UNIDAD 1

Grafos

Introducción a la teoría de grafos

Aplicaciones

Caminos más cortos

Grafos dirigidos y ponderados:

- Obtener en el camino más corto desde un vértice ‘u’ a ‘todos’ los demás del grafo,
- Obtener los caminos más cortos desde todos los vértices a uno ‘destino’,
- Obtener el camino más corto desde un vértice ‘u’ a un vértice ‘v’
- Obtener los caminos más cortos entre todos los pares de vértices.

Caminos más cortos

Algoritmo de Dijkstra

Formulación: Sea G= (V,E) grafo dirigido y ponderado con costos no negativos en las aristas, dado un vértice s $\in$ V, se desea obtener cada uno de los caminos más cortos de s al resto de vértices v $\in$ V.

Caminos más cortos

Dijkstra resuelve eficientemente el problema.

Para aristas ponderadas con costos negativos en el grafo, la solución puede ser errónea: Bellman-Ford permite costos negativos en las aristas.

Los algoritmos de búsqueda de caminos más cortos explotan la propiedad de que el camino más corto entre dos vértices contiene, a su vez, caminos más cortos entre los vértices que forman el camino.

Caminos más cortos

Dado un grafo $G=(V,E)$ y un vértice s, encuentre el camino más corto desde s a todos los demás nodos en V.

- Existen dos variantes:
  - Las ramas NO tienen costo. En este caso, la longitud de una ruta es simplemente la cantidad de ramas incluidas en la trayectoria.
  - Las ramas tienen asociado un costo. Este caso es más complicado en particular si se permiten distancias negativas. En caso de ciclos con costo negativo no existe solución.

Caminos más cortos

Algoritmo de Dijkstra.

Conjuntos que mantiene:

- Conjunto de vértices S que contiene los vértices para los que la distancia mas corta desde el origen ya es conocida, S= $\varnothing$ en el inicio,
- Conjunto de vértices Q=V-S que contiene, para cada vértice, la distancia mas corta desde el origen pasando a través de vértices que pertenecen a S => para cada vértice u $\in$ Q se mantiene la distancia mas corta desde el origen utilizando uno de los caminos mas cortos ya calculados.

Caminos más cortos. Algoritmo de Dijkstra

Estrategia del algoritmo:

- Se extrae del conjunto Q el vértice u cuya distancia provisoria D[u] es la menor. Se puede afirmar que esta distancia es la menor posible entre el vértice origen s y u.
- La razón de esta aseveración es que, debido a que los pesos de las aristas son no negativos, no será posible encontrar otro camino mas corto desde s hasta u pasando a través de algún otro vértice perteneciente a Q, ya que sus distancias provisorias eran mayores o iguales que la distancia provisional a u.

Caminos más cortos. Algoritmo de Dijkstra

- Al mismo tiempo, la distancia provisoria se correspondia con el camino mas corto posible pasando a traves de algun camino mas corto ya calculado; o lo que es lo mismo, con el camino mas corto posible utilizando vertices de S.

- Ya no es posible encontrar un camino mas corto desde s hasta u utilizando algun otro vertice del grafo, ya que $V = S \cup Q$, por lo que podemos concluir que hemos hallado el camino mas corto de s a u.

Caminos más cortos. Algoritmo de Dijkstra

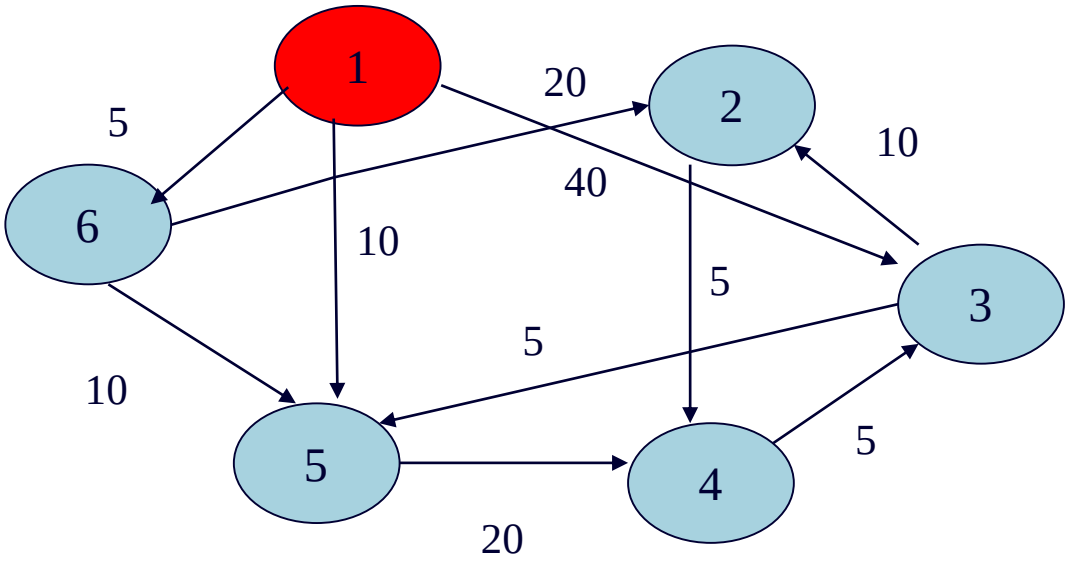
- Luego, se inserta el nuevo vértice u, para el que se ha calculado el camino más corto desde s, en el conjunto S ($S = S \cup \{u\}$). Al añadir u al conjunto S será posible acceder a los vértices v adyacentes a u a través del nuevo camino más corto calculado desde s hasta u.
- Podría ocurrir que las distancias provisorias de los vértices $v \cup Q$ adyacentes a u fueran mejoradas utilizando el nuevo camino, por lo que, si esto ocurre, se actualiza la distancia provisoria de estos vértices al nuevo valor calculado.

Función Dijkstra

```
Dijkstra (G,W,S)
para (cada vertice v ∈ V)
        D[v]=infinito
        P[v] = NULO
fin
D[s]=0
S=∅
Q=V
Mientras (Q≠ ∅)
      u= extraer_min(Q)
      S=S∪ {u}
      para (cada vertice v ∈ V adyacente a u)
        si D[v] > D[u] + W[u][v]
            D[v]=D[u] + W[u][v]
           P[v]=u
        fin
       fin
 fin
```

Función Dijkstra

Ejemplo



Seguimiento: Camino mas corto desde el vértice 1 a todos los demás

Función Dijkstra. Análisis de la complejidad temporal

```
Dijkstra (G,W,S)
para (cada vertice v ∈ V)
        D[v]=infinito
        P[v] = NULO
fin
D[s]=0
S=∅
Q=V
Mientras (Q≠ ∅)
     u= extraer_min(Q)
     S=S∪ {u}
     para (cada vertice v ∈ V adyacente a u)
      si D[v] > D[u] + W[u][v]
          D[v]=D[u] + W[u][v]
         P[v]=u
       fin
      fin
 fin
```

Representación del grafo: lista de adyacencia.

El ciclo ´Mientras´es el que determina el costo

- el bucle finalizará cuando Q = ∅,

  ⇒ se realiza |V| veces,

- extraer_min se realiza |V| veces

Se repite tantas veces como adyacentes tenga el vértice extraído de Q

⇒ se realiza |E| veces (max nro de adyacentes que tengan los vértices)

Función Dijkstra. Análisis de la complejidad temporal

extraer_min:

- Si los elementos de Q (vértice cuya distancia desde el origen sea la mínima respecto a todos los vértices de Q) están organizados en un arreglo sin orden $\Rightarrow O(|V|)$ y se realiza $|V|$, luego extraer_min: $O(|V|^2)$

Costo de *extraer_min* mas costo del ciclo,
COSTO TOTAL DIJKSTRA: $O(|V|^2 + |E|) = O(|V|^2)$

Caminos más cortos

Grafos dirigidos y ponderados:

- Obtener en el camino más corto desde un vértice ‘u’ a ‘todos’ los demás del grafo,
- Obtener los caminos más cortos desde todos los vértices a uno ‘destino’,
- Obtener el camino más corto desde un vértice ‘u’ a un vértice ‘v’
- Obtener los caminos más cortos entre todos los pares de vértices.

Caminos más cortos. Algoritmo de Floyd

Obtención del camino más corto entre cada para de vértices de un grafo rotulado (puede ser dirigido o no dirigido).

Floyd obtiene una tabla que brinda el costo menor requerido para ir de un vértice a otro.

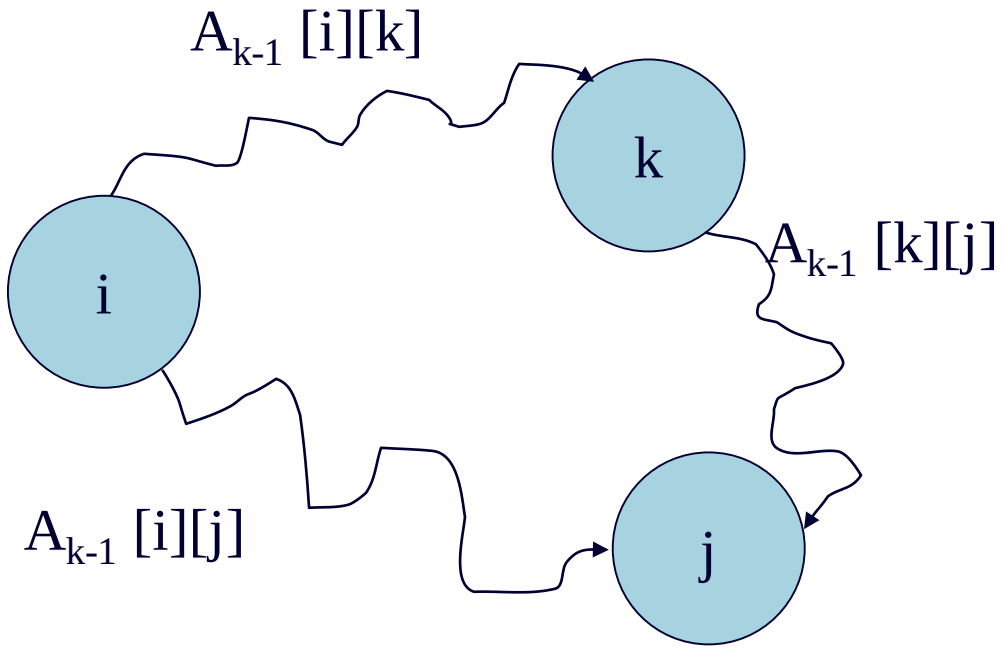
- v $\rightarrow$w tiene un costo no negativo
- Suponemos los vértices numerados 1,2,…n.
- Floyd usa una matriz ´A´ en la cual va a tener las longitudes de los caminos más cortos.

Caminos más cortos. Algoritmo de Floyd

- Inicialmente A[i][j]=C[i][j] $\forall$ i $\neq$ j
- Si no $\exists$ arco de i a j $\Rightarrow$ C[i][j] = infinito
- Cada elemento de la diagonal = 0
- Se itera n veces sobre la matriz A
- Luego de la k-ésima iteración, A[i][j] tendrá la longitud mas pequeña de cualquier camino desde i a j sin pasar por una cantidad de vértices > a k.

Caminos más cortos. Algoritmo de Floyd

Gráficamente:



Caminos más cortos. Algoritmo de Floyd

$$A_k[i][j] = \min \begin{cases} A_{k-1}[i][j] \\ A_{k-1}[i][k] + A_{k-1}[k][j] \end{cases}$$

Valor de la matriz A después de la k-ésima iteración

Caminos más cortos. Algoritmo de Floyd

Floyd es un algoritmo Programación Dinámica

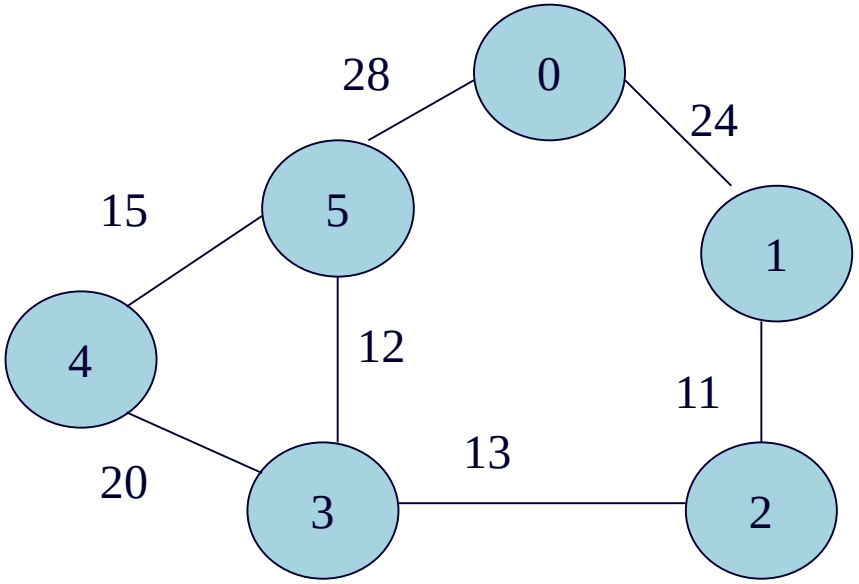
Para calcular $A_k[i][j]$ comparamos:

$A_{k-1}[i][j]$: costo de ir de i a j sin pasar a través de k o de cualquier otro vértice numerado mas alto,

$A_{k-1}[i][k]+A_{k-1}[k][j]$: costo de ir primero desde i a k y luego de k a j.

Caminos más cortos. Algoritmo de Floyd

Ejemplo: Grafo no dirigido ponderado



Matriz de costos

0	24	inf	inf	inf	28
24	0	11	inf	inf	inf
inf	11	0	13	inf	inf
inf	inf	13	0	20	12
inf	inf	inf	20	0	15
28	inf	inf	12	15	0

Iteraciones: pasando por vértice ´0´, ´1´, ´2´, ´3´, ´4´, ´5´.

Caminos más cortos. Función Floyd

```
Floyd (int A[][])
int i, k, j;
for(i=1;i<=n;i++)
  {for(j=1;j<=n;j++)
    {A[i][j]= C[i][j];
    for(i=1;i<=n;i++)
       A[i][j]= 0;
    for(k=1;k<=n;k++)
       {for(i=1;i<=n;i++)
          {for(j=1;j<=n;j++)
             if (A[i][k]+ A[k][j] < A[i][j])
               {A[i][j]=A[i][k]+A[k][j];}}}}}
```

Caminos más cortos. Función Floyd

Complejidad temporal del Algoritmo de Floyd:

$$O(n^3)$$

Caminos más cortos

Floyd garantiza la solución óptima para resolver el camino más corto entre cada para de vértices de un grafo.

Floyd se puede adaptar para resolver el siguiente problema: Determinar si ∃ un camino entre dos vértices.

Algoritmo de Warshall

Caminos más cortos. Algoritmo de Warshall

Warshall usa la matriz de adyacencia A[i][j]=1 si ∃ una arista entre los vértices i y j, si no, se asigna 0.

La matriz resultado se calcula aplicando, en la k-ésima iteración sobre la matriz A:

$A_k [i][j] = A_{k-1} [i][j] \vee (A_{k-1} [i][k] \wedge A_{k-1} [k][j])$

Caminos más cortos. Algoritmo de Warshall

Esto indica: hay un camino de i a j que no pasa por un vértice con número mayor que k, si:

1) Ya ∃ un camino de i a j que no pasa por un vértice > que k-1, o
2) Si hay un camino de i a k que no paso por un vértice numerado > que k-1 y un camino de k a j que no pasa por un vértice con número > que k-1.

Caminos más cortos. Algoritmo de Warshall

```
Funcion Warshall
for (i=1;i<=n;i++)
  {for(j=1;j<=n;j++)
      A[i][j]=C[i][j];}
 for (k=1;k<=n;k++)
   {for (i=1;i<=n;i++)
     {for (j=1;j<=n;j++)
        { if (A[i][j]=FALSE)
           A[i][j]=A[i][k] && A[k][j]
    }}}
```

Complejidad temporal

$O(n^3)$

Árboles de costo mínimo

Árbol de expansión de costo mínimo

Árbol de expansión

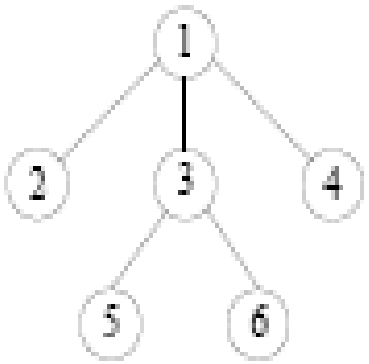
Introduciremos algunos conceptos necesarios para la exposición del problema que se pretende resolver.

Un grafo no dirigido, acíclico y conexo se denomina **árbol libre**.

Un árbol libre cumple las siguientes propiedades:

1. Un árbol libre con n>= 1 nodos contiene exactamente n − 1 arcos.
2. Si se añade una nueva arista a un árbol libre, se crea un ciclo.
3. Cualquier par de vértices están conectados por un único camino.

Árboles de costo mínimo



Un **árbol de expansión de un grafo no dirigido** G = (V , E), es un árbol libre
T = (V´,E´), tal que **V´ = V y E´⊆ E**; es decir T contiene **todos** los vértices de G, y las aristas de T son aristas de G.
Dado un grafo no dirigido y ponderado mediante una función de ponderación
w, el **costo de un árbol de expansión** T es la suma de los costes de todos los arcos del árbol.

$$w(T) = \sum_{(u,v) \in T} w(u.v)$$

Árboles de costo mínimo

Un **árbol de expansión de un grafo será de costo mínimo**, si se cumple que su costo es el menor posible respecto al costo de cada uno de los posibles árboles de expansión que contiene el grafo.

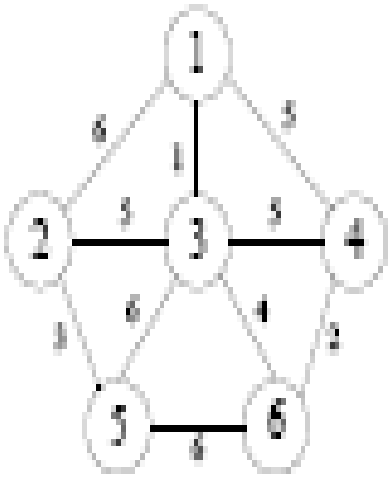
Árboles de costo mínimo

El propósito de obtener un árbol de recubrimiento de costo mínimo es obtener un nuevo grafo que sólo contenga las aristas imprescindibles para una optimización global de las conexiones entre todos los nodos.

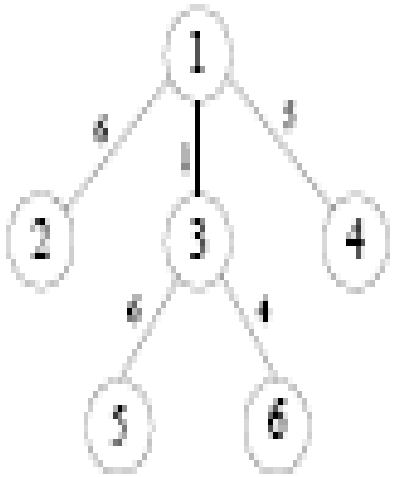
"optimización global": algunos pares de nodos pueden no quedar conectados entre ellos con el mínimo costo posible.

Árboles de costo mínimo

Ejemplos de árboles de expansión



Árbol de expansión
de costo 22



Árbol de expansión
de costo 15