Universidad de Sonora
Departamento de Matemáticas

Tesis

Una Implementación Computacional del Algoritmo de Ford-Fulkerson y sus Generalizaciones

Que para obtener el título de

Licenciado en Matemáticas

Presenta

Irene Rodríguez Castillo

Hermosillo, Sonora, 1 de Octubre de 1993

A mis padres, mis hermanos, a mi tia Julieta,...

a Rafael,...

a Yolanda.

Agradecimientos

Agradezco profundamente a mi padre por la confianza que ha depositado en mí y a mi madre por su constante apoyo, así como a mi maestro y director de este trabajo Pedro Flores Pérez porque trabajó conjuntamente conmigo en él.

Contenido

Introducción

A raíz del enorme crecimiento de las ciudades, la industria y el comercio se han presentado problemas tales como: control de tráfico, optimización de procesos industriales, planeación de proyectos, etc., que requieren solución.

Imaginemos lo que pasaría en una ciudad con un elevado número de habitantes si el tráfico en ésta no se controlara debidamente; se llagaría a una situación caótica, por lo tanto es importante resolver estos problemas.

Una de las técnicas usadas para ayudar a resolver este tipo de problemas es la modelación de los mismos en una red y se han elaborado algoritmos que resuelven un problema específico en ella, los cuales han resultado ser muy eficientes.

En nuestra región, existe poca difusión de esta técnica de solución y más aún existe poca paquetería comercial disponible de los algoritmos de flujo en redes que además resulta ser de un costo muy elevado, de aquí que cualquier contribución al respecto será bien recibida. Por lo tanto, el objetivo del trabajo aquí presentado es apoyar a la difusión de los algoritmos clásicos de flujo en redes, más concretamente, al algoritmo para resolver los problemas de flujo máximo y mínimo en una red e implementar computacionalmente dicho algoritmo, así mismo, pretendemos apoyar a los futuros estudiantes de flujo en redes con el material y el software desarrollado.

El trabajo está dividido en tres capítulos donde se desarrolla el material teórico y un único anexo que presenta el programa de la implementación del algoritmo para resolver el problema de flujo máximo en una red y la manera de ejecutarlo.

En el primer capítulo se presentan ejemplos de problemas en los cuales se requiere encontrar el flujo máximo en la red que los modela y la teoría básica para el desarrollo y justificación de un método de solución al problema, el cual da pie al algoritmo utilizado y presentado en este mismo capítulo.

En el capítulo 2 presentamos variantes del problema de flujo máximo en una red, así como el procedimiento a seguir para solucionar cada una de estas variantes. Al resolver un problema planteado como una de las variantes de flujo máximo, encontraremos un método general para conocer una solución factible en una red. Al final del capítulo se resuelve un ejemplo de un problema donde se presentan las variantes de flujo máximo.

En el tercer y último capítulo expondremos la teoría del problema de flujo mínimo haciendo

analogías con el problema de flujo máximo y justificamos teóricamente un método de solución que puede deducirse de manera natural de uno de los teoremas, en este capítulo presentados. Demostramos un teorema que nos permite utilizar herramientas de la teoría del problema de flujo máximo, específicamente, el algoritmo, para resolver el problema de flujo mínimo en una red, evitándonos el trabajo de implementar para éste el método de solución.

En los capítulos 1 y 2 se presentan las estructuras de datos utilizadas para la implementación del algoritmo que resuelve el problema de flujo máximo y sus variantes. Para mayor información sobre estructura de datos puede verse [4].

Capítulo 1

El Problema de Flujo Máximo

Un problema de Flujo máximo en una red consiste en encontrar la mayor cantidad de flujo que se pueda mandar desde ciertos lugares llamados fuentes a otros llamados destinos a través de conductos que tienen restricciones respecto al flujo que pueden transportar.

En este capítulo se presentarán ejemplos de este tipo de problemas, planteándolos en una red y especificando las restricciones que pueden aparecer en nodos o en arcos, según sea el caso.

El problema de Flujo máximo está intimamente relacionado con otro, llamado el problema de la Cortadura mínima en una red, ya que al encontrar nosotros la solución a uno de los problemas se encuentra también la solución del otro. Dadas las definiciones de una cortadura y su capacidad se hará más clara la idea del problema de Cortadura mínima en una red que consiste en encontrar la manera más "barata" (en terminos de su capacidad) de "desconectar" los nodos fuentes y los destinos, logrando que no se encuentre camino entre estos nodos.

Esta teoría será cubierta también en este capítulo donde además, presentaremos el algoritmo de *Ford-Fulkerson* para resolver este tipo de problemas, así mismo, justificaremos teóricamente la convergencia de dicho algoritmo y presentaremos las estructuras de datos usadas para su implementación. Por último se resolverá un problema en una corrida de escritorio utilizando estas estructuras de datos.

1.1 Planteamiento de problemas

Los siguientes problemas fueron tomados de [3] y podrán resolverse después de analizar la teoría presentada en este trabajo.

En el primero de estos problemas se requiere encontrar el flujo máximo en la red que lo modela. En el segundo, al encontrar el flujo máximo en la red, se genera un conjunto de arcos que tienen la característica de que al ser removidos de la red, es imposible que pase flujo de los nodos fuente a los destino. Este conjunto de arcos es el que constituye la cortadura mínima, de la cual se darán más detalles más adelante.

1.1.1 El Problema del Pintor

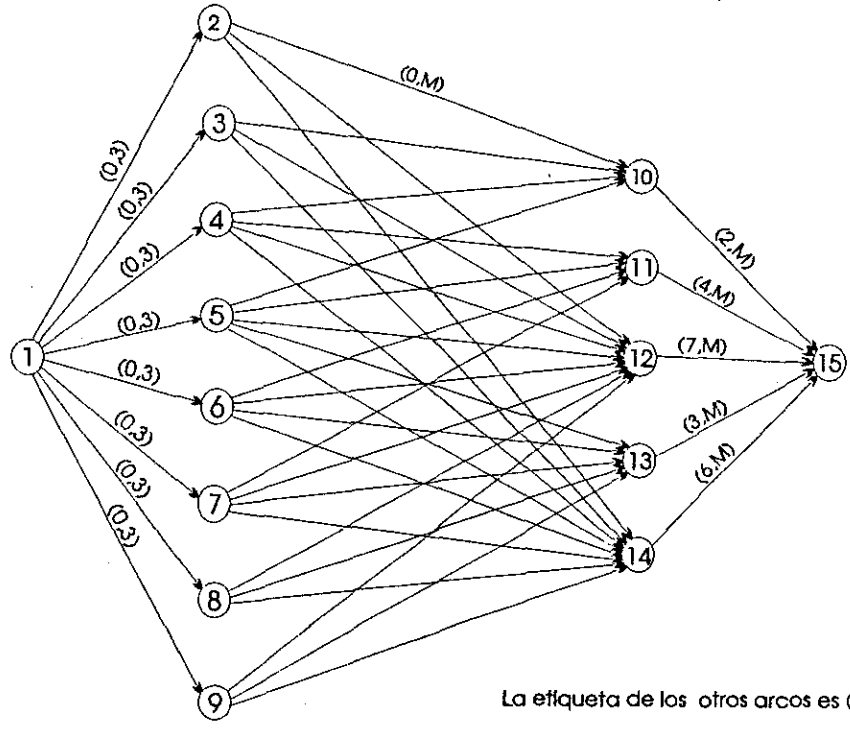
La familia Romero pinta casi todas las casas de Hermosillo, Sonora. Dicha familia es tan grande que hay tres equipos de trabajadores disponibles durante los meses de primavera y verano. Eduardo Romero, el patriarca de la familia y cabeza de la compañía, fué contratado para realizar cinco trabajos durante esta estación. Cada trabajo puede ser empezado inmediatamente o después de cierta fecha y debe ser completado en un cierto tiempo establecido de antemano.

El señor Romero estima el tiempo requerido en semanas, para cualquiera de los trabajos y la información para cada trabajo se dá a continuación.

Trabajo	Dia de inicio	Dia de terminación	Tiempo estimado (semanas)
A	Abril 1	Abril 28	2
B	Abril 14	Mayo 12	4
C	Abril 1	Mayo 26	7
D	Abril 21	Mayo 26	3
E	Abril 1	Mayo 26	6

El señor Romero ha hecho una práctica el asignar a sus equipos de forma tal que pueden trabajar solo en un trabajo. Así dos equipos no están realizando el mismo trabajo al mismo tiempo, por lo tanto podemos ver a un equipo de trabajadores como una semana de realización del trabajo al que se le asigne.

Usando estos datos el señor Romero quiere saber si es posible terminar todos los trabajos en el tiempo estimado. La siguiente figura da una red que puede ser usada para responder esta pregunta.



La etiqueta de los otros arcos es (0,M).

Los nodos del 2 al 9 representan cada una de las semanas entre Abril 1 y Mayo 26. Los nodos del 10 al 14 representan los trabajos de A a E respectivamente. Los nodos 1 y 15 son el principio y el fin para el problema de flujo máximo modelado. Los arcos originados del nodo 1 tienen todos capacidad 3, reflejando el hecho de que tres equipos de trabajo son disponibles en cada semana. Los arcos que terminan en el nodo 15 tienen todos capacidad igual al número de semanas requeridos para completar el trabajo asociado con el nodo orígen de cada arco. El flujo en esta red puede verse como semanas de trabajo realizado incluso en los arcos originados del nodo 1, ya que dos equipo no están realizando el mismo trabajo al mismo tiempo como ya mencionamos. M denota un número muy grande usado en arcos con flujo no acotado.

Nótese que las semanas 2 y 3 están ligadas solamente a empleos A, C y E, la semana 4 está ligada a todos excepto al empleo D, y las semanas de la 5 a la 9 están ligadas a todos los empleos.

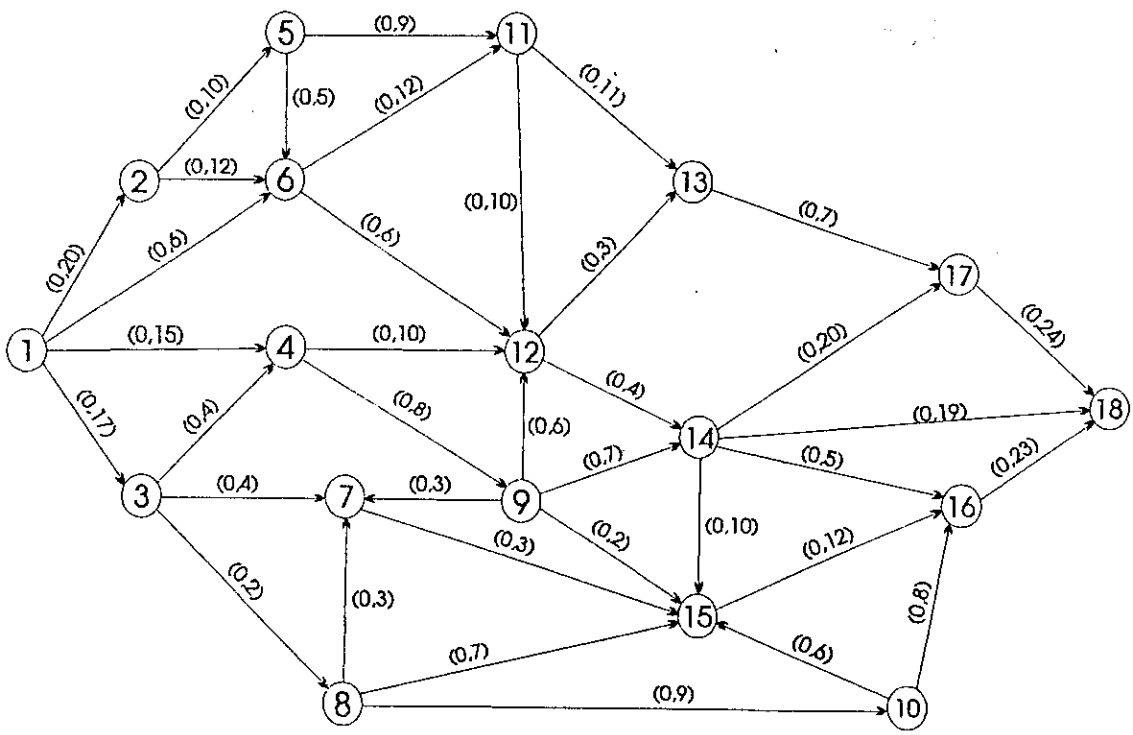
Si es posible finalizar todos los empleos a tiempo, el flujo máximo en la red será igual a 22; esto es, todos los arcos que terminan en el nodo 15 tendrán flujos igual a sus capacidades.

1.1.2 La Eliminación de la Plaga de Maíz

El Servicio de Control de Plaga de México tiene que evitar la migración de la plaga de maíz de este año para evitar que llege a los graneros y con ésto sufrir una severa pérdida económica.

En años anteriores se han mapeado extensivamente los caminos de migración de la plaga y se ha determinado que el costo de fumigación de cada camino es proporcional a la fuerza de la plaga que use el camino. Usando esta información, el servicio desea determinar cuáles son los caminos que deben fumigar para evitar que la plaga llege a los graneros y además con el menor costo posible.

La representación esquemática de los caminos de migración esta dada en la red de la siguiente figura que será usada para resolver el problema.



En la figura los nodos son puntos de unión para los arcos, los cuales representan la migración de la plaga; los nodos 1 y 18 son el principio y el fin de la migración respectivamente. la capacidad de los arcos es el máximo de individuos en miles que se espera que usen un camino en particular. Estas capacidades son tomadas de datos históricos.

La cortadura mínima en la red anterior es el conjunto de arcos que al ser removidos de la ella ya no se encuentra camino alguno entre el nodo 1 y el 18, donde la suma de las capacidades máximas de esos arcos es mínima, por lo tanto el servicio de plaga debe encontrar la cortadura mínima en la red ya que si los arcos de la cortadura son fumigados extensamente se puede evitar que la plaga llege al nodo 18, y la capacidad de cada arco de la cortadura corresponde al costo su fumigación el cual se requiere que sea lo más barato posible.

Al final de este trabajo, en un anexo, se resolverán los dos problemas que hemos presentado.

El las siguientes secciones se verá un método "natural" para encontrar solución a problemas de flujo máximo y al mismo tiempo se encuentra solución al problema de cortadura mínima en la red que modela los problemas. El método natural del que hablamos, es la idea principal del algoritmo de *Ford-Fulkerson*, tema central de este trabajo, y que consiste a grandes rasgos en encontrar "conductos" que llamaremos cadenas aumentantes, en la red por los cuales se pueda mejorar el flujo existente en ella.

Se verá también la teoría en la que se basa el algoritmo mencionado, demostrando que, bajo ciertas condiciones, se llega efectivamente al óptimo flujo en la red.

1.2 Planteamiento teórico del problema

Supongamos que se tiene el problema de mandar la mayor cantidad posible de cierto producto desde una ciudad productora "s" a una ciudad demandante "t" pasando posiblemente por otras ciudades de "paso" (ciudades que no son productoras ni demandantes) y que en las rutas a utilizar existe una capacidad máxima de transporte de ese producto.

La red asociada a este tipo de problema la denotaremos por $R = [X, A, q]$ donde:

- X representa el conjunto de ciudades (nodos). Supondremos que la cardinalidad de X es n.
- A representa el conjunto de caminos (arcos) entre las ciudades entre las cuales es posible transportar el producto, es decir,
  $$A = \{(i,j)/i,j \in X, \text{es posible mandar producto de } i \text{ a } j\ \}$$
  Denotaremos la cardinalidad de A por m.
- q es una función $q : A \to \mathcal{Z}$ donde a cada $(i,j) \in A$ le asocia la capacidad máxima de transporte de i a j. A q le llamaremos función de capacidad.

Si las capacidades máximas en los arcos son reales, el algoritmo utilizado para resolver el problema, no converge al óptimo flujo en la red. Al final del capítulo presentaremos un ejemplo donde

se muestra lo antes mencionado. Así pués, nos remitiremos a usar enteros para las cotas máximas en los arcos, ya que si algunos tienen cota racional se pueden convertir a enteros multiplicandolas por el mínimo multiplo de diez tal que todas sean enteras. Tomaremos este criterio, aún para racionales con expansión decimal infinita, como 1/3, ya que, al final de cuentas, la precisión de las computadoras no permiten trabajarlos con toda su expansión.

Representaremos por v la cantidad de flujo en la red del nodo "s" al nodo "t" y por f_{ij} la cantidad de flujo del nodo i al nodo j. Entonces nuestro problema consiste en:

Maximizar v

sujeta a:

1.
$$\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{k\in\Gamma^-(i)} f_{ki} = \begin{cases} v & \text{si } i = s \\ 0 & \text{si } i \neq s \text{ ó } i \neq t \\ -v & \text{si } i = t \end{cases}$$

2. $$0 \leq f_{ij} \leq q(i,j) \text{ para todo } (i,j) \in A$$

a los nodos s y t se le llama nodo fuente y destino respectivamente, $\Gamma^+(i)$ es el conjunto de nodos sucesores de i y $\Gamma^-(i)$ es el conjunto de nodos antecesores de i. A 1. se le conoce como "ecuaciones de conservación de flujo" y su lado izquierdo es la cantidad de flujo que sale menos la cantidad de flujo que entra a cada nodo, que toma los valores que se indican a su derecha. La ecuación (2) significa que el flujo de cada arco debe ser menor o igual a la capacidad máxima de flujo permitida para él. Un flujo factible en una red se define de la siguiente manera:

Definición: Un flujo factible en una red $R = [X, A, q]$ es una función $f : A \rightarrow \mathcal{Z}$ que cumple las condiciones 1. y 2. escritas arriba. Se dice que f es máximo cuando genera el mayor valor posible de v.

Para exponer el método para solucionar este problema son necesarios los siguientes conceptos:

Definición: Sea $R = [X, A, q]$ una red. Llamaremos cadena a una sucesión de la forma "nodo-arco-nodo" denotada por C y representada por $C = \{i_1, a_1, i_2, a_2, ..., a_k, i_{k+1}\}$ donde cada nodo i_j está conectado por el arco a_j al nodo i_{j+1}.

Observemos que si f es un flujo factible en R y $C = \{s = i_1, a_1, i_2, a_2, ..., a_k, i_{k+1} = t\}$ una cadena desde s a t, los arcos de C se pueden separar en dos subconjuntos F (del inglés Forward: hacia adelante) y B (del inglés Backward: hacia atrás), tales que:

$$a_j \in F \Leftrightarrow a_j = (i_j, i_{j+1})$$

es decir, los arcos que están en F son los arcos de C que tienen el sentido de s a t y

$$a_j \in B \Leftrightarrow a_j = (i_{j+1}, i_j)$$

los arcos que están en B son los arcos de C que tienen el sentido de t a s.

Denotemos por $q_{ij} = q(i,j)$ la capacidad máxima del arco (i,j), es decir, el máximo flujo que puede soportar dicho arco.

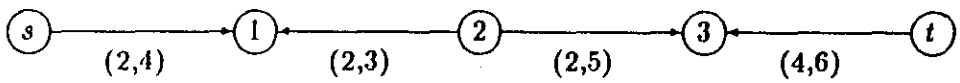
Definición: Sea $R = [X, A, q]$ una red y f un flujo factible en ella. Una cadena de s a t es aumentante si $f_{ij} < q_{ij}$ para todo $(i,j) \in F$ y $f_{ij} > 0$ para todo $(i,j) \in B$.

Ejemplos:

En las siguientes cadenas de s a t la pareja de números asociada a cada arco es (f_{ij}, q_{ij}).

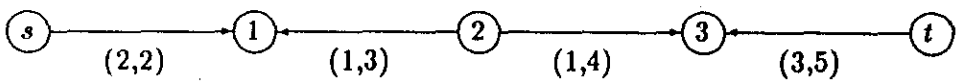
Sean $F = \{(s,1),(2,3)\}$ y $B = \{(2,1),(t,3)\}$ en todas las cadenas.

1.



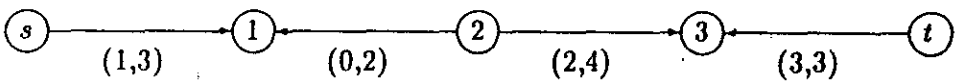
Esta cadena es aumentante ya que en todos los arcos de F se cumple que $f_{ij} < q_{ij}$ y en los de B se cumple que $f_{ij} > 0$.

2.



Esta cadena no es aumentante ya que en el arco $(s,1) \in F$, $f_{s1} = q_{s1}$

3.



Esta cadena no es aumentante ya que en el arco $(2,1) \in B$, $f_{21} = 0$

A través de una cadena aumentante C se puede aumentar el flujo de s a t construyendo así, un nuevo flujo factible f' de valor mayor que el que tiene: f. La manera de construir este nuevo flujo factible es la siguiente:

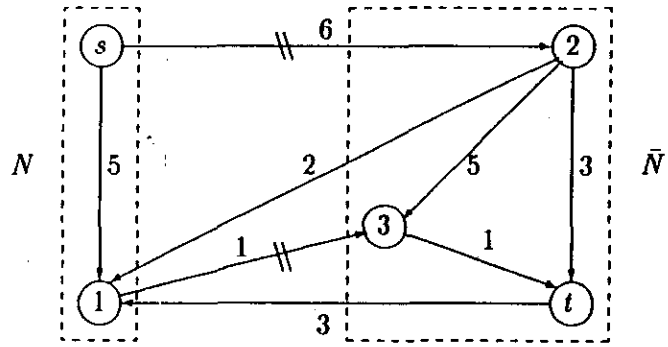
$$f'_{ij} = f_{ij} + z \qquad \forall\ (i,j) \in F$$

$$f'_{ij} = f_{ij} - z \qquad \forall\ (i,j) \in B$$

donde z es tal que:

Definición: El conjunto de arcos $(N, \bar{N})$ es una cortadura de R si $s \in N$ y $t \in \bar{N}$, donde s y t son el nodo fuente y el destino de R.

Ejemplo: En la siguiente red los arcos marcados forman una cortadura.



El número asociado a los arcos es la capacidad máxima de cada uno. En la red, si se remueven los arcos de la cortadura ya no existe camino alguno de s a t.

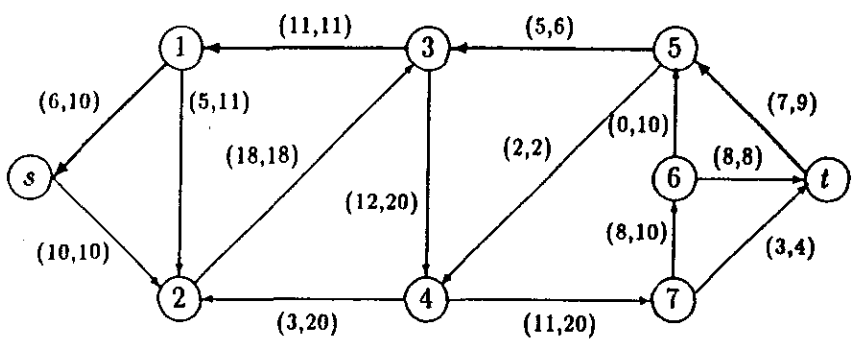
Definición: La capacidad de una cortadura $(N, \bar{N})$ que se denota por $q(N, \bar{N})$ es la suma de las capacidades máximas de los arcos que la forman. Una cortadura mínima es la que tiene mínima capacidad.

Así, la capacidad de la cortadura del ejemplo anterior es 7.

1.3 Método de solución

En vista de los conceptos de cadena aumentante y capacidad incremental se puede concluir que un método natural para determinar la solución al problema de encontrar el flujo máximo en una red es ir calculando cadenas aumentantes en ésta y a través de ellas incrementar el flujo lo más que se pueda.

Por ejemplo, supongamos que se desea determinar el flujo máximo en la red de la siguiente figura a partir del flujo factible definido en ella.



Las etiquetas de los arcos son (f_{ij}, q_{ij}). Se puede verificar fácilmente que el flujo en esta red es factible de valor $v = 4$.

El problema principal en esta red es encontrar una cadena aumentante, la idea para hacerlo es la siguiente:

Si nos fijamos en el nodo s y en todos los arcos que entran y salen de él podemos escoger uno de ellos tal que cumplan con las condiciones para formar parte de una cadena aumentante.

En la red el único arco adyacente a s, que cumple con tales condiciones es $(1, s)$, por lo tanto, si existe una cadena aumentante de s a t, ésta contiene al arco $(1, s)$.

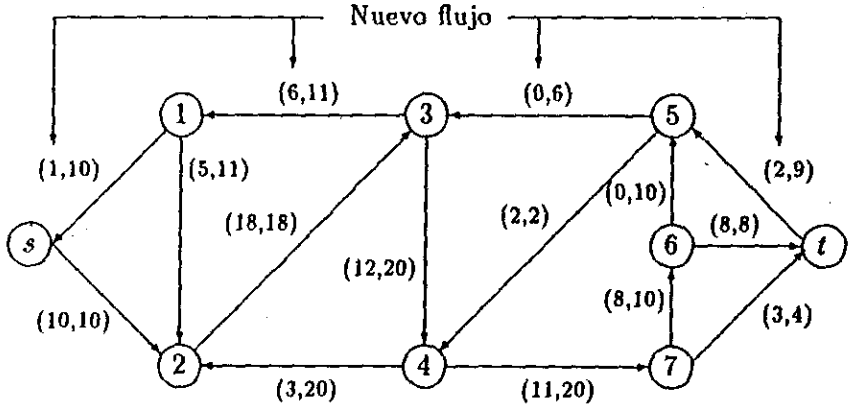
Ahora nos fijaremos en el nodo 1 (el otro extremo del arco en la cadena) y escogeremos uno de los arcos adyacentes a él que puedan entrar en la cadena, por ejemplo, el $(3, 1)$. Haremos lo mismo con el nodo 3 y suponiendo que el arco escogido en este caso es el $(5, 3)$ pasaremos al nodo 5 para revisar los arcos candidatos a entrar en la cadena con algún extremo en él. Si elegimos el arco $(t, 5)$, hemos encontrado entonces una cadena aumentante

$$C = \{s, (1, s), 1, (3, 1), 3, (5, 3), 5, (t, 5), t\}.$$

entre s y t a través de la cual podemos aumentar el flujo en la red; los arcos de esta cadena se encuentran remarcados en la figura anterior.

Obsérvese que todos los arcos de esta cadena tienen el sentido de t a s, entonces B está formado por todos los arcos de C y $F = \phi$, por lo tanto, actualizar el flujo a través de la cadena es restar al flujo actual de todos sus arcos la capacidad incremental que tiene valor $q(C) = 5$.

La red con el flujo actualizado es la siguiente:



El nuevo valor del flujo es 9.

En este procedimiento para encontrar cadenas aumentantes, se tiene que tener cuidado de no escoger un arco que ya se encuentre en la cadena ya que si ésto pasa estaríamos revisando el mismo nodo anterior. También debemos tener un registro de todos los candidatos a entrar en la cadena aumentante ya que si, por el camino que vamos, no la encontramos podemos seguir por otro

candidato, evitando comenzar desde el nodo s a buscarla.

Estos detalles son tomados en cuenta en el algoritmo de *Ford-Fulkerson* que consiste en lo siguiente:

Se inicia con cualquier flujo factible. Se calcula una cadena aumentante de la siguiente manera: El nodo fuente s tendrá dos etiquetas $[+s, \infty]$ indicando que de él se dispone de cualquier cantidad de flujo. Si j es un nodo etiquetado y puede enviarse flujo de j a i, entonces etiquetamos a i de la forma $[\pm j, f(i)]$ donde $\pm j$ será positivo si $i \in \Gamma^+(j)$, es decir, si puede aumentarse el flujo a través del arco (j, i) y negativo si $i \in \Gamma^-(j)$, es decir, si puede disminuirse el flujo a través del arco (i, j) y $f(i)$ es la cantidad de flujo que puede enviarse de j a i la cual se calcula como el mínimo entre $f(j)$ y h, donde:

$$h = \begin{cases} q_{ji} - f_{ji} & \text{si } i \in \Gamma^+(j) \\ f_{ij} & \text{si } i \in \Gamma^-(j) \end{cases}$$

Este proceso de asignación de etiquetas a los nodos se repite hasta que sea etiquetado el nodo destino t y se habrá obtenido una cadena aumentante de s a t que tendrá capacidad incremental igual a $f(t)$.

La cadena aumentante de s a t se determina con la primera etiqueta de los nodos y se actualiza el flujo a través de ella aumentando o disminuyendo el flujo de los arcos que la forman.

Si t no recibe etiquetas, entonces se tendrá el flujo máximo en la red. La justificación de este hecho se dá en la sección 2.4.

1.4 Algoritmo de Ford-Fulkerson

Se desea determinar el flujo máximo entre un nodo fuente y un destino en una red $R = [X, A, q]$.

DESCRIPCION

1. Iniciar con cualquier flujo factible f.
2. Encontrar una cadena aumentante, para ésto se hace lo siguiente:
   (a) Etiquetar el nodo s con $[+s, \infty]$.
   (b) Elegir un nodo etiquetado para examinarlo; supongase que j es el nodo elegido y que sus etiquetas son $[\pm k, f(j)]$.
      i. A todo $i \in \Gamma^+(j)$ que no tenga etiquetas y tal que $f_{ji} < q_{ji}$ se le asigna las etiquetas $[+j, f(i)]$, donde $f(i) = \min\{f(j), q_{ji} - f_{ji}\}$
      ii. A todo $i \in \Gamma^-(j)$ que no tenga etiquetas y tal que $f_{ij} > 0$ se le asigna las etiquetas $[-j, f(i)]$, donde $f(i) = \min\{f(j), f_{ij}\}$

      Luego el nodo j ha sido examinado.
   (c) Repetir el paso 2b hasta que:

i. El nodo destino t recibe etiquetas. Ir al paso 3.

ii. Todos los nodos etiquetados han sido examinados. Si el nodo destino t no tiene etiquetas el flujo factible f es máximo y termina el algoritmo.

3. Actualizar el flujo

(a) Sea $x = t$

(b) Revisar la etiqueta de x

i. Si es de la forma $[+z, f(x)]$ hacer

$$f_{zx} = f_{zx} + f(t)$$

ii. Si es de la forma $[-z, f(x)]$ hacer

$$f_{zx} = f_{zx} - f(t)$$

(c) Si $z = s$, borrar todas las etiquetas y regresar al paso 2. Si $z \neq s$, hacer $x = z$ y regresar al paso 3.(b).

1.5 Justificación teórica del algoritmo

La proposición y el teorema siguientes son una herramienta que justifica cuándo se alcanza la optimalidad.

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

$$v \leq q(N, \bar{N})$$

Demostración:

Primero veamos que

$$\sum_{j \in N} f_{ij} = \sum_{j \in (N \cap \Gamma^+(i))} f_{ij} + \cancelto{0}{\sum_{j \in (N - \Gamma^+(i))} f_{ij}} = \sum_{j \in (N \cap \Gamma^+(i))} f_{ij} \tag{1.1}$$

y

$$\sum_{j \in \bar{N}} f_{ij} = \sum_{j \in (\bar{N} \cap \Gamma^+(i))} f_{ij} + \cancelto{0}{\sum_{j \in (\bar{N} - \Gamma^+(i))} f_{ij}} = \sum_{j \in (\bar{N} \cap \Gamma^+(i))} f_{ij} \tag{1.2}$$

ahora; por (1.1) y (1.2).

$$\sum_{j \in \Gamma^+(i)} f_{ij} = \sum_{j \in (N \cap \Gamma^+(i))} f_{ij} + \sum_{j \in (\bar{N} \cap \Gamma^+(i))} f_{ij} = \sum_{j \in N} f_{ij} + \sum_{j \in \bar{N}} f_{ij} \tag{1.3}$$

análogamente:

$$\sum_{k\in\Gamma^-(i)} f_{ki} = \sum_{k\in N} f_{ki} + \sum_{k\in\bar{N}} f_{ki} \tag{1.4}$$

Y si sumamos las ecuaciones de conservación de flujo para todos los nodos de N y después separando N en los subconjuntos ajenos $N-\{s\}$ y $\{s\}$ tenemos:

$$\sum_{i\in N}\left(\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{k\in\Gamma^-(i)} f_{ki}\right) =$$

$$\sum_{i\in N-\{s\}}\left(\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{k\in\Gamma^-(i)} f_{ki}\right) + \left(\sum_{j\in\Gamma^+(s)} f_{sj} - \sum_{k\in\Gamma^-(s)} f_{ks}\right) = 0 + v = v$$

ésto se justifica con el hecho de que toda $i \in N-\{s\}$ es distinta de s y de t y para estas i's se cumple que

$$\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{k\in\Gamma^-(i)} f_{ki} = 0$$

además, para s

$$\sum_{j\in\Gamma^+(s)} f_{sj} - \sum_{k\in\Gamma^-(s)} f_{ks} = v$$

por otro lado utilizando (1.3) y (1.4).

$$v = \sum_{i\in N}\left(\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{k\in\Gamma^-(i)} f_{ki}\right) = \sum_{i\in N}\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{i\in N}\sum_{k\in\Gamma^-(i)} f_{ki} =$$

$$\sum_{i\in N}\left(\sum_{j\in N} f_{ij} + \sum_{j\in\bar{N}} f_{ij}\right) - \sum_{i\in N}\left(\sum_{k\in N} f_{ki} + \sum_{k\in\bar{N}} f_{ki}\right) =$$

$$\cancel{\sum_{i\in N, j\in N} f_{ij}} + \sum_{i\in N, j\in\bar{N}} f_{ij} - \cancel{\sum_{i\in N, k\in N} f_{ki}} - \sum_{i\in N, k\in\bar{N}} f_{ki} =$$

y como $0 \le f_{ij} \le q_{ij}$ para todo $(i,j) \in A$.

$$\sum_{i\in N, j\in\bar{N}} f_{ij} - \sum_{i\in N, k\in\bar{N}} f_{ki} \leq \sum_{i\in N, j\in\bar{N}} f_{ij} \leq \sum_{i\in N, j\in\bar{N}} q_{ij} = q(N,\bar{N})$$

concluimos que

$$v \leq q(N,\bar{N})$$

■

En el siguiente teorema se vé claramente la relación tan estrecha que tienen el concepto de Flujo Máximo y el de Cortadura Mínima.

TEOREMA: (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:

Como $v \leq q(N,\bar{N})$ por la proposición anterior, sólo basta demostrar que existe un flujo factible en R que tiene valor igual a la capacidad de una cortadura de R.

La idea de la demostración es construir una cortadura de R para comparar su capacidad y el flujo en R y verificar que efectivamente son iguales. Para ésto se construye primero el conjunto N, con $s \in N$, del tal manera que si $i \in N$ se puede aumentar el flujo desde s hasta i.

Necesitamos que $t \notin N$ al terminar de construirlo para que efectivamente $(N,\bar{N})$ con $\bar{N} = X - N$, sea una cortadura. Si éste no es el caso, de manera iterativa, podemos "forzar" a que t quede en $\bar{N}$.

El procedimiento es el siguiente:

Considérese cualquier flujo factible f en R y construyase el conjunto N de la siguiente manera:

1. Sea $N = \{s\}$
2. Tomamos $i \in N$ y $j \in \bar{N} = X - N$. Si $j \in \Gamma^+(i)$ y $f_{ij} < q_{ij}$ ó $j \in \Gamma^-(i)$ y $f_{ji} > 0$ se agrega j a N.

Se repite 2. hasta que no pueda agregarse nodo alguno a N. Entonces puede suceder que $t \in N$ ó $t \notin N$.

- Veremos primero el caso en que $t \in N$:

  Por la forma en que se construyó N, existe una cadena de s a t tal que $f_{ij} < q_{ij}$ para todo $(i,j) \in F$ y $f_{ij} > 0$ para todo $(i,j) \in B$ se ha encontrado una cadena aumentante C de s a t por lo que se puede mejorar el flujo de s a t de la siguiente manera:

Sea $q(C)$ la capacidad incremental de la cadena C y redefínase f y v como:

$$f_{ij} = \begin{cases} f_{ij} + q(C) & \text{si } (i,j) \in F \\ f_{ij} - q(C) & \text{si } (i,j) \in B \\ f_{ij} & \text{en otro caso} \end{cases}$$

y el nuevo valor de este flujo resulta ser

$$v = v + q(C)$$

Como este nuevo f es un flujo factible de valor v, entonces se puede repetir el procedimiento señalado anteriormente utilizando este flujo de mayor valor. Observemos que como $q_{ij} \in \mathcal{Z}$ $\forall$ $(i,j) \in A$ el flujo se incrementa en al menos una unidad en cada iteración, es decir $q(C) \geq 1$ por lo tanto el flujo máximo se obtiene en un número finito de pasos.

- Ahora veremos el caso en que $t \notin N$:

  Si $t \notin N$ entonces $t \in X - N = \bar{N}$ y $(N, \bar{N})$ es una cortadura de R. Por construcción tenemos que $f_{ij} = q_{ij}$ $\forall$ $(i,j) \in (N, \bar{N})$, es decir, el conjunto de arcos tales que $i \in N$ y $j \in \bar{N}$; y $f_{ij} = 0$ $\forall$ $(i,j) \in (\bar{N}, N)$, es decir, el conjunto de arcos tales que $i \in \bar{N}$ y $j \in N$ y entonces:

$$v = \sum_{i \in N, j \in \bar{N}} f_{ij} - \sum_{i \in N, k \in \bar{N}} \overbrace{f_{ki}}^{0} = \sum_{i \in N, j \in \bar{N}} f_{ij} = \sum_{i \in N, j \in \bar{N}} q_{ij} = q(N, \bar{N})$$

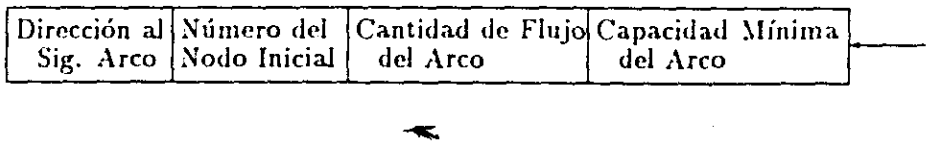
■

El conjunto N durante esta demostración corresponde al conjunto de nodos etiquetados en la presentación del algoritmo ya que se construyen de la misma manera. Por ésto, la justificación de optimalidad y convergencia del algoritmo está dada por éste teorema. Nótese además que siempre que se encuentra el flujo máximo en una red se encuentra también una cortadura mínima.

Para la implementación del algoritmo utilizamos estructura de datos para guardar la información necesaria de la red, éstas se presentan a continuación.

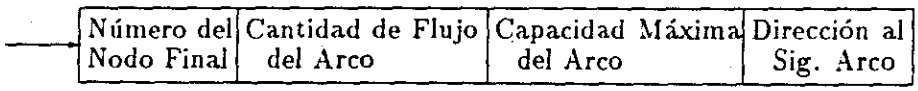
1.6 Estructura de datos

Se utilizan listas enlazadas de estructura de datos para nodos y para arcos para formar lo que se conoce con el nombre de "Gráfica de estructura de datos" la cual tiene la siguiente forma:



- Para guardar la información de los arcos sucesores se utilizan registros que contenga:
  - El número de nodo final del arco que lo conecta con el nodo en cuestión.
  - La cantidad de flujo a través del arco.
  - La capacidad máxima del arco.
  - La dirección al siguiente elemento en la lista de arcos sucesores.

  La forma en la que será acomodada la información de los arcos sucesores se vé en la siguiente ampliación de la estructura de los mismos.



- Para la pila donde se guarda los nodos etiquetados y no examinados se utilizan registros que contengan:
  - La dirección del nodo en la lista de los mismos.
  - La dirección al siguiente elemento en la pila.

1.7 Descripción de las iteraciones del algoritmo

En esta sección describiremos, en pseudocódigo, la implementación del algoritmo de *Ford-Fulkerson* apegándonos al procedimiento de etiquetado presentado anteriormente, aunque no sea la mejor manera de implementarlo. Al finalizar la descripción comentaremos los detalles de una implementación más eficiente.

Primeramente definiremos las variables que usaremos en la descripción de las iteraciones:

- En la estructura de los Nodos:
  - Arco_Antecesor : Dirección del arco antecesor.
  - Num_Nodo : Número de nodo.
  - Ant_Cad_Aum : Antecesor en la cadena aumentante.
  - Cap_Cad_Aum : Capacidad de la cadena aumentante.
  - Siguiente : Dirección al siguiente nodo en la lista.

  - Arco_Sucesor : Dirección del arco sucesor.

- En la estructura de los Arcos antecesores:
  - Siguiente : Dirección al siguiente arco antecesor en la lista.
  - Nodo_Inicial : Número del nodo inicial del arco.
  - Flujo : Cantidad de flujo del arco.
  - Cap_Min : Capacidad mínima del arco.
- En la estructura de los Arcos sucesores:
  - Nodo_Final : Número del nodo final del arco.
  - Flujo : Cantidad de flujo del arco.
  - Cap_Max : Capacidad máxima del arco.
  - Siguiente : Dirección al siguiente arco sucesor en la lista.
- En la estructura de la Pila:
  - Nodo : Dirección del nodo en la lista de los mismos.
  - Siguiente : Dirección al siguiente nodo en la pila.
- N_rev : Nodo en revisión.
- Arc_Ant : Dirección del arco antecesor que se va a revisar.
- Arc_Suc : Dirección del arco sucesor que se va a revisar.
- N_Cad : Nodo en la cadena aumentante.

Definiremos Infinito = MAXINT (máximo valor entero con el que la computadora puede trabajar).

Se construye la gráfica de estructura de datos con la información de la red.

Se inicializan las variables en la estructura de todos los nodos de la gráfica:

$$\text{Ant\_Cad\_Aum} = 0 \qquad \text{Cap\_Cad\_Aum} = \text{Infinito}$$

La pila está vacía.

Se comienza con un flujo factible igual a cero en todos los arcos, es decir, se inicializa, en las estructuras de arcos antecesores y sucesores, la variable:

$$\text{Flujo} = 0$$

Realización de una iteración

1. Se busca una cadena aumentante entre s y t.

   Se toma y etiqueta el nodo a partir del cual queremos encontrar la cadena aumentante, es decir, el nodo s, se etiquetan los nodos extremos de sus arcos antecesores y sucesores que no lo esten y se van guardando en una pila. Se saca un elemento de la pila y se repite el procedimiento hasta etiquetar el nodo t, si ésto no sucede entonces no existe cadena aumentante entre s y t y el flujo en la red es óptimo. Esto se hace de la siguiente manera:

   (a) Se etiqueta el nodo s haciendo Ant_Cad_Aum = $+s$.

   (b) Se hace N_rev = s

   (c) Se hace Arc_Suc = primer arco sucesor de N_rev

      i. Si Arc_Suc existe

         A. Se compara Cap_Max y Flujo de Arc_Suc.

         B. Si son iguales, se pasa a E.

         C. Si son distintos y Ant_Cad_Aum del nodo final de Arc_Suc es cero, es decir, si el nodo final de Arc_Suc no está etiquetado, se etiqueta el nodo final con Ant_Cad_Aum = + N_rev, se calcula la diferencia de Cap_Max y Flujo de Arc_Suc y si:

            - La diferencia es menor o igual a Cap_Cad_Aum de N_rev se hace Cap_Cad_Aum del nodo final de Arc_Suc igual a la diferencia mencionada arriba.
            - La diferencia es mayor que Cap_Cad_Aum de N_rev se hace Cap_Cad_Aum del nodo final de Arc_Suc igual a Cap_Cad_Aum de N_rev.

         D. Si el nodo final del arco es el nodo t, ir al paso 2.; si no, se mete a la pila.

         E. Se hace Arc_Suc = siguiente arco sucesor en la lista de N_rev. Regresar a 1.(c) i.

      ii. Si Arc_Suc no exite pasar a 1.(d).

   (d) Se hace Arc_Ant = primer arco antecesor de N_rev

      i. Si Arc_Ant existe

         A. Si el flujo de Arc_Ant es distinto a su capacidad mínima y su nodo inicial no está etiquetado se etiqueta el nodo inicial con Ant_Cad_Aum = − N_rev y Cap_Cad_Aum = Flujo de Arc_Ant − Cap_Min de Arc_Ant.

         B. Si el nodo inicial del arco es el nodo t, ir al paso 2.; si no, se mete a la pila.

         C. Se hace Arc_Ant = siguiente arco antecesor en la lista de N_rev. Regresar a 1.(d) i.

      ii. Si Arc_Suc no exite pasar a 1.(e).

   (e) Se saca un elemento de la pila y N_rev toma su valor. Regresar a 1.(c). Si la pila ya no tiene elementos y no se ha etiquetado el nodo t no existe cadena aumentante entre s y t y el flujo en la red es óptimo. Se termina el algoritmo.

2. Se ha encontrado una cadena aumentante entre s y t, ahora se actualizará el flujo en esa cadena aumentante.

   Se recorre la cadena aumentante "hacia atrás", es decir, desde el nodo t hasta el nodo s, haciendo uso de la etiqueta Ant_Cad_Aum, sumando al flujo actual de cada arco que tiene sentido de s a t la capacidad incremental de la cadena encontrada y restándola al flujo actual de cada arco que tiene sentido de t a s. A continuación se ilustra el procedimiento:

   (a) Se hace N_Cad = t

   (b) Si N_Cad es distinto del nodo s y

      i. Si Ant_Cad_Aum de N_Cad es positivo

         A. Se busca Ant_Cad_Aum en la lista de arcos antecesores de N_Cad, cuando se encuentra se actualiza el flujo de ese arco haciendo Flujo = Flujo + Cap_Cad_Aum de t.

         B. Se busca N_Cad en la lista de arcos sucesores de Ant_Cad_Aum, cuando se encuentra se actualiza el flujo de ese arco haciendo Flujo = Flujo + Cap_Cad_Aum de t.

      ii. Si Ant_Cad_Aum de N_Cad es negativo

         A. Se busca −Ant_Cad_Aum en la lista de arcos sucesores de N_Cad, cuando se encuentra se actualiza el flujo de ese arco haciendo Flujo = Flujo - Cap_Cad_Aum de t.

         B. Se busca N_Cad en la lista de arcos antecesores de −Ant_Cad_Aum, cuando se encuentra se actualiza el flujo de ese arco haciendo Flujo = Flujo - Cap_Cad_Aum de t.

      iii. Se hace N_Cad = Ant_Cad_Aum y se regresa a 2.(b).

3. Se etiquetan todos los nodos de nuevo con Ant_Cad_Aum = 0 y Cap_Cad_Aum = Infinito. Se vacía la pila y se regresa a 1.

Como se había mencionado ésta no es la manera más eficiente de implementar el algoritmo de *Ford-Fulferson*. La razón es que, en el proceso de etiquetado para encontrar una cadena aumentante, se realizan operaciones de diferencia inecesarias al calcular la capacidad incremental de la cadena, ya que se etiquetan muchos nodos que no forman parte de la cadena que hemos encontrado; ésto es más notable si la red en la que deseamos encontrar el flujo máximo es muy grande.

Lo que puede hacerse para evitar tanto cálculo, es etiquetar sólo con el antecesor de la cadena aumentante para encontrarla. Después, ya que se tiene la cadena aumentante, recorrer la gráfica para calcular la capacidad incremental de la cadena, de la siguiente manera:

- Se define previamente una variable para ir guardando la posible capacidad de la cadena aumentante al revisar cada nodo. Esta variable será Cap_Cad y se inicializa con Infinito, es decir, Cap_Cad = Infinito.

- Se hace N_Cad = t.

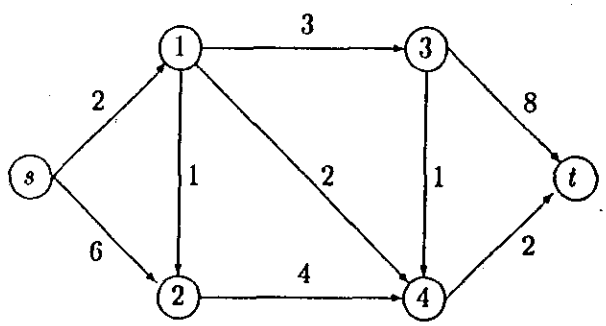
- Se revisa Ant_Cad_Aum de N_Cad y si:
  - Es positivo, se busca Ant_Cad_Aum en la lista de nodos y se busca N_Cad en la lista de arcos sucesores de Ant_Cad_Aum. Se hace Cap_Cad =min { Cap_Cad, Cap_Cad_Aum - Flujo de ese arco }.
  - Es negativo, se busca −Ant_Cad_Aum en la lista de nodos y se busca N_Cad en la lista de arcos antecesores de −Ant_Cad_Aum. Se hace Cap_Cad = min { Cap_Cad, Flujo de ese arco }.
- Si Ant_Cad_Aum de N_Cad es el nodo s, ir al siguiente punto. Si no es el nodo s se hace N_Cad = Ant_Cad_Aum de N_Cad. Regresar al punto anterior.

Se recorre de nuevo la gráfica como se indica en el paso 2. para subir el flujo a través de la cadena encontrada.

A continuación veremos un ejemplo de una red donde se requiere encontrar el flujo máximo en ella, el cual lo resolvemos usando los diagramas de las estructuras de datos utilizadas en la implementación del algoritmo de *Ford-Fulkerson*. Presentaremos también en cada iteración la red del ejemplo especificando en cada nodo la etiqueta que le corresponde según se haya etiquetado en la gráfica de estructura de datos.

1.8 Ejemplo

Determinar el flujo máximo entre s y t de la siguiente red.



El valor asignado a los arcos es la capacidad máxima de cada uno de ellos.

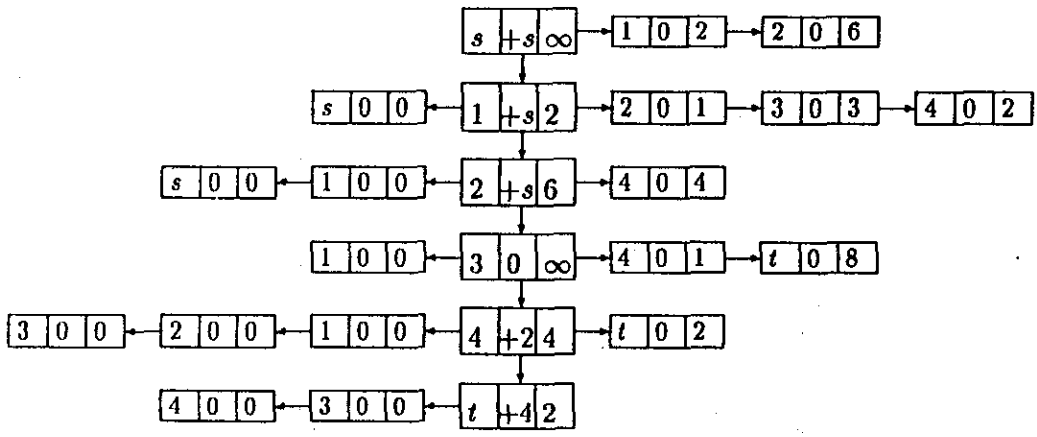
Solución

En la gráfica de estructura de datos acomodaremos la información de la manera que se ilustró en la sección 2.5.1. y para dar las corridas de escritorio se omitirán las casillas donde se tiene la dirección de algún elemento de las listas ya sea de nodos o de arcos.

Corrida de escritorio

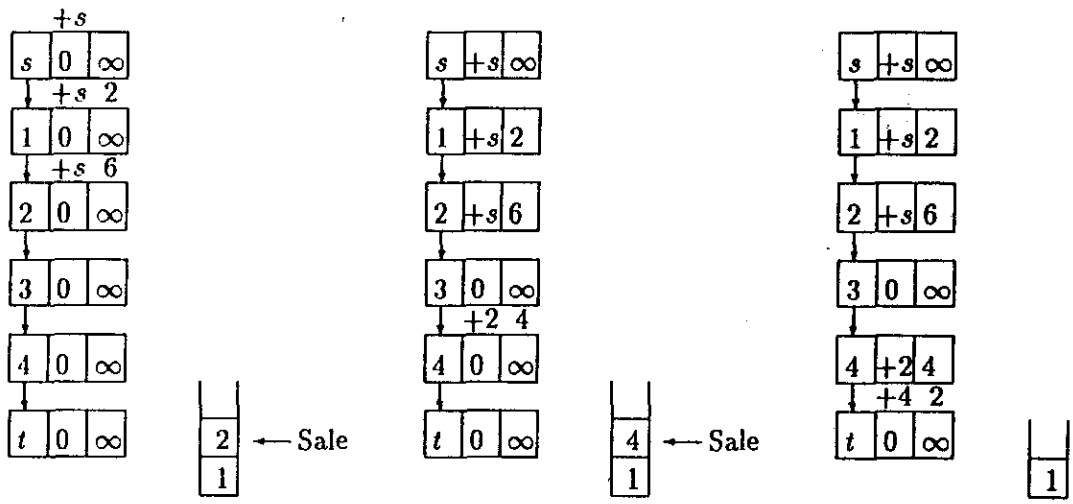
Primera Iteración

La siguiente, es la gráfica con las etiquetas para encontrar una cadena aumentante y la capacidad incremental de tal cadena.



La manera en la que llegamos a la gráfica anterior la señalamos enseguida.

Para no trabajar con toda la gráfica, en la siguiente figura sólo aparece la lista de nodos ya que los arcos no tendrán ninguna modificación en este paso.

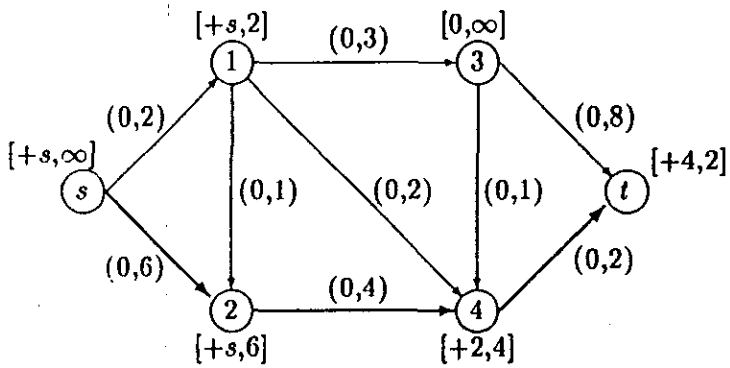


Se etiqueta el nodo s.
Se etiquetan cada uno de los arcos sucesores que no lo esten y se meten a la pila.

Se saca el nodo 2 de la pila.
Se etiquetan los arcos suc. y ant. que no lo esten y se meten a la pila.

Se saca el nodo 4 de la pila.
Se etiquetan los arcos suc. y ant. que no lo esten y como se etiquetó el nodo t, terminamos.

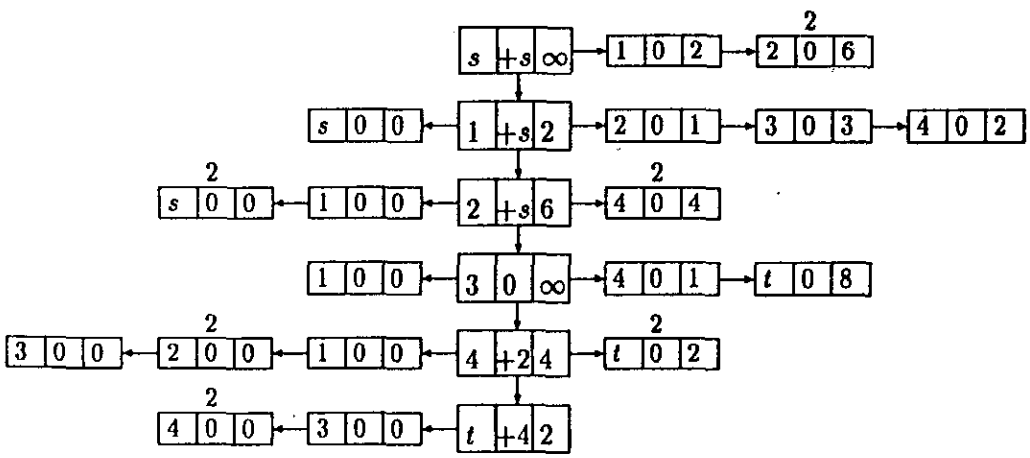
La red en este paso es:

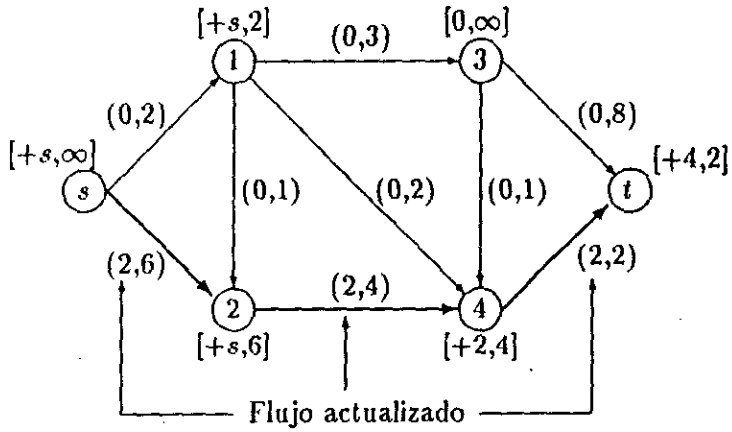


Los arcos más gruesos son los que forman la cadena aumentante por la cual se aumentará el flujo en la red.

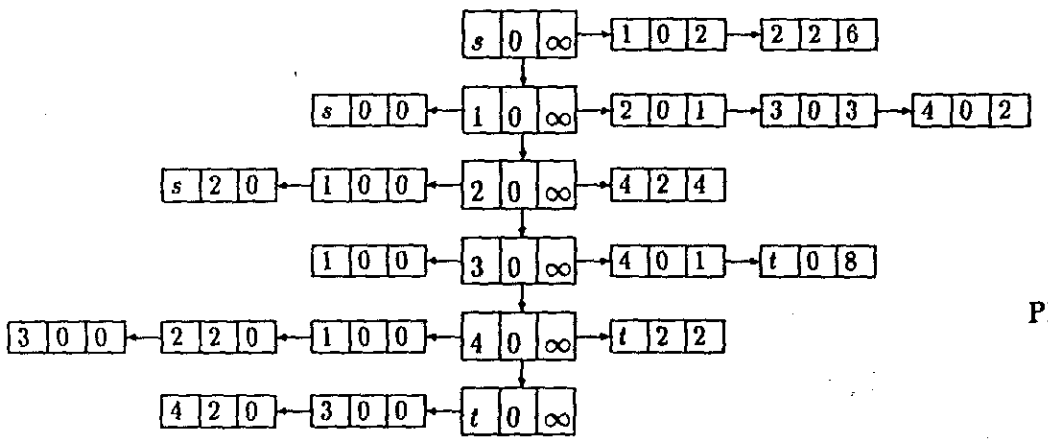
Actualizando el flujo entre s y t.

Se toma el nodo t y su antecesor en la cadena aumentante (+4), como es positivo se busca en la lista de antecesores y se le suma al flujo de ese arco la capacidad de la cadena aumentante (2), se busca el nodo t en la lista de sucesores del nodo 4 y se le suman 2 unidades al flujo de ese arco. Se realiza el mismo procedimiento con los nodos 4 y 2. Cuando llegamos al nodo s ya hemos actualizado el flujo en la red.

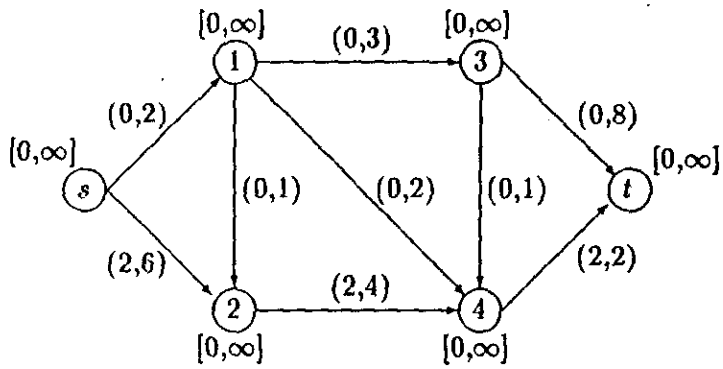




Actualizar etiquetas en la red y en la gráfica de estructura de datos.

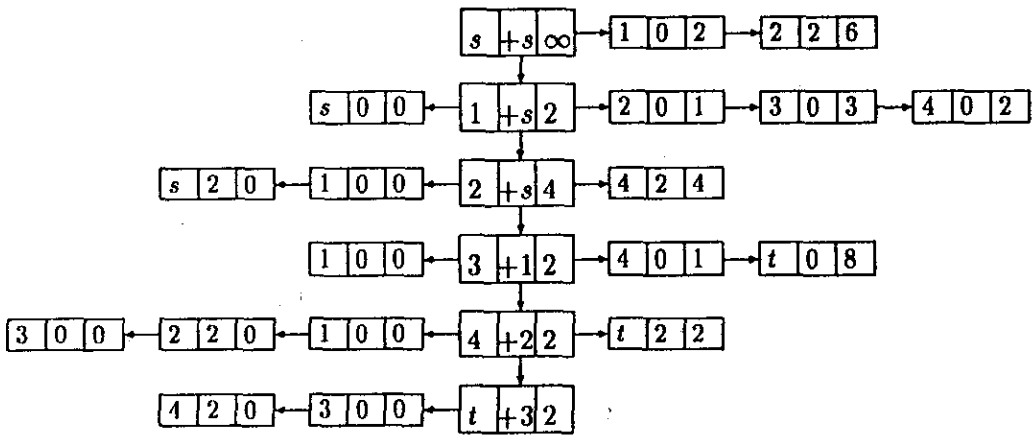


Pila vacía

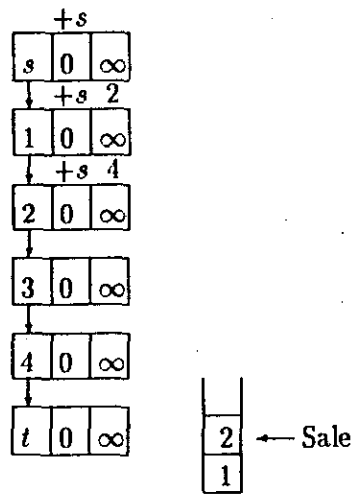


Segunda Iteración

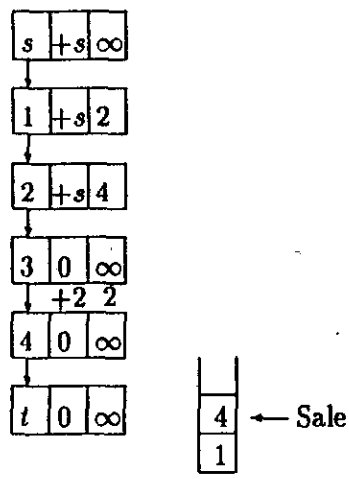
La siguiente, es la gráfica con las etiquetas para encontrar una cadena aumentante y la capacidad incremental de tal cadena.



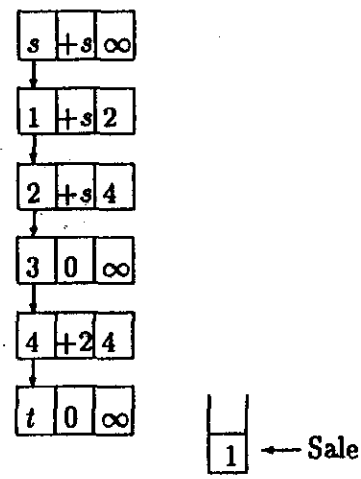
Para llegar a la gráfica anterior se hizo lo siguiente:



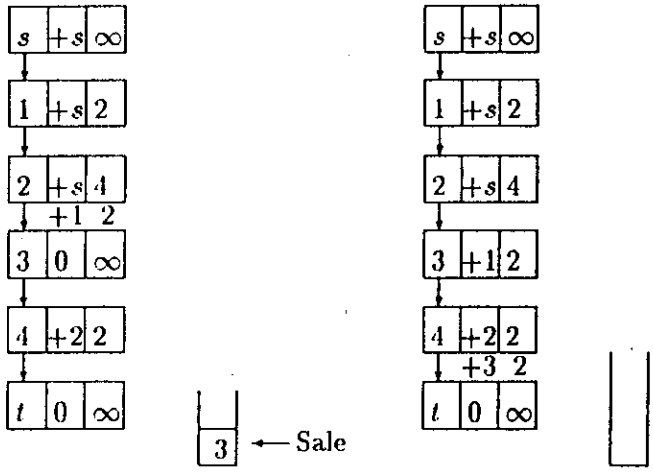
Se etiqueta el nodo s.
Se etiquetan cada uno de los arcos sucesores que no lo esten y se meten a la pila.



Se saca el nodo 2 de la pila.
Se etiquetan los arcos suc. y ant. que no lo esten y se meten a la pila.



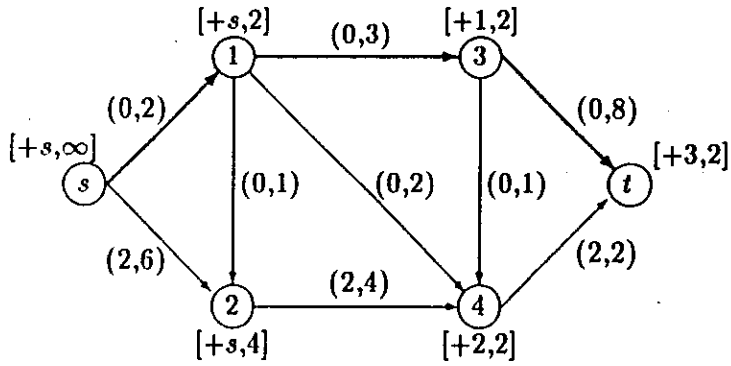
Se saca el nodo 4 de la pila, observese que ninguno de los arcos adyacentes a él se puede etiquetar.



Se saca el nodo 1 de la pila. Se etiquetan los arcos suc. y ant. que no lo esten y se meten a la pila.

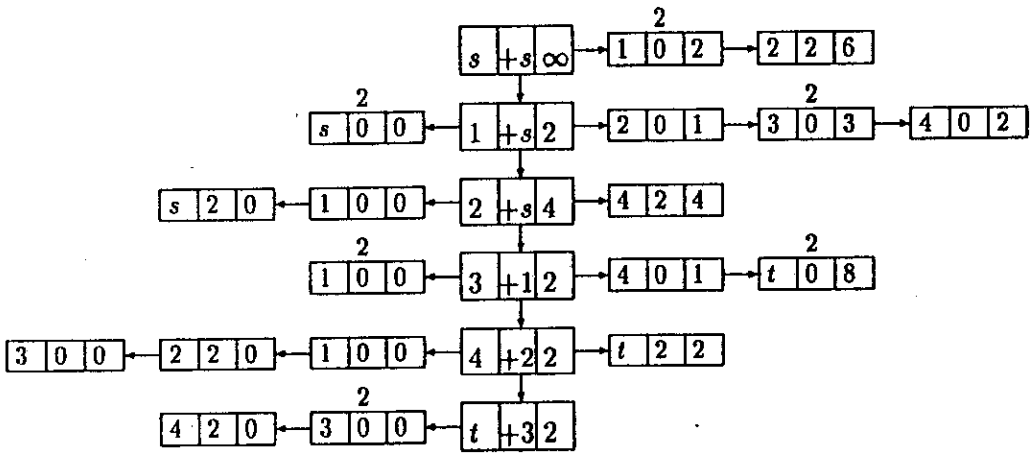
Se saca el nodo 3 de la pila. Se etiquetan los arcos suc. y ant. que no lo esten y como se etiquetó el nodo t, terminamos.

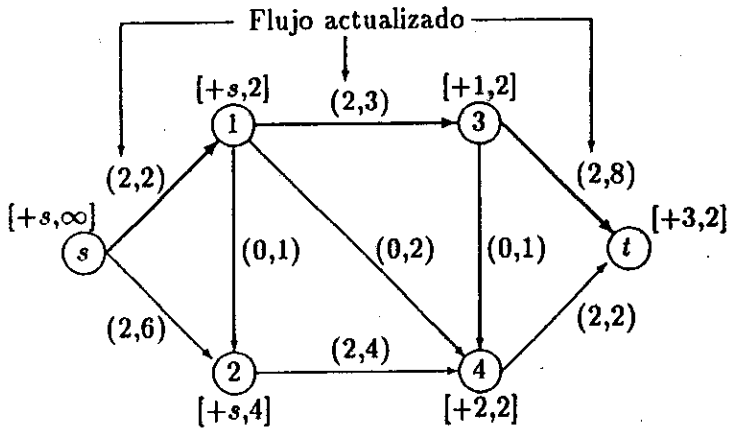
La red en este paso es:



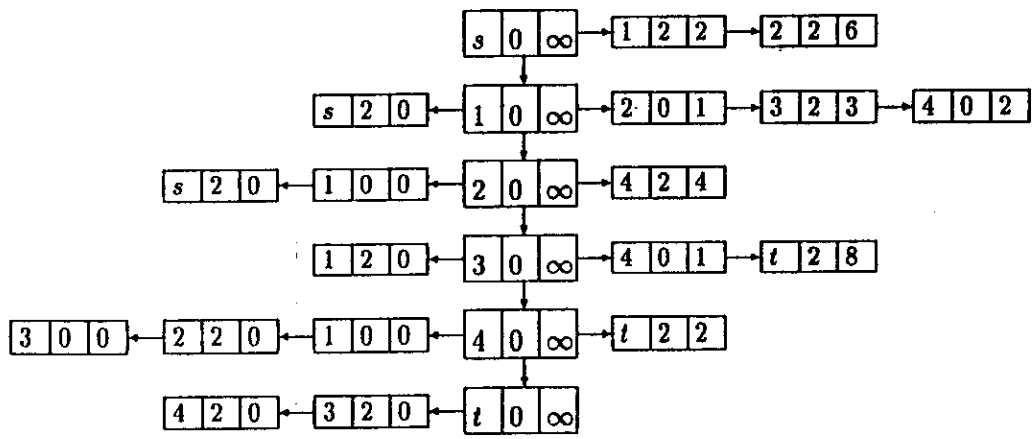
Actualizando el flujo.

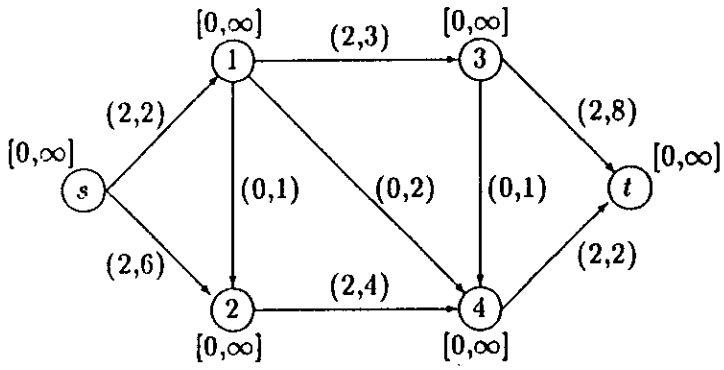
Se toma el nodo t y su antecesor en la cadena aumentante (+3), como es positivo se busca en la lista de antecesores y se le suma al flujo de ese arco la capacidad de la cadena aumentante (2), se busca el nodo t en la lista de sucesores del nodo 3 y se le suman 2 unidades al flujo de ese arco. Se realiza el mismo procedimiento con los nodos 3 y 1. Cuando llegamos al nodo s ya hemos actualizado el flujo en la red.





Actualizar etiquetas en la red y en la gráfica de estructura de datos.

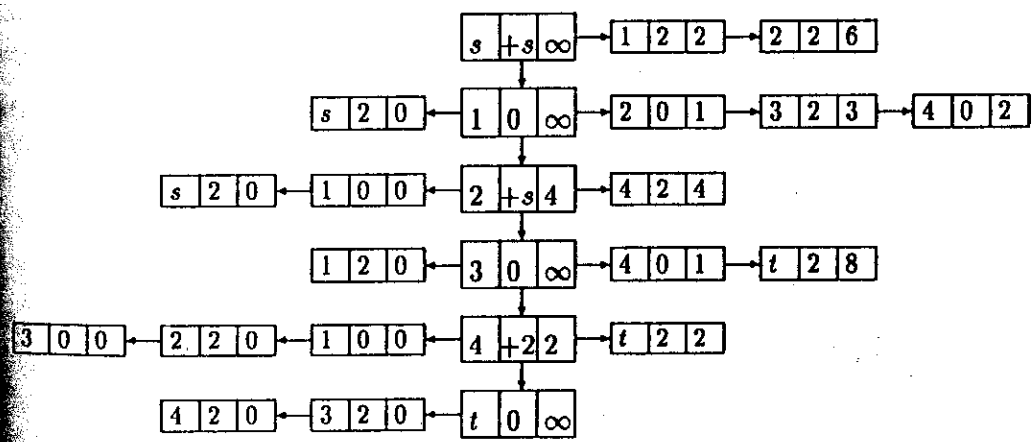




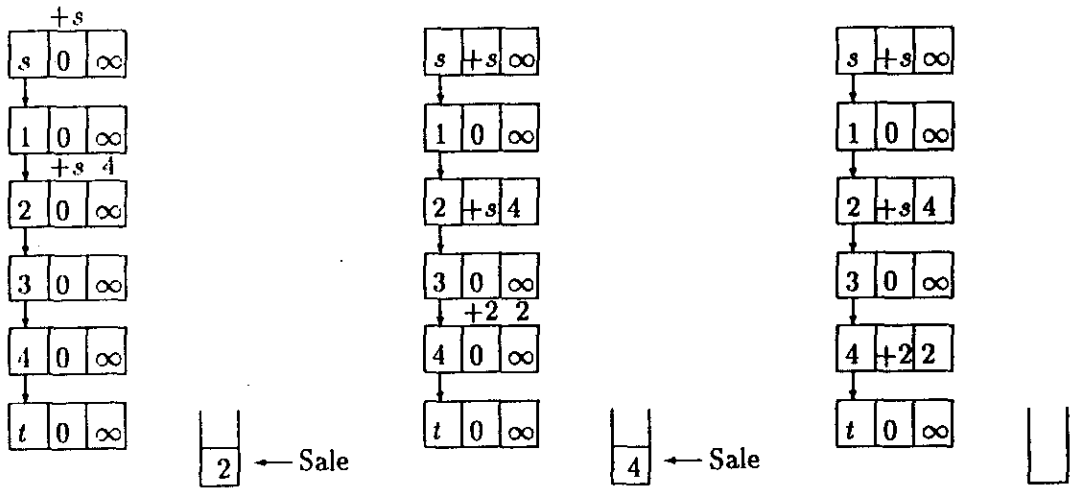
Pila vacía

Tercera Iteración

siguiente, es la gráfica con las etiquetas para encontrar una cadena aumentante y la capacidad ntal de tal cadena.



legar a la gráfica anterior se hizo lo siguiente:

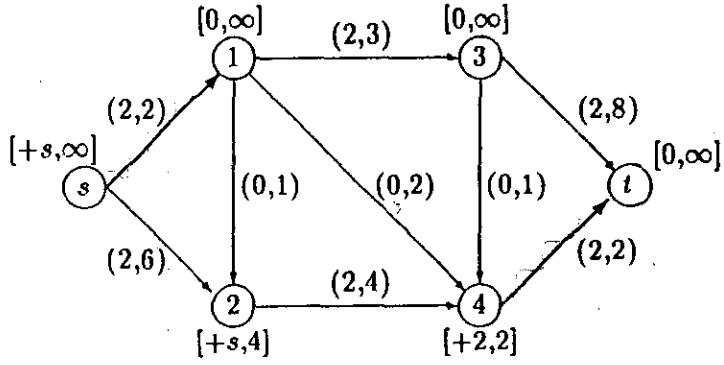


Se etiqueta el nodo s. Se etiquetan cada uno de los arcos sucesores que no lo esten y se meten a la pila.

Se saca el nodo 2 de la pila. Se etiquetan los arcos suc. y ant. que no lo esten y se meten a la pila.

Se saca el nodo 4 de la pila, observese que ninguno de los arcos adyacentes a él se puede etiquetar y como ya no hay nodos en la pila, terminamos.

En esta iteración no se encontró cadena aumentante entre s y t, por lo tanto el flujo que tienen los arcos de la red es óptimo.



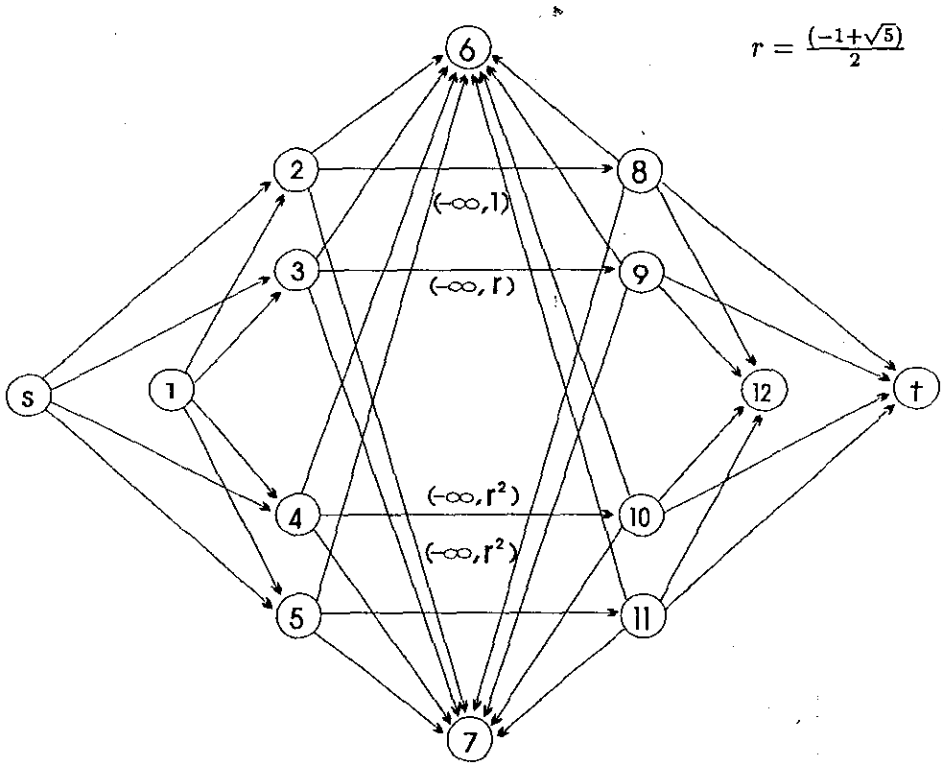
Recordando la demostración constructiva del teorema de Flujo Máximo - Cortadura Mínima todos los nodos etiquetados en esta red constituyen el conjunto N y los no etiquetados, el conjunto $\bar{N}$, por lo tanto una cortadura mínima producida por el flujo máximo que tiene esta red son los arcos tales que su extremo inicial esté etiquetado y su extremo final no.

Los arcos más gruesos, en la figura, son los que forman la cortadura mínima.

Un procedimiento para encontrar la cortadura mínima teniendo la gráfica de estructura de datos

con las etiquetas producidas al buscar una cadena aumentante, es recorrer la gráfica y tomar cada nodo etiquetado revisando todos sus arcos sucesores; si el nodo final del arco no esta etiquetado, entonces este arco pertenece a la cortadura.

Como se mencionó anteriormente, si las capacidades máximas en los arcos son reales, el algoritmo de *Ford-Fulkerson*, no converge al óptimo flujo en la red. El siguiente problema tomado de [2] es un ejemplo de lo antes mencionado.



Las etiquetas en esta red son la capacidad mínima y máxima permitida de flujo para cada arco. En los arcos donde no aparecen etiquetas, las capacidades mínima y máxima de flujo son $-\infty$ e ∞ respectivamente.

El termino capacidad mínima de un arco es una de las variantes que se pueden presentar en el tipo de problema que trataremos en este trabajo y será abordado en el segundo capítulo del mismo. La idea presentada en [2] para demostrar que el algoritmo no converge al flujo óptimo en la red de la figura mostrada anteriormente, es verificar que en ella el supremo de flujo no corresponde a la solución que dá el algoritmo.

Capítulo 2

Variantes del Problema de Flujo Máximo

Las variantes del problema de flujo máximo son las siguientes:

- Existan varias fuentes y destinos.
- Los arcos tengan restricciones mínimas para flujo a través de ellos.
- Algunos nodos tengan restricciones mínimas o máximas para el flujo que pasa por ellos.

Cada una de éstas variantes se reducirán al problema sencillo de flujo máximo entre un nodo fuente s y un nodo destino t tratado en el capítulo 1, construyendo una gráfica auxiliar en cada caso a la cual se le aplica el algoritmo de *Ford-Fulkerson* para encontrar el flujo máximo y después se recupera la gráfica original con la solución de flujo máximo en ella. Presentaremos también las estructuras de datos usadas en cada una de ellas. Al final del capítulo se expondrá la idea de la implementación en el caso que se presenten todas las variantes, ya que es cuando tiene caso comentar a fondo los detalles de la implementación.

2.1 Varias fuentes o varios destinos

Supóngase que la red $R = [X, A, q]$ tiene m fuentes s_1, s_2,..., s_m, n destinos t_1, t_2,...t_n y que puede enviarse flujo de cualquier fuente a cualquier destino.

El problema es determinar el flujo máximo que puede enviarse, a través de la red, de todos los orígenes a todos los destinos.

Sea $R' = [X', A', q']$ donde:

$$X' = X \cup \{s, t\}$$

$$A' = A \cup \{(s, s_i)/i = 1, 2, ..., m\} \cup \{(t_j, t)/j = 1, 2, ..., n\}$$

$$q'_{ij} = \begin{cases} q_{ij} & \text{si } (i,j) \in A \\ \infty & \text{si } i = s \quad \acute{o} \quad j = t \end{cases}$$

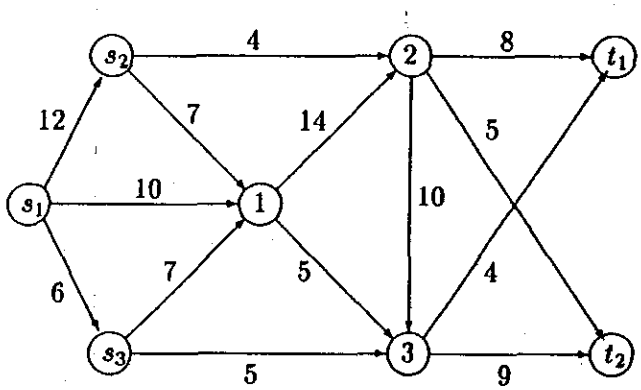
El problema se reduce a determinar el flujo máximo entre el nodo fuente s y el nodo destino t de la red R'.

Una vez resuelto el problema en R' bastará con eliminar los elementos de R' que no están en R y el flujo definido en los arcos de R será óptimo.

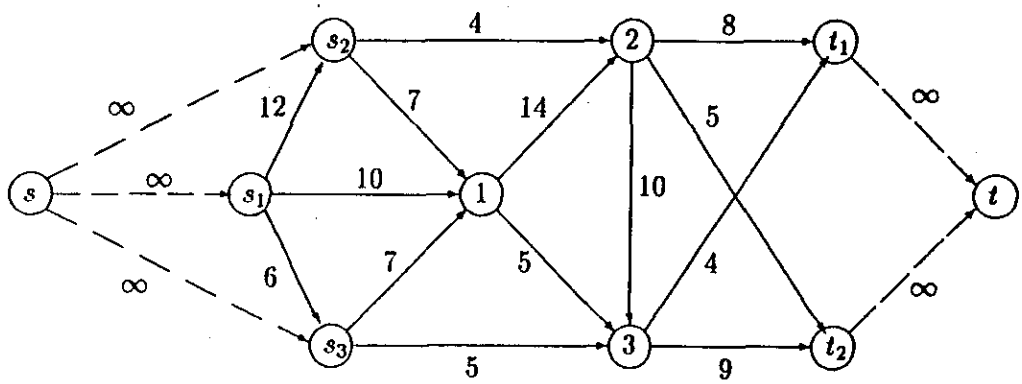
Para resolver el problema se utilizará el algoritmo de *Ford-Fulkerson.*

Ejemplo:

Supongamos que tenemos la red R de la siguiente figura, en la cual deseamos obtener el flujo máximo. Los nodos s_1, s_2 y s_3 son las fuentes de donde se desea enviar flujo a los nodos destino t_1 y t_2.



La red R' auxiliar que se utilizaría en este caso es:



Estructura de datos

Las estructuras de datos adicionales usada para implementar esta variante del problema de flujo máximo son las siguientes:

- La información de cuáles nodos son fuente se guarda en una pila para la cual se utilizan registros que contengan:
  - La dirección del nodo fuente.
  - La dirección al siguiente elemento en la pila de nodos fuente.
- La información de cuáles nodos son destino se guarda en una pila para la cual se utilizan registros que contengan:
  - La dirección del nodo destino.
  - La dirección al siguiente elemento en la pila de nodos destino.

La implementación de esta variante del problema de flujo máximo es muy sencilla ya que lo único que hay que hacer es agregar a la gráfica con los datos de la red los nodos s y t auxiliares y después, haciendo uso de las pilas con los nodos fuentes y destinos, se agregan los arcos auxiliares de la forma que se mencionó anteriomente.

2.2 Restricciones mínimas en arcos

Supóngase que tenemos un problema en el que se requiere que la cantidad de producto que pase por los arcos en R sea mayor que un cierto valor. La red que modela el problema es $R = [X, A, r, q]$ donde q es la función de capacidad máxima de flujo asociada a los arcos de R y $r : A \rightarrow \mathcal{Z}$ es la función que asocia, a cada arco $(i, j) \in A$, la cota inferior r_{ij} permitida para el flujo a través de (i, j).

Las condiciones de factibilidad del problema las dan las ecuaciones de conservación de flujo y $r_{ij} \leq f_{ij} \leq q_{ij}$, para todo $(i, j) \in A$ (r_{ij} no todos cero).

Note que el caso expuesto en el capítulo 2 es un caso particular de éste donde $r_{ij} = 0$ para todo $(i, j) \in A$.

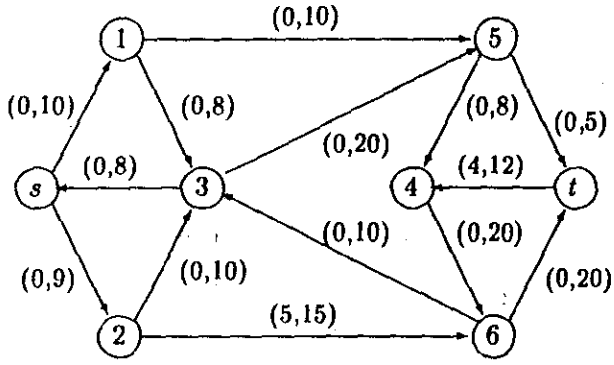
Un primer problema, para encontrar el flujo máximo en la red R, sería encontrar un flujo factible inicial en ella; ya que no podríamos tomar el flujo igual a cero en todos los arcos por ser los r_{ij} no todos cero.

Para determinar una primera solución factible se utilizará la red. $R' = [X \cup \{s', t'\}, A \cup A' \cup \{(t, s), (s, t)\}, q']$ donde:

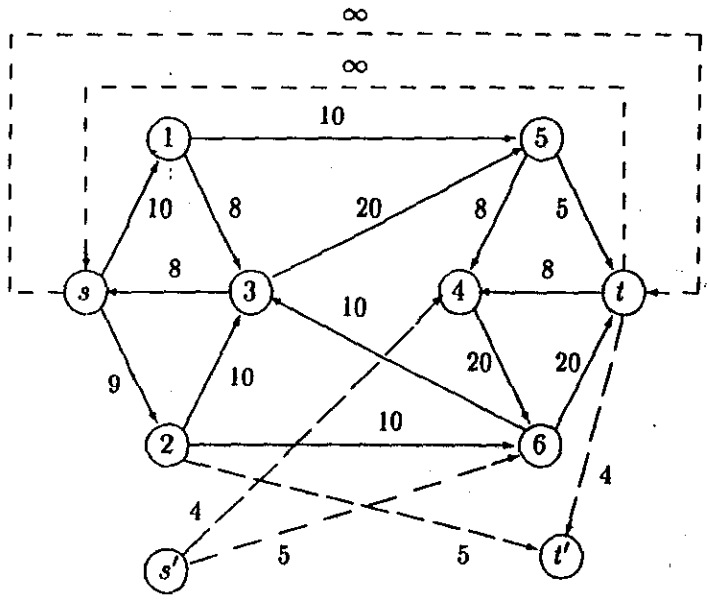
- Para todo $(i, j) \in A$, si $r_{ij} \neq 0$ se agregan a la red R el arco (s', j) con $q'_{s'j} = r_{ij}$; si ya existe este arco se define $q'_{s'j} = q'_{s'j} + r_{ij}$; y el arco (i, t') con $q'_{it'} = r_{ij}$; si ya existe este arco se define $q'_{it'} = q'_{it'} + r_{ij}$.

- $q'_{ij} = q_{ij} - r_{ij}$, para todo $(i,j) \in A$
- $q_{ts} = q_{st} = \infty$
- Las cotas inferiores para el flujo a través de los arcos de R' son todas cero.

Ejemplo:



(Cap. mín., Cap. máx.)



Las etiquetas en esta figura son las capacidades máximas de cada uno de los arcos. La capacidad mínima de todos los arcos es cero.

La razón de poner dos arcos auxiliares con extremos en s y t con distinto sentido es que, al buscar el flujo máximo de s' a t', s y t se convierten en nodos intermedios y debe permitirse la circulación de flujo entre ellos ya sea de entrada o de salida. Al final del capítulo 3. se da un ejemplo en el que al poner solo el arco (t,s) no se encuantra una primera solución factible en una red con cotas mínimas distintas de cero en arcos a pesar de que sí existe tal solución.

En general puede suceder que, al querer agregar un arco auxiliar (i,j) con i, j nodos de la red original, se tengan arcos paralelos en el mismo sentido. Para evitar este problema se puede agregar un nodo auxiliar k, y en lugar de agregar el arco (i,j), agregar los arcos (i,k) y (k,j), ésto es lo que se hace en la implementación del procedimiento para encontrar un primer flujo factible en una red con cotas inferiores distintas de cero en arcos.

Si después de haber aplicado el algoritmo de *Ford-Fulkerson*, para s' y t' en R', el flujo máximo encontrado tiene valor igual a la suma de las cotas inferiores de los arcos de R, puede demostrarse que existe un flujo factible en R.

El siguiente teorema demuestra constructivamente lo anteriormente dicho y proporciona una manera de obtener el flujo factible deseado.

TEOREMA: *Sea F el flujo máximo de valor v' en R' y sea F_{ts} y F_{st} el flujo a través de los arcos (t,s) y (s,t) respectivamente. Si*

$$v' = \sum_{(i,j)\in A} r_{ij}$$

entonces existe un flujo factible de valor $F_{ts} - F_{st}$ en R.

Demostración:

Como F es factible en R' entonces se cumple que:

(i)

$$\sum_{j\in\Gamma^+(i)} F_{ij} - \sum_{k\in\Gamma^-(i)} F_{ki} = \begin{cases} v' & \text{si } i = s' \\ 0 & \text{si } i \neq s', t' \\ -v' & \text{si } i = t' \end{cases}$$

(ii) $\qquad 0 \leq F_{ij} \leq q'_{ij} \quad$ para todo $\quad (i,j) \in A \cup A' \cup \{(t,s)\}$

Por otro lado, por ser

$$v' = \sum_{(i,j)\in A} r_{ij}$$

se tiene que el flujo a través de los arcos de la forma (s',j) y (i,t') es igual a la capacidad máxima de dichos arcos, es decir,

$$F_{s'j} = \sum_{i\neq s'} r_{ij} \qquad y \qquad F_{it'} = \sum_{j\neq t'} r_{ij}$$

Sea f definido como $f_{ij} = F_{ij} + r_{ij}$, para todo $(i,j) \in A$. Por demostrar que f es factible de valor $F_{ts} - F_{st}$ en R.

Para probar ésto se debe verificar que se cumple:

1.

$$\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{k\in\Gamma^-(i)} f_{ki} = \begin{cases} F_{ts} - F_{st} & \text{si } i = s \\ 0 & \text{si } i \neq s,t \\ -(F_{ts} - F_{st}) & \text{si } i = t \end{cases}$$

2. $$r_{ij} \leq f_{ij} \leq q_{ij}, \quad \text{para todo} \quad (i,j) \in A$$

Primeramente se probará 1. para $i \in X$ con $i \neq s,t$:

$$\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{k\in\Gamma^-(i)} f_{ki} = \sum_{j\neq t'} F_{ij} + \sum_{j\neq t'} r_{ij} - \sum_{k\neq s'} F_{ki} - \sum_{k\neq s'} r_{ki} =$$

$$\sum_{j\neq t'} F_{ij} + F_{it'} - \sum_{k\neq s'} F_{ki} - F_{s'i} = \sum_{j} F_{ij} - \sum_{k} F_{ki} = 0 \quad (por\ (i))$$

Ahora, si $i = s$:

$$\sum_{j} f_{sj} - \sum_{k} f_{ks} = \sum_{j\neq t',t} F_{sj} + \sum_{j\neq t',t} r_{sj} - \sum_{k\neq s',t} F_{ks} - \sum_{k\neq s',t} r_{ks} =$$

$$\sum_{j\neq t} F_{sj} - \sum_{k\neq t} F_{ks} = F_{ts} - F_{st} \quad (por\ (i))$$

La prueba de la ecuación de conservación de flujo para t es análoga.

Para probar 2. tenemos de (ii) que

$$0 + r_{ij} \leq F_{ij} + r_{ij} \leq q'_{ij} + r_{ij},$$

pero $f_{ij} = F_{ij} + r_{ij}$ y $q'_{ij} = q_{ij} - r_{ij}$, por lo tanto:

$$r_{ij} \leq f_{ij} \leq q_{ij}$$

Luego f es un flujo factible de valor $F_{ts} - F_{st}$ en R

■

Sin embargo, no siempre existe un flujo factible en redes con éstas características, ésto será probado en el siguiente teorema.

Sea R' definida anteriormente.

TEOREMA: *Sea F el flujo máximo de s' a t', de valor v' en R'. Si*

$$v' \neq \sum_{(i,j)\in A} r_{ij},$$

entonces no existe ningún flujo factible en R.

Demostración:

Se demostrará el teorema por contradicción. Supongamos que existe un flujo factible de valor v en R. Entonces se cumplen:

(i)

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

(ii) $$r_{ij} \leq f_{ij} \leq q_{ij}, \text{ para todo } (i,j) \in A$$

Defínase el flujo F en R' como:

- $F_{ij} = f_{ij} - r_{ij}$, para todo $(i,j) \in A$
- $F_{ts} - F_{st} = v$
- $$F_{it'} = \sum_j r_{ij},$$

  donde r_{ij} es la cota inferior de los arcos de la forma $(i,j) \in A$
- $$F_{s'j} = \sum_k r_{kj},$$

  donde r_{kj} es la cota inferior de los arcos de la forma $(k,j) \in A$

Por demostrar que F es un flujo factible, en R', de valor

$$v' = \sum_{(i,j)\in A} r_{ij}.$$

Para ello deberá verificarse que se cumplen:

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

2. $$0 \leq F_{ij} \leq q'_{ij}, \text{ para todo } (i,j) \in A \cup A' \cup \{(t,s),(s,t)\}$$

Probaremos 1. para $i \neq s', t', s, t$:

$$\sum_j F_{ij} - \sum_k F_{ki} = F_{it'} + \sum_{j \neq t'} F_{ij} - F_{s'i} - \sum_{k \neq s'} F_{ki} =$$

$$F_{it'} + \sum_{j \neq t'} f_{ij} - \sum_{j \neq t'} r_{ij} - F_{s'i} - \sum_{k \neq s'} f_{ki} + \sum_{k \neq s'} r_{ki}$$

como $r_{it'} = r_{s'i} = 0$ para todo $i \in X$ entonces:

$$\sum_{j \neq t'} r_{ij} = \sum_j r_{ij} \qquad y \qquad \sum_{k \neq s'} r_{ki} = \sum_k r_{ki}$$

$$\sum_j F_{ij} - \sum_k F_{ki} = F_{it'} + \sum_{j \neq t'} f_{ij} - \sum_j r_{ij} - F_{s'i} - \sum_{k \neq s'} f_{ki} + \sum_k r_{ki}$$

ahora como

$$F_{it'} = \sum_j r_{ij} \qquad y \qquad F_{s'i} = \sum_k r_{ki}$$

$$\sum_j F_{ij} - \sum_k F_{ki} = \sum_{j \neq t'} f_{ij} - \sum_{k \neq s'} f_{ki} = 0 \quad (por\ (i))$$

Sea $i = s$:

$$\sum_j F_{sj} - \sum_k F_{ks} = F_{st'} + F_{st} + \sum_{j \neq t', j \neq t} F_{sj} - F_{s's} - F_{ts} - \sum_{k \neq s', k \neq t} F_{ks} =$$

$$F_{st'} + F_{st} + \sum_{j \neq t', j \neq t} f_{sj} - \sum_{j \neq t', j \neq t} r_{sj} - F_{s's} - F_{ts} - \sum_{k \neq s', k \neq t} f_{ks} + \sum_{k \neq s', k \neq t} r_{ks}$$

como $r_{st'} = r_{s's} = r_{ts} = r_{st} = 0$

$$F_{st'} + F_{st} + \sum_{j \neq t', j \neq t} f_{sj} - \sum_j r_{sj} - F_{s's} - F_{ts} - \sum_{k \neq s', k \neq t} f_{ks} + \sum_k r_{ks}$$

ahora como

$$F_{st'} = \sum_j r_{sj} \qquad y \qquad F_{s's} = \sum_k r_{ks}$$

$$\sum_j F_{sj} - \sum_k F_{ks} = \sum_{j\neq t',j\neq t} f_{sj} - \sum_{k\neq s',k\neq t} f_{ks} + F_{st} - F_{ts} = v - v = 0 \quad (por\ (i))$$

De forma análoga se prueba 1. para $i = t$.

Tenemos también que $\Gamma^-(s') = \Gamma^+(t') = \phi$ por lo que:

$$\sum_j F_{s'j} - \sum_k F_{ks'} = \sum_j F_{s'j} = \sum_{(i,j)\in A} r_{ij}$$

y

$$\sum_j F_{t'j} - \sum_k F_{kt'} = -\sum_k F_{kt'} = -\sum_{(i,j)\in A} r_{ij}$$

Ahora probaremos 2.

Por (ii) tenemos que:

$$r_{ij} - r_{ij} \leq f_{ij} - r_{ij} \leq q_{ij} - r_{ij}$$

como $F_{ij} = f_{ij} - r_{ij}$ y $q'_{ij} = q_{ij} - r_{ij}$

$$0 \leq F_{ij} \leq q'_{ij} \text{ para todo } (i,j) \in A.$$

El flujo definido para los arcos de la forma (i,t') y (s',j) es igual a su capacidad y la capacidad de los arcos (t,s), (s,t) es infinita. Luego, 2. se cumple para todo $(i,j) \in A \cup A' \cup \{(t,s),(s,t)\}$, ésto prueba que F es un flujo factible en R' de valor

$$\sum_{(i,j)\in A} r_{ij}.$$

En la red R' el conjunto de arcos con nodo inicial s' forman una cortadura de capacidad

$$\sum_{(i,j)\in A} r_{ij}$$

por lo que F es un flujo máximo en R'. Hemos llegado a una contradicción por lo tanto se concluye la afirmación del teorema.

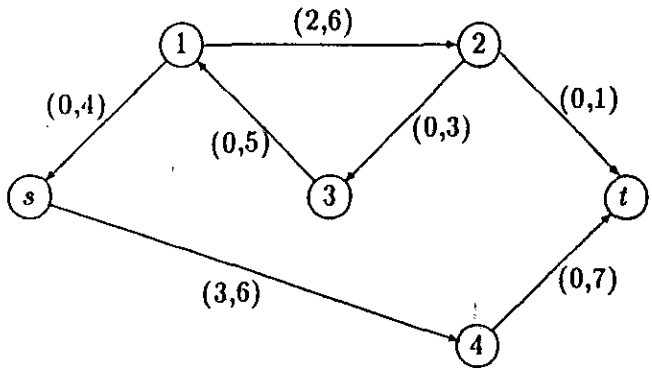
■

Una vez que se ha resuelto el problema de encontrar un flujo factible en la red R, si existe, se procede a encontrar el flujo máximo en la esta red.

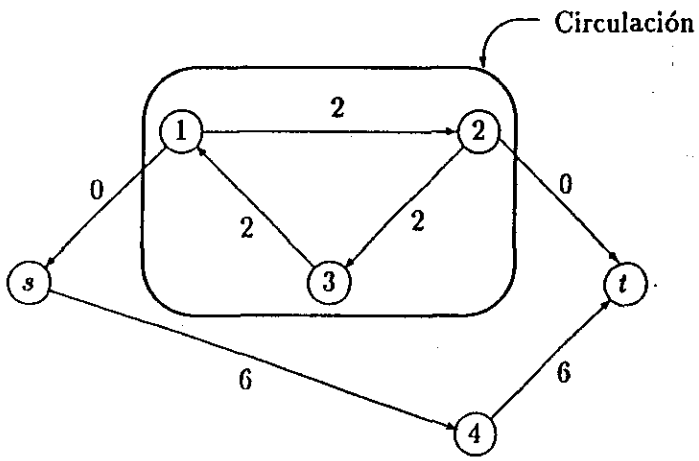
Para ésto, se aplica nuevamente el algoritmo de *Ford-Fulkerson*, tomando como primer flujo factible, el encontrado con el procedimiento anterior y con las restricciones del problema original en cada arco.

Cabe hacer notar que el procedimiento anterior puede dar orígen a circulaciones de flujo en una red, por ejemplo:

Supóngase que en la red de la siguiente figura se requiere encontrar el flujo máximo de s a t satisfaciendo las restricciones establecidas en ella. Las etiquetas de los arcos son (r_{ij}, q_{ij})



Al aplicar el algoritmo de *Ford-Fulkerson* para encontrar una primera solución factible y después para maximizar el flujo de s a t en la red anterior se obtuvo el flujo ilustrado en la siguiente figura donde se muestra una circulación de flujo por los nodos 1, 2 y 3.



Estructura de datos

La estructura de datos adicional usada para implementar esta variante del problema de flujo máximo es la siguiente:

- La información de cuáles arcos tienen restricciones mínimas distintas de cero se guarda en una pila para la cual se utilizan registros que contengan:

- La dirección del nodo inicial del arco.
- La dirección del nodo final del arco.
- La capacidad mínima del arco.
- La dirección del siguiente elemento en la pila.

En la implementación de esta variante se utiliza la pila con la información antes mencionada para, después de haber agregado los nodos s' y t' auxiliares, agregar los arcos auxiliares desde estos nodos a los nodos inicial o final de los arcos guardados en la pila según sea el caso.

2.3 Restricciones mínimas y máximas en nodos

En esta sección expondremos por separado los dos tipos de restricciones que se pueden presentar en nodos, aunque en la implementación los veremos como un caso singular, es decir, cuando se presenten sólo restricciones máximas en nodos, la restricción mínima será cero para todo nodo restringido y cuando se presenten restricciones mínimas distintas de cero, la capacidad máxima de cada nodo restringido será finita. Las modificaciones en la implementación para el caso en que la capacidad mínima en nodos es distinta de cero y la máxima infinita, son directas pero no se realizan en la versión presentada en este trabajo.

2.3.1 Restricciones máximas

Sea $R = [X, A, q, k]$ una red donde q es la función de capacidad máxima asociada a los arcos de R y k es una función que asocia a cada nodo la capacidad máxima de flujo que puede pasar por él.

Un flujo factible f en R es una función que cumple:

1.
$$\sum_{j\in\Gamma^+(i)} f_{ij} - \sum_{k\in\Gamma^-(i)} f_{ki} = \begin{cases} v & \text{si } i = s \\ 0 & \text{si } i \neq s,t \\ -v & \text{si } i = t \end{cases}$$

2.
$$0 \leq f_{ij} \leq q_{ij} \quad \forall \quad (i,j) \in A$$

y además

3.
$$\sum_{j\in\Gamma^-(i)} f_{ji} \leq k(i) \quad \forall \ i \in X$$

Para encontrar el flujo máximo entre s y t en R, utilizaremos una red auxiliar R' construida de la siguiente manera:

A cada $i \in X$ tal que $k(i) < \infty$ le corresponde un arco (i, i') en R' con capacidad máxima $k(i)$; todos lod arcos con extremo final i en R corresponden, en R', arcos de la forma (j', i) si $k(j) < \infty$, si $k(j) = \infty$ los arcos permanecen igual. Todos los arcos con extremo inicial i en R corresponden, en R', a arcos de la forma (i', j). Todos los arcos de R conservan su capacidad máxima.

A partir de cualquier flujo factible en R' puede definirse otro del mismo valor en R y viceversa. Sea f' un flujo factible de valor v en R' y definase f como $f_{ij} = f'_{i'j}$, si $k(i) \leq \infty$ en R y $f_{ij} = f'_{ij}$ en otro caso. Verifiquemos que f es un flujo factible en R:

Si $i \in X$ tiene capacidad máxima $k(i) \leq \infty$, se tiene que:

$$\sum_{j \in \Gamma^+(i)} f_{ij} - \sum_{k \in \Gamma^-(i)} f_{ki} = \sum_{j \in \Gamma^+(i')} f'_{i'j} - \sum_{k \in \Gamma^-(i)} f'_{ki} =$$

$$\sum_{j \in \Gamma^+(i')} f'_{i'j} - f'_{ii'} = \begin{cases} v & \text{si } i = s \\ 0 & \text{si } i \neq s, t \\ -v & \text{si } i = t \end{cases}$$

Por otro lado:

$$\sum_{j \in \Gamma^-(i)} f_{ji} = \sum_{j \in \Gamma^-(i)} f'_{ji} = f'_{ii'} \leq k(i)$$

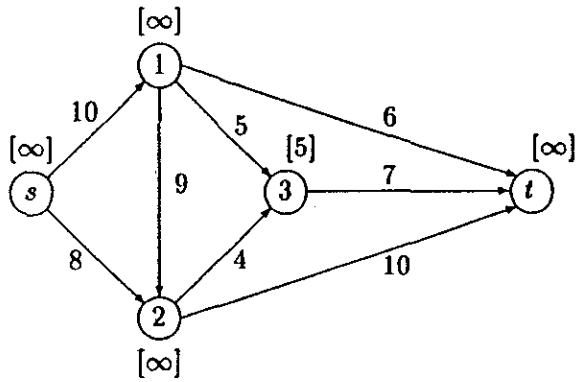
luego, f es un flujo factible de valor v en R.

Determinar el flujo máximo en R es equivalente entonces, a determinarlo en R'. Por lo tanto, puede concluirse, que para resolver este tipo de problemas puede utilizarse el algoritmo de *Ford-Fulkerson*.

Para este tipo de problemas la cortadura mínima de R' puede corresponder a un conjunto de arcos y nodos de R.

Ejemplo

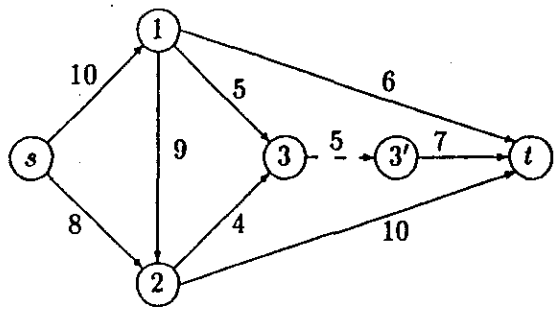
Supongamos que en la siguiente red deseamos encontrar el flujo máximo entre s y t.



[Capacidad máxima]

Las etiquetas de los arcos son las capacidades máximas de cada uno. Todos los arcos tienen capacidad mínima cero.

El nodo 3 es el único que tiene capacidad máxima finita por lo tanto es el que "partiremos", construyendo así, la red auxiliar para resolver el problema de encontrar el flujo máximo en la red original.



2.3.2 Restricciones mínimas

Sea $R = [X, A, q, h]$ donde q es la capacidad máxima asociada a cada arco y h es una función que asocia a cada nodo la capacidad mínima que se requiere que pase por él.

Un flujo factible f en R es aquel que cumple con 1. (a) y 1. (b) y además

$$\sum_{j \in \Gamma^-(i)} f_{ji} \geq h(i) \quad \forall \ i \in X$$

Para resolver el problema de encontrar el flujo máximo en R utilizaremos R' construida como sigue:

Si $i \in X$ es tal que $h(i) > 0$, se le asocia un arco (i, i') con capacidad mínima $h(i)$ y máxima ∞. Todos los arcos con extremo final i en R corresponden, en R', a arcos de la forma (j', i) si $h(j) > 0$;

si $h(i) = 0$ los arcos permanecen igual. Todos los arcos con extremo inicial i en R corresponden, en R', a arcos de la forma (i', j). Todos los arcos de R conservan su capacidad mínima y máxima.

La red R' construída de esta manera, es una tal que, el encontrar un flujo factible en ella es encontrar f' tal que satisfaga las ecuaciones de conservación de flujo y además

$$r_{ij} \leq f'_{ij} \leq q'_{ij} \quad \forall \ (i,j) \in A'$$

donde $A' = A \cup \{(i,i') : h(i) > 0\} - \{(i,j) : h(i) > 0\} \cup \{(i',j) : h(i) > 0, j \in \Gamma^+(i)\}$

$$r_{ij} = \begin{cases} h(i) & \text{si } j = i' \\ 0 & \text{en otro caso} \end{cases}$$

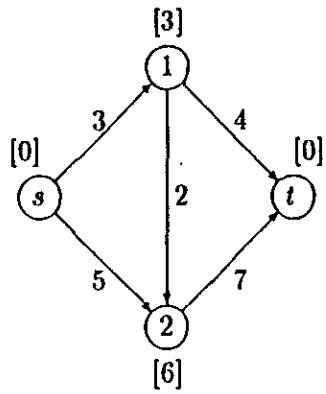
y

$$q'_{ij} = \begin{cases} \infty & \text{si } j = i' \\ q_{ij} & \text{en otro caso} \end{cases}$$

Nótese que encontrar el flujo máximo en R' es un problema como los de la sección 3.2., por lo tanto, podemos resolverlo aplicando el método que se menciona en tal sección.

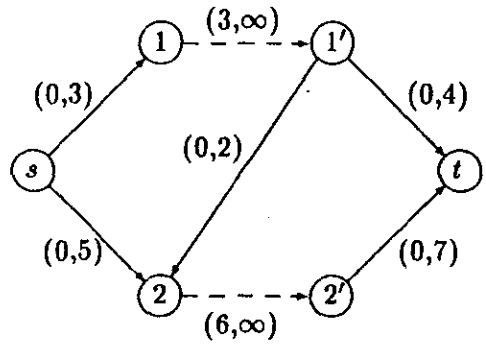
Una vez encontrado el flujo máximo en R' se puede demostrar que es también un flujo máximo en R de manera análoga que en 3.3.1.

Ejemplo:



Las etiquetas de los arcos son la capacidad máxima de cada uno de ellos y la de los nodos es la capacidad mínima de flujo requerido para ellos.

Los nodos 1 y 2 son los que tienen capacidad mínima distinta de cero por lo que serán "partidos" para formar la siguiente red auxiliar.



Véase la sección 3.2. para resolver el problema en esta red.

Estructura de datos

La estructura de datos adicional usada para implementar esta variante del problema de flujo máximo es la siguiente:

- La información de cuáles nodos tienen restricciones mínimas y máximas distintas de cero se guarda en una pila para la cual se utilizan registros que contengan:
  - El número del nodo restringido.
  - Capacidad mínima del nodo.
  - Capacidad máxima del nodo.
  - Dirección del nodo auxiliar que hace pareja con el nodo restringido.
  - Dirección del siguiente elemento en la pila.

Con esta pila se implementa la variante del problema aquí presentada tomando la información de los nodos restringidos en la pila para agregar arcos auxiliares en la gráfica como se mencionó en este caso y si existen nodos con restricción mínima se procede a agregar arcos auxiliares como se mencionó en la sección anterior.

2.4 Implementación de las variantes del problema

Presentaremos en esta sección el procedimiento a seguir para la implementación de todas las variantes del problema de flujo máximo. Primeramente mostraremos el programa principal con sus dos secciones, la de captura y la de la organización de la información y aplicación del algoritmo teniendo cuidado en los detalles importantes de la implementación. después presentamos los procedimientos, que en sí, realizan las conexiones auxiliares de cada variante.

Programa Principal

Captura

Se realiza la captura de la información de la red y la información adicional que indique si el problema al que se quiere aplicar el algortimo tiene o no variantes como varios fuentes y destinos, restricciones mínimas y máximas en nodos o restricciones mínimas en arcos gardando todos los datos para después hacer los cambios pertinentes en la red, aplicar el algoritmo y recuperar la gráfica original, si se encontró solución al problema. Esto se hace de la siguiente manera:

1. Se captura un indicador con valor cero o uno, que dirá si en la red existen varios fuentes y varios destinos.
2. Se capturan los arcos originales de la red agregando a la gráfica de estructura de datos los nodos inicial y final del arco capturado, después se agrega el arco como sucesor del nodo inicial y como antecesor del nodo final.
3. Se revisa si el arco tiene capacidad mínima distinta de cero, si así sucede se mete el arco a la pila de arcos restringidos.
4. Se capturan los nodos que tienen capacidades mínimas y máximas en la red y se meten a la pila de nodos restringidos.
5. Se revisa el indicador de varios fuentes y varios destinos y si:
   - Es cero, es decir, si no hay varios fuentes y destinos, se captura el número del nodo fuente y el número del nodo destino.
   - Es uno, es decir, si existen varios fuentes y destinos, se capturan primero los nodos fuentes y se van metiendo en la pila de nodos fuentes uno por uno, se hace lo mismo para los nodos destinos. Se agregan al final de la gráfica los nodos s (fuente) y t (destino) auxiliares.
6. termina la captura

Organización de la información y aplicación del algoritmo

Cuando en la gráfica organizamos la información para hacer la conexiones auxiliares adecuadamente, seguimos un orden al tomar los datos, éste debe ser:

Primero hacer los cambios necesarios, en la gráfica, en el caso de que existan restricciones máximas en nodos. Después, hacer los cambios en el caso de que existan en la red nodos o arcos con restricción mínima. Y por último, modificar la gráfica en el caso de que la red tenga varios fuentes y varios destinos.

Teniendo mucho cuidado al aplicar el algoritmo, para hacerlo en los nodos adecuados, se obtiene una primera solución factible, si es que se requiere, y se aplica de nuevo el algoritmo para obtener el flujo máximo. Se despliega la solución después de haber recuperado la información original de la red.

A continuación mostramos el procedimiento.

1. Se llama a una función que agrega al final de la gráfica los nodos auxiliares que hacen pareja con los nodos restringidos y pone los arcos que los conectan. A estos nodos auxiliares los llamaremos "cuates".
2. Si existen arcos o nodos con restricción mínima, se agregan al final de la gráfica los nodos s' y t' auxiliares.
3. Se revisa si algún nodo en la pila de nodos restringidos tiene capacidad mínima, si así es, se mete el arco que conecta al nodo restringido con su cuate a la pila de arcos con restricción mínima.
4. Se llama a una función que agrega los arcos auxiliares, en el caso que existan arcos con restricción mínima, con extremos en s' y t' según sea el caso.
5. Se llama a una función que agrega los arcos auxiliares en el caso de que haya en la red varios fuentes y varios destinos.
6. Si el nodo t es restringido el algoritmo se deberá aplicar a su cuate. Para ésto se toma como nodo t al cuate.
7. Si la pila de arcos con restricción no está vacía se llama a una función que encuentra una solución factible en la red. En esta función se agregan las conexiones entre los nodos s y t. Si se encuentra la solución, se quitan las conexiones puestas si existen arcos o nodos con restricción mínima distinta de cero y se hacen los cambios pertinentes.
8. Se aplica el algoritmo a la gráfica para encontrar el flujo máximo entre los nodos s y t .
9. Se liberan todas las conexiones auxiliares.
10. Se escribe la solución obtenida.

Procedimiento para agregar nodos y arcos auxiliares si existen nodos con restricciones

1. Se toma un elemento de la pila de nodos restringidos.
2. Se agrega al final de la gráfica el nodo cuate del nodo restringido que sacamos de la pila.
3. Se cambian los sucesores del nodo restringido a sucesores de su cuate.
4. Se agrega el nodo cuate como sucesor del nodo restringido.
5. Se busca cada nodo sucesor del nodo restringido y se le cambia como antecesor al nodo restringido por el nodo cuate.
6. Se agrega el nodo restringido como antecesor del nodo cuate.

A continuación mostramos el procedimiento.

1. Se llama a una función que agrega al final de la gráfica los nodos auxiliares que hacen pareja con los nodos restringidos y pone los arcos que los conectan. A estos nodos auxiliares los llamaremos "cuates".
2. Si existen arcos o nodos con restricción mínima, se agregan al final de la gráfica los nodos s' y t' auxiliares.
3. Se revisa si algún nodo en la pila de nodos restringidos tiene capacidad mínima, si así es, se mete el arco que conecta al nodo restringido con su cuate a la pila de arcos con restricción mínima.
4. Se llama a una función que agrega los arcos auxiliares, en el caso que existan arcos con restricción mínima, con extremos en s' y t' según sea el caso.
5. Se llama a una función que agrega los arcos auxiliares en el caso de que haya en la red varios fuentes y varios destinos.
6. Si el nodo t es restringido el algoritmo se deberá aplicar a su cuate. Para ésto se toma como nodo t al cuate.
7. Si la pila de arcos con restricción no está vacía se llama a una función que encuentra una solución factible en la red. En esta función se agregan las conexiones entre los nodos s y t. Si se encuentra la solución, se quitan las conexiones puestas si existen arcos o nodos con restricción mínima distinta de cero y se hacen los cambios pertinentes.
8. Se aplica el algoritmo a la gráfica para encontrar el flujo máximo entre los nodos s y t .
9. Se liberan todas las conexiones auxiliares.
10. Se escribe la solución obtenida.

Procedimiento para agregar nodos y arcos auxiliares si existen nodos con restricciones

1. Se toma un elemento de la pila de nodos restringidos.
2. Se agrega al final de la gráfica el nodo cuate del nodo restringido que sacamos de la pila.
3. Se cambian los sucesores del nodo restringido a sucesores de su cuate.
4. Se agrega el nodo cuate como sucesor del nodo restringido.
5. Se busca cada nodo sucesor del nodo restringido y se le cambia como antecesor al nodo restringido por el nodo cuate.
6. Se agrega el nodo restringido como antecesor del nodo cuate.

Procedimiento para agregar arcos auxiliares si existen arcos con restricciones

1. Se toma un elemento de la pila de arcos restringidos.
2. Se revisa si algún nodo restringido es el extremo inicial del arco restringido, ya que si:
   - Sí es y el extremo final es el nodo cuate o si no es, el arco auxiliar se agregará tomando el nodo restringido y su extremo final.
   - El arco restringido no es el que conecta al nodo restringido y su cuate, el arco auxiliar se agregará tomando el nodo cuate y su extremo final.
3. Se busca si ya existe el nodo t' como sucesor del nodo inicial del arco restringido y si:
   - Sí existe como sucesor se le suma a su capacidad máxima la capacidad mínima del arco restringido.
   - No existe como sucesor se agrega el arco con extremo inicial el nodo inicial del arco restringido y final el nodo t' con capacidad máxima la capacidad mínima del arco restringido.
4. Se busca si ya existe el nodo s' como antecesor del nodo final del arco restringido y si:
   - Sí existe como antecesor se le suma a su capacidad máxima la capacidad mínima del arco restringido.
   - No existe como antecesor se agrega el arco con extremo inicial el nodo s' y final el nodo el nodo final del arco restringido con capacidad máxima la capacidad mínima del arco restringido.
5. La capacidad máxima del arco restringido se cambia por la diferencia de la que tenía anteriormente y su capacidad mínima.
6. Se van sumando las capacidades mínimas de todos los arcos restringidos.

Procedimiento para agregar arcos auxiliares si existen varios fuentes y varios destinos

1. Se toma el nodo fuente auxiliar y se le agrega a la lista de sus arcos sucesores tantos arcos como nodos fuentes tenga la red, con extremo final de cada arco el número del nodo fuente y capacidad máxima infinita
2. A cada nodo fuente se le agrega un arco antecesor con extremo inicial el nodo fuente auxiliar y capacidad mínima cero.
3. Se revisa si algún nodo destino es un nodo restringido ya que si
   - Si lo es, al nodo cuate se le agrega un arco sucesor con extremo final el nodo destino y capacidad máxima infinita. Se agrega al nodo t auxiliar un arco antecesor con extremo inicial el número de nodo del cuate del destino y capacidad mínima cero.

Procedimiento para agregar arcos auxiliares si existen arcos con restricciones

1. Se toma un elemento de la pila de arcos restringidos.
2. Se revisa si algún nodo restringido es el extremo inicial del arco restringido, ya que si:
   - Sí es y el extremo final es el nodo cuate o si no es, el arco auxiliar se agregará tomando el nodo restringido y su extremo final.
   - El arco restringido no es el que conecta al nodo restringido y su cuate, el arco auxiliar se agregará tomando el nodo cuate y su extremo final.
3. Se busca si ya existe el nodo t' como sucesor del nodo inicial del arco restringido y si:
   - Sí existe como sucesor se le suma a su capacidad máxima la capacidad mínima del arco restringido.
   - No existe como sucesor se agrega el arco con extremo inicial el nodo inicial del arco restringido y final el nodo t' con capacidad máxima la capacidad mínima del arco restringido.
4. Se busca si ya existe el nodo s' como antecesor del nodo final del arco restringido y si:
   - Sí existe como antecesor se le suma a su capacidad máxima la capacidad mínima del arco restringido.
   - No existe como antecesor se agrega el arco con extremo inicial el nodo s' y final el nodo el nodo final del arco restringido con capacidad máxima la capacidad mínima del arco restringido.
5. La capacidad máxima del arco restringido se cambia por la diferencia de la que tenía anteriormente y su capacidad mínima.
6. Se van sumando las capacidades mínimas de todos los arcos restringidos.

Procedimiento para agregar arcos auxiliares si existen varios fuentes y varios destinos

1. Se toma el nodo fuente auxiliar y se le agrega a la lista de sus arcos sucesores tantos arcos como nodos fuentes tenga la red, con extremo final de cada arco el número del nodo fuente y capacidad máxima infinita
2. A cada nodo fuente se le agrega un arco antecesor con extremo inicial el nodo fuente auxiliar y capacidad mínima cero.
3. Se revisa si algún nodo destino es un nodo restringido ya que si
   - Si lo es, al nodo cuate se le agrega un arco sucesor con extremo final el nodo destino y capacidad máxima infinita. Se agrega al nodo t auxiliar un arco antecesor con extremo inicial el número de nodo del cuate del destino y capacidad mínima cero.

- Si no lo es, al nodo destino se le agrega un arco sucesor con extremo final el nodo destino y capacidad máxima infinita. Se agrega al nodo t auxiliar un arco antecesor con extremo inicial el número del nodo destino y capacidad mínima cero.

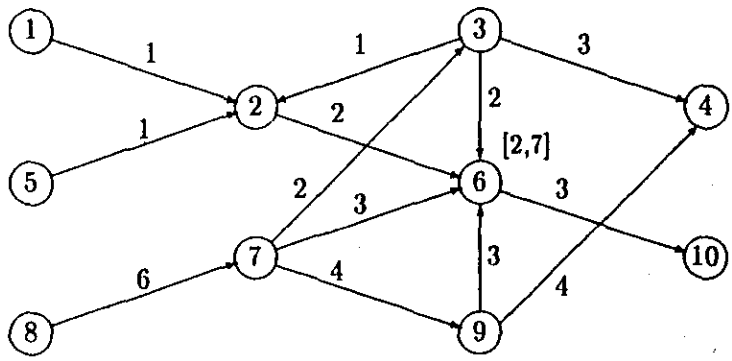
En este trabajo se anexan los listados del programa principal y las funciones más importantes de la implementación computacional del algoritmo de *Ford-Fulkerson*.

A continuación presentaremos un ejemplo de una red en la cual se requiere encontrar el flujo máximo ilustrando el procedimiento anterior.

2.5 Ejemplo

Encontrar el flujo máximo en la red de la siguiente figura satisfaciendo las restricciones establecidas para el flujo a través de ella.

Los nodos fuente en la figura son el 1,5 y 8 mientras que los destino son el 4 y 10.



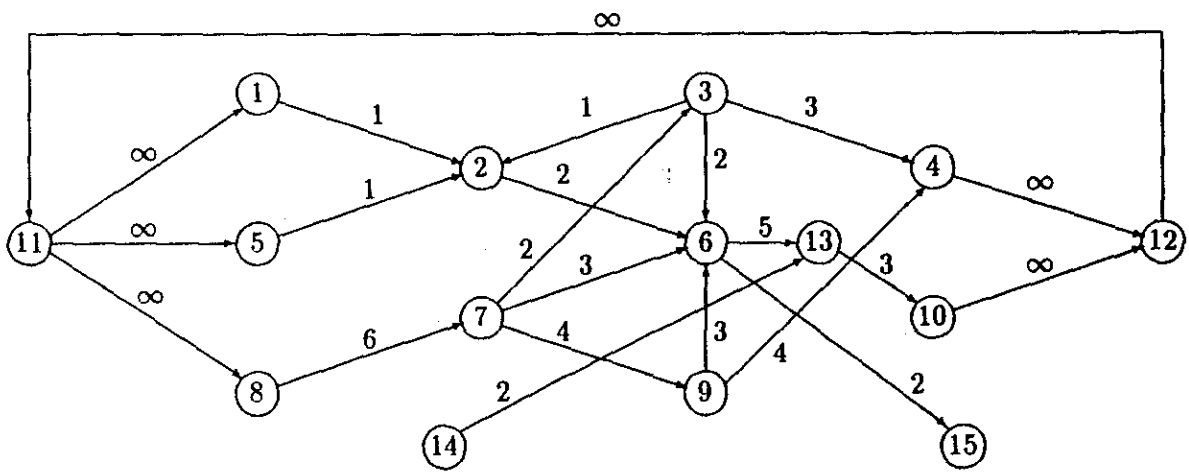
El valor asignado a los arcos es su capacidad máxima y la etiqueta del nodo 6 es [Capacidad mínima, Capacidad máxima].

En esta red se presentan todas las restricciones posibles para flujo, excepto en arcos. En ella se puede ilustrar el procedimiento de la sección anterior ya que el caso en que se presentan restricciones en arcos queda ilustrado cuando existen restricciones mínimas en nodos porque se realizan las mismas modificaciones en la red.

Solución

Dado que en la red existen varios fuentes y varios destinos agregamos a ella los nodos 11 y 12 que serán el fuente y el destino auxilares respectivamente. Como el 6 es el único nodo restringido agregamos el nodo 13 que será su cuate y el arco (6,13) con capacidad mínima 2 y máxima 7, ahora existe un arco con restricción, por lo tanto, se agregan los nodos 14 y 15, s' y t' respectivamente, y los arcos (14,13) y (6,15) ambos con capacidad máxima 2 y la nueva capacidad máxima del arco (6,13) será 5. Se agregan los arcos (11,1), (11,5), (11,8), (4,12), (10,12) y (12,11) todos con capacidad máxima ∞. Todos los arcos, originales y auxiliares tienen restricción mínima cero.

La red modificada es:

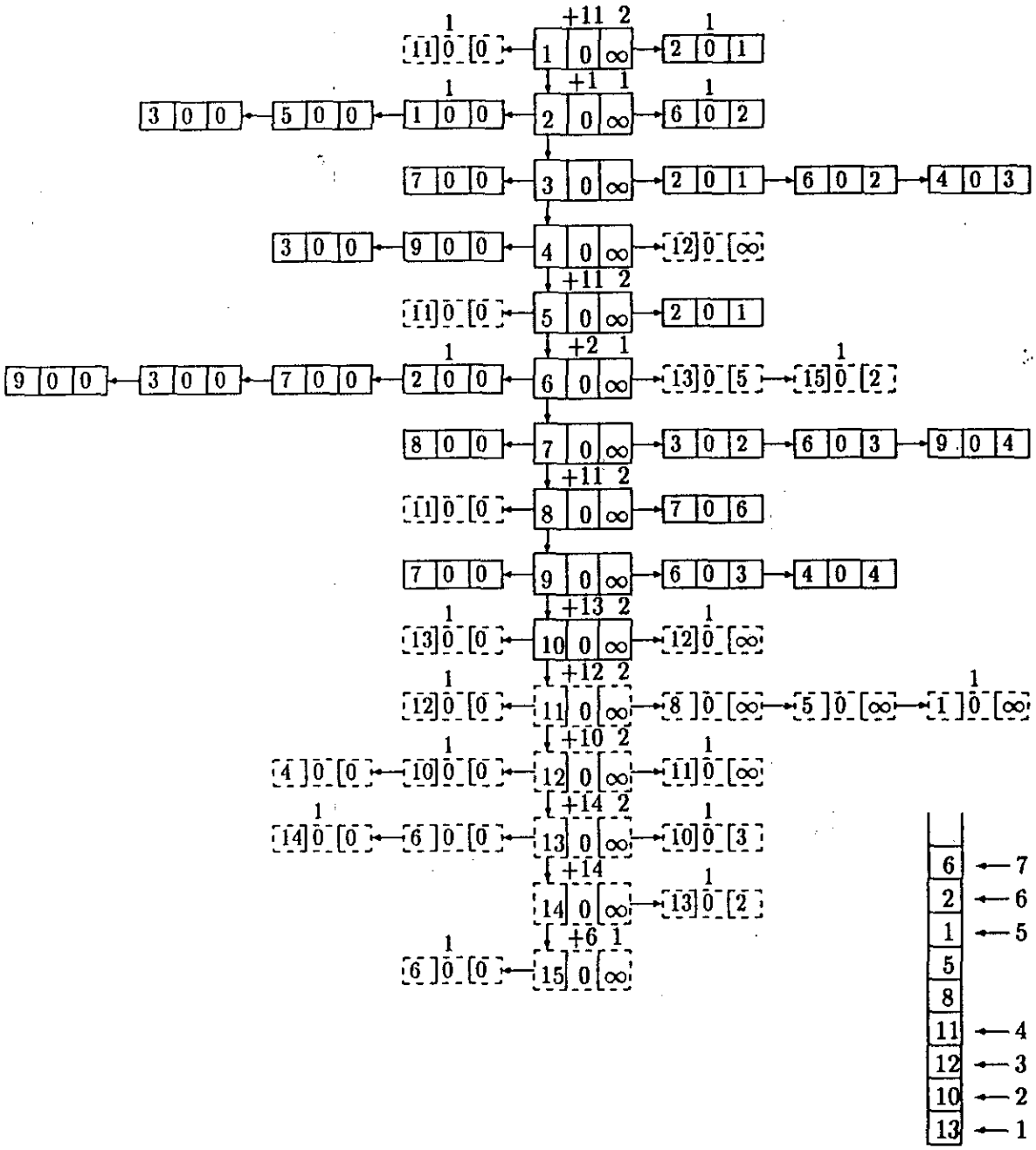


En este caso buscaremos un primer flujo factible en la red aplicando el algoritmo *Ford-Fulkerson* a los nodos 14 como fuente y 15 como destino.

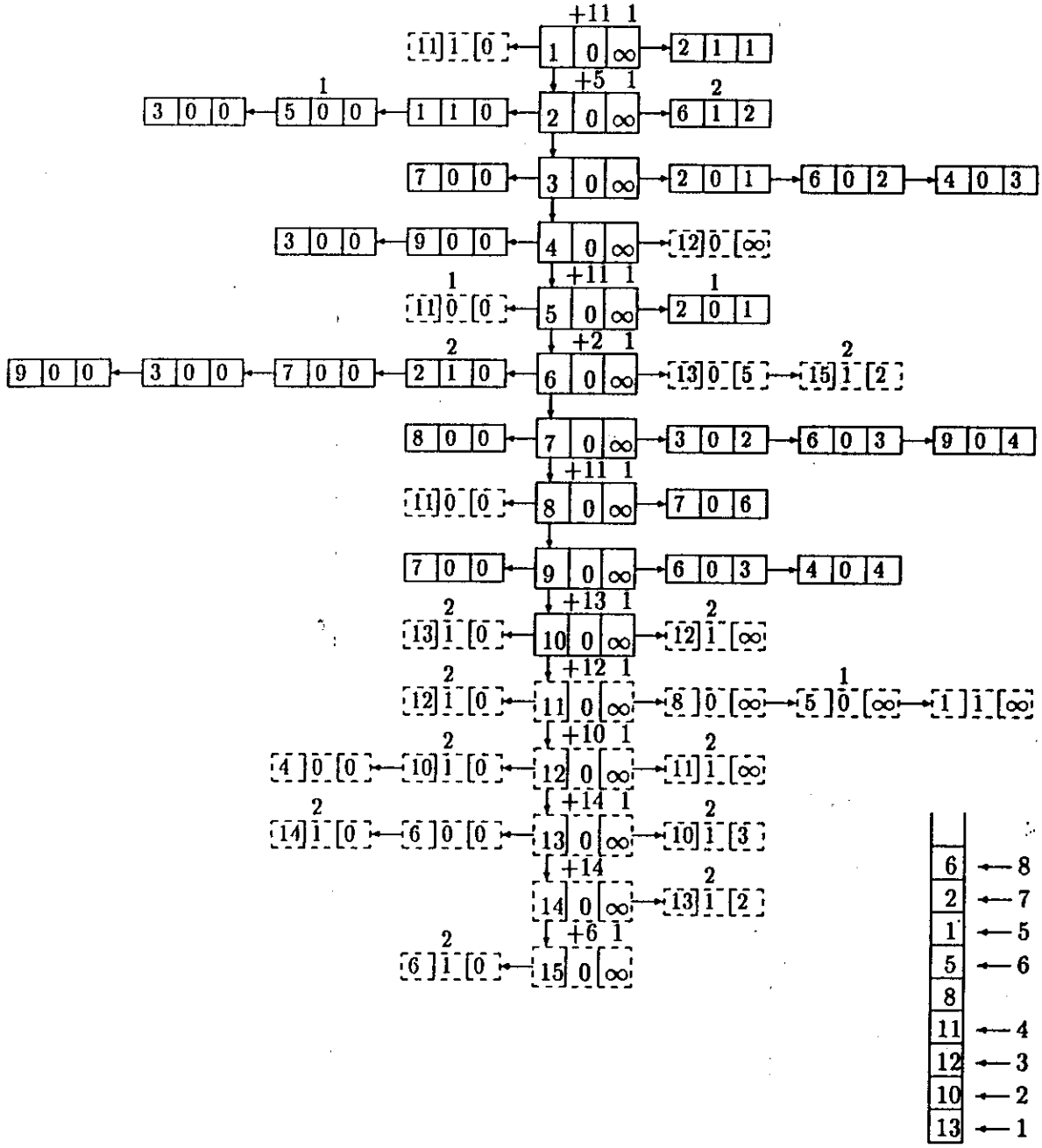
En cada iteración pondremos solo los cambios en las etiquetas de los nodos y en el flujo de los arcos en la gráfica de estructura de datos. Para más detalles en las iteraciones se puede revisar el ejemplo del capítulo 1 ya que es el mismo procedimiento.

A la derecha de los nodos en la pila especificamos el orden en que salieron de ella.

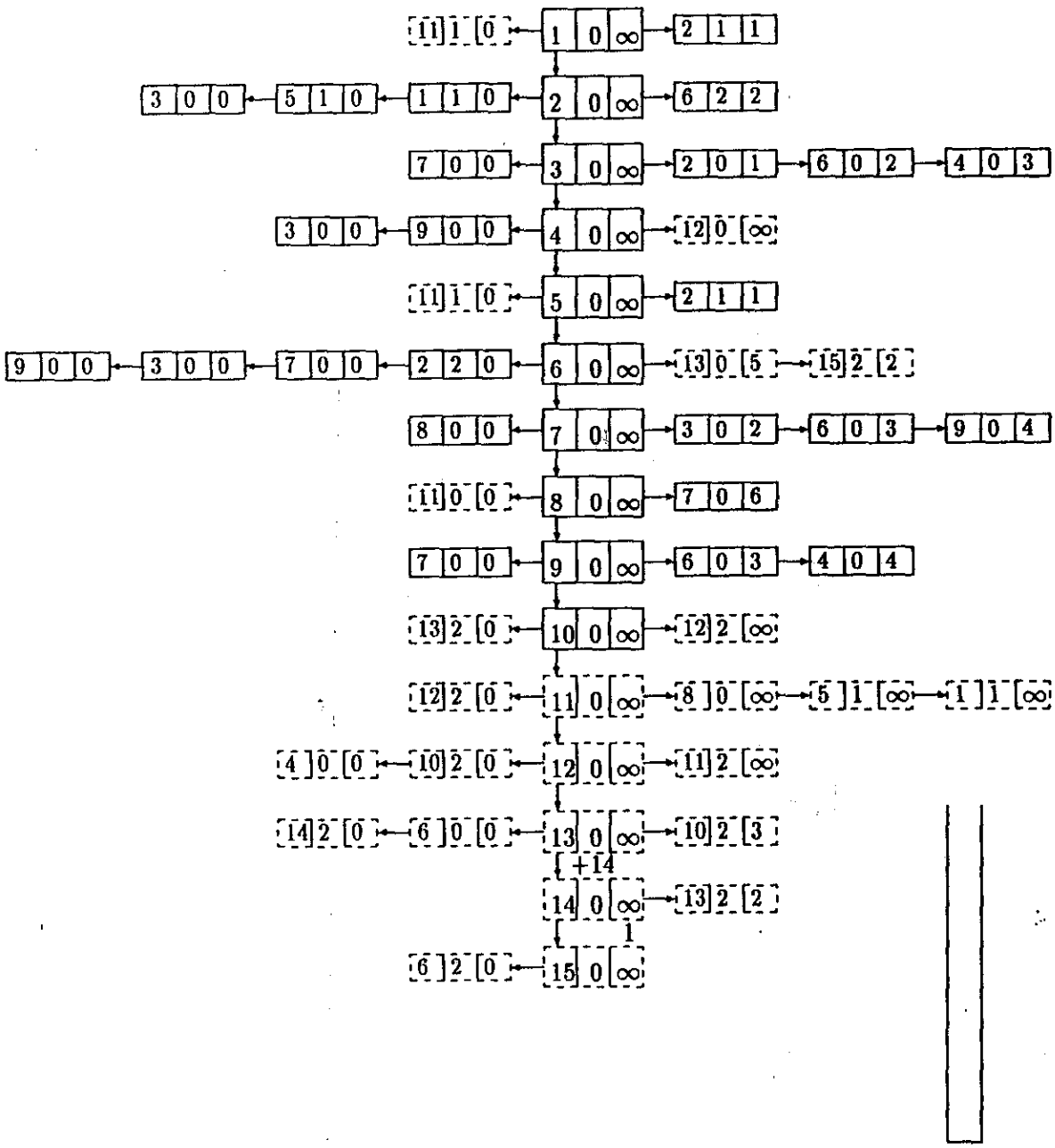
Primera iteración



Segunda iteración



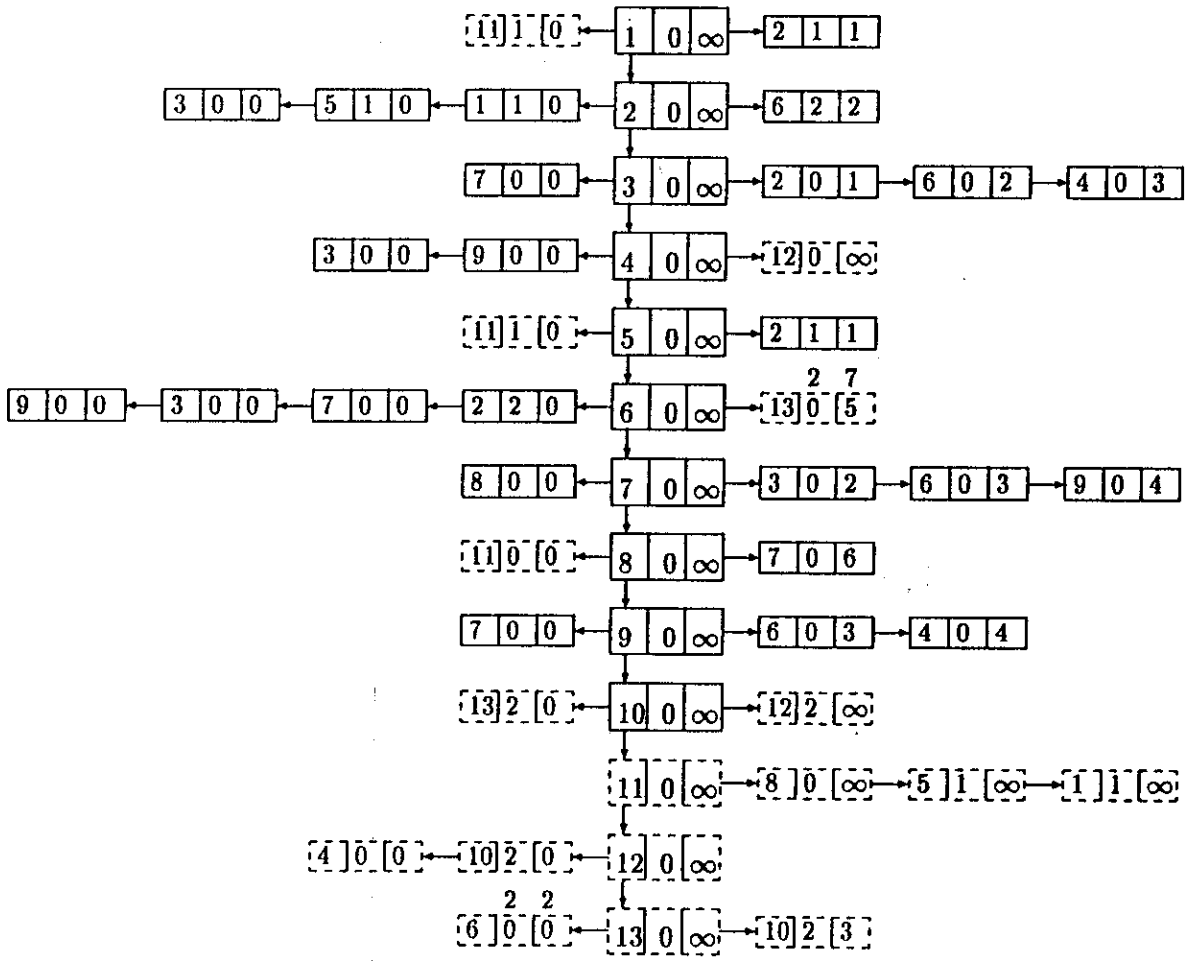
Tercera iteración



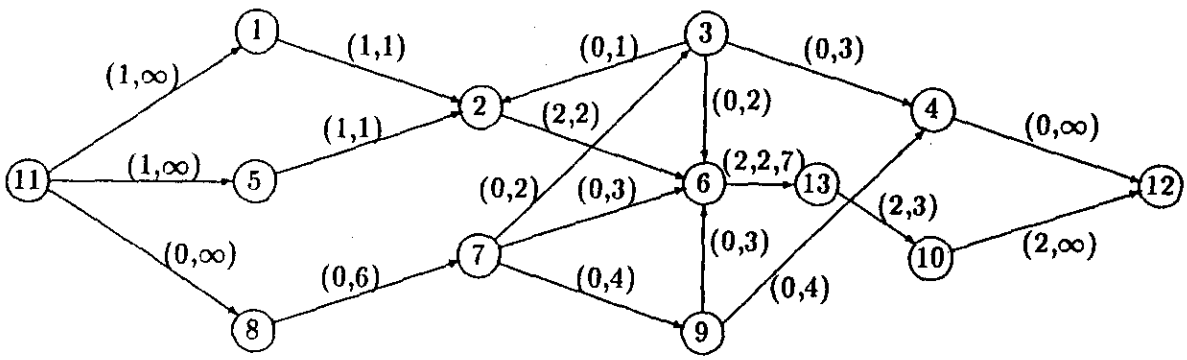
Se ha encontrado el flujo máximo del nodo 14 al 15 y su valor es 2 y como el nodo 6 tiene

capacidad mínima 2, por el primer teorema del la sección 2.2., existe un flujo factible en la red sin los nodos 14 y 15 y sin los arcos (4,13),(6,15) y (12,11) por lo tanto éstos se desconectan de la red.

En la gráfica de estructura de datos actualizada se aplicará de nuevo el algoritmo de *Ford-Fulkerson* para maximizar el flujo en la red original a partir del flujo factible de valor 2. El algoritmo se aplica entre los nodos 11 y 12 después de realizar los cambios en el arco (6,13) poniendole su capacidad mínima y máxima original (en la red auxiliar de capacidad máxima del nodo 6), es decir 2 y 7 respectivamente. Los cambios se ven encima de las casillas correspondientes.

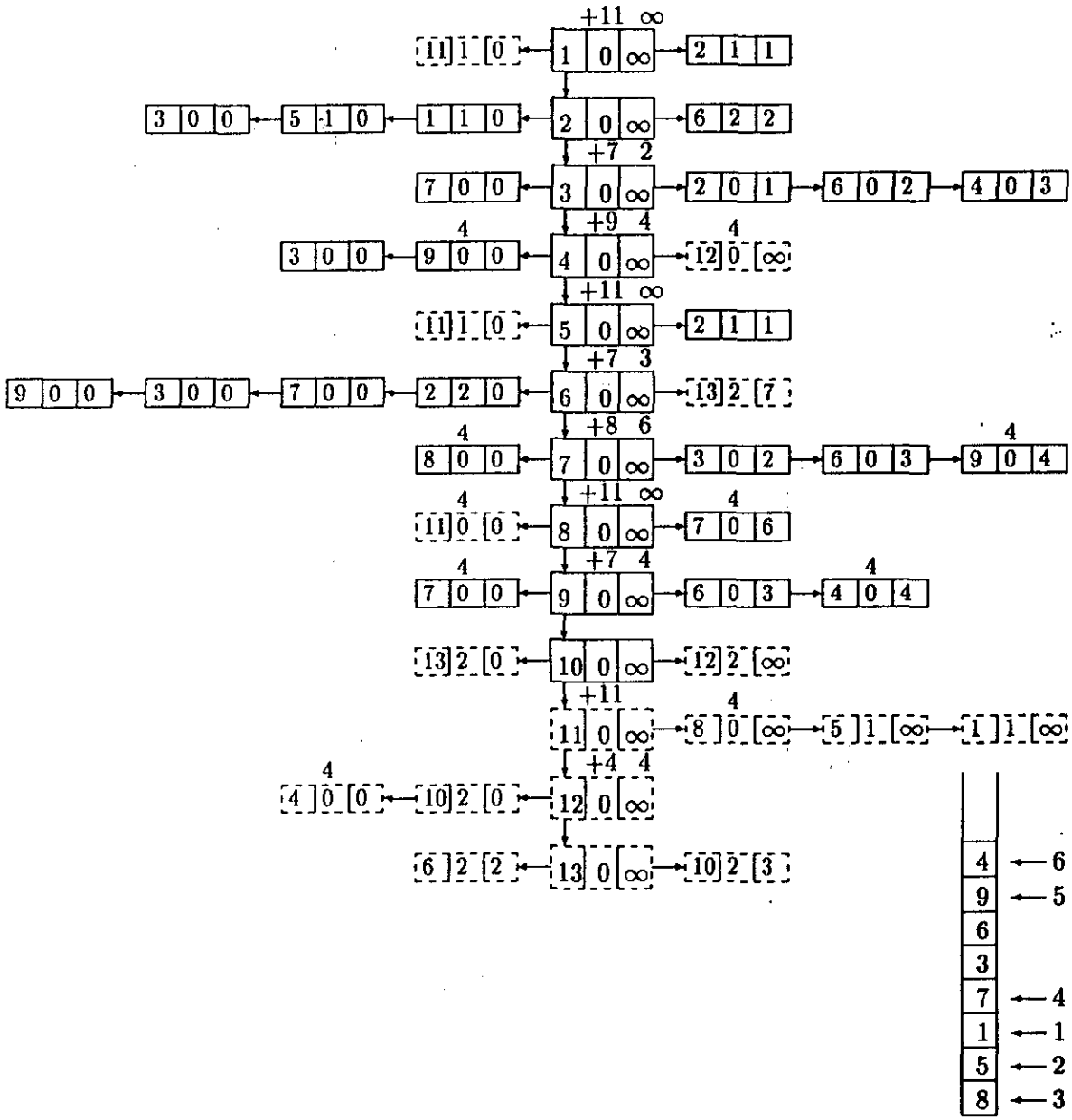


La red actualizada es:

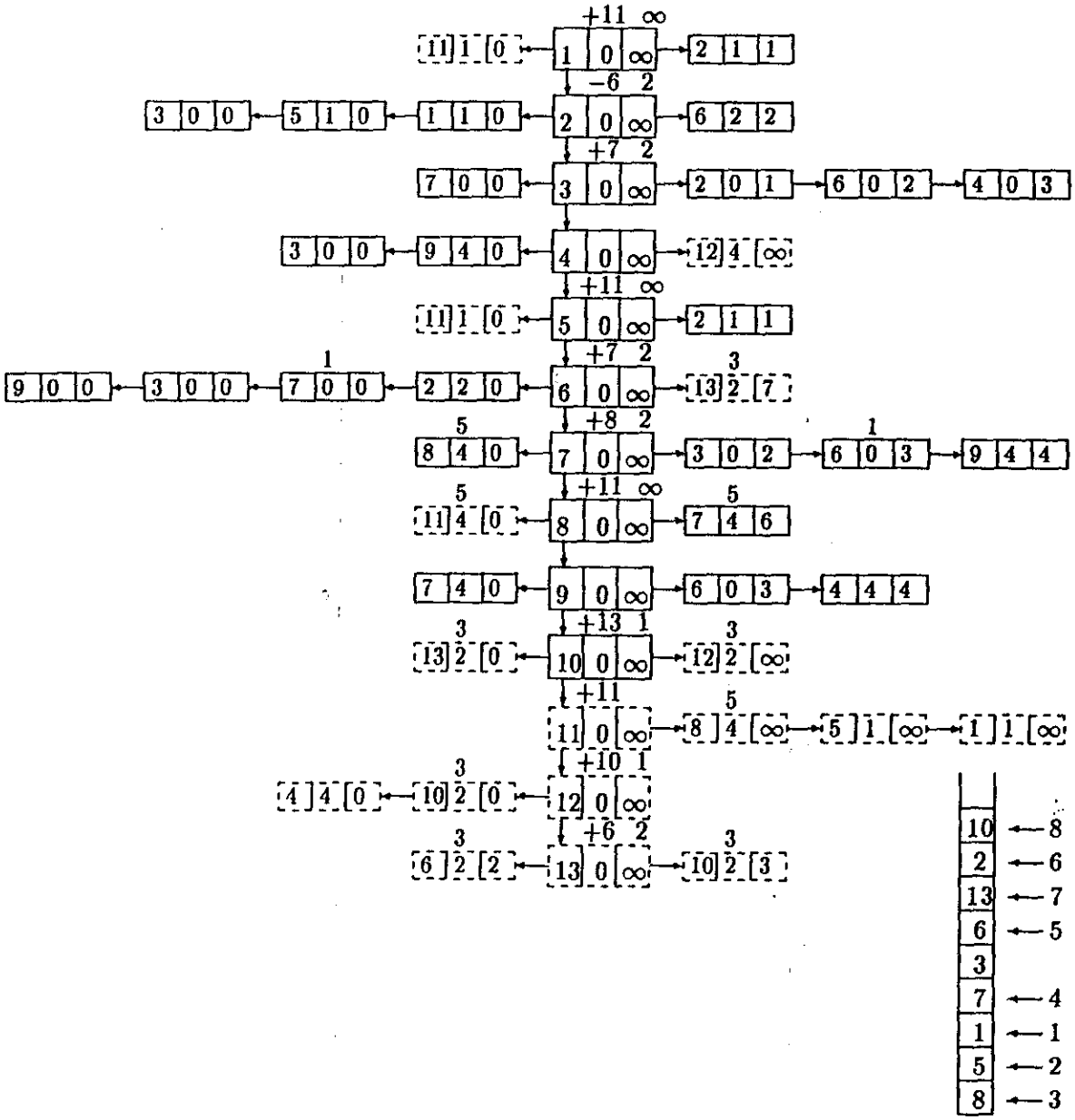


Las etiquetas de todos los arcos es (f_{ij}, q_{ij}) excepto del arco (6,13) que es (r_{ij}, f_{ij}, q_{ij}), la capacidad mínima de los demás arcos es cero.

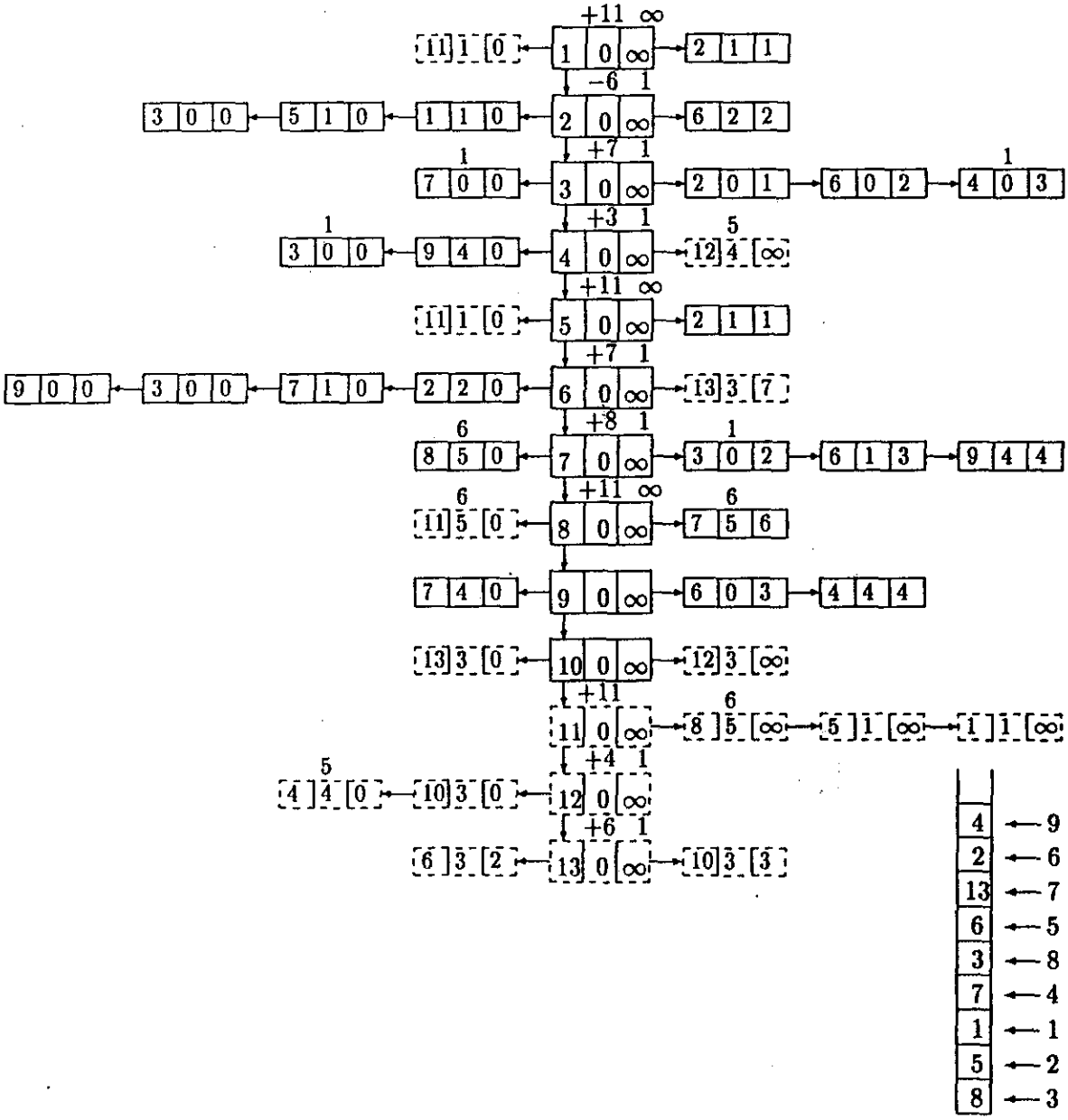
Cuarta iteración



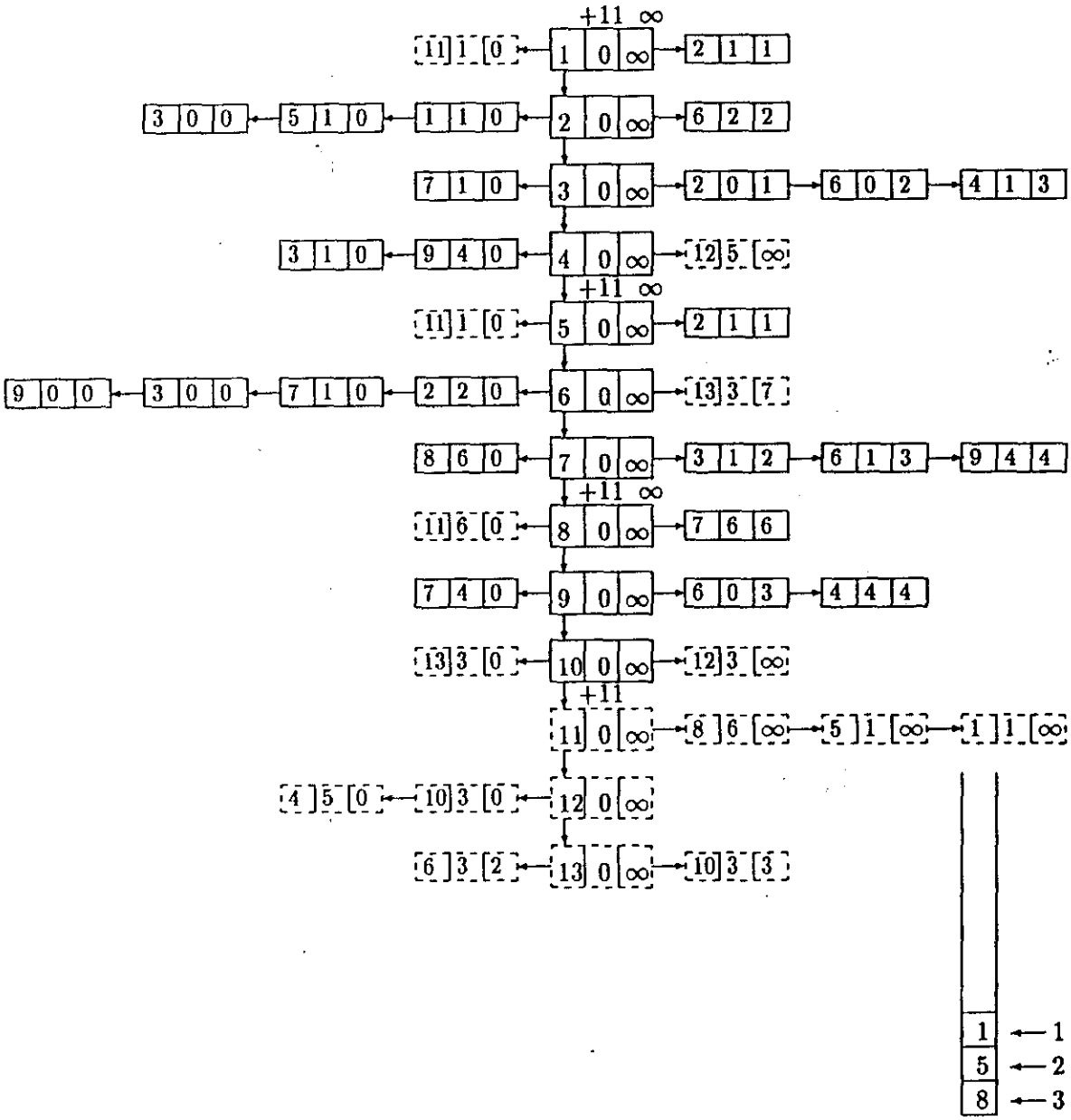
Quinta iteración



Sexta iteración

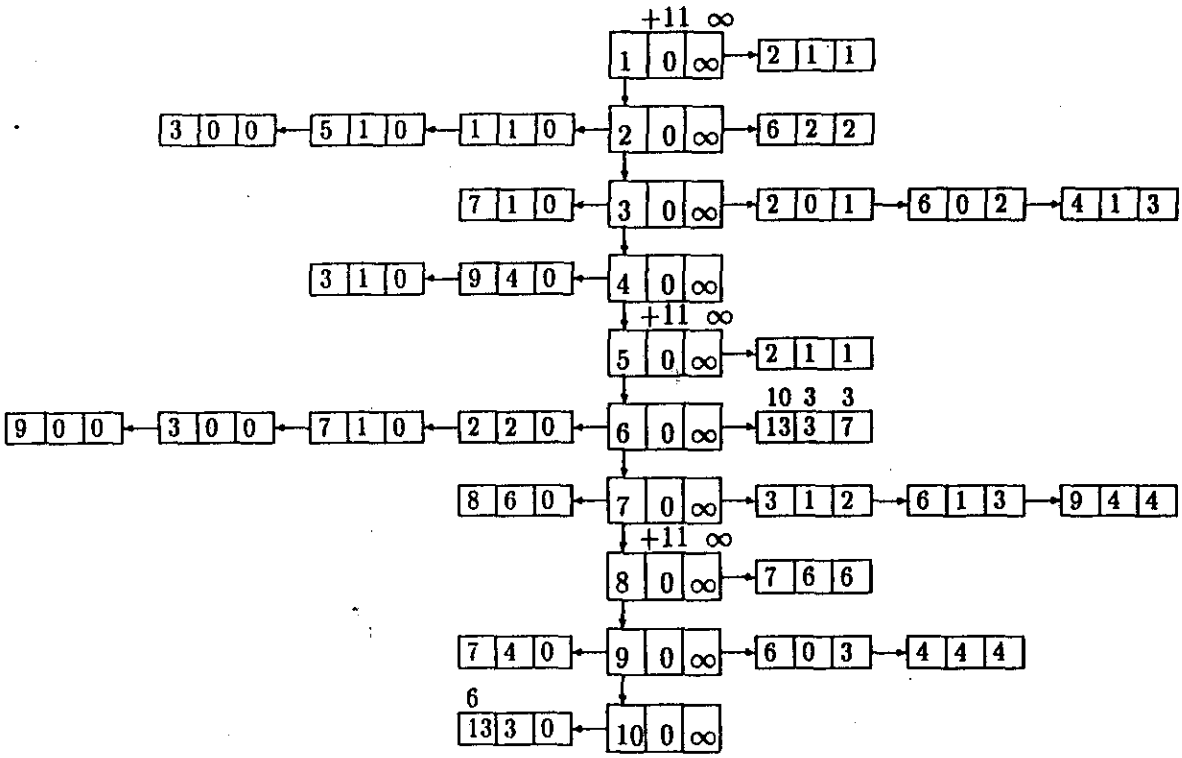


Septima iteración

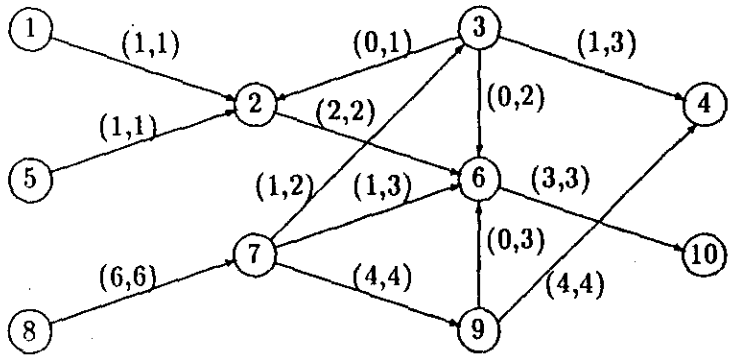


Se llegó al óptimo flujo en la red. Los arcos que constituyen la cortadura mínima son el (1,2), (5,2) y (8,7), arcos de la red del problema original.

Se desconectan todas las restantes conexiones auxiliares y se cambia el arco sucesor del nodo 13 al nodo 6 y en el arco antecesor del nodo 10 se cambia su número de nodo inicial por el nodo 6; la gráfica del problema original con su flujo máximo queda entonces:



La red del problema original con su flujo máximo es:



Hemos presentado la teoría de flujo máximo para una red con todas sus variantes pero ésta no es la única opción de flujo en una red, también puede tenerse el problema de encontrar el flujo mínimo en ella satisfaciendo las restricciones que puedan presentarse. Este es el tema que abordaremos en el siguiente capítulo donde presentaremos un método para encontrar el flujo mínimo en una red y lo justificaremos teóricamente.

Capítulo 3

El Problema de Flujo Mínimo

En este capítulo presentaremos un problema similar al de flujo máximo que es el de encontrar el flujo mínimo en una red, es decir, mientras que en un problema de flujo máximo se desea enviar del nodo fuente la mayor cantidad de flujo posible hasta el nodo destino, en el problema de flujo mínimo se desea enviar la menor cantidad de flujo posible desde el nodo fuente hasta el nodo destino. También se presenta el método de solución para el problema de flujo mínimo, aunque no se desarrollan a detalle los aspectos teóricos del mismo ya que son muy parecidos al del capítulo 1., lo que haremos es generalizar los conceptos preliminares al método y dar una justificación del mismo.

Sea $R = [X, A, r, q]$ la red que modela un problema de flujo mínimo con s como fuente y t como destino. Intuitivamente pudiera parecer que si $r_{ij} = 0$ para todo $(i,j) \in A$ el flujo mínimo de s a t es cero, pero ésto no es necesariamente cierto; al final de este capítulo se presenta un ejemplo donde el flujo mínimo de s a t es distinto de cero en una red donde todas las restricciones mínimas en arcos son cero.

En teoría lo que deseamos es:

Minimizar v

sujeta a:

1.
$$\sum_{j \in \Gamma^+(i)} f_{ij} - \sum_{k \in \Gamma^-(i)} f_{ki} = \begin{cases} v & \text{si } i = s \\ 0 & \text{si } i \neq s \text{ ó } i \neq t \\ -v & \text{si } i = t \end{cases}$$

2.
$$r_{ij} \leq f_{ij} \leq q_{ij} \text{ para todo } (i,j) \in A$$

donde r_{ij} y q_{ij} son las capacidades mínima y máxima del arco (i,j) respectivamente.

Este es un problema semejante al de la sección 2.2., por lo tanto para encontrar una primera solución factible a éste procederemos de la manera que se indica en tal sección.

Una vez que en la red se tiene definido un primer flujo factible se procede a encontrar el mínimo flujo en ella. El procedimiento para bajar el flujo definido en una red será presentado en la siguiente sección.

3.1 Método de solución

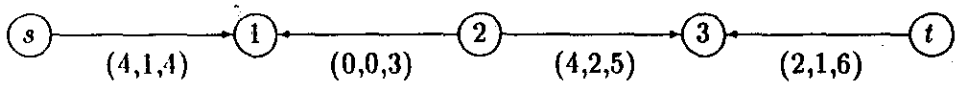
Para obtener un método para disminuir el flujo en una red utilizaremos conceptos análogos a cadenas aumentantes que llamaremos "cadenas disminuyentes" y a cortadura mínima llamada "cortadura máxima". Enunciaremos y demostraremos teoremas y proposiciones que justifican que el método que encontraremos nos lleva efectivamente al flujo mínimo en una red, una vez presentados los siguientes conceptos:

Definición: Sea $R = [X, A, q]$ una red y f un flujo factible en ella. Una cadena de s a t es disminuyente si $f_{ij} > r_{ij}$ para todo $(i,j) \in F$ y $f_{ij} < q_{ij}$ para todo $(i,j) \in B$. F y B definidos como en la sección 1.2.

Ejemplo:

En la siguiente cadena de s a t la etiqueta asociada a cada arco es (f_{ij}, r_{ij}, q_{ij}).

Sean $F = \{(s,1),(2,3)\}$ y $B = \{(2,1),(t,3)\}$.



Esta cadena es disminuyente ya que en todos los arcos de F se cumple que $f_{ij} > r_{ij}$ y en los de B se cumple que $f_{ij} < q_{ij}$.

A través de una cadena disminuyente C, como su nombre lo indica, se puede disminuir el flujo de s a t construyendo un nuevo flujo factible f' de menor valor que el que tiene: f. El nuevo flujo factible se construye de la siguiente manera:

$$f'_{ij} = f_{ij} - z \qquad \forall\ (i,j) \in F$$

$$f'_{ij} = f_{ij} + z \qquad \forall\ (i,j) \in B$$

donde z es tal que:

$$f_{ij} - z \geq r_{ij} \qquad \forall\ (i,j) \in F$$

$$f_{ij} + z \leq q_{ij} \qquad \forall\ (i,j) \in B$$

Si f es de valor v entonces f' es de valor $v - z$.

Al igual que para flujo máximo, se tiene una definición que da el mejor valor de z para disminuir el flujo a través de C.

Definición: La capacidad de decremento de la cadena disminuyente C de s a t es la máxima cantidad de flujo en que puede disminuirse el flujo a través de ella; la denotaremos con $d(C)$ y se calcula:

$$d(C) = \min\{\min_{(i,j)\in F}[f_{ij} - r_{ij}], \min_{(i,j)\in B}[q_{ij} - f_{ij}]\}$$

La capacidad de decremento de la cadena disminuyente del elemplo anterior es:

$$d(C) = \min\{\min_{(s,1),(2,3)}[f_{ij} - r_{ij}], \min_{(2,1),(t,3)}[q_{ij} - f_{ij}]\}$$

como:

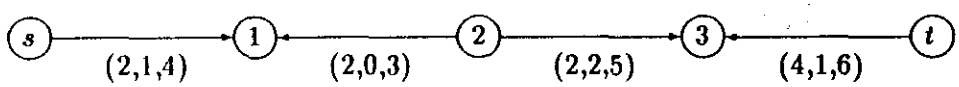
$$\min_{(s,1),(2,3)}[f_{ij} - r_{ij}] = \min[4-1, 4-2] = 2$$

$$\min_{(2,1),(t,3)}[q_{ij} - f_{ij}] = [3-0, 6-2] = 3$$

entonces:

$$d(C) = \min\{2, 3\} = 2$$

y el flujo actualizado en esta cadena es:



Observese en este caso que cuando se actualiza el flujo en una cadena disminuyante por lo menos un arco de la misma llaga a su capacidad máxima o mínima.

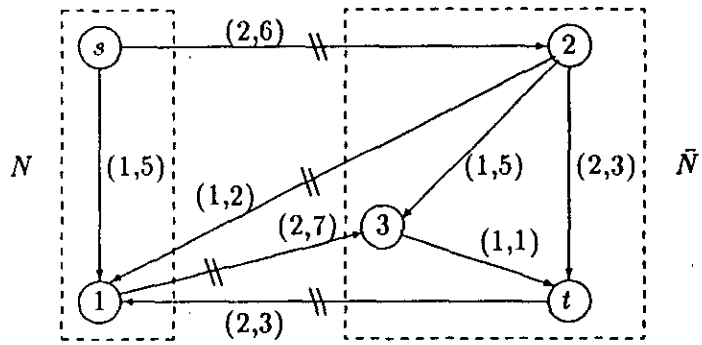
Análogo al teorema de Flujo Máximo - Cortadura Mínima presentaremos un teorema que relacione el flujo mínimo y la cortadura máxima, pero antes tenemos los siguientes conceptos:

Redefiniremos el concepto de cortadura .

Sea $R = [X, A, r, q]$ una red y $N \in X$, $\bar{N} = X - N$. Se denota $(N, \bar{N})$ al conjunto de arcos que tienen extremo inicial en N y extremo final en $\bar{N}$; y $(\bar{N}, N)$ al conjunto de arcos que tienen extremo inicial en $\bar{N}$ y extremo final en N y sea $[N, \bar{N}] = (N, \bar{N}) \cup (\bar{N}, N)$.

Definición: $[N, \bar{N}]$ es una cortadura de R si $s \in N$ y $t \in \bar{N}$, donde s es el nodo fuente y t el nodo destino de R.

Ejemplo: En la siguiente red todos los arcos marcados forman una cortadura.



La siguiente es una generalización de capacidad de una cortadura.

Definición: La capacidad máxima de una cortadura $[N, \bar{N}]$ se denota por $q[N, \bar{N}]$ y se calcula:

$$q[N, \bar{N}] = \sum_{(i,j)\in(N,\bar{N})} q_{ij} - \sum_{(i,j)\in(\bar{N},N)} r_{ij}$$

Así, la capacidad máxima de la cortadura del ejemplo anterior es 10.

Nótese que la capacidad de la cortadura definida en el capítulo 1. es un caso particular de ésta ya que en aquella definición $r_{ij} = 0$ para todo $(i, j) \in A$.

Análogamente:

Definición: La capacidad mínima de una cortadura $[N, \bar{N}]$ se denota por $d[N, \bar{N}]$ y se calcula:

$$d[N, \bar{N}] = \sum_{(i,j)\in(N,\bar{N})} r_{ij} - \sum_{(i,j)\in(\bar{N},N)} q_{ij}$$

La capacidad mínima de la cortadura del ejemplo anterior es -1.

Para tener una analogía con el teorema de Flujo Máximo - Cortadura Mínima tenemos los siguientes conceptos:

Definición: Una cortadura mínima es la que tiene mínima capacidad máxima.

Análogamente:

Definición: Una cortadura máxima es la que tiene máxima capacidad mínima.

3.1.1 Base teórica para el método de solución

La proposición y los teoremas de esta sección son la base teórica del método de solución para el problema de flujo mínimo en una red. Dicho método se puede deducir de manera natural de la

demostración del teorema de Flujo Mínimo - Cortadura Máxima que presentaremos más adelante.

PROPOSICIÓN: *Sea $R[X, A, r, q]$ una red, f un flujo factible en ella de valor v y $[N, \bar{N}]$ una cortadura de R, entonces:*

$$d[N, \bar{N}] \leq v \leq q[N, \bar{N}]$$

Demostración:

De la demostración de la proposición de la sección 1.5. tenemos que cualquier flujo factible cumple que

$$v = \sum_{i \in N, j \in \bar{N}} f_{ij} - \sum_{i \in N, k \in \bar{N}} f_{ki} = \sum_{(i,j) \in (N,\bar{N})} f_{ij} - \sum_{(i,j) \in (\bar{N},N)} f_{ij}$$

y como $f_{ij} \leq q_{ij}$ para todo $(i,j) \in A$ y $f_{ij} \geq r_{ij}$ para todo $(i,j) \in A$

$$v = \sum_{(i,j) \in (N,\bar{N})} f_{ij} - \sum_{(i,j) \in (\bar{N},N)} f_{ij} \leq \sum_{(i,j) \in (N,\bar{N})} f_{ij} - \sum_{(i,j) \in (\bar{N},N)} r_{ij} \leq$$

$$\sum_{(i,j) \in (N,\bar{N})} q_{ij} - \sum_{(i,j) \in (\bar{N},N)} r_{ij} = q[N, \bar{N}]$$

análogamente:

$$v = \sum_{(i,j) \in (N,\bar{N})} f_{ij} - \sum_{(i,j) \in (\bar{N},N)} f_{ij} \geq \sum_{(i,j) \in (N,\bar{N})} r_{ij} - \sum_{(i,j) \in (\bar{N},N)} f_{ij} \geq$$

$$\sum_{(i,j) \in (N,\bar{N})} r_{ij} - \sum_{(i,j) \in (\bar{N},N)} q_{ij} = d[N, \bar{N}]$$

por lo tanto

$$d[N, \bar{N}] \leq v \leq q[N, \bar{N}]$$

■

El siguiente es el teorema generalizado de Flujo Máximo - Cortadura Mínima del capítilo 1.

TEOREMA: (Flujo Máximo - Cortadura Mínima)

En una red $R = [X, A, r, q]$ el valor del flujo máximo es igual a la capacidad de la cortadura mínima.

Demostración:

El procedimiento de demostración es exactamente el mismo del teorema de Flujo Máximo - Cortadura Mínima del capítulo 1., solo que ahora N se construye de la siguiente manera:

1. Sea $N = \{s\}$
2. Tomamos $i \in N$ y $j \in \bar{N}$. Si $j \in \Gamma^+(i)$ y $f_{ij} < q_{ij}$ ó $j \in \Gamma^-(i)$ y $f_{ji} > r_{ji}$ se agrega j a N.

La única diferencia con el teorema de Flujo máximo - Cortadura Mínima es que todas las comparaciones con cero se hacen ahora con r_{ij} del arco (i,j).

Por tanto, al terminar de iterar en el caso de que $t \notin N$, por construcción, tendremos que $f_{ij} = q_{ij}$ para todo $(i,j) \in (N, \bar{N})$ y $f_{ij} = r_{ij}$ para todo $(i,j) \in (\bar{N}, N)$ y

$$v = \sum_{(i,j)\in(N,\bar{N})} f_{ij} - \sum_{(i,j)\in(\bar{N},N)} f_{ij} = \sum_{(i,j)\in(N,\bar{N})} q_{ij} - \sum_{(i,j)\in(\bar{N},N)} r_{ij} = q[N,\bar{N}]$$

■

El siguiente teorema relaciona el flujo mínimo en una red con la cortadura de máxima capacidad mínima y da un método para encontrar tal flujo mínimo en una red, de la misma manera que lo hace el teorema de Flujo Máximo - Cortadura Mínima.

TEOREMA: (Flujo Mínimo - Cortadura Máxima)

En una red $R = [X, A, r, q]$ el valor del flujo mínimo es igual a la capacidad de la cortadura máxima.

Demostración:

Como $v \geq d[N, \bar{N}]$ donde v es el valor de cualquier flujo factible, sólo basta probar que existe un flujo factible en R que tiene valor igual a la capacidad mínima de una cortadura.

El procedimiento para esta demostración es análogo al de Flujo Máximo - Cortadura Mínima solo que ahora en lugar de encontrar cadenas aumentantes y subir el flujo en la red a través de ellas, se buscan cadenas disminuyentes y se baja el flujo definido en la red a través de tales cadenas.

A continuación se da el procedimiento:

Considérese cualquier flujo factible f en R y construyase el conjunto N como sigue:

1. Sea $N = \{s\}$
2. Tomamos $i \in N$ y $j \in \bar{N} = X - N$. Si $j \in \Gamma^+(i)$ y $f_{ij} > r_{ij}$ ó $j \in \Gamma^-(i)$ y $f_{ji} < q_{ji}$ se agrega j a N.

Se repite 2. hasta que no pueda agregarse nodo alguno a N. Entonces sucede que si:

- $t \in N$:

  Por construcción de N, existe una cadena C de s a t tal que $f_{ij} > r_{ij}$ para todo $(i,j) \in F$ y $f_{ij} < q_{ij}$ para todo $(i,j) \in B$, por lo tanto C es disminuyente y a través de ella se puede mejorar el flujo de s a t, es decir, disminuirlo de la siguiente manera:

  Sea $d(C)$ la capacidad de decremento de la cadena C y redefinase f y v como:

$$f_{ij} = \begin{cases} f_{ij} - d(C) & \text{si } (i,j) \in F \\ f_{ij} + d(C) & \text{si } (i,j) \in B \\ f_{ij} & \text{en otro caso} \end{cases}$$

  y el nuevo valor de este flujo será:

$$v = v - d(C)$$

  éste es el valor de un nuevo flujo factible f, entonces se puede repetir el procedimiento señalado anteriormente utilizando este flujo de menor valor. El flujo se decrementa en al menos una unidad por ser r_{ij} y q_{ij} enteros para todo $(i,j) \in A$, por lo tanto, el flujo mínimo se obtiene en un número finito de pasos.

- $t \notin N$:

  Entonces $t \in \bar{N}$ y $[N, \bar{N}]$ es una cortadura de R. Por construcción tenemos que $f_{ij} = r_{ij}$ para todo $(i,j) \in (N, \bar{N})$ y $f_{ij} = q_{ij}$ para todo $(i,j) \in (\bar{N}, N)$ y

$$v = \sum_{(i,j)\in(N,\bar{N})} f_{ij} - \sum_{(i,j)\in(\bar{N},N)} f_{ij} = \sum_{(i,j)\in(N,\bar{N})} r_{ij} - \sum_{(i,j)\in(\bar{N},N)} q_{ij} = d[N,\bar{N}]$$

■

TEOREMA: *Sea $R = [X, A, r, q]$ una red y $s, t \in X$. Entonces el valor del flujo mínimo de s a t es igual a menos el valor del flujo máximo de t a s.*

Demostración:

Esto es trivial ya que por definición una cadena disminuyente de s a t es una cadena aumentante de t a s y la capacidad de decremento de s a t es igual a la capacidad incremental de t a s, así, el valor en que se disminuye el flujo de s a t es el mismo en el que se aumenta el flujo de t a s.

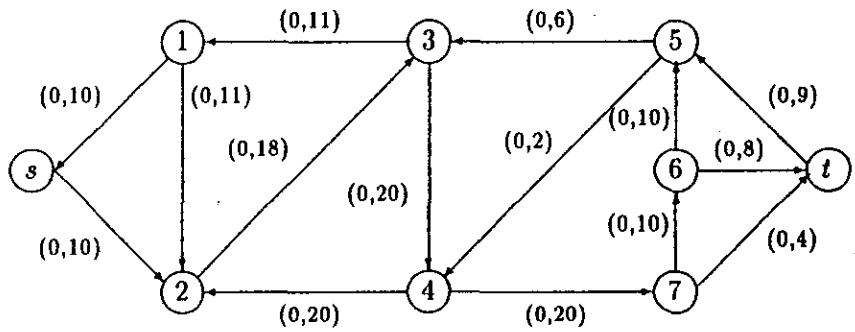
Si de s a t no existe cadena disminuyente entonces no existe cadena aumentante de t a s y el valor del flujo definido en R es mínimo de s a t y máximo de t a s, por lo tanto el valor del flujo en s es $-v$ donde v es el valor del flujo máximo de t a s.

■

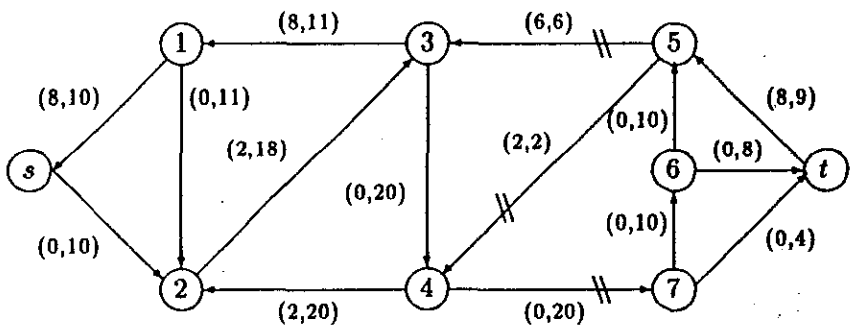
El resultado anterior es muy importante ya que permite que en lugar de implementar un algoritmo para encontrar el mínimo flujo en una red, basandose en el procedimiento de demostración del teorema de Flujo Mínimo - Cortadura Máxima, se utilise el mismo *Ford-Fulkerson*, ya que la cortadura producida por éste tiene la misma capacidad que una producida por un método iterativo de flujo mínimo, es decir, una cortadura mínima producida por el flujo máximo de t a s es una cortadura máxima producida por el flujo mínimo de s a t.

3.1.2 Ejemplos

1. Encontrar el flujo mínimo del nodo s al t en la red de la siguiente figura. Las etiquetas de los arcos son (f_{ij}, q_{ij}), la cota mínima de flujo para todos los arcos es cero.

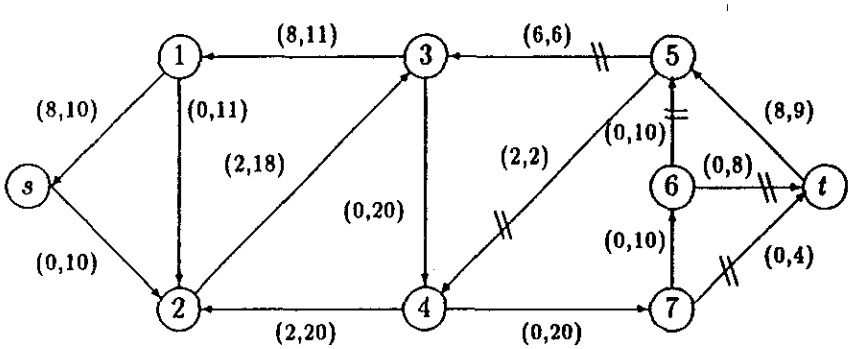


Resolviendo este problema con un método de etiquetado y disminución de flujo a través de cadenas disminuyentes, se obtiene como resultado la siguiente red donde se marcan los arcos de la cortadura máxima generada:



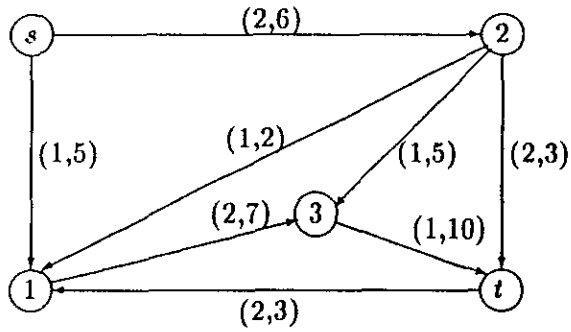
Observese que el valor del flujo mínimo de s a t es v=0−(6+2)=−8.

Por otra parte, aplicando el algoritmo de *Ford-Fulkerson* para flujo máximo de t a s se obtiene la red:

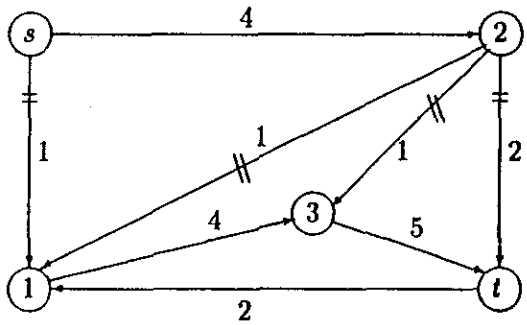


Los arcos marcados en la figura son los que constituyen la cortadura mínima generada. El valor del flujo máximo de t a s es v=(6+2)−(0+0+0)=8.

2. Encontrar el flujo mínimo de s a t en la red de la siguiente figura donde los arcos presentan restricciones mínimas y máximas ilustradas en la pareja ordenada respectivamente en cada uno de ellos.



Aplicando el algoritmo de *Ford-Fulkerson* de t a s obtenemos la red:



Los arcos marcados constituyen la cortadura mínima generada por *Ford-Fulkerson* y el número asociado a cada arco es su flujo correspondiente en la solución óptima obtenida.

Nótese que el valor del flujo máximo de t a s es -5, por lo tanto el valor del flujo mínimo de s a t es 5.

En este ejemplo, al querer encontrar el flujo máximo de t a s, si solo se agrega el arco auxiliar (s,t), no se encuentra una primera solución factible por las restricciones establecidas, sin embargo, si se agrega además de ese arco el arco (t,s) la primera solución factible es encontrada y mejorada por el algoritmo de *Ford-Fulkerson*. Por esta razón, cuando tenemos un problema de flujo máximo con restricciones mínimas distintas de cero en arcos debemos agregar los dos arcos auxiliares como se comentó en la sección 2.2.

Conclusiones

El trabajo que presentamos es parte de un proyecto de desarrollo de software para flujo en redes llamado PAREIMM que aún no ha llegado a su término. El objetivo del proyecto es implementar los algoritmos clásicos de flujo en redes y utilizarlos como paquetería.

La experiencia obtenida al realizar este trabajo es que al elaborar software es preciso tener una buena base teórica y computacional. Esparamos que en lo futuro se desarrolle software con éxito y se difunda su realización.

Apéndice A

Implementación del Algoritmo de Ford-Fulkerson

El programa de la implementación del algoritmo de *Ford-Fulkerson*, es parte de un proyecto de desarrollo de software para flujo en redes llamado PAREIMM. Dicho proyecto se constituye por un programa manager, manejador de pantallas y editor de problemas, del cual se ejecutan algoritmos clásicos de flujo en redes donde se incluye el de *Ford-Fulkerson*. Sin embargo, el programa también ser ejecutado fuera del manager PAREIMM, y para ésto se requiere que el primer argumento sea FORD y el segundo el nombre del archivo donde esté guardado el problema el cual deberá tener la información como la presentaremos en la sección A.1.1., y el programa dará la solución del problema en otro archivo que tendrá las características descritas en la sección A.1.2. El código del programa se presenta en la sección A.2.

A.1 Archivos de Captura y Solución

En esta sección presentaremos la manera en que deben hacerse los archivos de captura de problemas para ser ejecutados por el programa de la implementación del algoritmo de *Ford-Fulkerson* y la manera en que dicho programa guarda la solución en un archivo.

A.1.1 Captura

Para capturar los datos de una red donde se desea aplicar el programa de la implementación del algoritmo de *Ford-Fulkerson* presentado en este anexo se utiliza un archivo cuyo nombre llevará la extención .PRB y sus forma será la siguiente:

El archivo con el problema puede tener dos formas, éstas dependerán de si en la red que modela al problema existen varios nodos fuente y varios destinos.

1. **Cuando existe solo un nodo fuente y un destino.**

- El archivo debe tener una primera linea que contenga el número 4, éste indica que el tipo de problema es de flujo máximo.
- Una segunda linea que contenga el número 0 que indicará que la variante de varios fuentes y destinos no se presenta en el problema que se captura.
- Las lineas posteriores llevarán la información de los arcos de la red en cuatro columnas que serán: el nodo inicial, nodo final, capacidad mínima y capacidad máxima del arco. Al terminar de poner los arcos pertenecientes a la red se agregará un renglón con 4 ceros (0 0 0 0) indicando la terminación de la captura de los arcos.
- Después se capturan los nodos pertenecientes a la red que presentes restricciones. Esto se hace en tres columnas que serán: número del nodo restringido, capacidad mínima y capacidad máxima del nodo. Se agregará una linea con 3 ceros (0 0 0) indicando la terminación de la captura de los nodos.
- La última linea del archivo llevará un par de números que serán el nodo fuente y destino de la red, en ese orden.

2. **Cuando existen varios fuentes y varios destinos.**

- El archivo debe tener una primera linea que contenga el número 4, éste indica que el tipo de problema es de flujo máximo.
- Una segunda linea que contenga el número 1 que indicará que en el problema que se captura se presenta la variante de varios fuentes y destinos.
- Las lineas posteriores llevarán la información de los arcos de la red en cuatro columnas que serán: el nodo inicial, nodo final, capacidad mínima y capacidad máxima del arco. Al terminar de poner los arcos pertenecientes a la red se agregará un renglón con 4 ceros (0 0 0 0) indicando la terminación de la captura de los arcos.
- Después se capturan los nodos pertenecientes a la red que presentes restricciones. Esto se hace en tres columnas que serán: número del nodo restringido, capacidad mínima y capacidad máxima del nodo. Se agregará una linea con 3 ceros (0 0 0) indicando la terminación de la captura de los nodos.
- Cada renglón posterior, contendrá uno de los nodos fuente de la red; se agregará un renglón con el número 0 cuando se terminen de capturar dichos nodos.
- Cada siguiente renglón contendrá uno de los nodos destino de la red; se agregará un renglón con el número 0 cuando se terminen de capturar dichos nodos.

Todos los números que se capturen tendrán que separarse por un espacio.

Ejemplos

Presentaremos el archivo correspondiente a uno de los dos ejemplos del capítulo 1. donde existe un solo fuente y un solo destino, y el de un problema donde se presentan varios fuentes y destinos.

Nombre del archivo: plaga.prb (La eliminación de la plaga de maíz)

```
4
0
1  2  0  20
1  6  0  6
1  4  0  15
1  3  0  17
2  5  0  10
2  6  0  12
3  4  0  4
3  7  0  4
3  8  0  2
4  12  0  10
4  9  0  8
5  11  0  9
5  6  0  5
6  11  0  12
6  12  0  6
7  15  0  3
8  7  0  3
8  15  0  7
8  10  0  9
9  7  0  3
9  12  0  6
9  14  0  7
9  15  0  2
10  15  0  6
10  16  0  8
11  12  0  10
11  13  0  11
12  13  0  3
12  14  0  4
13  17  0  7
14  17  0  20
14  18  0  19
14  16  0  5
14  15  0  10
15  16  0  12
16  18  0  23
17  18  0  24
0  0  0  0
0  0  0
1  18
```

Nombre del archivo: prob.prb

```
4
1
1 2 0 7
5 2 1 5
8 7 0 5
2 6 0 4
7 3 0 2
6 7 0 3
7 9 0 4
3 2 0 1
3 6 1 2
9 6 0 3
9 4 0 4
3 4 0 3
6 10 1 1
0 0 0 0
2 1 6
0 0 0
1
5
8
0
4
10
0
```

A.1.2 Solución

El programa de la implementación del algoritmo de *Ford-Fulkerson* llena un archivo con la información de la solución a un problema, dicho archivo tendrá el mismo nombre que el usuario haya escogido para el problema con la extención .SOL, y contendrá lo siguiente:

- La primera linea del archivo contendrá el número 4, indicando que el tipo de problema del que es solución es de flujo máximo.
- Las lineas posteriores llevarán la información del flujo solución de los arcos de la red en tres columnas que serán: el nodo inicial, nodo final, capacidad y flujo del arco. Al terminar de poner los arcos se agregará un renglón con 3 ceros (0 0 0) indicando la terminación de la lista de arcos.
- En cada renglón posterior llavará el número de los nodos que quedaron marcados en la última iteración, ésto con el fin de saber cuales arcos constituyen la cortadura mínima. La última linea del archivo llevará el número 0 para indicar su terminación.

En caso de que no se pueda encontrar solución aparecerá en pantalla un mensaje indicando qué tipo de problema se ha presentado.

Ejemplos

Se presentarán los archivos de solución de los dos problemas presentados en el capítulo 1.

Nombre del archivo: plaga.sol (La eliminación de la plaga de maíz)

```
4
1 2 0
1 6 4
1 4 15
1 3 5
2 5 0
2 6 0
3 4 0
3 7 3
3 8 2
4 12 7
4 9 8
5 11 0
5 6 0
6 11 4
6 12 0
7 15 3
8 7 0
8 15 0
8 10 2
9 7 0
9 12 0
9 14 6
9 15 2
10 15 0
10 16 2
11 12 0
11 13 4
12 13 3
12 14 4
13 17 7
14 17 0
14 18 10
14 16 0
```

```
14 15 0
15 16 5
16 18 7
17 18 7
0 0 0
1
2
3
4
5
6
7
11
12
13
0
```

Como éste es un problema con cotas mínimas en arcos todas cero, los arcos que constituyen la cortadura mínima son los que tienen como extremo inicial un nodo marcado y estremo final un nodo no marcado, por lo tanto los arcos de la cortadura mínima del problema de la plaga de maíz son el (13,17), (12,14), (4,9), (7,15) y (3,8) que son los que se tienen que fumigar para evitar que la plaga llege a los graneros. Ver la figura de la sección 1.1.2.

Nombre del archivo: pintor.sol (El problema del pintor)

```
4
1 2 3
1 3 3
1 4 3
1 5 3
1 6 3
1 7 3
1 8 3
1 9 3
2 10 1
2 12 0
2 14 2
3 10 1
3 12 0
3 14 2
4 10 0
4 11 2
4 12 1
4 14 0
```

```
5  10  0
5  11  0
5  12  3
5  13  0
5  14  0
6  11  0
6  12  3
6  13  0
6  14  0
7  11  2
7  12  0
7  13  1
7  14  0
8  12  0
8  13  0
8  14  3
9  12  0
9  13  2
9  14  1
10  15  2
11  15  4
12  15  7
13  15  3
14  15  8
0 0 0
1
0
```

El problema del pintor se planteó en una red que si al encontrarse el flujo máximo en ella su valor es 22 (suma del tiempo estimado de todos los trabajos), entonces se podrían terminar los trabajos a tiempo. El valor del flujo máximo encontrado por *Ford-Fulkerson* es 24 lo cual quiere decir que dos equipos de trabajo del señor Romero quedan desocupados en una semana de trabajo o que dos semanas no se trabajan.

A continuación presentaremos el código del programa de la implementación del algoritmo de *Ford-Fulkerson* con todas sus variantes.

A.2 Programa

```
/*
 ************************************************************************
    VERANO 93                                                  PAREIMM
    NOMBRE DEL ARCHIVO: FORD.C
       Este archivo aplica el algoritmo FORD_FULKERSON.
    FUNCIONES QUE INCLUYE:
    Main: Organiza la informacion de entrada y salida y
  pone algunas conecciones auxiliares para el
  Algoritmo.
 ************************************************************************
    */
#include <io.h>
#include <math.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <values.h>
#include <process.h>

#define         IO_ERR          0
#define         NO_PROBLEM      1
#define         NO_SOL          2
#define         NO_MEM          3
#define         TOO_BIG         4
#define         NO_FORD         5
#define         PROBLEMAS       6
#define         SOL_CERO        7

#define EXITO 100

#define         INFINITO  MAXINT

typedef unsigned short unsh;

struct Nodo{
   unsh Num_Nodo;                          // Numero del nodo.
   int Ant_Cad_Aum;                        // Antecesor en cadena aumentante.
   unsh Cap_Cad_Aum;                       // Capacidad de cadena aumentante.
   struct Arco_Sucesor *Arco_Sucesor;      // Direccion del primer elemento.
  //de la lista de arcos sucesores.
   struct Arco_Antecesor *Arco_Antecesor; // Direccion del primer elemento.
  //de la lista de arcos antecesores.
   struct Nodo *Siguiente;                 // Direccion al siguiente elemento.
};
struct Arco_Sucesor{
   unsh Nodo_Final;                  // Nodo final del arco.
```

```
   unsh Flujo;                          // Flujo del arco.
   unsh Cap_Max;                        // Capacidad Maxima del arco.
   struct Arco_Sucesor *Siguiente;      // Direccion al siguiente elemento.
};
struct Arco_Antecesor{
   unsh Nodo_Inicial;                    // Nodo inicial del arco.
   unsh Flujo;                           // Flujo del arco.
   unsh Cap_Min;                         // Capacidad minima del arco.
   struct Arco_Antecesor *Siguiente;     // Direccion al siguiente elemento.
};
struct Plato{
   struct Nodo *Nodo;            // Direccion de un nodo.
   struct Plato *Siguiente;      // Direccion al siguiente elemento.
};
struct Fuente_Plato{
   struct Nodo *Dir_f;                // Direccion de un nodo fuente.
   struct Fuente_Plato *Siguiente;    // Direccion al siguiente elemento.
};
struct Destino_Plato{
   struct Nodo *Dir_d;                  // Direccion de un nodo destino.
   struct Destino_Plato *Siguiente;     // Direccion al siguiente elemento.
};
struct Res_Nodo_Plato{
   unsh Nodo_Res;                       // Numero del nodo restringido.
   unsh Res_Min;                        // Restriccion Minima del nodo.
   unsh Res_Max;                        // Restriccion Maxima del nodo.
   struct Nodo *Dir_Cuate;              // Direccion del nodo cuate (auxiliar).
   struct Res_Nodo_Plato *Siguiente;    // Direccion al siguiente elemento.
};
struct Plato_Arco{
   struct Nodo *Dir_Nodo_Inicial;    // Direccion del nodo inicial.
   struct Nodo *Dir_Nodo_Final;      // Direccion del nodo final.
   unsh Cap_Min;                     // Capacidad Minima del arco.
   struct Plato_Arco *Siguiente;     // Direccion al siguiente elemento.
};
struct Nodo *Grafica;   // Direccion al primer elemento de la lista de nodos.

unsh PAREIMM;

#include "ford_fun.h"  //  Archivo de funciones de agregado y borrado de
       // elementos.
#include "ford_alg.h"  // Archivo de funciones que aplican el algoritmo.

void main( int argc,char *argv[] )

/*    Esta funcion organiza la informacion de entrada y salida y pone
   algunas conecciones auxiliares para el algoritmo.

 Opera de la siguiente manera:

 a.- Crea la lista de nodos incluyendo los auxiliares en el caso de que
```

```
    existan varias fuentes, varios destinos, arcos o nodos restringidos.
 b.- Captura los arcos de la digrafica.
 c.- Si existen arcos restringidos o nodos con restriccion minima aplica
    el algoritmo mediante la funcion Primera_Solucion.
 d.- Aplica el algoritmo mediante la funcion Flujo_Maximo.
 e.- Quita los nodos y arcos auxiliares mediante la funcion
    Libera_Conecciones.
 f.- Despliega la solucion obtenida mediante la funcion Imprimir.

    Las variables principales que utiliza son:

        Grafica: Direccion del primer elemento de la lista de nodos.
        Nodo_Inicial: Numero del nodo inicial en la captura de la grafica.
        Nodo_Final: Numero del nodo final en la captura de la grafica.
        N_Res: Numero de nodos Restringidos.
        Nodo_Res: Numero del nodo restringido.
        Res_Min: Restriccion minima del nodo restringido.
        Res_Max: Restriccion maxima del nodo restringido.
        Cap_Min: Capacidad minima del arco restringido.
        Cap_Max: Capacidad maxima del arco restringido.

    Las funciones que se utilizan son:

        Nodo_Busca: Busca un elemento en la lista de nodos.
        Nodo_Agrega: Agrega un elemento a la lista de nodos.
        Agregar_Arco_Sucesor: Agrega un elemento a la lista de arcos
    sucesores de un nodo.
        Agr_Arco_Antecesor: Agrega un elemento a la lista de arcos
  antecesores de un nodo.
        Mete_Plato_Arco: Agrega un elemento a la pila de arcos restringidos.
        Agarres: Agrega arcos auxiliares si existen arcos con restriccion.
        Mete_Fuente_Plato: Agrega un elemento a la pila de nodos fuente.
        Mete_Destino_Plato: Agrega un elemento a la pila de nodos destino.
        Poner_Nodo_Res: Agrega arcos auxiliares si existen nodos restringidos.
        Fuentes_Destinos: Agrega arcos auxiliares si existen varias fuentes
y varios destinos.
        Primera_Solucion: Aplica el algoritmo Ford_Fulkerson para encontrar
una primera solucion.
        Flujo_Maximo: Aplica el algoritmo Ford_Fulkerson.
        Libera_Conecciones: Libera las conecciones auxiliares.
        Imprimir: Imprime la solucion obtenida.                          */
{
   char Nomb_Prob[16];
   int Resultado;
   unsh Tipo_Prob,Nodo_Inicial,Nodo_Final,Ind,s,t,Nodo_f,Nodo_d,Nodo_Res,Var;
   unsh Indicador,Ind_NRes_Min=0,Ind_ARes_Min=0,Vf_Vd,Cap_Max,Res_Min;
   unsh Res_Max,Cap_Min,Suma_Res=0,Num_Nodos=0;
   struct Nodo *Dir_t,*Dir_s,*Buscando,*Apt,*Dir_NI,*Dir_NF,*Ant_s,*Dir_f;
   struct Nodo *Dir_d,*Dir_tp,*Dir_sp,*Ant_sp,*temp;
   struct Arco_Sucesor *temps;
   struct Plato_Arco *Pila_Arcos;
```

```
   struct Fuente_Plato *Pila_f;
   struct Destino_Plato *Pila_d;
   struct Res_Nodo_Plato *Pila_Res,*Aptres;

   FILE *Arch;

   void Libera_Conecciones ( struct Res_Nodo_Plato *,unsh,struct Nodo *,
 struct Nodo *,struct Fuente_Plato *,struct Destino_Plato * );
   void Imprimir ( void );
   void Ar_Ant_Borra ( struct Arco_Antecesor **,unsh );
   void Ar_Suc_Borra ( struct Arco_Sucesor **,unsh );
   void Se_Acabo ( void );
   void Chk_Nomb_PRB( char * );
   void Chk_Nomb_SOL( char * );
   unsh Primera_Solucion ( unsh,struct Plato_Arco *,struct Res_Nodo_Plato *,
   struct Nodo *,struct Nodo *,struct Nodo *, struct Nodo *, unsh * );
   unsh Flujo_Maximo ( unsh,unsh );
   unsh Mete_Fuente_Plato ( struct Fuente_Plato **,struct Nodo * );
   unsh Mete_Destino_Plato ( struct Destino_Plato **,struct Nodo * );
   unsh Res_Mete_Plato ( struct Res_Nodo_Plato **,unsh,unsh ,unsh  );
   unsh Mete_Plato_Arco ( struct Plato_Arco **,struct Nodo *,struct Nodo *,
      unsh  );
   unsh Agregar_Arco_Sucesor ( struct Arco_Sucesor **,unsh,unsh  );
   unsh Agr_Arco_Antecesor ( struct Arco_Antecesor **,unsh,unsh  );
   unsh Poner_Nodo_Res ( struct Res_Nodo_Plato *,unsh,unsh * );
   unsh Agarres ( struct Plato_Arco *,struct Res_Nodo_Plato *,
    struct Nodo *Dir_sp,struct Nodo *Dir_tp,unsh  *suma_Res );
   unsh Fuentes_Destinos ( struct Fuente_Plato *,struct Destino_Plato *,
struct Res_Nodo_Plato *,struct Nodo *,struct Nodo * );
   struct Nodo *Nodo_Agrega ( unsh,unsh * );
   struct Nodo *Nodo_Busca ( unsh,int * );

   Pila_Arcos=NULL;
   Pila_f=NULL;
   Pila_d=NULL;
   Pila_Res=NULL;

   if ( access("pareimm.tmp", 0) ) { //  Si no fue llamado por el manager
    PAREIMM = 0;
    if ( argc == 1 ) {
      cputs("\nUso: ford problema.prb\r\n\n");
      cputs("La extension .PRB es obligada para el nombre del archivo\r\n");
      cputs("que contiene al problema. Tal archivo debe tener una\r\n");
      cputs("primera linea que solo contenga el numero 4, una segunda\r\n");
      cputs("el numero 0 o 1, despues de los arcos una linea que \r\n");
      cputs("contenga 4 ceros (0 0 0 0), despues de los nodos 3 ceros \r\n");
      cputs("(0 0 0), en cada linea posterior los nodos fuente y destino\r\n");
      cputs("seguidas c/u de un cero si hay uno solo poner los numeros \r\n");
      cputs("separados por un espacio.\r\n);
      exit( 1 );
    }
```

```
    cputs("\nPAREIMM  Version 1.0, 1992-1993\r\nDEPARTAMENTO DE MATEMATICAS");
    cputs("\r\nUNIVERSIDAD DE SONORA\r\n");
  } else PAREIMM = 1;

  strcpy(Nomb_Prob, argv[1]);

  Chk_Nomb_PRB( Nomb_Prob );
  if ( access(Nomb_Prob, 0) ) {
     if ( PAREIMM )
exit( IO_ERR );
     else {
cprintf("\nERROR: Archivo %s no encontrado\r\n", Nomb_Prob);
exit( 1 );
     }
  }
  if ( !( Arch = fopen(Nomb_Prob, "rt") ) ) {
     if ( PAREIMM )
exit( IO_ERR );
     else {
cprintf("\nERROR: Archivo %s no abierto\r\n", Nomb_Prob);
exit( 1 );
     }
  }
  fscanf( Arch, "%d", &Tipo_Prob );
  if ( Tipo_Prob != 4 ) {
     fclose( Arch );
     if ( PAREIMM )
exit( NO_FORD );
     else {
cprintf("\nERROR: Archivo %s no adecuado\r\n", Nomb_Prob);
exit( 1 );
     }
  }
  fscanf(Arch,"%d", &Vf_Vd );
  if ( Vf_Vd==0 ) Var=0;
  else Var=2;
  do {
    fscanf(Arch,"%u %u %d %d",&Nodo_Inicial,&Nodo_Final,&Cap_Min,&Cap_Max );
    if ( Nodo_Inicial == 0  &&  Nodo_Final == 0 ) continue;
    Dir_NI=Nodo_Agrega ( Nodo_Inicial,&Num_Nodos );
    if( !Dir_NI ) {
fclose( Arch );
if ( PAREIMM )
  exit( NO_MEM );
else {
  cprintf("\nERROR: Memoria insuficiente para continuar\r\n",Nomb_Prob);
  exit( 1 );
}
    }
    Dir_NF=Nodo_Agrega ( Nodo_Final,&Num_Nodos );
    if( !Dir_NF ) {
```

```
fclose( Arch );
if ( PAREIMM )
   exit( NO_MEM );
else {
   cprintf("\nERROR: Memoria insuficiente para continuar\r\n",Nomb_Prob);
   exit( 1 );
}
     }
     Ind=Agr_Arco_Antecesor ( & ( Dir_NF->Arco_Antecesor ),Nodo_Inicial,0 );
     if( !Ind ) {
fclose( Arch );
if ( PAREIMM )
   exit( NO_MEM );
else {
   cprintf("\nERROR: Memoria insuficiente para continuar\r\n",Nomb_Prob);
   exit( 1 );
}
     }
     Ind=Agregar_Arco_Sucesor ( & ( Dir_NI->Arco_Sucesor ),Nodo_Final,Cap_Max );
     if( !Ind ) {
fclose( Arch );
if ( PAREIMM )
   exit( NO_MEM );
else {
   cprintf("\nERROR: Memoria insuficiente para continuar\r\n",Nomb_Prob);
   exit( 1 );
}
     }
     if ( Cap_Min > 0 ) {
Resultado=Mete_Plato_Arco ( &Pila_Arcos,Dir_NI,Dir_NF,Cap_Min );
if ( Resultado==0 ) {
   fclose( Arch );
   if ( PAREIMM )
      exit( NO_MEM );
   else {
      cprintf("\nERROR: Memoria insuficiente para continuar\r\n",Nomb_Prob);
      exit( 1 );
   }
}
Ind_ARes_Min=1;
     }
     if ( Num_Nodos > 20 ) {
fclose( Arch );
if ( PAREIMM )
   exit( TOO_BIG );
else {
   cprintf("\nERROR: Archivo %s demasiado grande\r\n",Nomb_Prob);
   exit( 1 );
}
     }
   } while ( Nodo_Inicial != 0  ||  Nodo_Final != 0 );
```

```
   do {
      fscanf(Arch, "%u %d %d", &Nodo_Res, &Res_Min, &Res_Max );
      if ( Nodo_Res == 0 ) continue;
      if ( Res_Min != 0 ) Ind_NRes_Min=1;
      Resultado=Res_Mete_Plato ( &Pila_Res,Nodo_Res,Res_Min,Res_Max );
      if ( Resultado==0 ) {
      fclose( Arch );
 if ( PAREIMM )
    exit( NO_MEM );
 else {
    cprintf("\nERROR: Memoria insuficiente para continuar\r\n",Nomb_Prob);
    exit( 1 );
 }
      }
   } while( Nodo_Res != 0 );

   if ( !Var ) {
      fscanf ( Arch,"%u %u",&s,&t );
      Buscando=Nodo_Busca ( s,&Resultado );
      if ( !Buscando ) Dir_s=Grafica;
      else Dir_s=Buscando->Siguiente;
      Buscando=Nodo_Busca ( t,&Resultado );
      if ( !Buscando ) Dir_t=Grafica;
      else Dir_t=Buscando->Siguiente;
   } else {
      do {
fscanf ( Arch,"%u",&Nodo_f );
if ( Nodo_f==0 ) continue;
Buscando=Nodo_Busca ( Nodo_f,&Resultado );
if ( Buscando==NULL )  Dir_f=Grafica;
else Dir_f=Buscando->Siguiente;
Resultado=Mete_Fuente_Plato ( &Pila_f,Dir_f );
if ( Resultado==0 ) {
   fclose( Arch );
   if ( PAREIMM )
    exit( NO_MEM );
   else {
    cprintf("\nERROR: Memoria insuficiente para continuar\r\n",Nomb_Prob);
    exit( 1 );
   }
}
     }while ( Nodo_f!=0 );
     do {
fscanf ( Arch,"%u",&Nodo_d );
if ( Nodo_d==0 ) continue;
Buscando=Nodo_Busca ( Nodo_d,&Resultado );
if ( Buscando==NULL )  Dir_d=Grafica;
else Dir_d=Buscando->Siguiente;
Resultado=Mete_Destino_Plato ( &Pila_d,Dir_d );
if ( Resultado==0 ) {
   fclose( Arch );
```

```
    if ( PAREIMM )
     exit( NO_MEM );
    else {
     cprintf("\nERROR: Memoria insuficiente para continuar\r\n",Nomb_Prob);
     exit( 1 );
    }
 }
      }while ( Nodo_d!=0 );
      Ant_s=Grafica;
      while ( Ant_s->Siguiente ) Ant_s=Ant_s->Siguiente;
      Dir_s=Nodo_Agrega ( Ant_s->Num_Nodo+1,&Num_Nodos );
      Dir_t=Nodo_Agrega ( Ant_s->Num_Nodo+2,&Num_Nodos );
   }
   fclose( Arch );

   Apt=Grafica;
   while ( Apt->Siguiente ) Apt=Apt->Siguiente;

   Ind=Poner_Nodo_Res ( Pila_Res,Apt->Num_Nodo,&Num_Nodos );
   if ( Ind==0 ) {
      fclose( Arch );
      if ( PAREIMM )
       exit( NO_MEM );
      else {
       cprintf("\nERROR: Memoria insuficiente para continuar\r\n", Nomb_Prob);
       exit( 1 );
      }
   }

   if ( Pila_Arcos || Ind_NRes_Min ){
      Ant_sp=Grafica;
      while ( Ant_sp->Siguiente ) Ant_sp=Ant_sp->Siguiente;
      Dir_sp=Nodo_Agrega ( Ant_sp->Num_Nodo+1,&Num_Nodos );
      Dir_tp=Nodo_Agrega ( Ant_sp->Num_Nodo+2,&Num_Nodos );
   }

   Aptres=Pila_Res;
   while ( Aptres ) {
    if ( Aptres->Res_Min > 0 ) {
     Buscando=Nodo_Busca ( Aptres->Nodo_Res,&Resultado );
     if ( Buscando==NULL )  Apt=Grafica;
     else Apt=Buscando->Siguiente;

     Resultado=Mete_Plato_Arco ( &Pila_Arcos,Apt,Aptres->Dir_Cuate,Aptres->Res_Min );
     if ( Resultado==0 ) {
fclose( Arch );
if ( PAREIMM )
   exit( NO_MEM );
else {
   cprintf("\nERROR: Memoria insuficiente para continuar\r\n", Nomb_Prob);
   exit( 1 );
```

```
}
     }
    }
    Aptres=Aptres->Siguiente;
   }

   Ind=Agarres ( Pila_Arcos,Pila_Res,Dir_sp,Dir_tp,&Suma_Res );
   if ( Ind==0 ) {
      fclose( Arch );
      if ( PAREIMM )
       exit( NO_MEM );
      else {
       cprintf("\nERROR: Memoria insuficiente para continuar\r\n", Nomb_Prob);
       exit( 1 );
      }
   }

   Ind=Fuentes_Destinos ( Pila_f,Pila_d,Pila_Res,Dir_s,Dir_t );
   if ( Ind==0 ) {
      fclose( Arch );
      if ( PAREIMM )
       exit( NO_MEM );
      else {
       cprintf("\nERROR: Memoria insuficiente para continuar\r\n", Nomb_Prob);
       exit( 1 );
      }
   }

   Aptres=Pila_Res;
   while ( Aptres && Aptres->Nodo_Res!=t )  Aptres=Aptres->Siguiente;
   if ( Aptres )  Dir_t=Aptres->Dir_Cuate;
   if ( Pila_Arcos ) {
      Indicador=Primera_Solucion ( Suma_Res,Pila_Arcos,Pila_Res,Dir_s,Dir_t,
 Ant_sp,Dir_sp,&Num_Nodos );
      if ( Indicador ==0 ) {
fclose( Arch );
if ( PAREIMM )
 exit ( NO_SOL );
else {
 cprintf ( "No existe solucion factible al problema por la restricciones
establecidas." );
 exit ( 1 );
}
      }
      if ( Indicador==2 ) {
fclose( Arch );
if ( PAREIMM )
 exit( NO_MEM );
else {
 cprintf("\nERROR: Memoria insuficiente para continuar\r\n", Nomb_Prob);
 exit( 1 );
```

```
}
     }
     if ( Indicador ==10 ) {
fclose( Arch );
if ( PAREIMM )
   exit ( NO_SOL );
else {
   cprintf ( "Problemas en la implementacion." );
   exit ( 1 );
}
     }
  }
// FORD-FULKERSON

  Indicador=Flujo_Maximo ( Dir_s->Num_Nodo,Dir_t->Num_Nodo );

  if ( Indicador==0 && Ind_ARes_Min==0 && Ind_NRes_Min==0 ) {
     fclose ( Arch );
     if ( PAREIMM )
      exit ( SOL_CERO );
     else {
      cprintf ( "\nEl flujo entre los nodos fuente y los destino es cero\n" );
      exit ( 0 );
     }
  }

  if(Var || Pila_Res) Libera_Conecciones(Pila_Res,Var,Ant_s,Dir_t,Pila_f,Pila_d);

  Chk_Nomb_SOL( Nomb_Prob );
  if ( !( Arch = fopen(Nomb_Prob, "wt") ) ) {
     if ( PAREIMM ) exit( IO_ERR );
     else {
cprintf("\nERROR: Archivo %s no abierto\r\n", Nomb_Prob);
exit( 1 );
     }
  }
  if ( fprintf ( Arch, "4\n" ) == EOF ) {
     fclose( Arch );
     unlink( Nomb_Prob );
     if ( PAREIMM )
exit( IO_ERR );
     else {
cputs( "\nERROR: Problemas al escribir la solucion\r\n" );
exit( 1 );
     }
  }
  temp=Grafica;
  while ( temp ) {
   temps=temp->Arco_Sucesor;
   while ( temps ) {
    if (fprintf(Arch,"%u %u %u\t\n",temp->Num_Nodo,temps->Nodo_Final,
```

```
      temps->Flujo )==EOF) {
       fclose( Arch );
       unlink( Nomb_Prob );
       if ( PAREIMM ) exit( IO_ERR );
       else {
  cputs("ERROR: Problemas al escribir la solucion\r\n");
  exit( 1 );
       }
     }
     temps=temps->Siguiente;
    }
    temp=temp->Siguiente;
   }
   if ( fprintf(Arch, "0 0 0\n" ) == EOF ) {
      fclose( Arch );
      unlink( Nomb_Prob );
      if ( PAREIMM )
exit( IO_ERR );
      else {
cputs("\nERROR: Problemas al escribir la solucion\r\n");
exit( 1 );
      }
   }
   temp=Grafica;
   while ( temp ) {
      if( temp->Ant_Cad_Aum )
if ( fprintf(Arch, "%u\n",temp->Num_Nodo) == EOF ) {
    fclose( Arch );
    unlink( Nomb_Prob );
    if ( PAREIMM )
       exit( IO_ERR );
    else {
       cputs("\nERROR: Problemas al escribir la solucion\r\n");
       exit( 1 );
    }
}
      temp=temp->Siguiente;
  }

  if ( fprintf(Arch, "0" ) == EOF ) {
     fclose( Arch );
     unlink( Nomb_Prob );
     if ( PAREIMM )
exit( IO_ERR );
     else {
cputs("\nERROR: Problemas al escribir la solucion\r\n");
exit( 1 );
     }
  }
  fclose( Arch );
  if ( !PAREIMM ) {
```

```
      cputs("\nSolucion grabada en: ");
      cputs(Nomb_Prob);
      putch('\n');
   }
   exit ( EXITO );

}

//   ---------------------------------------------------------------------
void Chk_Nomb_PRB ( char *Nomb )
{
  unsh Pos;
  unsh Pos_Char(char *, char);

  Pos = ( Pos_Char(Nomb, '.') ? Pos_Char(Nomb, '.') - 1 : strlen(Nomb) );

  Nomb[Pos] = '.';
  Nomb[Pos+1] = 'P';
  Nomb[Pos+2] = 'R';
  Nomb[Pos+3] = 'B';
  Nomb[Pos+4] = NULL;
}

//   ---------------------------------------------------------------------
void Chk_Nomb_SOL ( char *Nomb )
{
  unsh Pos;
  unsh Pos_Char(char *, char);

  Pos = ( Pos_Char(Nomb, '.') ? Pos_Char(Nomb, '.') - 1 : strlen(Nomb) );

  Nomb[Pos] = '.';
  Nomb[Pos+1] = 'S';
  Nomb[Pos+2] = 'O';
  Nomb[Pos+3] = 'L';
  Nomb[Pos+4] = NULL;
}

//   ---------------------------------------------------------------------
unsh Pos_Char(char *s, char c)
{
  unsh i=1;

  while ( *s ) {
    if ( *s == c )
      return ( i );
    s++;
    i++;
  }
  return ( 0 );
}
```

```
/*
 ************************************************************************
     VERANO 93                                          PAREIMM
     NOMBRE DEL ARCHIVO: FORD_FUN.H
Este archivo agraga y borra elementos de listas
     utilizadas por el algoritmo.
     FUNCIONES QUE INCLUYE:
Nodo_Busca: Busca un elemento en la lista de nodos.
Nodo_Agrega: Agrega un elemento a la lista de nodos.
Agregar_Arco_Sucesor: Agrega un elemento a la lista de
     arcos sucesores de un nodo.
Agr_Arco_Antecesor: Agrega un elemento a la lista de
   arcos antecesores de un nodo.
Mete_Plato_Arco: Agrega un elemento a la pila de arcos
restringidos.
Mete_Plato: Agrega un elemento a la pila de nodos por
   revisar.
Quita_Plato: Quita un elemento a la pila de nodos por
    revisar.
Mete_Fuente_Plato: Agrega un elemento a la pila de
  nodos fuente.
Mete_Destino_Plato: Agrega un elemento a la pila de
   nodos destino.
Imprimir: Imprime la solucion obtenida.
B_L_Ant: Borra la lista de antecesores de un nodo.
B_L_Suc: Borra la lista de sucesores de un nodo.
Borrar_Lista: Borra la lista de nodos.
Se_Acabo: Libera la Grafica.
 ************************************************************************

struct Nodo *Nodo_Agrega ( unsh Nodo,unsh *Num_Nodos )

/*    Esta funcion crea y llena un lista de nodos

      Las variables principales que se utilizan son:

 *p: Direccion al primer elemento de la lista de nodos.
 r: Direccion del nuevo elemento de la lista de nodos.          */
{
   int estancia=0;
   struct Nodo *buscando,*r;
   struct Nodo *Nodo_Busca ( unsh,int * );

   buscando=Nodo_Busca( Nodo,&estancia );
   if(estancia<0 && !buscando)  return ( Grafica );
   if(estancia<0 && buscando)  return ( buscando->Siguiente );
   r=(struct Nodo *) malloc ( sizeof (struct Nodo) );
   if(r==NULL) return (NULL);
   r->Num_Nodo=Nodo;
   r->Ant_Cad_Aum=0;
```

```
   r->Cap_Cad_Aum=INFINITO;
   r->Arco_Antecesor=NULL;
   r->Arco_Sucesor=NULL;
   (*Num_Nodos)++;
   switch(estancia){
      case 0:
 r->Siguiente=NULL;
 Grafica=r;
 break;
      case 1:
 r->Siguiente=Grafica;
 Grafica=r;
 break;
      case 2:
 r->Siguiente=buscando->Siguiente;
 buscando->Siguiente=r;
 break;
      case 3:
 r->Siguiente=NULL;
 buscando->Siguiente=r;
 break;
   }
   return (r);
}

// ------------------------------------------------------------------------
struct Nodo *Nodo_Busca ( unsh Nodo,int *Resultado )

/*    Busca un nodo, retorna el anterior

      Las variables principales que se utilizan son:

 Nodo: Numero del nodo buscado.
 anterior: Direccion al anterior del nodo buscado.          */

{
   struct Nodo *s,*anterior;

   s=Grafica;
   if(s==NULL){
      *Resultado=0;        // La lista esta vacia
      return(NULL);
   }
   if(Nodo<s->Num_Nodo){
      *Resultado=1;        // El arco no esta pero
      return(NULL);    //  cabe al principio
   }
   if(Nodo==s->Num_Nodo){
      *Resultado=-1;    // El arco esta al principio
      return(NULL);
   }
```

```
   anterior=s;
   s=s->Siguiente;
   while(Nodo>s->Num_Nodo && s!=NULL){
      anterior=s;
      s=s->Siguiente;
   }
   if((anterior->Siguiente)->Siguiente==NULL && Nodo==(anterior->Siguiente)->Num_Nodo)
      *Resultado=-3;
   if(anterior->Siguiente==NULL && Nodo>anterior->Num_Nodo)
      *Resultado=3;
   if(anterior->Siguiente!=NULL && Nodo<(anterior->Siguiente)->Num_Nodo)
      *Resultado=2;
   if((anterior->Siguiente)->Siguiente!=NULL && Nodo==(anterior->Siguiente)->Num_Nodo)
      *Resultado=-2;
   return(anterior);
}

// ------------------------------------------------------------
unsh Agregar_Arco_Sucesor ( struct Arco_Sucesor  **p,unsh Nodo_Final,unsh  Cap_Max )

/*   Esta funcion inserta un Arco_Sucesor a la digrafica

      Las variables principales que se utilizan son:
      *p: Direccion del primer elemento de la lista de arcos sucesores.
      r: Direccion del nuevo elemento de la lista de arcos.            */

{
   struct Arco_Sucesor *t,*r,*Anterior;

   r=new Arco_Sucesor;  // Pidiendo memoria
   if ( r==NULL ) return ( 0 );  //      No hubo
   r->Nodo_Final=Nodo_Final;
   r->Flujo=0;
   r->Cap_Max=Cap_Max;
   r->Siguiente=NULL;
   t=*p;
   if ( *p==NULL )  *p=r;
   else{
      while ( t ) {
 Anterior=t;
 t=t->Siguiente;
      }
      Anterior->Siguiente=r;
   }
   return ( 1 );
}

// ------------------------------------------------------------
unsh Agr_Arco_Antecesor ( struct Arco_Antecesor **p,unsh Nodo_Inicial,unsh  Cap_Min )

/*   Esta funcion inserta un arco antecesor a la digrafica
```

```
      Las variables principales que se utilizan son:
      *p: Direccion del primer elemento de la lista de arcos antecesores.
      r: Direccion del nuevo elemento de la lista de arcos.          */

{
   struct Arco_Antecesor *t,*r,*Anterior;

   r=new Arco_Antecesor;          // Pidiendo memoria
   if ( r==NULL ) return ( 0 );  //      No hubo
   r->Nodo_Inicial=Nodo_Inicial;
   r->Flujo=0;
   r->Cap_Min=Cap_Min;
   r->Siguiente=NULL;
   t=*p;
   if ( *p==NULL )  *p=r;
   else{
      while ( t ) {
 Anterior=t;
 t=t->Siguiente;
      }
      Anterior->Siguiente=r;
   }
   return ( 1 );
}

// ------------------------------------------------------------------
unsh Mete_Plato_Arco(struct Plato_Arco **prin,struct Nodo *Nodo_Inicial,
struct Nodo *Nodo_Final,unsh  Cap_Min)

/* Esta funcion mete un elemento en la pila de arcos restringidos.

      Las variables principales que se utilizan son:

 *prin: Direccion del primer elemento de la pila.
 R: Direccion del nuevo elemento de la pila.
 Nodo_Inicial: Nodo inicial del arco.
 Nodo_Final: Nodo final del arco.
 Cap_Min: Capacidad minima del arco.                              */

{
   struct Plato_Arco *temp,*R;

   temp=*prin;
   R=new Plato_Arco;
   if ( R==NULL )  return ( 0 );
   R->Dir_Nodo_Inicial=Nodo_Inicial;
   R->Dir_Nodo_Final=Nodo_Final;
   R->Cap_Min=Cap_Min;
   R->Siguiente=temp;
   *prin=R;
```

```
   return ( 1 );
}

//      ------------------------------------------------------------------
unsh Mete_Plato ( struct Plato **prin,struct Nodo *Nodo )

/* Esta funcion mete un elemento en la pila de nodos por revisar.

      Las variables principales que se utilizan son:

 *prin: Direccion del primer elemento de la pila.
 r: Direccion del nuevo elemento de la pila.                    */

{
   struct Plato *temp,*r;

   temp=*prin;
   r=new Plato;
   if ( r==NULL )  return ( 0 );
   r->Nodo=Nodo;
   r->Siguiente=temp;
   *prin=r;
   return ( 1 );
}

//      ------------------------------------------------------------------
unsh Mete_Fuente_Plato ( struct Fuente_Plato **prin,struct Nodo *Dir_f )

/*    Esta funcion mete un elemento en la pila de nodos fuente.

      Las variables principales que se utilizan son:

 *prin: Direccion del primer elemento de la pila.
 r: Direccion del nuevo elemento de la pila.
 Dir_f: Direccion del nodo fuente.                              */

{
   struct Fuente_Plato *temp,*r;
   temp=*prin;

   r=new Fuente_Plato;
   if ( r==NULL )  return ( 0 );
   r->Dir_f=Dir_f;
   r->Siguiente=temp;
   *prin=r;
   return ( 1 );
}

//      ------------------------------------------------------------------
unsh Mete_Destino_Plato ( struct Destino_Plato **prin,struct Nodo *Dir_d )
```

```
/*    Esta funcion mete un elemento en la pila de nodos destino.

      Las variables principales que se utilizan son:

 *prin: Direccion del primer elemento de la pila.
 r: Direccion del nuevo elemento de la pila.
 Dir_d: Direccion del nodo destino.                              */

{
   struct Destino_Plato *temp,*r;

   temp=*prin;
   r=new Destino_Plato;
   if ( r==NULL )  return ( 0 );
   r->Dir_d=Dir_d;
   r->Siguiente=temp;
   *prin=r;
   return ( 1 );
}

//      ------------------------------------------------------------
unsh Res_Mete_Plato(struct Res_Nodo_Plato **prin,unsh Nodo_Res,unsh Res_Min,
unsh Res_Max )

/*    Esta funcion mete un Plato en la pila de nodos restringidos.

      Las variables principales que se utilizan son:

 *prin: Direccion del primer elemento de la pila.
 r: Direccion del nuevo elemento de la pila.
 Nodo_Res: Numero del nodo restringido.
 Res_Min: Restriccion minima del nodo.
 Res_Max: Restriccion maxima del nodo.                           */

{
   struct Res_Nodo_Plato *temp,*r;

   temp=*prin;
   r=new Res_Nodo_Plato;
   if ( r==NULL )  return ( 0 );
   r->Nodo_Res=Nodo_Res;
   r->Res_Min=Res_Min;
   r->Res_Max=Res_Max;
   r->Dir_Cuate=NULL;
   r->Siguiente=temp;
   *prin=r;
   return ( 1 );
}

// ------------------------------------------------------------
struct Nodo *Quita_Plato ( struct Plato **prin )
```

```
/*    Esta funcion saca un elemento de la pila de nodos por revisar.

      La variable principal que se utiliza es:

 *prin: Direccion al primer elemento de la pila.                */

{
   struct Nodo *Nodo;
   struct Plato *temp;

   temp=*prin;
   *prin=temp->Siguiente;
   Nodo=temp->Nodo;
   delete temp;
   return ( Nodo );
}

// ------------------------------------------------------------
void Borrar_Lista ( void )

/*    Esta funcion borra una lista de nodos

      La variable principal que se utiliza es:

      Grafica: Direccion del primer elemento de la lista de nodos.        */

{
   struct Nodo *y;

   while ( Grafica ) {
      y=Grafica;
      Grafica= Grafica->Siguiente;
      free ( y );
   }
}

// ------------------------------------------------------------
void B_L_Ant ( struct Arco_Antecesor **p )

/*    Esta funcion borra la lista de arcos  antecesores de un nodo.

      La variable principal que se utiliza es:

      *p: Direccion del primer elemento de la lista de arcos antecesores. */

{
   struct Arco_Antecesor *y;

   while ( *p ) {
      y=*p;
```

```
      *p= ( *p ) ->Siguiente;
      delete y;
   }
}

// ------------------------------------------------------------------
void B_L_Suc ( struct Arco_Sucesor **p )

/*    Esta funcion borra la lista de arcos sucesores de un nodo.

      La variable principal que se utiliza es:

      *p: Direccion del primer elemento de la lista de arcos sucesores. */

{
   struct Arco_Sucesor *y;

   while ( *p ) {
      y=*p;
      *p= ( *p ) ->Siguiente;
      delete y;
   }
}

// ------------------------------------------------------------------
void Ar_Suc_Borra ( struct Arco_Sucesor **p, unsh i )

/*    Esta funcion borra un elemento de una lista de arcos sucesores.

      La variable principal que se utiliza es:

      *p: Direccion del primer elemento de la lista de arcos sucesores.
      i: Numero del nodo final del arco a borrar.                    */

{
   struct Arco_Sucesor *t,*Anterior;

   t=*p;
   if ( (*p)->Nodo_Final == i ){
      delete t;
      *p=NULL;
      return;
   }
   while (  t && t->Nodo_Final != i ){
     Anterior=t;
     t=t->Siguiente;
   }
   if ( t )  Anterior->Siguiente=t->Siguiente;
   delete t;
   return;
}
```

```
// ------------------------------------------------------------
void Ar_Ant_Borra ( struct Arco_Antecesor **p, unsh i )

/*    Esta funcion borra un elemento de una lista de arcos antecesores

      La variable principal que se utiliza es:

      *p: Direccion del primer elemento de la lista de arcos antecesores.
      i: Numero del nodo inicial del arco a borrar.                        */

{
   struct Arco_Antecesor *t,*Anterior;

   t=*p;
   Anterior=*p;
   if ( (*p)->Nodo_Inicial == i ){
      delete t;
      *p=NULL;
      return;
   }
   while (  t && t->Nodo_Inicial != i ){
     Anterior=t;
     t=t->Siguiente;
   }
   if ( t )  Anterior->Siguiente=t->Siguiente;
   delete t;
   return;
}

// ------------------------------------------------------------
void Se_Acabo ( void )

/*    Esta funcion libera toda la Grafica.

      La variable principal que se utiliza es:

 Grafica: Direccion del primer elemento de la lista de nodos.

      Las funciones que se utilizan son:

 Borra_Lista: Borra la lista de Nodos.
 B_L_Ant: Borra la lista de arcos Antecesores de un Nodo.
 B_L_Suc: Borra la lista de arcos Antecesores de un Nodo.        */

{
   struct Nodo *Apt;

   void Borrar_Lista ( void );
   void B_L_Ant ( struct Arco_Antecesor ** );
   void B_L_Suc ( struct Arco_Sucesor ** );
```

```
   Apt=Grafica;
   while ( Apt ) {
      B_L_Ant ( & ( Apt->Arco_Antecesor )  );
      B_L_Suc ( & ( Apt->Arco_Sucesor )  );
      Apt=Apt->Siguiente;
   }
   Borrar_Lista ( );
}

/*
 ************************************************************************
     VERANO 93                                                  PAREINM
     NOMBRE DEL ARCHIVO: FORD_ALG.H
Este archivo aplica las funciones principales
utilizadas por el algoritmo.
     FUNCIONES QUE INCLUYE:
Agarres: Agrega arcos auxiliares si existen arcos con
restriccion
Flujo_Maximo: Aplica el algoritmo Ford_Fulkerson.
Enc_Cad_Aum: Encuentra una cadena aumentante.
Poner_Nodo_Res: Agrega arcos auxiliares si existen
       nodos con restriccion.
Fuentes_Destinos: Agrega arcos auxiliares si existen
 varias fuentes y varios destinos.
Primera_Solucion: Aplica el algoritmo Ford_Fulkerson
 para encontrar una primera solucion.
Libera_Conecciones:  Libera las conecciones auxiliares.
 ************************************************************************
    */
unsh Agarres(struct Plato_Arco *Pila_Arco,struct Res_Nodo_Plato *Pila_Res,
     struct Nodo *Dir_sp,struct Nodo *Dir_tp,unsh  *Suma_Res)

/* Esta funcion agrega los arcos auxiliares con restricciones

      Opera de la siguiente manera:

   a.- Se toma cada arco con restriccion y se cambian las capacidades:
      La capacidad minima se hace cero y la maxima igual a la diferencia
      entre las restricciones del arco.
   b.- Revisar si estan si hay arco entre los nodos inicial y t-prima, de ser
      asi se aumenta a la capacidad maxima de este arco la capacidad
      minima del arco restringido, si no, el arco se agrega con capacidad
      maxima la restriccion minima del arco restringido y minima cero.
   c.- Hacer (b) con el nodo final del arco restringido y s-prima.
   d.- Las restricciones minimas de los arcos restringidos se suman
      y se guardan en Suma_Res.

      Las variables principales que se utilizan son:

Pila_Arco: Direccion de la pila de arcos restringidos.
```

```
Dir_sp: Direccion del nodo fuente (auxiliar para primera solucion).
Dir_tp: Direccion del nodo destino (auxiliar para primera solucion).
Suma_Res: Suma de capacidades minimas de arcos con restriccion.

     Las funciones que se utilizan son:

Agregar_Arco_Sucesor: Agrega un elemento a la lista de
     arcos sucesores de un Nodo.
Agr_Arco_Antecesor: Agrega un elemento a la lista de
   arcos antecesores de un Nodo.              */

{
   unsh Agregar_Arco_Sucesor ( struct Arco_Sucesor **,unsh,unsh  );
   unsh Agr_Arco_Antecesor ( struct Arco_Antecesor **,unsh,unsh  );

   struct Nodo *apuntador;
   struct Plato_Arco *apt_arcos;
   struct Res_Nodo_Plato *aptres;
   struct Arco_Sucesor *Dir_Arc_NF,*si_o_no,*esta_o_no,*temp;
   struct Arco_Antecesor *Dir_Arc_NI;
   unsh  Difer;
   unsh z,y;

   apt_arcos=Pila_Arco;
   while ( apt_arcos ) {
      aptres=Pila_Res;
      while ( aptres &&  ( apt_arcos->Dir_Nodo_Inicial ) ->Num_Nodo!=
  aptres->Nodo_Res ) aptres=aptres->Siguiente;
      if(aptres==NULL||(apt_arcos->Dir_Nodo_Final)->Num_Nodo==
(aptres->Dir_Cuate)->Num_Nodo) apuntador=apt_arcos->Dir_Nodo_Inicial;
      else apuntador=aptres->Dir_Cuate;
      Dir_Arc_NF= apuntador->Arco_Sucesor;
      while(Dir_Arc_NF && Dir_Arc_NF->Nodo_Final!=
      (apt_arcos->Dir_Nodo_Final)->Num_Nodo) Dir_Arc_NF=Dir_Arc_NF->Siguiente;
      Dir_Arc_NI= ( apt_arcos->Dir_Nodo_Final ) ->Arco_Antecesor;
      while ( Dir_Arc_NI && Dir_Arc_NI->Nodo_Inicial != apuntador->Num_Nodo )
     Dir_Arc_NI=Dir_Arc_NI->Siguiente;
      Difer=Dir_Arc_NF->Cap_Max - apt_arcos->Cap_Min;
      temp= apuntador->Arco_Sucesor;
      esta_o_no=temp;
      while ( temp && temp->Nodo_Final!=Dir_tp->Num_Nodo ) {
esta_o_no=temp;
temp=temp->Siguiente;
      }
      if ( temp ) {
if ( temp== apuntador->Arco_Sucesor )
  esta_o_no->Cap_Max=esta_o_no->Cap_Max + apt_arcos->Cap_Min;
else (esta_o_no->Siguiente)->Cap_Max=(esta_o_no->Siguiente)->Cap_Max+
  apt_arcos->Cap_Min;
      }
      else{
```

```
 y=Agregar_Arco_Sucesor(&(apuntador->Arco_Sucesor),Dir_tp->Num_Nodo,
  apt_arcos->Cap_Min);
 if ( y==0 )  return ( 0 );
 z=Agr_Arco_Antecesor(&(Dir_tp->Arco_Antecesor),apuntador->Num_Nodo,0);
 if ( z==0 )  return ( 0 );
      }
      temp=Dir_sp->Arco_Sucesor;
      si_o_no=temp;
      while(temp&&temp->Nodo_Final!=(apt_arcos->Dir_Nodo_Final)->Num_Nodo){
 si_o_no=temp;
 temp=temp->Siguiente;
      }
      if ( temp ) {
 if(temp==Dir_sp->Arco_Sucesor) si_o_no->Cap_Max=si_o_no->Cap_Max+
   apt_arcos->Cap_Min;
 else(si_o_no->Siguiente)->Cap_Max=(si_o_no->Siguiente)->Cap_Max+
   apt_arcos->Cap_Min;
      }
      else{
 y=Agregar_Arco_Sucesor(&(Dir_sp->Arco_Sucesor),
    (apt_arcos->Dir_Nodo_Final)->Num_Nodo,apt_arcos->Cap_Min);
 if ( y==0 )  return ( 0 );
 z=Agr_Arco_Antecesor(&((apt_arcos->Dir_Nodo_Final)->Arco_Antecesor),
  Dir_sp->Num_Nodo,0);
 if ( z==0 )  return ( 0 );
      }
      *Suma_Res=*Suma_Res+apt_arcos->Cap_Min;
      Dir_Arc_NF->Cap_Max=Difer;
      apt_arcos=apt_arcos->Siguiente;
   }
   return ( 1 );
}

// ------------------------------------------------------------------
unsh Flujo_Maximo ( unsh s,unsh t )

/* Esta funcion encuentra el flujo maximo entre s y t

Opera de la siguiente manera:

     a.- Encuentra una cadena aumentante entre los nodos s y t.
     b.- Actualiza el flujo en los arcos.

Las variables principales que se utilizan son:
   s: Nodo fuente.
   t: Nodo destino.
   Contador: Cuenta las veces que entra a la funcion. Si
    Contador=1 y no se encuentra cadena aumentante,
    el problema no tiene solucion.

Las funciones que se utilizan son:
```

```
   Nodo_Busca: Busca un elemento en la lista de Nodos.
   Enc_Cad_Aum: Encuentra una cadena aumentante.              */

{
   struct Nodo *Enc_Cad_Aum ( unsh,unsh );
   struct Nodo *Nodo_Busca ( unsh,int * );

   struct Nodo *apt,*inter,*ninter,*nfinal,*temp;
   struct Arco_Antecesor *tempa;
   struct Arco_Sucesor *temps;

   unsh Contador=0;
   unsh  Cambia;
   int Resultado;

   apt=NULL;
   inter=NULL;
   ninter=NULL;
   nfinal=NULL;
   temp=NULL;
   tempa=NULL;
   temps=NULL;
     nfinal=Enc_Cad_Aum ( s,t );
     Contador=Contador+1;
     if ( nfinal==NULL )
if ( Contador==1 )  return ( 0 );
else return ( 1 );
     temp=nfinal;
     while ( temp->Num_Nodo!=s )
       if ( temp->Ant_Cad_Aum>0 ) {
 tempa=temp->Arco_Antecesor;
 while(tempa && tempa->Nodo_Inicial!=temp->Ant_Cad_Aum) tempa=tempa->Siguiente;
 if(!tempa) return (10);
 Cambia=tempa->Flujo + nfinal->Cap_Cad_Aum;
 tempa->Flujo=Cambia;
 inter=Nodo_Busca ( tempa->Nodo_Inicial,&Resultado );
 if ( inter==NULL )  ninter=Grafica;
 else ninter=inter->Siguiente;
 temps=ninter->Arco_Sucesor;
 temps->Flujo=Cambia;
 temp=ninter;
 continue;
      }else{
  temps=temp->Arco_Sucesor;
  while(temps->Nodo_Final!=(-1*temp->Ant_Cad_Aum)) temps=temps->Siguiente;
  Cambia=temps->Flujo - nfinal->Cap_Cad_Aum;
  temps->Flujo=Cambia;
  inter=Nodo_Busca ( temps->Nodo_Final,&Resultado );
  if ( inter==NULL) ninter=Grafica;
  else ninter=inter->Siguiente;
```

```
  tempa=ninter->Arco_Antecesor;
  while ( tempa->Nodo_Inicial!=temp->Num_Nodo) tempa=tempa->Siguiente;
  tempa->Flujo=Cambia;
  temp=ninter;
       }
     apt=Grafica;
     while ( apt){
apt->Ant_Cad_Aum=0;
apt->Cap_Cad_Aum=INFINITO;
apt=apt->Siguiente;
     }
   }
}

// ------------------------------------------------------------------
struct Nodo *Enc_Cad_Aum ( unsh s,unsh t)

/* Esta funcion encuentra una cadena aumentante entre s y t

Opera de la siguiente manera:

      a.- Marca el nodo s como inicial de la cadena.
      b.- Revisa los sucesores que no han sido tocados por el algoritmo
 (no etiquetados) y si se puede aumentar el flujo por los arcos,
 marca sus nodos extremos y si el nodo extremo no es el t
 los mete en una pila.
      c.- Saca un elemento de la pila. Ir a (b).

      Las variables principales que se utilizan son:

 s: Nodo fuente.
 t: Nodo destino.
 Pila: Pila de nodos por revisar.

      Las funciones que se utilizan son:

 Nodo_Busca: Busca un elemento en la lista de Nodos.
    restringidos.
 Mete_Plato: Agrega un elemento a la pila de Nodos por
    revisar.
 Quita_Plato: Quita un elemento a la pila de Nodos por
     revisar.                                          */

{
 void Mete_Plato ( struct Plato **,struct Nodo * );
 struct Nodo *Quita_Plato ( struct Plato ** );
 struct Nodo *Nodo_Busca ( unsh,int * );

 struct Nodo *apts,*anrev,*b_n_s,*b_n_a;
 struct Arco_Sucesor *temps;
 struct Arco_Antecesor *tempa;
```

```
 struct Plato *Pila;

 int Resultado;
 unsh  Difer;

 Pila=NULL;
 apts=Nodo_Busca ( s,&Resultado );  // buscando s
 if ( apts==NULL )  anrev=Grafica;  // anrev: apuntador al Nodo por revisar.
 else anrev=apts->Siguiente;
 anrev->Ant_Cad_Aum=anrev->Num_Nodo;
 while ( anrev ) {
  temps=anrev->Arco_Sucesor;  // apuntador a la lista de arcos
  while ( temps ) {
   Difer=temps->Cap_Max-temps->Flujo;
   if ( Difer==0 ) {
      temps=temps->Siguiente;
      continue;
   }
   b_n_s=Nodo_Busca(temps->Nodo_Final,&Resultado);// busca el Nodo final del arco
   if ( b_n_s==NULL ) {
    if (  ( Grafica ) ->Ant_Cad_Aum==0 ) {
     ( Grafica ) ->Ant_Cad_Aum=anrev->Num_Nodo;
     if ( Difer<=anrev->Cap_Cad_Aum )   ( Grafica ) ->Cap_Cad_Aum=Difer;
     else  ( Grafica ) ->Cap_Cad_Aum=anrev->Cap_Cad_Aum;
     if (  ( Grafica ) ->Num_Nodo==t )  return ( Grafica );
     Mete_Plato ( &Pila,Grafica );
    }
   }else{
    if (  ( b_n_s->Siguiente ) ->Ant_Cad_Aum==0 ) {
     ( b_n_s->Siguiente ) ->Ant_Cad_Aum=anrev->Num_Nodo;
     if(Difer<=anrev->Cap_Cad_Aum ) ( b_n_s->Siguiente ) ->Cap_Cad_Aum=Difer;
     else  ( b_n_s->Siguiente ) ->Cap_Cad_Aum=anrev->Cap_Cad_Aum;
     if (  ( b_n_s->Siguiente ) ->Num_Nodo!=t )
     Mete_Plato ( &Pila,b_n_s->Siguiente );
    }
   }
   temps=temps->Siguiente;
  }
  tempa=anrev->Arco_Antecesor;
  while ( tempa ) {
   if ( tempa->Flujo==tempa->Cap_Min ) {
      tempa=tempa->Siguiente;
   }else{
    b_n_a=Nodo_Busca ( tempa->Nodo_Inicial,&Resultado );// busca el Nodo inicial del arco
    if ( b_n_a==NULL ) {
     if (  ( Grafica ) ->Ant_Cad_Aum==0 ) {
 ( Grafica ) ->Ant_Cad_Aum=- ( anrev->Num_Nodo );
 ( Grafica ) ->Cap_Cad_Aum=tempa->Flujo - tempa->Cap_Min;
if (  ( Grafica ) ->Num_Nodo==t )  return ( Grafica );
Mete_Plato ( &Pila,Grafica );
     }
```

```
    }else{
     if (  ( b_n_a->Siguiente ) ->Ant_Cad_Aum==0 ) {
 ( b_n_a->Siguiente ) ->Ant_Cad_Aum=- ( anrev->Num_Nodo );
 ( b_n_a->Siguiente ) ->Cap_Cad_Aum=tempa->Flujo - tempa->Cap_Min;
if (( b_n_a->Siguiente ) ->Num_Nodo==t ) return ( b_n_a->Siguiente );
Mete_Plato ( &Pila,b_n_a->Siguiente );
     }
    }
    tempa=tempa->Siguiente;
   }
  }
  if ( Pila==NULL )  anrev=NULL;
  else anrev=Quita_Plato ( &Pila );
 }
 return ( NULL );
}

// ------------------------------------------------------------------
unsh Poner_Nodo_Res(struct Res_Nodo_Plato *Pila_Res,unsh Suma,unsh *Num_Nodos)

/*    Aqui se parte el Nodo restringido en dos y se hacen los
     cambios debidos

      Opera de la siguiente manera:

   a.- Se agrega un arco auxiliar de el nodo restringido a un nodo
      auxiliar ( Cuate ).
   b.- Los sucesores del nodo restringido pasan a ser sucesores del cuate.

      Las variables principales que se utilizan son:

 Pila_Res: Direccion del primer elemento de la pila de
  nodos restringidos.
 Suma:  Numero de nodos de la digrafica mas dos.
 N_Res: Numero de nodos con restriccion.

      Las funciones que se utilizan son:

 Nodo_Busca: Busca un elemento en la lista de Nodos.
 Agregar_Arco_Sucesor: Agrega un elemento a la lista de
      arcos sucesores de un Nodo.
 Agr_Arco_Antecesor: Agrega un elemento a la lista de
    arcos antecesores de un Nodo.                */

{
 struct Nodo *Nodo_Agrega ( unsh,unsh * );
 struct Nodo *Nodo_Busca ( unsh,int * );

 unsh Agregar_Arco_Sucesor ( struct Arco_Sucesor **,unsh,unsh  );
 unsh Agr_Arco_Antecesor ( struct Arco_Antecesor **,unsh,unsh  );
```

```
 struct Nodo *Dir_Suc_Res,*Dir_n2i,*b;
 struct Arco_Antecesor *apt_ant;
 struct Arco_Sucesor *temporal;
 struct Res_Nodo_Plato *aptres;
 unsh i=1,y,z;
 int Resultado;

 aptres=Pila_Res;
 while ( aptres ) {
  Dir_n2i=Nodo_Agrega ( Suma+i,Num_Nodos );
  aptres->Dir_Cuate=Dir_n2i;
  b=Nodo_Busca ( aptres->Nodo_Res,&Resultado );
  if ( b==NULL ) {
   temporal= ( Grafica ) ->Arco_Sucesor;
   Dir_n2i->Arco_Sucesor=temporal;
   Grafica->Arco_Sucesor=NULL;
   y=Agregar_Arco_Sucesor(&(Grafica->Arco_Sucesor),Suma+i,aptres->Res_Max);
  }else{
   temporal= ( b->Siguiente ) ->Arco_Sucesor;
   Dir_n2i->Arco_Sucesor=temporal;
   ( b->Siguiente ) ->Arco_Sucesor=NULL;
   y=Agregar_Arco_Sucesor(&((b->Siguiente)->Arco_Sucesor),Suma+i,aptres->Res_Max);
  }
  if( y==0 ) return( 0 );
  while ( temporal ) {
   Dir_Suc_Res=Nodo_Busca ( temporal->Nodo_Final,&Resultado );
   if ( Dir_Suc_Res==NULL )  apt_ant= ( Grafica ) ->Arco_Antecesor;
   else apt_ant= ( Dir_Suc_Res->Siguiente ) ->Arco_Antecesor;
   while((apt_ant)->Nodo_Inicial!=aptres->Nodo_Res)apt_ant=apt_ant->Siguiente;
    ( apt_ant ) ->Nodo_Inicial=Suma+i;
   temporal=temporal->Siguiente;
  }
  z=Agr_Arco_Antecesor(&(Dir_n2i->Arco_Antecesor),aptres->Nodo_Res,0);
  if( z==0 ) return( 0 );
  aptres=aptres->Siguiente;
  i++;
 }
 return(1);
}

// ------------------------------------------------------------------
unsh Fuentes_Destinos ( struct Fuente_Plato *Pila_f,struct Destino_Plato *Pila_d,
struct Res_Nodo_Plato *Pila_Res,struct Nodo *Dir_s,
struct Nodo *Dir_t )

/*    Aqui se conectan los fuentes con el s y los destinos con el t

      Opera de la siguiente menera:

   a.- Se agregan arcos auxiliares de el nodo auxiliar s a los nodos
       fuentes.
```

```
   b.- Se agregan arcos auxiliares de los nodos destino a el nodo
       auxiliar t.

       Las variables principales que se utilizan son:

 Pila_f: Direccion al primer elemento de la pila de nodos fuente.
 Pila_d: Direccion al primer elemento de la pila de nodos destino.
 Pila_Res: Direccion al primer elemento de la pila de nodos
  restringidos.
 Dir_s: Direccion del nodo fuente.
 Dir_t: Direccion del nodo destino.

       Las funciones que se utilizan son:

 Agregar_Arco_Sucesor: Agrega un elemento a la lista de
       arcos sucesores de un Nodo.
 Agr_Arco_Antecesor: Agrega un elemento a la lista de
    arcos antecesores de un Nodo.          */

{
 unsh Agregar_Arco_Sucesor ( struct Arco_Sucesor **,unsh,unsh  );
 unsh Agr_Arco_Antecesor ( struct Arco_Antecesor **,unsh,unsh  );

 struct Nodo *apuntador;
 struct Fuente_Plato *temp;
 struct Destino_Plato *TEMP;
 struct Res_Nodo_Plato *aptres;

 unsh y,z;

 temp=Pila_f;
 while ( temp ) {
  y=Agregar_Arco_Sucesor(&(Dir_s->Arco_Sucesor),(temp->Dir_f)->Num_Nodo,INFINITO);
  if ( y==0 )  return ( 0 );
  z=Agr_Arco_Antecesor(&((temp->Dir_f)->Arco_Antecesor),Dir_s->Num_Nodo,0);
  if ( z==0 )  return ( 0 );
  temp=temp->Siguiente;
 }
 TEMP=Pila_d;
 while ( TEMP ) {
  aptres=Pila_Res;
  while(aptres && (TEMP->Dir_d)->Num_Nodo!=aptres->Nodo_Res) aptres=aptres->Siguiente;
  if ( aptres )  apuntador=aptres->Dir_Cuate;
  else apuntador=TEMP->Dir_d;
  z=Agr_Arco_Antecesor (&(Dir_t->Arco_Antecesor),apuntador->Num_Nodo,0);
  if ( z==0 )  return ( 0 );
  y=Agregar_Arco_Sucesor(&(apuntador->Arco_Sucesor),Dir_t->Num_Nodo,INFINITO);
  if ( y==0 )  return ( 0 );
  TEMP=TEMP->Siguiente;
 }
 return ( 1 );
```

```
}

// ------------------------------------------------------------------
unsh Primera_Solucion ( unsh  Suma_Res,struct Plato_Arco *Pila_Arco,
struct Res_Nodo_Plato *Pila_Res,struct Nodo *Dir_s,struct Nodo *Dir_t,
struct Nodo *Ant_sp,struct Nodo *Ant_tp, unsh *Num_Nodos)

/*
     Opera de la siguiente manera:

   a.- Se agrega un arco de t a s.
   b.- Se aplica el algoritmo entre s-prima y t-prima.
   c.- Si el flujo total encontrado en el paso inmediato anterior no
      es igual a Suma_Res no   hay solucion,  en caso contrario se quitan
      los arcos auxiliares agregados en Agarres.

      Las variables principales que se utilizan son:

 Suma_Res: Suma de capacidades minimas de los arcos con
  restriccion.
 Pila_Arco: Direccion al primer elemento de la pila de
   arcos restringidos.
 Pila_Res: Direccion al primer elemento de la pila de nodos
  restringidos.
 Dir_s: Direccion del nodo fuente.
 Dir_t: Direccion del nodo destino.
 Ant_sp: Direccion al anterior del nodo fuente auxiliar.
 Ant_tp: Direccion al anterior del nodo destino auxiliar.

      Las funciones que se utilizan son:

 Flujo_Maximo: Aplica el algoritmo Ford_Fulkerson.
 Agregar_Arco_Sucesor: Agrega un elemento a la lista de
      arcos sucesores de un Nodo.
 Agr_Arco_Antecesor: Agrega un elemento a la lista de
    arcos antecesores de un Nodo.
 B_L_Ant: Borra la lista de antecesores de un Nodo.
 B_L_Suc: Borra la lista de sucesores de un Nodo.
 Ar_Ant_Borra: Borra un elemento en la lista de arcos
      antecesores.
 Ar_Suc_Borra: Borra un elemento en la lista de arcos
      sucesores.                                        */

{
 unsh Flujo_Maximo ( unsh,unsh  ) ;
 unsh Agregar_Arco_Sucesor ( struct Arco_Sucesor **,unsh,unsh  );
 unsh Agr_Arco_Antecesor ( struct Arco_Antecesor **,unsh,unsh  );
 void B_L_Ant ( struct Arco_Antecesor ** );
 void B_L_Suc ( struct Arco_Sucesor ** );
 void Ar_Ant_Borra ( struct Arco_Antecesor **,unsh );
 void Ar_Suc_Borra ( struct Arco_Sucesor **,unsh );
```

```
struct Nodo *Nodo_Agrega ( unsh,unsh * );

struct Nodo *apuntador,*ante,*Dir_nst,*Dir_nts;
struct Arco_Antecesor *apt_temporal;
struct Arco_Sucesor *apt_mientras;
struct Plato_Arco *apt_arcos;
struct Res_Nodo_Plato *aptres;

unsh si_o_no,z,y;
unsh  Flujo=0;

apuntador=Grafica;
while( apuntador ){
   ante=apuntador;
   apuntador = apuntador -> Siguiente;
}
Dir_nst=Nodo_Agrega ( ante->Num_Nodo + 1, Num_Nodos );
Dir_nts=Nodo_Agrega ( ante->Num_Nodo + 2, Num_Nodos );

     // creando el arco  ( s,nst )
z=Agr_Arco_Antecesor ( & ( Dir_nst->Arco_Antecesor ) ,Dir_s->Num_Nodo,0 );
if ( z==0 )  return ( 2 );
y=Agregar_Arco_Sucesor ( & ( Dir_s->Arco_Sucesor ) ,Dir_nst->Num_Nodo,INFINITO );
if ( y==0 )  return ( 2 );

     // creando el arco  ( nst,t )
z=Agr_Arco_Antecesor ( & ( Dir_t->Arco_Antecesor ) ,Dir_nst->Num_Nodo,0 );
if ( z==0 )  return ( 2 );
y=Agregar_Arco_Sucesor(&(Dir_nst->Arco_Sucesor),Dir_t->Num_Nodo,INFINITO);
if ( y==0 )  return ( 2 );

     // creando el arco  ( t,nts )
z=Agr_Arco_Antecesor ( & ( Dir_nts->Arco_Antecesor ) ,Dir_t->Num_Nodo,0 );
if ( z==0 )  return ( 2 );
y=Agregar_Arco_Sucesor(&(Dir_t->Arco_Sucesor),Dir_nts->Num_Nodo,INFINITO);
if ( y==0 )  return ( 2 );

     // creando el arco  ( nts,s )
z=Agr_Arco_Antecesor ( & ( Dir_s->Arco_Antecesor ) ,Dir_nts->Num_Nodo,0 );
if ( z==0 )  return ( 2 );
y=Agregar_Arco_Sucesor(&(Dir_nts->Arco_Sucesor),Dir_s->Num_Nodo,INFINITO);
if ( y==0 )  return ( 2 );

     // FORD-FULKERSON
si_o_no=Flujo_Maximo((Ant_sp->Siguiente)->Num_Nodo,(Ant_tp->Siguiente)->Num_Nodo);
if ( si_o_no==0 )  return ( -1 );
if ( si_o_no==10) return(10);
apt_temporal= ( Ant_tp->Siguiente ) ->Arco_Antecesor;
while ( apt_temporal ) {
   Flujo=Flujo+apt_temporal->Flujo;
   apt_temporal=apt_temporal->Siguiente;
```

```
 }
 if ( Flujo!=Suma_Res )  return ( 0 );
 else{
  apt_arcos=Pila_Arco;
  while ( apt_arcos ) {
   aptres=Pila_Res;
   while(aptres && (apt_arcos->Dir_Nodo_Inicial)->Num_Nodo!=aptres->Nodo_Res)
     aptres=aptres->Siguiente;
   if(aptres==NULL||(apt_arcos->Dir_Nodo_Final)->Num_Nodo==(aptres->Dir_Cuate)->Num_Nodo)
apuntador=apt_arcos->Dir_Nodo_Inicial;
   else apuntador=aptres->Dir_Cuate;
   apt_mientras=apuntador->Arco_Sucesor;
   while(apt_mientras->Nodo_Final!=(Ant_tp->Siguiente)->Num_Nodo)
 apt_mientras=apt_mientras->Siguiente;
   if(apt_arcos->Cap_Min<apt_mientras->Cap_Max)
       apt_mientras->Cap_Max=apt_mientras->Cap_Max-apt_arcos->Cap_Min;
   else Ar_Suc_Borra(&(apuntador->Arco_Sucesor),(Ant_tp->Siguiente)->Num_Nodo);
   apt_temporal= ( apt_arcos->Dir_Nodo_Final ) ->Arco_Antecesor;
   while((apt_temporal)&&(apt_temporal->Nodo_Inicial!=(Ant_sp->Siguiente)->Num_Nodo))
 apt_temporal=apt_temporal->Siguiente;
   if(apt_temporal) Ar_Ant_Borra(&((apt_arcos->Dir_Nodo_Final)->Arco_Antecesor),
   ( Ant_sp->Siguiente ) ->Num_Nodo );
   apt_mientras=apuntador->Arco_Sucesor;
   while(apt_mientras->Nodo_Final!=(apt_arcos->Dir_Nodo_Final)->Num_Nodo)
 apt_mientras=apt_mientras->Siguiente;
   apt_mientras->Cap_Max=apt_mientras->Cap_Max + apt_arcos->Cap_Min;
   apt_mientras->Flujo=apt_mientras->Flujo + apt_arcos->Cap_Min;

   apt_temporal= ( apt_arcos->Dir_Nodo_Final ) ->Arco_Antecesor;
   while(apt_temporal->Nodo_Inicial!=apuntador->Num_Nodo)
 apt_temporal=apt_temporal->Siguiente;
   apt_temporal->Flujo=apt_temporal->Flujo + apt_arcos->Cap_Min;
   apt_temporal->Cap_Min=apt_arcos->Cap_Min;
   apt_arcos=apt_arcos->Siguiente;
  }
 }
      // borrando los sucesores de s'
 B_L_Suc ( & (  ( Ant_sp->Siguiente ) ->Arco_Sucesor )  );

      // borrando los Antecesores de t'
 B_L_Ant ( & (  ( Ant_tp->Siguiente ) ->Arco_Antecesor )  );

      //borrando arcos con extremo ficticio adyacentes a s
 Ar_Suc_Borra ( & ( Dir_s->Arco_Sucesor ) , Dir_nst->Num_Nodo );
 Ar_Ant_Borra ( & (  Dir_s->Arco_Antecesor ) , Dir_nts->Num_Nodo );

      //borrando arcos con extremo ficticio adyacentes a t
 Ar_Suc_Borra ( & ( Dir_t->Arco_Sucesor ) , Dir_nts->Num_Nodo );
 Ar_Ant_Borra ( & (  Dir_t->Arco_Antecesor ) , Dir_nst->Num_Nodo );

      // borrando los sucesores de nst y nts
```

```
 B_L_Suc ( & (  Dir_nst->Arco_Sucesor )  );
 B_L_Suc ( & (  Dir_nts->Arco_Sucesor )  );

      // borrando los Antecesores de nst y nts
 B_L_Ant ( & (  Dir_nst->Arco_Antecesor )  );
 B_L_Ant ( & (  Dir_nts->Arco_Antecesor )  );

      // borrando los nodos s', t' y los ficticios
 if( Dir_nst )  free ( Dir_nst );
 if( Dir_nts )  free ( Dir_nts );
 if( Ant_tp->Siguiente  )  free ( Ant_tp->Siguiente );
 if ( Ant_sp->Siguiente )  free ( Ant_sp->Siguiente );
 Ant_sp->Siguiente=NULL;
 return ( 1 );
}

// ------------------------------------------------------------------
void Libera_Conecciones ( struct Res_Nodo_Plato *Pila_Res,unsh Var,
   struct Nodo *Ant_s,struct Nodo *Dir_t,struct Fuente_Plato *Pila_f,
   struct Destino_Plato *Pila_d )

/*    Esta funcion libera las conecciones de los nodos con restriccion
     y de los varios fuentes y varios destinos.

      Las variables principales que se utilizan son:

 Pila_Res: Direccion al primer elemento de la pila de nodos
  restringidos.
 Var: Indica si hay varias fuentes y varios destinos.
 Ant_s: Direccion del nodo anterior al fuente.
 Dir_t: Direccion del nodo destino.
 Pila_f: Direccion al primer elemento de la pila de nodos
fuente.
 Pila_d: Direccion al primer elemento de la pila de nodos
destino.

      Las funciones que se utilizan son:

 Nodo_Busca: Busca un elemento en la lista de Nodos.
 B_L_Ant: Borra la lista de antecesores de un Nodo.
 B_L_Suc: Borra la lista de sucesores de un Nodo.                */

{
 struct Nodo *Nodo_Busca ( unsh,int * );
 void B_L_Ant ( struct Arco_Antecesor ** );
 void B_L_Suc ( struct Arco_Sucesor ** );

 struct Res_Nodo_Plato *aptres;
 struct Arco_Sucesor *temporal;
 struct Arco_Antecesor *apt_ant;
 struct Nodo *b,*Dir_Suc_Res,*apt,*aptc;
```

```
struct Fuente_Plato *temp;
struct Destino_Plato *TEMP;

int Resultado;

aptres=Pila_Res;
while ( aptres ) {
 if((aptres->Dir_Cuate)->Arco_Antecesor)delete(aptres->Dir_Cuate)->Arco_Antecesor;
 b=Nodo_Busca ( aptres->Nodo_Res,&Resultado );
 if ( b==NULL ) {
  temporal= ( aptres->Dir_Cuate ) ->Arco_Sucesor;
  if (  ( Grafica ) ->Arco_Sucesor )  delete ( Grafica ) ->Arco_Sucesor;
  ( Grafica ) ->Arco_Sucesor=temporal;
  ( aptres->Dir_Cuate ) ->Arco_Sucesor=NULL;
 }else{
  temporal= ( aptres->Dir_Cuate ) ->Arco_Sucesor;
  if((b->Siguiente)->Arco_Sucesor) delete (b->Siguiente)->Arco_Sucesor;
  ( b->Siguiente ) ->Arco_Sucesor=temporal;
  ( aptres->Dir_Cuate ) ->Arco_Sucesor=NULL;
 }
 while ( temporal ) {
  Dir_Suc_Res=Nodo_Busca ( temporal->Nodo_Final,&Resultado );
  if ( Dir_Suc_Res==NULL )  apt_ant= ( Grafica ) ->Arco_Antecesor;
  else apt_ant= ( Dir_Suc_Res->Siguiente ) ->Arco_Antecesor;
  while(apt_ant->Nodo_Inicial!=(aptres->Dir_Cuate)->Num_Nodo)
  apt_ant=apt_ant->Siguiente;
  ( apt_ant ) ->Nodo_Inicial=aptres->Nodo_Res;
  temporal=temporal->Siguiente;
 }
 aptres=aptres->Siguiente;
}

if ( Pila_Res ) {
 apt=Nodo_Busca ( ( Pila_Res->Dir_Cuate ) ->Num_Nodo,&Resultado );
 aptc=apt->Siguiente;
 apt->Siguiente=NULL;
 while ( aptc->Siguiente ) {
  apt=aptc->Siguiente;
  aptc->Siguiente= ( aptc->Siguiente ) ->Siguiente;
  free ( apt );
 }
 free ( aptc );
}

if ( Var ) {
    //borrando los Sucesores de s  ( los fuentes )
  B_L_Suc ( & (  ( Ant_s->Siguiente ) ->Arco_Sucesor )  );

    //borrando los antecesores de t  ( los destino )
  B_L_Ant ( & ( Dir_t->Arco_Antecesor )  );
```

```
   temp=Pila_f;
   while ( temp ) {
      B_L_Ant ( & (  ( temp->Dir_f ) ->Arco_Antecesor )  );
      temp=temp->Siguiente;
   }

   TEMP=Pila_d;
   while ( TEMP ) {
      B_L_Suc ( & (  ( TEMP->Dir_d ) ->Arco_Sucesor )  );
      TEMP=TEMP->Siguiente;
   }

   apt=Ant_s->Siguiente;
   free ( apt->Siguiente );
   free ( apt );
   Ant_s->Siguiente=NULL;
 }
}
```

Bibliografía

[1] Hernández Ayuso, María del Carmen. *Análisis de Redes.* Publicaciones del Departamento de Matemáticas de la Facultad de Ciencias, UNAM.

[2] Hernández Ayuso, María del Carmen. *Seminario de Análisis Combinatorio.* Publicaciones del Departamento de Matemáticas de la Facultad de Ciencias, UNAM.

[3] Kennington, J. L. & Helgason, R. V. *Algorithms for Network Programming.* Edit John Wiley & Sons. 1980.

[4] Rodríguez Alcántar, Edelmira. *Estructras de datos para flujo en redes.* Proyecto PAREIMM. Publicaciones internas del Departamento de Matemáticas, Universidad de Sonora.