GRAFOS

Algoritmos

Temario

- El problema de los caminos más cortos desde un vértice a otro
  - Algoritmo de Dijkstra.
- El problema de los caminos más cortos entre todos los pares de vértices.
  - Algoritmo de Floyd.
  - Repetir el Dijkstra n veces (n igual al número de vértices)
- El problema del árbol de extensión mínima a partir de un grafo
  - Algoritmo de Prim
  - Algoritmo de Kruskal

2

Ruta más corta

Un grafo con pesos: es un grafo en el cual se asignan valores a las aristas.

La longitud de un camino en un grafo con pesos es la suma de los pesos de las aristas en el camino.

Con frecuencia se desea determinar la ruta más corta entre dos vértices dados.

Dijkstra escribió el algoritmo que resuelve este problema.

3

Algoritmo de Dijkstra

- PROCEDURE DIJKSTA
- BEGIN
- S={nodo inicial};
- FOR j=1 to n do
- D[j] = C[nod inicial,j];
- FOR i=1 to n-1 BEGIN
- elige un vértice w en V-S tal que D[w] sea un mínimo;
- agregar w a S;
- FOR cada vértice v en V-S DO
- D[v] = min { D[v], D[w] + C[w,v];
- END
- END

4

Caminos más cortos

- En muchas aplicaciones (p.e. redes de transporte) las aristas tienen pesos diferentes.
- *Problema*: Hallar los caminos de peso total mínimo desde un vértice determinado (fuente) a todos los demás vértices.

5

Algoritmo de Dijkstra

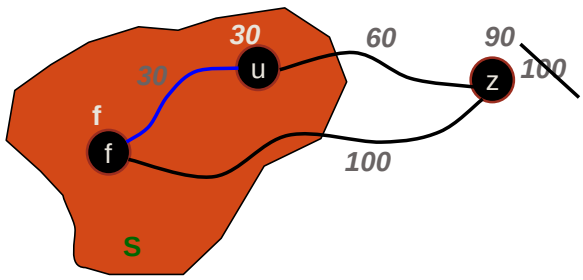
- La idea principal es realizar una búsqueda a lo ancho "ponderada" empezando por el vértice inicial *f*.
- De manera iterativa se construye un conjunto de vértices seleccionados S que se toman del conjunto de vértices candidatos C según el menor peso (distancia) desde *f*.
- El algoritmo termina cuando no hay más vértices de G fuera del conjunto formado.
- El paradigma de programación usado corresponde al ***método voraz***, en el que se trata de optimizar una función sobre una colección de objetos (menor peso).
- Se usa un vector d[*v*] para almacenar la distancia de *v* a *f*.

6

Algoritmo de Dijkstra

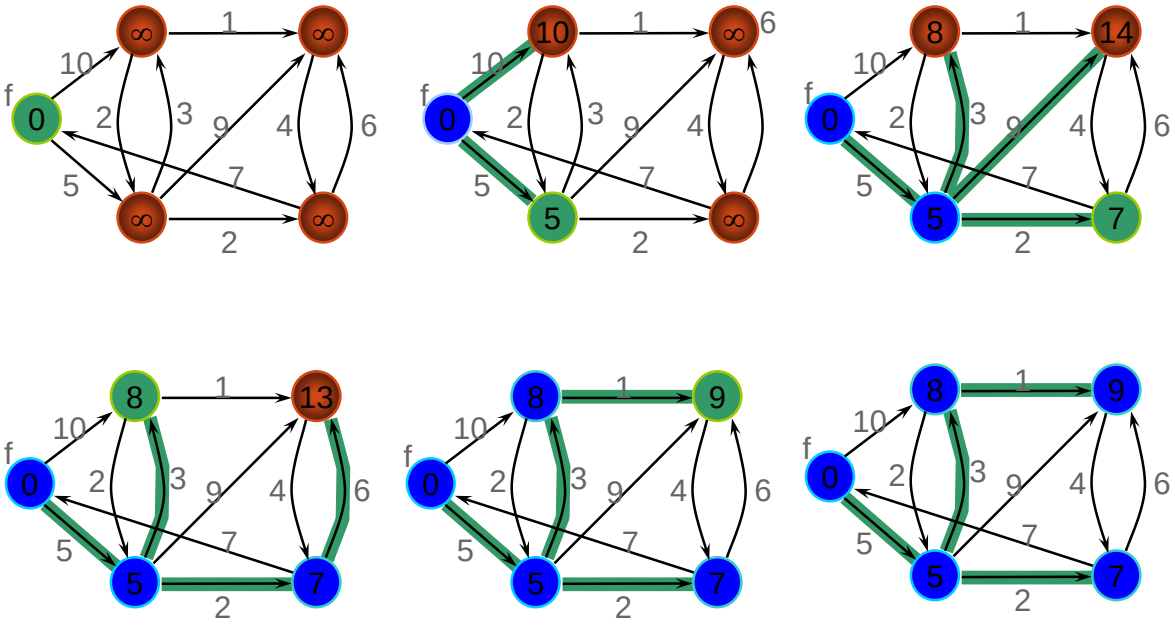
- Cuando se añade un vértice al conjunto S, el valor de d[*v*] contiene la distancia de *f* a *v*.
- Cuando se añade un nuevo vértice *u* al conjunto S, es necesario comprobar si *u* es una mejor ruta para sus vértices adyacentes *z* que están en el conjunto C.
- Para ello se actualiza d con la ***relajación*** de la arista (*u*, *z*):

$$d[z] = \min(d[z], d[u] + w[u, z])$$



7

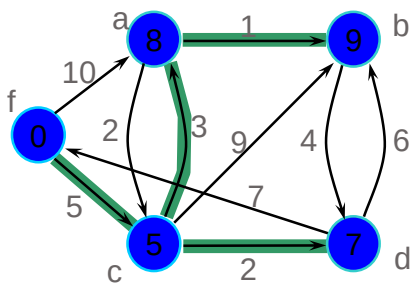
Algoritmo de Dijkstra



8

Algoritmo de Dijkstra

- Para reconstruir los vértices del camino más corto desde el vértice fuente a cada vértice:
  - Se usa otro array p[*v*] que contiene el vértice anterior a *v* en el camino.
  - Se inicializa p[*v*] = *f* para todo v ≠ *f*
  - Se actualiza p siempre que d[*v*] > d[*w*] + w(*w*, *v*) a:
    p[*v*] = *w*.
  - El camino a cada vértice se halla mediante una traza hacia atrás en el array p.



p[a] = c p[b] = a p[c] = f p[d] = c

9

Algoritmo de Dijkstra

- Complejidad:
  - Con matrices de adyacencia: $O(n^2)$.
  - Con listas de adyacencia: $O(n^2)$.
  - Se puede usar estructuras de datos más eficientes, como una cola de prioridad para buscar el vértice con menor d[*v*] se consigue O(e*log n).

10

Caminos más cortos

- *Problema*: Hallar los caminos mínimos entre cualquier par de nodos de un grafo.
- Se puede usar el algoritmo de Dijkstra tomando cada vértice como fuente.
- Existe otra alternativa: *Algoritmo de Floyd*.

11

Algoritmo de Floyd

- Utiliza una matriz $A_k[i][j]$, que contiene el camino más corto que pasa por los primeros k primeros vértices.
- Inicialmente $A_k[i][j] = C[i][j] \ \forall \ i \neq j$. Si no hay arista de i a j $C[i][j] = \infty$ y los elementos diagonales se ponen a 0.
- En la iteración k (nodo k como pivote) se calcula, para cada camino de v a w, si es más corto pasando por k aplicando:

  $A_k[i][j] = \min (A_{k-1}[i][j] , A_{k-1}[i][k] + A_{k-1}[k][j]), \forall \ i \neq j.$
- Como no varía la fila y la columna k en la iteración k, sólo es necesario una matriz A.

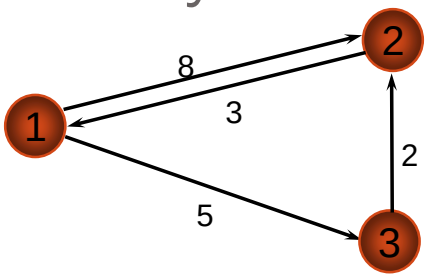
12

Algoritmo de Floyd

```
Floyd (n, C, A)
{
   A[i][j] = C[i][j]
   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]
}
```

13

Algoritmo de Floyd



C[*i*][*j*]	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

A_1[*i*][*j*]	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

A_2[*i*][*j*]	1	2	3
1	0	8	5
2	3	0	8
3	5	2	0

A_3[*i*][*j*]	1	2	3
1	0	7	5
2	3	0	8
3	5	2	0

14

Algoritmo de Floyd

- Para obtener los caminos se procede como en Dijkstra. Se usa una matriz P[i][j] para almacenar el camino:
  - P[i][j] = 0 si el camino es directo.
  - En otro caso, si

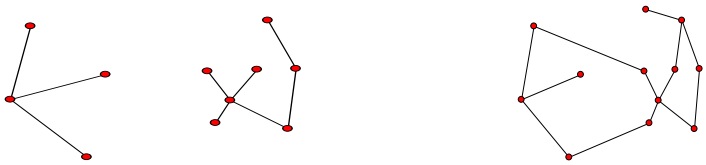
    if (A[i][k] + A[k][j] < A[i][j])

    P[i][j] = k
- La complejidad del algoritmo es:
  - Con matriz de adyacencia: O(n^3).
  - Para grafos dispersos es mejor usar la versión Dijkstra con lista de adyacencia que toma O(ne log n).

15

ARBOLES

- Un grafo se dice un árbol si es conexo y no tiene ciclos.
- Los primeros dos grafos son árboles:



16

ARBOLES

- Por tanto, un grafo es un árbol ⇔ entre cada par de vértices existe un camino y sólo uno.
- Un grafo se dice un bosque si sus componentes conexas son árboles.
- **Teorema.-** Sea G(V,E) un grafo. Son equivalentes

  a) G es un árbol

  b) Cada par de vértices distintos de V esta conectado por un único camino.

  c) G es conexo y toda arista de G es de separación

  d) G no tiene ciclos y $|V| = |E| + 1$

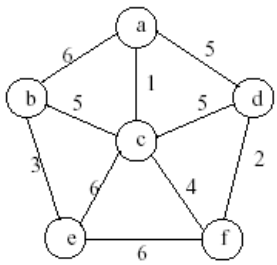
  e) G es conexo y $|V| = |E| + 1$

  f) G no tiene ciclos pero al añadirle una arista a G se crea un único circuito

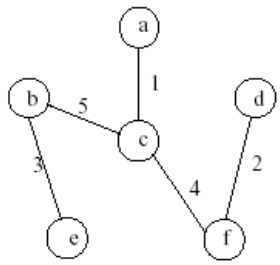
17

ARBOL GENERADOR

- **Definición.-** Sea G un grafo, un árbol generador de G es un subgrafo conexo de G que tiene los mismos vértices que G y no tiene circuitos.



Grafo no dirigido



Arbol generador de costo mínimo

18

ARBOL GENERADOR

- Supongamos que a cada arista se le asocia un número positivo (su peso). Un árbol generador se dice de peso mínimo si la suma de los pesos de las aristas que lo componen es lo menor posible
- Para *calcular el árbol de peso mínimo* existen 2 algoritmos:
  - Kruskal: Se van escogiendo las aristas de menor peso hasta conseguir un árbol de peso mínimo
  - Prim: Consiste en ir borrando las aristas de mayor peso posible y que no sean aristas de separación.
- Puede haber más de un árbol generador de peso mínimo, pero todos deben tener el mismo peso.

19

ALGORITMO DE PRIM

La idea básica consiste en añadir, en cada paso, una arista de peso mínimo a un árbol previamente construido. Más explícitamente:

Paso 1. Se elige un vértice u de G y se considera el árbol S={u}

Paso 2. Se considera la arista e de mínimo peso que une un vértice de S y un vértice que no es de S, y se hace S=S+e

Paso 3. Si el nº de aristas de T es n-1 el algoritmo termina. En caso contrario se vuelve al paso 2

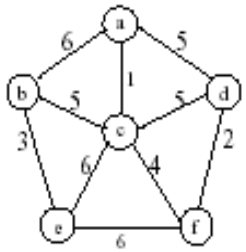
20

ALGORITMO DE PRIM

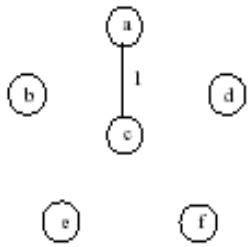
```
void Prim( GRAFO G, CONJUNTO T )
{
   CONJUNTO_V  U;
   VERTICE     v;

   T = ∅;
   U = { cualquier vértice de G };
   while( U != V ) {
     Sea (u,v) es el arco de menor costo tal que
         u ∈ U  y  v ∈ V-U;
     T = T ∪ { (u,v) };
     U = U ∪ { v };
   }
}
```

21

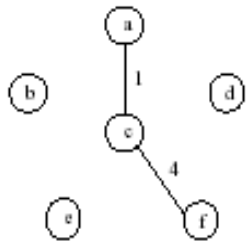
ALGORITMO DE PRIM



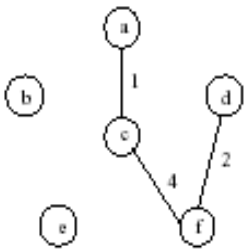
Grafo original



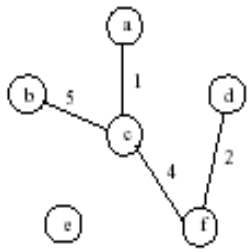
a)



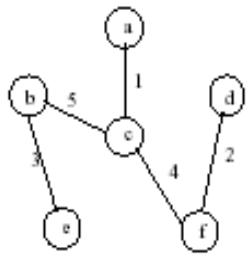
b)



c)



d)



e)

22

ALGORITMO DE KRUSKAL

- La idea básica consiste en elegir sucesivamente las aristas de mínimo peso sin formar ciclos.
- **Paso 1**. Se elige la arista de mínimo peso e y se considera S={e}.
- **Paso 2**. Sea e' la arista de mínimo peso tal que e'ÏS y S+e' es un grafo acíclico. Se hace S=S+e'.
- **Paso 3.** Si S tiene n-1 aristas, el algoritmo termina. En caso contrario se vuelve al paso 2.

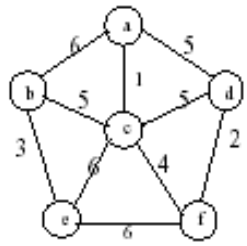
23

ALGORITMO DE KRUSKAL

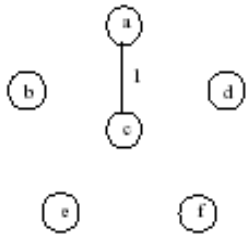
```
void Kruskal( GRAFO G, CONJUNTO T )
{
   CONJUNTO_V  U;
   VERTICE     v;

   T = ∅;
   for( cada vértice v de G )
     construye un árbol con v;
   Ordena los arcos de G en orden no decreciente;
   while( Haya más de un árbol ) {
     Sea (u,v) es el arco de menor costo tal que el árbol
     de u es diferente al árbol de v;
     Mezcla los árboles de u y de v en uno solo;
     T = T ∪ { (u,v) };
   }
}
```

24

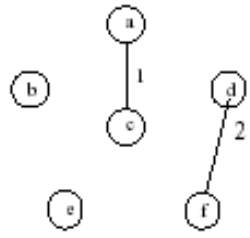
ALGORITMO DE KRUSKAL



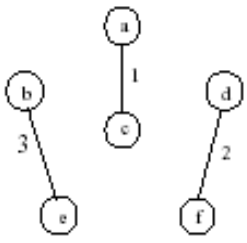
Grafo original



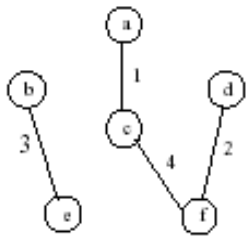
a)



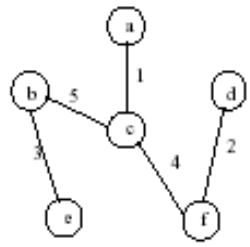
b)



c)



d)



e)

25