Benemérita Universidad Autónoma de Puebla

Teoría de grafos

Para el club de ACM de la BUAP

Conceptos

Los grafos son una manera de representar relaciones binarias, y sirven como modelo matemático para representar el mundo real.

Definición: Grafo Dirigido (Digrafo). Un grafo dirigido es un conjunto V y en relación E asociada a el, esto es: G = (V, E) donde:

$$E \subset V \times V$$

Los elementos de V se llaman vértices , y los de E se llaman arcos. Los grafos son por definición dirigidos, ya que el orden en que aparecen los elementos de E es relevante. Cuando se escribe (a, b) se está denotando el arco que va de a hastá b.

Definición: Grafo No dirigido (Grafo) Un grafo es no dirigido si y solo si E es simétrica, esto es:

$$(a, b) \in E \Longrightarrow (b, a) \in E \qquad \forall a, b \in E$$

Definicón: Un grafo G=(V,E) es un multigrafo si existen $a, b \in V, a \neq b$, con dos o mas aristas de la forma (a, b) (para grafos dirigidos), o $\{a, b\}$ (para grafos no dirigidos)

Definición: Sea G=(V,E) un grafo o multigrafo dirigido. Para cualquier $v \in V$.
a) El grado de entrada de v es el número de aristas de G que llegan a v y se denota con Γ^+
b) El grado de salida de v es el número de aristas de G que parten de v y se denota con Γ^-

RECORRIDO EN PROFUNDIDAD: GRAFOS NO DIRIGIDOS

Sea $G = (V, E)$ un grafo no dirigido formado por todos aquellos nodos que deseamos visitar. Supongamos que de alguna manera es posible marcar un nodo para mostrarlo que ya ha sido visitado. Para efectuar un recorrido en profundidad del grafo, se selecciona cualquier nodo v∈V como punto de partida. Se marca este nodo para mostrar que ya ha sido visitado. A continuación, si hay un nodo adyacente a v que no haya sido visitado todavía, se toma este nodo como nuevo punto de partido, y se invoca recursivamente el procedimiento de recorrido en profundidad. Al volver de la llamada recursiva, si hay otro nodo adyacente a v que no haya sido visitado, se toma este nodo como punto de partida siguiente, se vuelve a llamar recursivamente al procedimiento, y así sucesivamente. Cuando están marcados todos los nodos adyacentes a v, el recorrido que comenzara en v ha finalizado. Si quedan algún nodo de G que no haya sido visitado, tomamos cualquiera de ellos como nuevo punto de partida, y volvemos a invocar el procedimiento. Se sigue así hasta que esten marcados todos los nodos de G. Véase el procedimiento

Procedimiento recorridop(G)
— Para cada $v \in V$ hacer marca[v] ← no visitado
— Para cada $v \in V$ hacer
— — Si marca[v] visitado entonces rp(v)

Procedimiento rp(v)
— (el nodo v no ha sido visitadoanteriormente)
— marca[v] ←visitado
— para cada nodo w adyacente a v hacer
— — si marca[w] visitado entonces rp(w)

El algoritmo se llama de recorrido en profundidad porque inicia tantas llamadas recursivas como sea posible antes de volver de una llamada. La recursión solo se detiene cuando la exploración del grafo se ve bloqueada y no puede proseguir. En ese momento, la recursión "retrocede" para que sea posible estudiar posibilidades alternativas en niveles más elevado. El algoritmo requiere un tiempo que esta en $\theta(n)$ para las llamadas al procedimiento, y un tiempo que esta en $\theta(a)$ para inspeccionar las marcas. Por tanto el tiempo de ejecución esta en $\theta(\max(a, n))$.

El recorrido en profundidad de un grafo conexo asocia al grafo un árbol de recubrimiento. Si el grafo que esta explorando no es conexo, entonces un recorrido en profundidad le asocia no solo a un único árbol, sino a todo un bosque de árboles, uno para cada componente conexo del grafo.

RECORRIDO EN PROFUNDIDAD: GRAFOS DIRIGIDOS

El algoritmo es esencialmente el mismo que para los grafos no dirigidos; la diferencia reside en la interpretación de la palabra "adyacente". En un grafo dirigido el nodo w es adyacente al nodo v, si existe la arista dirigida (v,w). Si existe (v,w) pero (w,v) no existe, entonces w es adyacente a v, pero v no es adyacente a w. El tiempo que requiere esta algoritmo también esta en $\theta(\max(a, n))$, En este caso , sin embargo, las aristas utilizadas para visitar todos los nodos de un grafo dirigido G=(N,A) pueden formar un bosque de varios árboles aunque G sea conexo.

RECORRIDO EN ANCHURA O AMPLITUD

Cuando un recorrido en profundidad llega a un nodo v, intenta a continuación visitar algún vecino de v, después algún vecino del vecino, y así sucesivamente. Cuando un recorrido en anchura llega a algún nodo v, por otra parte, primero visita todos los vecinos de v, Solo examina nodos mas alejados después de haber hecho esto. A diferencia del recorrido en profundidad, el recorrido en anchura no es naturalmente recursivo.

Para el algoritmo de recorrido en anchura , necesitaos un tipo cola que admita las dos operaciones poner y quitar. Este tipo representa una lista de elementos que hay que manejar por el orden "primero en llegar, primero en salir" (FIFO, que indica "first in, fist out"). La función primero denota el elemento que ocupa la primera posición de la cola. Véase a continuación el algoritmo

procedimiento ra(G[V,A], v)
— para cada nodo u∈V-{v}
— — marca[u] = novisitado
— — d[u] ← ∞
— — p[u] ← NIL
— Q ← cola vacia
— marca[v] ← visitado
— poner v en Q
— d[v] ← 0
— p[v] ← NIL
— mientras Q no esta vacia hacer
— — u ← primero(Q)
— — quitar u de Q
— — para cada nodo w adyacente a u hacer
— — — si marca[w] ≠ visitado entonces marca[w] ←visitado
— — — — poner w en Q
— — — — d[w] ← d[u] + 1;
— — — — p[v] ← u

Necesitamos un programa principal que de comienzo al recorrido:

procedimiento recorrido(G)
— para cada v ∈ N hacer marca[v] ← no visitado
— para cad v ∈ N hacer
— — si marca[w]≠visitado entonces ra(v)

Podemos ver que en el arreglo P se encuentra el camino para llegar de un nodo **v** a cualquier nodo **w** , si P[**w**] contiene NIL entonces no se puede llegar de **v** a **w**, y en el arreglo **d** contiene la distancia mínima llegar de **v** a **w**.

El camino mas corto

Si G=(V,A) es un grafo no ponderado podemos definir que la distancia de la ruta mas corta δ(s,v) de s a v como el mínimo numero de enlaces del camino de s a v; si aquí no existe camino de s a v entonces δ(s,v)=∞. Un camino de largo δ(s,v) de s a v se dice que es un camino corto de s a v.

Algunas propiedades de la distancia mas corta.

Lema

Permita ser G =(V,A) ser un grafo dirigido o no dirigido , y permita ser s∈V ser un vertice arbitrario. Entonces, para todo vertice adyacente (u,v)∈A,

δ(s,v)$\leq$ δ(s,u)+1

Lema

Permita ser G=(V,A) ser un grafo dirigido o no dirigido, y supongamos que BFS(Breadth-first search) corre en G de un vertice dado s $\in$ V. Cuando termina, para cada vertice v∈V, el valor d[v] computado por BFS satisface d[v] $\geq$ δ(s,v)

ÁRBOL DE RECUBRIMIENTO MÍNIMO

Sea G = (N,A) un grafo conexo no dirigido en donde N es el conjunto de nodos y A es el conjunto de aristas. Cada arista posee una longitud no negativa. El problema consiste en hallar un subconjunto T de las arista de G tal que utilizando solamente las aristas de T, todos lo nodos deben quedar conectados, y además la suma de las longitudes de las aristas de T debe ser tan pequeña como sea posible.

Sea G'=(N,T) el grafo parcial formado por los nodos de G y las aristas de T, y supongamos que en N hay n nodos. Un grafo conexo con n nodos debe tener al menos n-1 aristas, así que este es el numero mínimo de aristas que puede haber en T, Por otra parte, un grafo con n nodos y mas de n-1 aristas contiene al menos un ciclo. Por tanto, si G' es conexo y T tiene mas de n-1 arista, se puede eliminar al menos un arista sin desconecta G', siempre y cuando seleccionemos una arista que forme parte de un ciclo. Un conjunto T con n o mas aristas no puede ser optimo. Se sigue que T debe tener exactamente n-1 arista, y como G' es conexo, tiene que ser un árbol.

El grafo G' se denomina árbol de recubrimiento mínimo o Árbol abarcador de costo mínimo para el grafo G.

La propiedad de AAM (Árbol Abarcador de costo mínimo)

Hay distintas formas de construir un árbol abarcador de costo mínimo, muchos de esos métodos utilizan la siguientes propiedad, que se denomina propiedad AAM. Sea G=(V,A) un grafo conexo con una función de costo definida en las aristas. Sea U algún subconjunto propio del conjunto de vértices V. Si (u,v) es una arista de costo mínimo tal que u $\in$ U y v $\in$ V-U, existe un árbol abarcador de costo mínimo que incluye (u,v) entre sus aristas.

Algoritmo Prim

Existen dos técnicas populares que explotan la propiedad AAM para construir un árbol abarcador de costo mínimo a partir de un grafo ponderado G=(V,A). El algoritmo de prim comienza cuando se asigna a un conjunto U un valor inicial {1}, en el cual "crece" un árbol abarcador, arista por arista. En cada paso localiza la arista mas corta (u,v) que conecta U y V-U, y después agrega v, el vertice en V-U, a U. Este se repite hasta que U = V.

```
procedure Prim(G:grafo, var T: conjunto de aristas)
— (Prim construye un árbol abarcador de costo mínimo T para G)
— var
— — U: conjunto de vértices
— — u, w: vértices
— begin
— — T:=0
— — U:={1}
— — while U ≠ V do begin
— — — sea (u,v) una arista de costo mínimo tal que u esta en U y v en V-U
— — — T:=T∪{(u,v)}
— — — U:=U∪{v}
— — end
end; (Prim)
```

La complejidad de tiempo del algoritmo de Prim es $O(n^2)$

Kruskal

El conjunto de aristas T esta vació inicialmente. A medida que progresa el algoritmo, se van añadiendo aristas a T. Mientras no haya encontrado una solución, el grafo parcial formado por los nodos de G y las aristas de T consta de varios componentes conexos. Los elementos de T que se incluyen en una componente conexa dada, forman un árbol de recubrimiento mínimo para los nodos de esta componente. Al final del Algoritmo, solo queda una componente conexa. así que T es un árbol de recubrimiento mínimo para todos los nodos de G.

Para construir componentes conexas mas y mas grandes, examinamos las aristas de G por orden creciente de longitudes. Si una arista une a dos nodos de componentes conexas distintas, se lo añadimos a T. Consiguientemente, las dos componentes conexas forman ahora una única componente. En caso contrario, se rechaza la arista: une a dos nodos de la misma componente conexa, y por tanto no se puede añadir a T sin formar un ciclo. El algoritmo se detiene cuando solo queda una componente conexa.

Es preciso efectuar rápidamente dos operaciones: buscar(x), que nos dice en que componente conexa se encuentra el nodo x, y fusionar(A,B), para fusionar dos componentes conexas.

funciona Kruskal (G=(N,A): grafo; longitud: A $\rightarrow R^+$): conjunto de aristas
— (Inicia)
— Ordenar A por longitudes crecientes
— n $\leftarrow$ el numero de nodos que hay en N
— T $\leftarrow \oslash$
— Iniciar n conjuntos, cada uno de los cuales contiene un elemento distinto de N
— (bucle voraz)
— repetir
— — e $\leftarrow$ (u,v) $\leftarrow$ arista mas corta, aun no considerada
— — compu $\leftarrow$ buscar(u)
— — compv $\leftarrow$ buscar(v)
— — si compu $\neq$ compv entonces
— — — fusionar(compu,compv)
— — — T $\leftarrow$ T $\cup$ {e}
— hasta que T contenga n-1 aritas
devolver T

El timpo total para el Algoritmo esta en θ(a log n).

Problemas

1.- Segundo mejor árbol de recubrimiento mínimo

Permitir G = (V,A) ser no dirigido, grafo conectado con una función de peso w:A$\rightarrow$**R**, y supongamos que $|A| \geq |V|$ y todos los adyacentes pesos son distintos

Un segundo mejor árbol de recubrimiento mínimo es definido como sigue, Permita Υ ser el conjunto de todos los árboles de recubrimiento de G, y permita ser T' un minmo árbol de recubrimiento mínimo de T tal que w(T) = $\min_{T'' \in \Upsilon - \{T'\}}\{$T"$\}$.

a. Mostrar que el mínimo árbol de recubrimiento es único, pero que el segundo menor árbol de recubrimiento mínimo no es necesariamente único

b. permita a T ser un árbol de recubrimiento mínimo de G. Probar que aquí existe una arista (u,v)$\in$T y (x,y) no pertenece a T tal que T -{(u,v)}$\cup${(x,y)} es un segundo mejor árbol de recubrimiento mínimo de G.

c. Permita a T se un árbol de recubrimiento de G y, para cualquiera dos vértices y,v $\in$ V, permitir max[u,v] ser una arista de máximo peso en el único camino entre u y v en T. Describe un O(V^2)-time algoritmo que, dado T, compute max[u,v] para todo u,v$\in$V.

d. Dar un eficiente algoritmo que compute el segundo mejor árbol de recubrimiento mínimo G.

CAMINO MAS CORTO Y SU RUTA

Dijkstra

Supóngase un grafo dirigido G=(V,A) en el cual cada arco tiene una etiqueta no negativa, y donde un vertice se especifica como origen. El problema es determinar el costo del camino mas corto del origen a todos los demás vértices de V. donde la longitud de un camino es la suma de los costos de los arcos del camino. Esto se conoce como el nombre de problema de los caminos mas cortos con un solo origen. Obsérvese que se hablara de caminos con "longitud" aun cuando los costos representen algo diferente, como tiempo.

Para resolver esta problema se manejara una técnica "ávida" conocida como "algoritmo de Dijkstra", que opera a partir de un conjunto S de vértices cuya distancia mas corta desde el origen ya es conocida. En principio, S contiene solo el vertice de origen. En cada paso, se agrega algún vertice restante v a S, cuya distancia desde el origen es la mas corta posible. Suponiendo que todos los arcos tienen costo no negativo, siempre es posible encontrar un camino mas corto entre el origen y v que pasa solo a través de los vértices de S, y que se llama especial. En cada paso del algoritmo, se utiliza un arreglo D para registrar la longitud del camino especial mas corto a cada vertice. Una vez que S incluye todos los vértices, todos los caminos son "especiales", así que D contendrá la distancia mas corta del origen a cada vertice.

```
procedure Dijkstra
— (Dijkstra calcula el costo de los caminos mas cortos entre el vertice 1 y todos los demás de un grafo dirigido)
begin
(1)— S:={1};
(2)— for i:=2 to n do
(3)— — D[i]:= C[1,i]; (asigna valir inicial a D)
(4)— for i:=1 to n-1 do begin
(5)— — elige un vertice w en V-S tal que D[w] sea un mínimo
(6)— — agrega w a S
(7)— — for cada vertice v en V-S do
(8)— — — D[v] := min(D[v],D[w]+C[w.v])
— — — end
end; Dijkstra
```

En algoritmo se supone que existe un grafo dirigido G=(V,A) en el que V={1,2,...,n} y el veritice 1 es el origen. C es un arreglo bidimensional de costos, donde C[i,j] es el costo de ir del vertice i al vertice j por el arco i $\rightarrow$ j, se supone que C[i,j] es ∞, un valor mucho mayor que cualquier costo real. EN cada paso, D[i], contiene la longitud del camino especial mas corto actual para el vertice i.

Para reconstruir el camino mas corto del origen a cada vertice, se agrega otro arreglo P de vertices, tal que P[v] contengo el vertice inmediato anterior a v en el camino mas corto. Se asigna P[v] valor inicial 1 para toda v $\neq$ 1. el arreglo P puede actualizarse después de la línea (8) de Dijkstra. Si D[w]+C[w,v]<D[v] en la línea (8) después se hace P[v]:=w. Al termino de Dijkstra, el camino a cada vertice puede encontrarse regresando por los vetices predecesores del arreglo P.

Floyd

Supóngase que se tiene un grafo dirigido atiquetado que da el tiempo de vuelo de ciertas ciudades, y se desea construir una tabla que brinde el menor tiempo requerido para volar entre dos ciudades cualesquiera. Este es un ejemplo de problema de los caminos mas cortos entre todos los pares. Para plantear el problema con precisión, se emplea un grafo dirigido G=(V,A) en el cual cada arco v $\leftarrow$ w tiene un costo no negativo C[v,w]. El problema es encontrar el camino de longitud mas corta entre v y w para cada par ordenado de vértices (v,w).

Podría resolverse este problema por medio del algoritmo de Dijkstra, tomando por turno cada vertice como vertice origen, pero una forma mas directa de solución es mediante el algoritmo creado por R.W. Floyd. Por conveniencia, se supone otra matriz A de n×n en la que se calculan las longitudes de los caminos mas cortos. Inicialmente se hace A[i,j]=C[i,j] para toda i≠j. Si no existe un arco que vaya de i a j, se supone que C[i,j]=∞. Cada elemento de la diagonal se hace 0.

Después se hacen n iteraciones en la matriz A. Al final de la k-esima iteración, A[i,j] tendrá por valor la longitud mas pequeña de cualquier camino que vaya desde el vertice i hasta el vertice j y que no pase por un vertice con numero mayor que k. Esto es, i y j, los vértices extremos del camino, pueden ser cualquier vertice, pero todo vertice intermedio debe ser menor o igual que k.

En la k-esima iteración se aplica la siguiente formula para calcular A.

$$A[i,j] = min \begin{cases} \text{A[i,j]} \\ \text{A[i,k]+A[k,j]} \end{cases}$$

Es evidente que el tiempo de ejecución de este programa es $O(n^3)$, que el programa esta conformado por el triple ciclo anidado **for**.

```
procedure Floy(var A: array[1..n, 1..n] of real; C:array[1..n, 1..n] of real)
— (Floyd calcula la matriz A de caminos mas cortos dada la matriz de costos de arcos C)
— var
— — i,j,k: integer;
— begin
— — for i:= 1 to n do
— — — for j:= 1 to n do
— — — — A[i,j] := C[i,j];
— — for i := 1 to n do
— — — A[i,i] := 0
— — for k:=1 to n do
— — — for i:=1 to n do
— — — — for j:=1 to n do
— — — — — if(A[i,k]+A[k,j])<A[i,j]) then
— — — — — — A[i,j]:=A[i,k]+A[k,j]
end; (Floyd)
```

PROBLEMAS CHIDOS

Flujo de una red

DESCRIPCION DEL PROBLEMA

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

-X representa al conjunto de poblados
-si $i, j \in$X, $(i, j) \in$A $\iff$ es posible transportar el producto del poblado i al poblado j
- $q : A \to \mathbf{R}$, donde para $(i, j) \in$A, $q(i, j) =$ capacidad máxima de transporte del poblado i al poblado j

Se desea entonces determinar la cantidad del producto a transportar del poblado i al poblado j de manera tal que en t se obtenga la máxima cantidad posible de tal producto. Debe observarse que la cantidad total que entre a un poblado i, distinto de s y de t, debe ser igual a la cantidad total que sale de el puesto que no tiene oferta ni demanda: por otro lado, la cantidad total que se obtenga en t debe ser igual a la cantidad total que sale de s puesto que no se genera ni se pierde el producto a través de la red; por ultimo la cantidad a enviar de i a j para $(i, j) \in$A, debe ser menor o igual que $q(i, j)$. En general, en un grafo R=(X,A,q), al numero $q(i, j)$ asociado a cada arco se le llama capacidad del arco (i, j) y a la cantidad a enviar a través de el se le llama flujo a través del arco (i, j). Un flujo factible en R se define de la siguiente manera:

Definición. Un flujo factible en un grafo R=(X,A,q)es una función f: A $\to$ **R** tal que:

(i)

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

(ii)

$$0 \leq f(i,j) \leq q(i,j), \forall (i,j) \in A.$$

Al numero v se le llama valor del flujo f y a las ecuaciones (i) se les conoce como "ecuaciones de conservacion del flujo"; a los vértices s y t (unicos con oferta y demanda respectivamente)se les llama origen y destino respectivamente. Por comodidad de notación se utilizara f_{ij} y q_{ij} para denotar $f(i, j)$ y $q(i, j)$ respectivamente.

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

Para exponer el método de solución para el problema es necesarios los siguientes conceptos:

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

$$a_j \in C^+ \iff a_j = (i_j, i_{j+1})$$
$$a_j \in C^- \iff a_j = (i_{j+1}, i_j)$$

Es decir, los arcos de C^+ son aquellos arcos de C que tienen el sentido de s hacia t; los arcos de C^- son aquellos arcos de C que tienen el sentido inverso. En base a esto considérese la siguiente definición.

Definición. Una cadena de s a t es aumentante si $f_{ij} < q_{ij}$ para todo $(i,j) \in C^+$ y $f_{ij} > 0$ para todo $(i,j) \in C^-$. Recibe el nombre de cadena aumentante ya que a través de ella puede enviarse flujo de s a t construyéndose, de este modo, un flujo factible de mayor valor. Puede verificarse facilmente que se construye un flujo factible f' de mayor valor que el flujo factible f definido en las primeras tres cadenas se produce como sigue:

$$\begin{aligned} f_{ij} &= f_{ij} + z, && \forall (i,j) \in C^+ \\ f_{ij} &= f_{ij} = z, && \forall (i,j) \in C^- \end{aligned}$$

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

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

$$q(C) = \min \left\{ \min_{(i,j) \in C^+} [q_{ij} - f_{ij}], \min_{(i,j) \in C^-} [f_{ij}] \right\}$$

Algoritmo de Ford y Fulkerson

Proposito. Determinar el flujo máximo entre origen y destino en un grafo $R = (X, A, q)$

Descripcion

1.- Iniciar con cualquier flujo factible f.

2.- Etiquetar el origen s con $[s, \infty]$.

3.- Elegir un vertice etiquetado y no examinado; sea j este vertice y sean $[\pm k, f(j)]$ sus etiquetas

(i) A todo $i \in \Gamma^+(j)$ que no este etiquetado y tal que $f_{ji} < q_{ji}$ asignar la etiqueta $[+j, f(i)]$, donde $f(i) = min\{f(j), q_{ji} - f_{ji}\}$.

(ii) A todo $i \in \Gamma^-(j)$ que no este etiquetado y tal que $f_{ij} > 0$ asignar la etiqueta $[-j, f(i)]$, donde $f(i) = min\{f(j), f_{ji}\}$. - Se dice ahora que el vertice j ha sido examinado-

4.- Repetir el paso 3 hasta que suceda (i) o (ii)

(i) El vértice destino t no tiene etiqueta y todos los vértices etiquetados han sido examinados. Terminar ya que el flujo factible f es máximo.

(ii) El vertice t recibe etiqueta. Ir al paso 5

5.- Sea $x = t$

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

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

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

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

7.- Si $z = s$, borrar todas las etiquetas y regresar al paso 2. Si $z \neq s$, hacer $x = z$ y regresar al paso 6.

Problemas.

1.- Que pasaría si en ves de ser un solo pueblo con oferta fueran varios, se podría utilizar el mismo algoritmo?

2.- Que pasaría si en ves de ser un solo pueblo con demanda fueran varios?

3.- Que pasaría si hubiera un limite en los pueblos además de las aristas?

Investigar, y si se puede aplicar el mismo algoritmo y que modificaciones se tienen que hacer.

Componentes Conexas

Entrada: Un grafo dirigido o no dirigido G.

Salida: Las componentes conexas del grafo G.

Discusión
Las componentes conexas de un grafo representan , en grandes términos , las piezas del grafo. Dos vértices están en la misma componente de G si y sólo si existe un camino entre estos.

Ordenacion Topologica

Entrada: Un grafo dirigido aciclico G = (V,E)

Salida: Encontrar una ordenación líneal de los vértices de V tal que para cada arista $(i, j) \in E$, vertice i esta a la izquierda del vertice j.

Discusión
La ordenación topológica ordena los vértices y aristas de un DAG (Directed Acyclic Graph) en una simple y consistente forma.

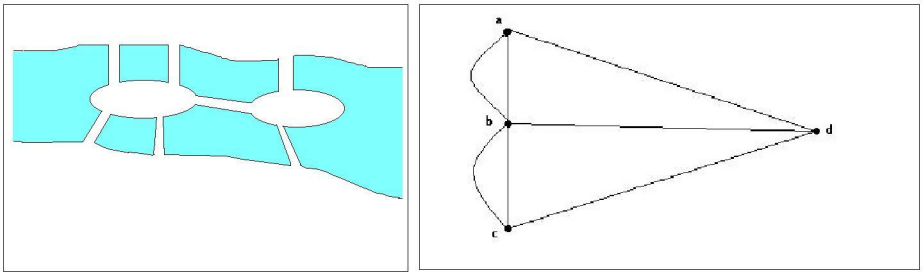
Vértices de articulación

Los vértices de articulación son aquellos que al eliminarlos del grafo este queda desconectado , para saber que vértices son articulaciones , solamente se necesita hacer una búsqueda en profundidad.

Ciclo Euleriano

Los siete puentes de $Konigsberg$. Durante el siglo $XVIII$, la ciudad de $Konigsberg$ (en Prusia Oriental) estaba dividida en cuatro zonas (incluida la isla de Kneiphof)por el rio Preel. Siete puentes comunicaban estas regiones, como se ve en la figura . Se decía que los habitantes hacían paseos dominicales tratando de encontrar una forma de caminar por la ciudad cruzando cada puente exactamente una vez y regresando al punto donde se había iniciado el paseo.

Con el fin de determinar si existía o no dicho circuito, Euler represento las cuatro zonas de la ciudad y los siete puentes con el munltigrafo que se muestra en la figura. Encontró cuatro vértices de grado par $\Gamma(a) = \Gamma(c) = \Gamma(d) = 3$ y $\Gamma(b) = 5$. También encontró que la existencia de tal circuito dependía del numero de vértices de grado impar en el grafo.



Definición Sea G = (V,E) un grafo o multigrafo no dirigido sin vértices aislador, Entonces G tiene un circuito euleriano si existe un circuito en G que recorre cada arista del grafo exactamente una vez. Si existe un recorrido abierto de a a b en G que recorre cada arista de G exactamente una vez, este recorrido se llamara recorrido euleriano

En este momento expondremos un teorema que nos servira para la creación de un algoritmo en tiempo polinomial

Teorema Sea G=(V,E) un grafo o multigrafo no dirigido sin vértices aislado. Entonces G tiene un circuito euleriano si y solo si G es conexo y todo vertice de G tiene grado par

Corolario Si G es un grafo o multigrafo no dirigido sin vértices aislados, entonces podemos construir un recorrido euleriano en G si y solo si G es conexo y tiene exactamente dos vértices de grado impar.

De aquí podemos partir a un algoritmo eficiente, tomemos como idea principal lo siguiente. Si todos sus nodos son de grado par, entonces el ciclo euleriano con el mínimo peso es la suma de todas sus aritas, bien pero que pasa si esto no se cumple, lo único que tenemos que hacer es volver a los nodos impares par, pero no podemos volver par así como así y garantizar que es optimo, tenemos que comparar todas las posibilidades que existen para alcanzar la solución, y de esto podemos escribir el siguiente algoritmo, que utiliza una función recursiva llamada Euler para poder computar todas la posibilidades.

Algoritmo (Para grafos no dirigidos y conexo)
— (leer el grafo si existe dos arista de v a w entonces escoger la de menor costo
— sumar todos los valores de las arista y colocárselas en T)
— con el algoritmo de FLOYD sacar las distancia mas cortas a todos los nodos y guárdarlas en G
— regresar T+ función Euler(G,1)

Función Euler(G=(V,A) con distancias ḿinimas; pos:Integer)
— var
— — min := ∞
— — temp
— for i:= pos to n begin
— — if vertice(i) tiene grado impar entonces
— — — pos := i
— — — salirdelciclo
— if i > n entoces retorna 0
— for i:=pos+1 to n begin
— — if(vertice(i) tiene grado impar entonces)
— — — temp = G[pos][i] + Euler(G,$i+1$)
— — — if(temp < max) max = temp
— retorna max

Para los grafos dirigidos básicamente es el mismo algoritmo puesto que el teorema nos dice:

Teorema Sea G(V,E) un grafo o multigrafo dirigido sin vértices aislados. El grafo G tiene un circuito euleriano dirigido si y solo si G es conexo y $\Gamma^+(v) = \Gamma^-(v)$ para todo $v \in V$.

Bibliografia

[1] G. Brassard y P. Brathey "FUNDAMENTO DE ALGORITMIA" Editoria PRENTICE HALL (1998)

[2] Thomas H. Cormen "Introduction To Algorithms, Second Edition" Editorial McGraw-Hill (2001)

[3] Steven S. Skiena "The Algorithm Design Manual" Editorial Springer (1998)

[4] Ralph P. Grimaldi "Matemáticas Discretas y Combinatoria" Editorial Prentice Hall (1998)

[5] ???????? "Teoría de Redes" Colección matemática

[6] ????? "Estructuras de datos en pascal" ??????