

RESEÑA DEL PROBLEMA

| | | |
|-----------------------------|---|-------------------------------------|
| Problema | : | Latin |
| Complejidad | : | Media |
| Tópico | : | Flujo en redes |
| Conocimientos | : | Algoritmos de flujo máximo en redes |
| Tamaño del Código | : | Corto |
| Codificación | : | Media |
| Estructuras de datos | : | Media |
| Propuesto por | : | Raidel Marichal, Cuba |

ANTECEDENTES

El problema de *Latin* busca que el estudiante identifique, implemente y ajuste un problema de flujo máximo de redes. Para el problema se tiene un ejemplo típico del caso donde se tienen muchas fuentes y muchos sumideros. Un elemento adicional que puede hacer al problema difícil, es el poco conocimiento que pueden tener los participantes sobre algoritmos en redes de flujo, por lo cual se busca incentivarlos en su estudio.

APROXIMACIONES

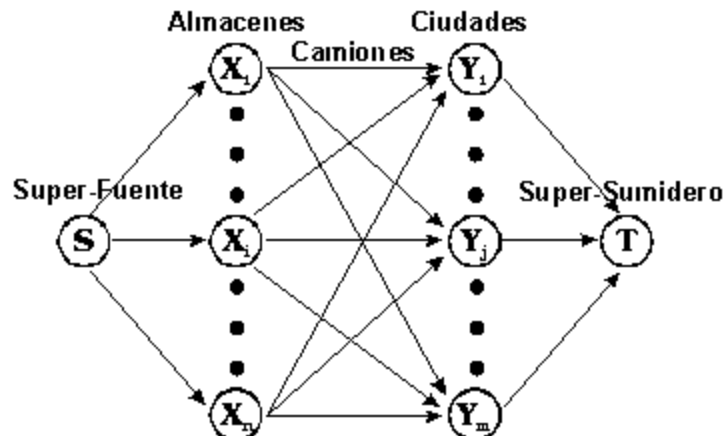
Es posible pensar un algoritmo recursivo donde para cada vértice, comenzando desde la fuente hacia el sumidero (avanzando por niveles), se analizan todas las posibilidades de distribuir el flujo que le llega entre sus vértices adyacentes (con los que tiene un arco), si no es posible que este flujo llegue al sumidero se reduce el flujo y se comienza de nuevo el proceso con todos los vértices. Claramente se ve su crecimiento altamente exponencial lo que lo hace incalculable para redes medianamente complejas y con valores de flujo grandes.

SOLUCION

La solución propuesta se basa en la construcción de una red de flujo a partir de la información de entrada y luego aplicar un algoritmo para flujo máximo en redes. Es claro el uso de flujo de redes ya que tenemos elementos (InfoLATIN) que se desplazan desde una o varias fuentes (Almacenes) y llegan a uno o varios puntos finales (Ciudades) a través de conexiones entre Almacenes y Ciudades (Camiones). Aquí la cantidad de InfoLATIN que fluye entre dos puntos será el flujo de la red y las limitaciones de flujo serán las capacidades máximas de esos conductos. Para el caso del Flujo desde los almacenes está limitado por el número de unidades de InfoLATIN que tiene dicho almacén, para el flujo entre almacenes y Ciudades la limitación estará dada por la capacidad del camión y por último para el flujo que llega a una ciudad está limitado por las unidades de InfoLATIN solicitadas por dicha ciudad. Lo que necesitamos indagar es la cantidad máxima de InfoLATIN que puede llegar a el sumidero desde la fuente cumpliendo las restricciones antes mencionadas (capacidad) y esto no es más que un flujo máximo en la red.

La red es construida de la siguiente forma (ver gráfica): Cada nodo X_i corresponde a un almacén y cada nodo Y_j corresponde a una ciudad con $i=1..n$ y $j=1..m$. Los vértices adicionales s y t son respectivamente la super-fuente y el super-sumidero y con ellos solucionamos el problema de tener múltiples fuentes y sumideros. Las capacidades de paso de cada arco o conducto son asignados de la siguiente forma: Cada arco desde s a X_i corresponde al número de unidades del producto en cada almacén, cada arco desde Y_j a t corresponde al número de unidades del producto solicitado por cada ciudad, por último, cada arco que sale de X_i hacia Y_j corresponde a

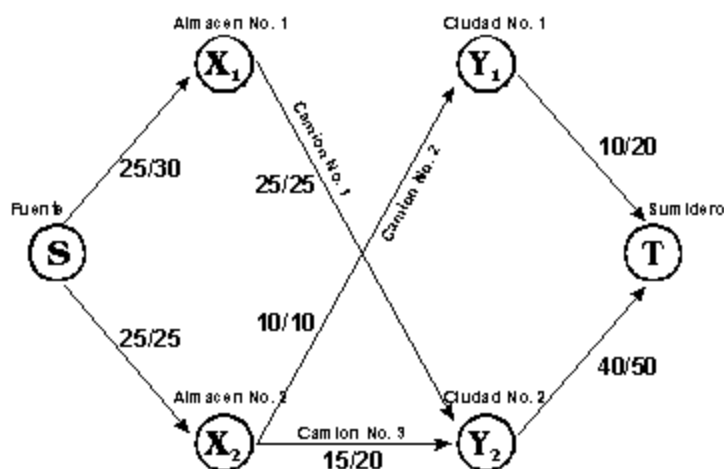
la capacidad de carga de cada camión que hay en cada almacén, si en un almacén no hay camión de alguna ciudad, su arco tendrá valor de 0.



El vértice s es representado por el vértice 0, $X_1..X_n$ son los vértices $1..n$, $Y_1..Y_m$ son los vértices $n+1..n+m$ y el vértice t corresponde al $n+m+1$.

Note que la distribución de la red siempre será la misma pues siempre habrán cuatro (4) niveles de vértices: Fuente(S), Almacenes(X_i), Ciudades(Y_j) y Sumidero(T).

Para la entrada ejemplo, la red y su flujo máximo lucirán de la siguiente forma, donde el primer valor de cada arco corresponde a su flujo y el segundo corresponde a su capacidad:



Después de construir la red de flujo aplicamos un algoritmo de flujo máximo en redes. Para la solución propuesta se ha elegido el LIFT-TO-FRONT ya que es el que mejor desempeño en cuanto tiempo presenta (menos de 1 segundo para todos los casos de prueba) debida a que tiene una complejidad de sólo $O(V^3)$, frente al $O(E^2 \cdot V)$ del FORD-FULKERSON y el $O(V^2 \cdot E)$ del PREFLOW-PUSH. A continuación se muestra una tabla comparativa de la iteraciones realizadas por cada algoritmo para el caso de prueba extremo para $|V|=202$ y $|E|=10200$, donde V corresponde al conjunto de vértices de la red y E es el conjunto de pares (u,v) correspondientes a los arcos o conductos de la red:

| ALGORITMO | COMPLEJIDAD | ITERACIONES |
|-----------------------|------------------|--------------------------------------|
| <i>FORD-FULKERSON</i> | $O(E^2 \cdot V)$ | $10200^2 \cdot 202 = 21.016.080.000$ |
| <i>PREFLOW-PUSH</i> | $O(V^2 \cdot E)$ | $202^2 \cdot 10200 = 416.200.800$ |
| <i>LIFT-TO-FRONT</i> | $O(V^3)$ | $202^3 = 8.242.408$ |

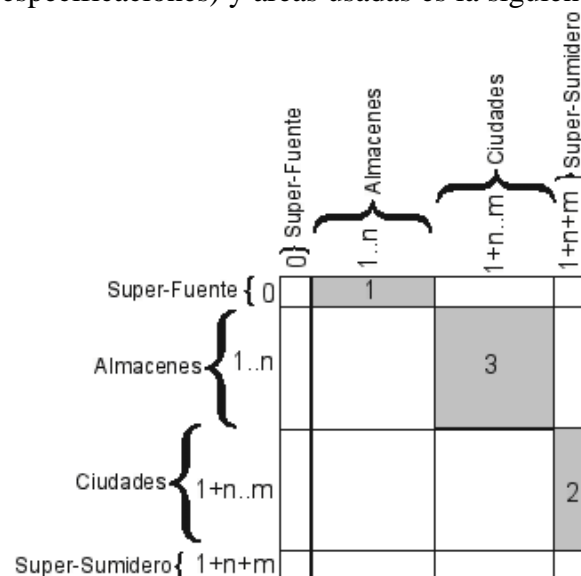
El algoritmo LIFT-TO-FRONT se basa en el hecho de asignar un nivel o altura $h[u]$ a cada vértice de modo que sólo puede existir flujo entre un vértice de mayor nivel hacia un vértice de menor nivel. Además se presume que la fuente entrega a sus vértices adyacentes (con los que tiene arco) todo el flujo que permite la capacidad de sus arcos de tal forma que el flujo que no ha sido inyectado desde el vértice u (exceso de flujo) se almacena en $e[u]$. El proceso de inyectar todo el exceso de flujo del nodo u hacia un nuevo vértice o de regreso hacia la fuente se llama *descargar(u)*; para el cual se puede tomar la decisión de inyectar nuevo flujo hacia otro vértice *inyectar(u)*, o incrementar su nivel para que pueda haber flujo con otros vértices *ascender(u)*. Los vértices son visitados en un orden particular, a diferencia del PREFLOW-PUSH, cuando se encuentra que un vértice ha subido de nivel después de un *descargar(u)*, se coloca al inicio de una lista y con esto se descargan de nuevo todos los demás vértices (de aquí el nombre del algoritmo).

CODIGO

La solución expuesta esta compuesta de varias funciones, la mayoría de las cuales pertenecen al algoritmo de flujo máximo de redes. A continuación se describen todas ellas.

leer()

Esta función inicializa la estructura y luego lee la información del archivo de entrada y construye la red correspondiente en *red[i][j].c*, indicando la capacidad de cada uno de los correspondientes arcos. Para conformar la red se deben incluir todos los vértices posibles por lo que es necesario utilizar una codificación para cada vértice así: el vértice s (super-fuente) es representado por el vértice 0 , $X_1..X_n$ (Almacenes) son los vértices $1..n$, $Y_1..Y_n$ (Ciudades) son los vértices $n+1..n+m$ y el vértice t (super-sumidero) corresponde al $n+m+1$. La distribución de *red[i][j]*, orden de lectura (de acuerdo a las especificaciones) y áreas usadas es la siguiente:



inicializar()

Esta función realiza varias funciones: Inicializa el vector de alturas $h[u]$ con nivel 0 excepto para $h[s]=|V|$, Inicializa el vector de excesos de flujo a 0, Inicializa el flujo entre s y todos su vértices adyacentes al máximo, almacenando el exceso de flujo correspondiente en $e[u]$, por último crea una lista de vértices adyacentes $N[u][k]$ para cada vértice u , además su posición actual en el recorrido de esta lista en $pN[u]$.

min(a, b)

Esta función devuelve como resultado el menor valor entre a y b .

flujo()

Función que ejecuta el algoritmo LIFT-TO-FRONT para encontrar el flujo máximo en la red. En primera instancia inicializa las estructuras necesarias llamando a *inicializar()*, luego crea una lista de vértices L con $u \in V - \{s, t\}$. Después de esto realiza una *descarga(u)* para cada uno de los nodos pertenecientes a L , teniendo en cuenta que si un vértice ha subido de nivel se coloca al principio de la lista y se vuelven a revisar de nuevo todos los demás vértices.

descargar(u)

Esta función se encarga de eliminar el exceso de flujo $e[u]$ del vértice con sobreflujo u , esto lo logra inyectando nuevo flujo a vértices donde exista una capacidad residual $c_f > 0$ o, cuando es imposible inyectar más flujo, se asciende de nivel en busca de nuevos vértices donde pueda existir flujo pues sólo se da entre vértices con una diferencia de nivel de una unidad.

inyectar(u, v)

Esta función se encarga de inyectar $df = \min(e[u], c_f)$ flujo entre el vértice u y el v teniendo en cuenta que deben estar a una diferencia de altura de una unidad ($h[u] = h[v] + 1$). También son actualizados los excesos de flujo de los vértices u y v .

ascender(u)

Esta función se encarga de incrementar el nivel del vértice u en una unidad con respecto al nivel más bajo de sus vértices adyacentes pertenecientes a la red residual E_f cuando para todos sus vértices adyacentes v se tiene que $h[u] \leq h[v]$.

imprimir()

Esta función escribe en el archivo de salida la información solicitada en el problema extrayendola del flujo máximo en $red[i][j].f$.

MEJORAS AL ALGORITMO

Como analizamos anteriormente este algoritmo tiene el mejor desempeño $O(V^3)$ frente a otros algoritmos que realizan la misma función (ver solución), pero sus requerimientos de memoria exigen crear una estructura que es parcialmente utilizada, por esto podría pensarse en realizar una redistribución de la información en $red[i][j]$ en busca de utilizarla óptimamente (usando listas enlazadas o codificación de las coordenadas) pero complicaría su implementación y el acceso a los datos no se daría en tiempo constante. Dados los límites máximos del problema no existen problemas con la memoria usada así que no es recomendable hacer esta optimización por la naturaleza de la competencia.

BIBLIOGRAFIA

- CORMEN Thomas, Leiserson Charles and Rivest Ronald. *Introduction to Algorithms*. The MIT Press, 1990.

Raúl Andrés Duque M.
 rduque@venus.uanarino.edu.co