - 1\} = -2.$

En este momento se termina de iterar, ya que se ha detectado un circuito negativo en el nodo 3.

Por último se presenta un ejemplo para mostrar el funcionamiento del algoritmo Primal.

Ejemplo B.3 *Determínese el flujo a costo mínimo de valor 5 en la red de la Figura B.7.*

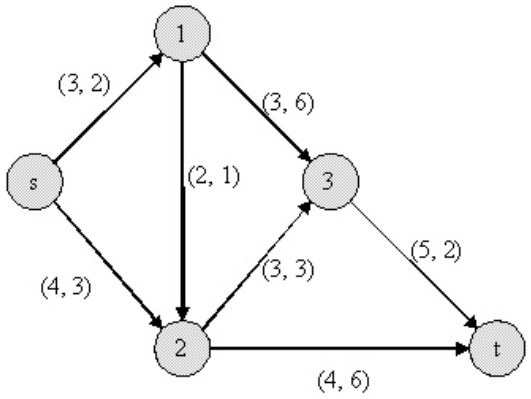


Figura B.7: Red para el ejemplo 3

Iteración 1. Se aplicará el algoritmo a partir de un flujo, de costo 44, determinado mediante el algoritmo de Ford y Fulkerson (Figura B.8). La red marginal con respecto a este flujo se muestra en la Figura B.9. Utilizando el algoritmo de Floyd, se determina el circuito negativo $1, s, 2, 3, 1$ de costo -2. De aquí se tiene $d = min\{q'_{1s}, q'_{s2}, q'_{23}, q'_{31}\} = 1$. Actualizando el flujo se obtiene el definido en la Figura B.10.

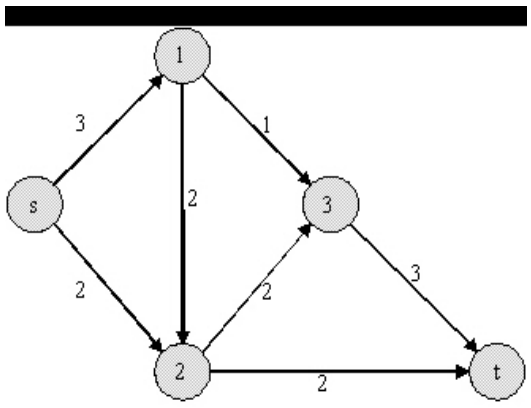


Figura B.8: Flujo de valor 5 de costo 44

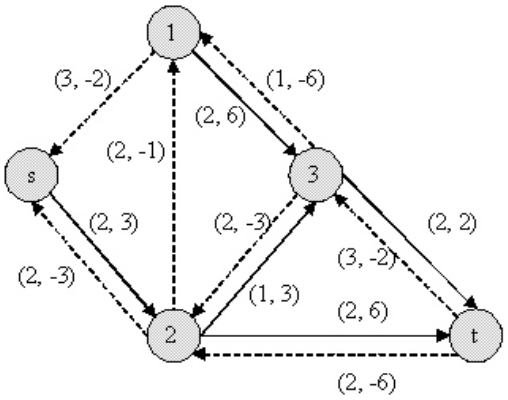


Figura B.9: Red marginal con respecto al flujo definido en la Figura B.8

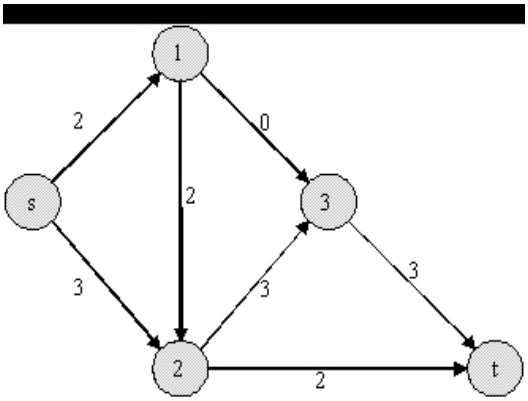


Figura B.10: Nuevo flujo de valor 5 de costo 42

Iteración 2. La red marginal con respecto a este nuevo flujo se muestra en la red de la Figura B.11. Al utilizar el algoritmo de Floyd se observa que en la red marginal no existen circuitos negativos por lo que el flujo de costo 42 definido en la Figura B.10 es el flujo a costo mínimo de valor 5.

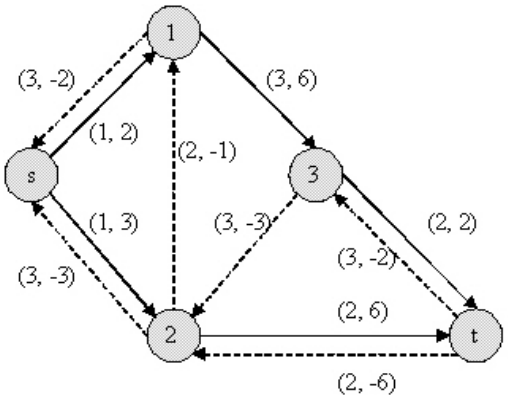


Figura B.11: Red marginal con respecto al flujo definido en la Figura B.10

Apéndice C

Ejemplos prácticos

En este apéndice se presentan problemas del mundo real en los cuales se aplican los algoritmos instrumentados en esta tesis para resolverlos.

Problema C.1 *El arbitraje consiste en negociar de una moneda a otra esperando aprovechar las pequeñas diferencias en índices de conversión entre varias monedas para alcanzar un beneficio. Por ejemplo, si \$1,00 en moneda estadounidense compra 0,7 libras británicas, £1 libra británica compra 9,5 francos franceses, y 1 franco francés compra 0,16 dólares estadounidenses, entonces un negociante puede comenzar con \$1,00 y ganar* $1 * 0.7 * 9.5 * 0.16 = 1.064$ *dólares obteniendo así un beneficio de 6,4 por ciento.*[11]

Este problema se puede resolver mediante el problema de rutas más cortas aplicando el algoritmo de Floyd-Warshall. Primeramente necesitamos transformar este problema en un problema de rutas más cortas, tomamos como ejemplo el siguiente problema:

	1	2	3	4
1	1	3.1	0.0023	0.35
2	0.21	1	0.00353	8.13
3	200	180.559	1	10.339
4	2.11	0.089	0.06111	1

Tabla C.1: Ejemplo para el problema de arbitraje

[11] *Problema tomado del concurso internacional de programación de la ACM.*

Primeramente debemos convertir este problema en un problema de rutas más cortas, para esto utilizamos las propiedades de los logaritmos de tal manera que para cada arco con un costo c_{ij} construimos un nuevo costo $c'_{ij} = -\log(c_{ij})$ y obtenemos lo siguiente:

	1	2	3	4
1	1	-.4913	2.6382	-1.5440
2	0.6777	1	2.4522	-0.9100
3	-2.3010	-2.2566	1	-1.0144
4	-0.3242	1.0506	1.2138	1

Tabla C.2: Problema modelado como uno de rutas más cortas

Convertimos los valores de los arcos a enteros

	1	2	3	4
1	1	-4913	26382	-15440
2	6777	1	24522	-9100
3	-23010	-22566	1	-10144
4	-3242	10506	12138	1

Tabla C.3: Valores convertidos a enteros

y obtenemos la siguiente red:

Y, corriendo el algoritmo de Floyd-Warshall de nuestro programa obtenemos:

Lo cual nos indica que se ha encontrado un circuito negativo en el nodo 4, dicho circuito es 4-1-4. Para este problema, significa que al hacer un cambio entre las monedas del país 4 y del país 1 se obtendría un beneficio.

Problema C.2 *El instituto mexicano del café desea exportar café a Francia. Las disponibilidades (en toneladas) de los puertos mexicanos son: Veracruz 120, Tampico 100, Tuxpan 100, Campeche 100. Las demandas en Francia son: Dunkerque 100, Burdeos 80, St. Nazaire 90, Le Havre 150. Los barcos que van de los puertos del Golfo a los puertos franceses tienen las siguientes capacidades*[12]*:*

[12] *Problema tomado de [AY97]*

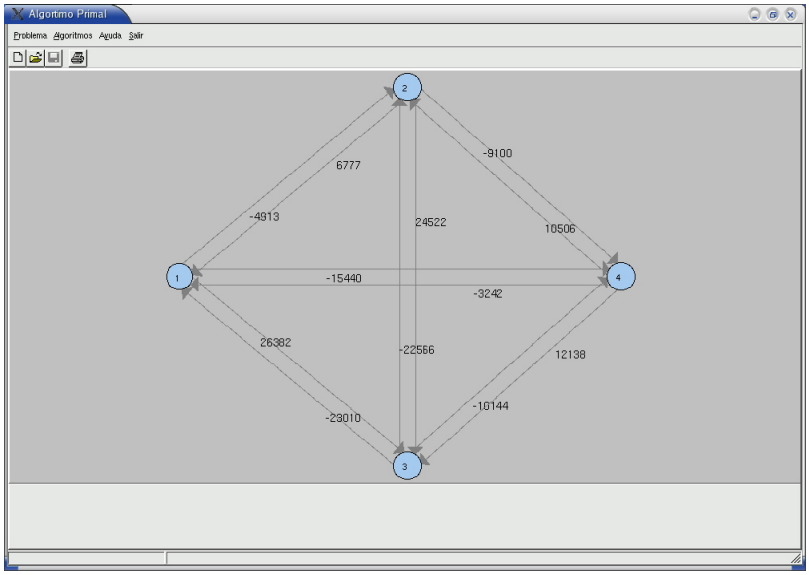


Figura C.1: Red para el Problema C.1

	Dunkerke	***Burdeos***	***St. Nazaire***	***Le Havre***
Veracruz	*70*	*30*	*20*	*0*
Tampico	*50*	*40*	*10*	*0*
Tuxpan	*0*	*20*	*40*	*80*
Campeche	*0*	*20*	*40*	*80*

Tabla C.4: Tabla de capacidades para el problema C.2

Este problema es de flujo máximo en una red. Primero modelamos el problema como una red de la siguiente manera:

Donde:

1: Fuente, 2: Veracruz, 3: Tampico, 4: Tuxpan, 5: Campeche, 6: Dunkerke, 7: Burdeos, 8: St. Nazaire, 9: Le Havre, 10: Destino.

Ahora, corriendo el programa obtenemos el siguiente flujo máximo:

La salida mostrada al correr al algoritmo de Ford y Fulkerson nos proporciona un flujo máximo de 380, como se puede observar en la Figura C.4

Problema C.3 *Una compañía de aerolíneas puede comprar gasolina a tres proveedores. Estos disponen de 2, 6 y 6 millones de galones respectivamente. El precio por millón de galones de gasolina entregada a cada una de las tres localidades en donde la compañía la necesita, así como*

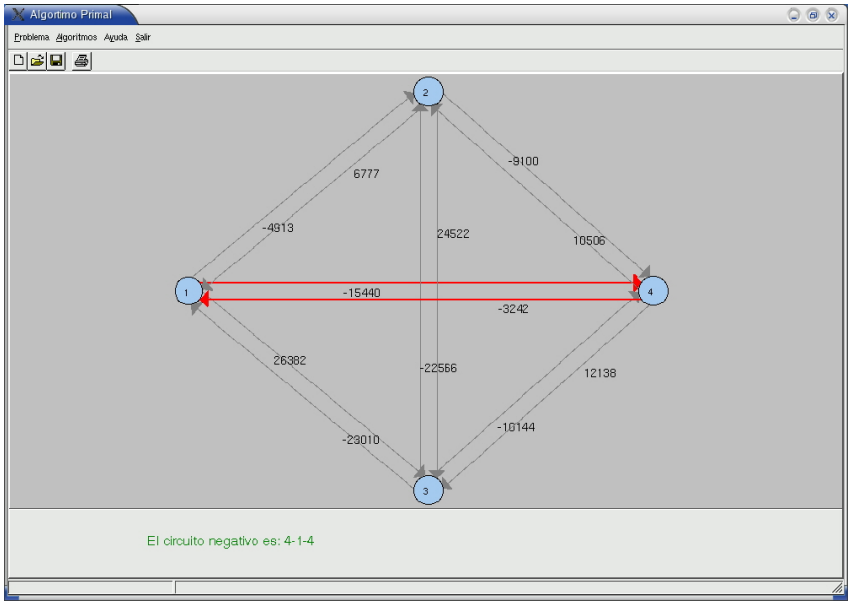


Figura C.2: Resultado del Problema C.1

la demanda es[13]*:*

	Localidad 1	***Localidad 2***	***Localidad 3***
Proveedor 1	*2*	*3*	*1*
Proveedor 2	*4*	*2*	*5*
Proveedor 3	*1*	*8*	*9*
Demanda	*5*	*3*	*2*

Tabla C.5: Tabla de precios para el problema C.3

Se desea determinar el plan de compra más barato.

El problema formulado como una red es el siguiente:

En este caso como no se menciona la capacidad que tiene para enviar de cada proveedor a cada localidad, se supone una capacidad infinita, pero en este caso se decidió poner una capacidad de 20 a cada arco. El problema formulado como una red es el siguiente:

Y, después de correr el algoritmo Primal del programa se obtiene:

El costo de compra más barato es 13, las rutas son 1-2-7-8, 1-3-6-8, 1-4-5-8 como se puede observar en la Figura C.6.

[13] *Problema tomado de [AY97]*

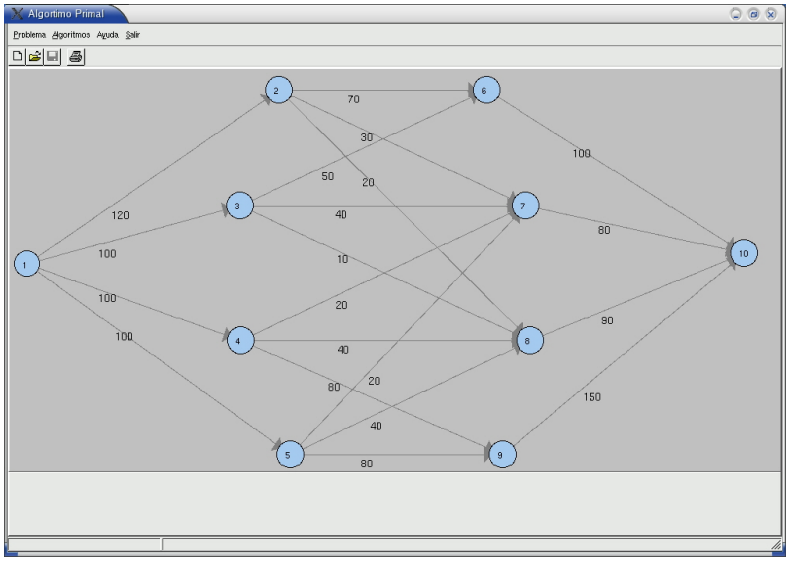


Figura C.3: Red para el Problema C.2

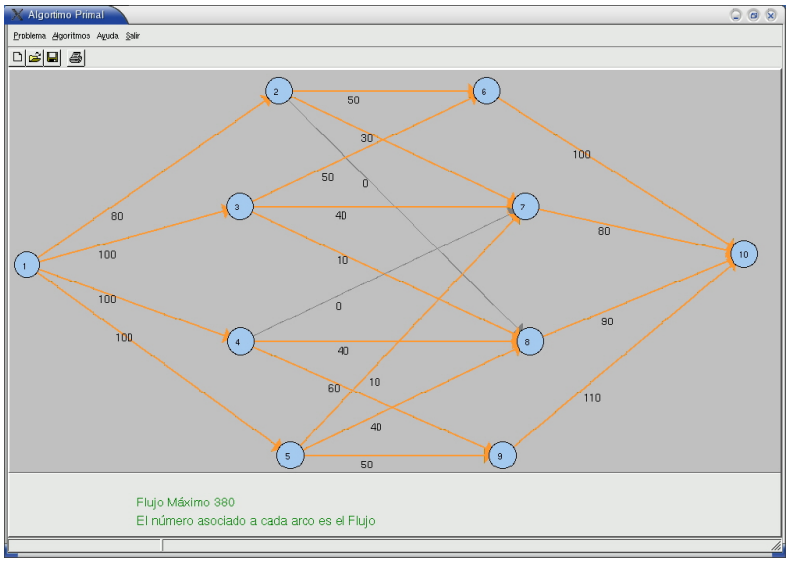


Figura C.4: Resultado del Problema C.2

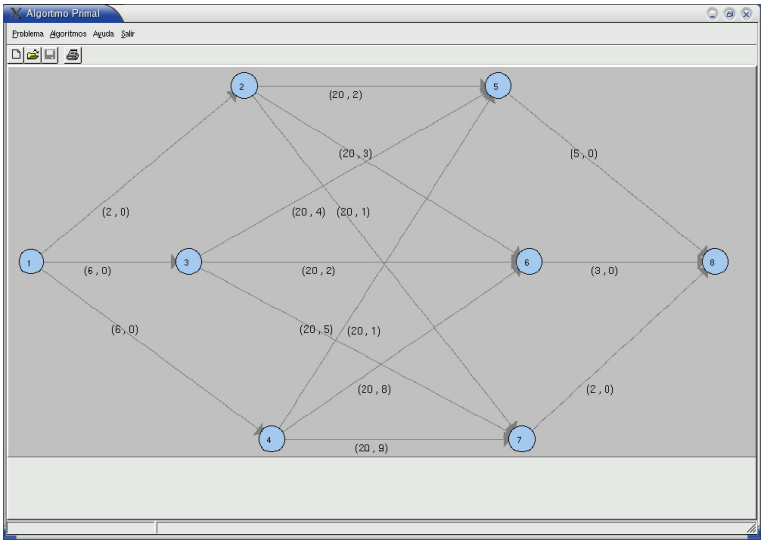


Figura C.5: Red asociada al Problema C.3

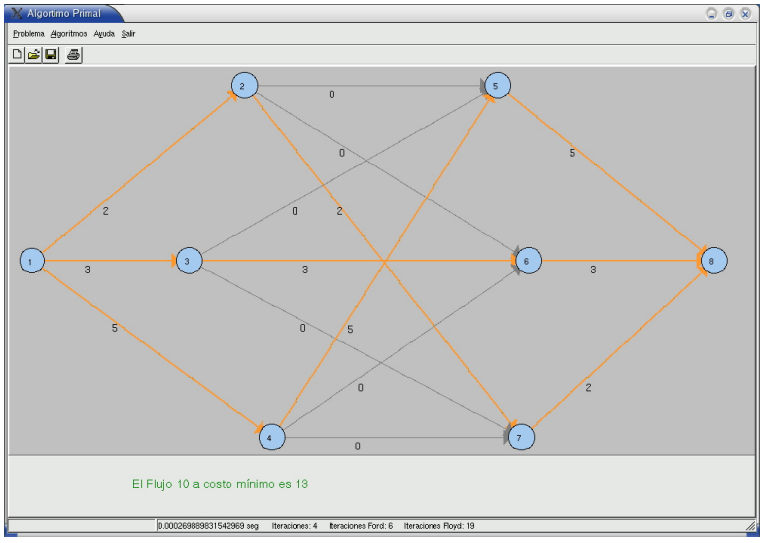


Figura C.6: Resultado del Problema C.3

Bibliografía

[NE96] NEOS SERVER (1996). *NEOS Guide Optimization Tree.* Recuperado Julio 4, 2004 de http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/index.html

[AY97] Ma. C. Hernández Ayuso (1997). Introducción a la teoría de redes. México: Sociedad Matemática Mexicana.

[CO01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein (2001). Introduction to Algorithms. USA: The MIT Press.

[TE93] Aaron M. Tenenbaum, Yedidyah Langsam, y Moshe A. Augenstein (1993). Estructuras de datos en C. México: Prentice-Hall Hispanoamericana.

[BA99] Giusepp D' Battista, Peter Eades, Roberto Tamassia y Ionnis G. Tollis (1999). Graph Drawing: Algorithms for the Visualization of Graphs USA: Prentice Hall

[CA02] Enrique Castillo, Antonio J. Conejo, Pablo Pedregal, Ricardo Garcia y Natalia Alguacil (2002). Formulación y Resolución de Modelos de Programación Matemática en Ingeniería y Ciencia. Universidad de Castilla-La Mancha. [En Linea] Disponible en: http://departamentos.unican.es/macc/personal/profesores/castillo/Libro.htm