Pseudocodigos-grafos-explicados-...

lili__

Algoritmos

2º Grado en Ingeniería Informática

Facultad de Informática
Universidad de A Coruña

ORDENACIÓN TOPOLÓGICA

```
función Ordenación topológica 2 (G:grafo): orden[1..n]
  Grado Entrada [1..n] := Calcular Grado Entrada (G);
  Crear Cola (C); contador := 1 ;
  para cada nodo v hacer
    si Grado Entrada [v] = 0 entonces Insertar Cola (v, C);
  mientras no Cola Vacía (C) hacer
    v := Eliminar Cola (C);
    Número Topológico [v] := contador; incrementar contador;
    para cada w adyacente a v hacer
      Grado Entrada [w] := Grado Entrada [w] - 1;
      si Grado Entrada [w] = 0 entonces Insertar Cola (w, C)
  fin mientras;
  si contador <= n entonces devolver error "el grafo tiene un ciclo"
  sino devolver Número Topológico
fin función
```

(1) → grado de entrada: se calcula contando cuantas flechas señalan a cada nodo

n° topológico →
nodo →

Crear cola:

Contador = 1

*Si grado = 0 ⇒ insertar:

1

Contador = 1

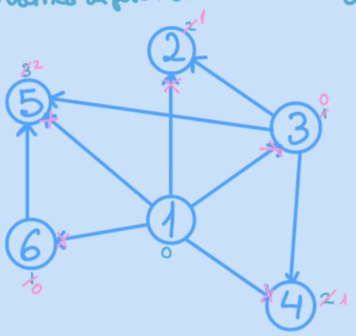
⇒ Entramos en el mientras:

V = eliminar 1ero de la cola = 1 ⇒

Num topológico v = 1 ⇒ contador = 1+1 = 2

n° topológico →	1					
nodo →	1	2	3	4	5	6

→ Vamos al para: en cada nodo adyacente al de 1, si el grado de entrada queda en 0: insertar en cola



*si hay varios se pone el pequeño primero

3	6

contador = 2

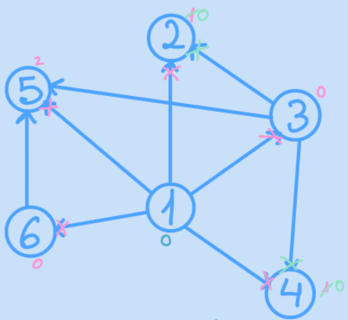
⇒ Seguimos en el mientras:

V = eliminar 1ero de la cola = 3 ⇒ 6

Num topológico v = 2 ⇒ contador = 2+1 = 3

	1		2			
nodo →	1	2	3	4	5	6

→ Vamos al para: en cada nodo adyacente al de 1, si el grado de entrada queda en 0: insertar en cola



6	2	4

contador = 2

⇒ Seguimos hasta completar la tabla

KRUSKAL

nº de nodos — nº de aristas

función Kruskal (G =(N,A)) : árbol
  Ordenar A según longitudes crecientes;
  n := |N|;
  T := conjunto vacío;
  inicializar n conjuntos, cada uno con un nodo de N;
  repetir
    a := (u,v) : arista más corta de A aún sin considerar;
    Conjunto U := Buscar (u);
    Conjunto V := Buscar (v);
    si Conjunto U <> Conjunto V **entonces**
      Fusionar (Conjunto U, Conjunto V);
      T := T U {a}
  hasta |T| = n-1;
 devolver T
fin función

⇒ Ordenamos por longitudes crecientes:

arista	2-3	3-4	1-4	1-5	3-5	5-6	1-2	1-3	1-6
longitud	1	1	2	2	3	3	4	5	6

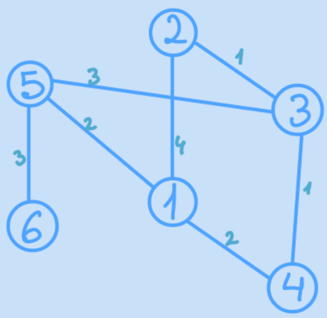
n=6

⇒ Inicializar conjuntos, 1 por nodo:
{1}, {2}, {3}, {4}, {5}, {6}

⇒ Empezamos el ciclo "repetir":

PRIM

```
función Prim 2 ( L[1..n,1..n] ) : árbol
 Distancia Mínima [1] := -1; ⇒ la del 1er nodo se cuenta como ya metido en el árbol
 T := conjunto vacío;
 para i := 2 hasta n hacer
  Más Próximo [i] := 1; ⇒ nodo anterior
  Distancia Mínima [i] := L[i,1] ⇒ costo más bajo conocido para llegar a i
 fin para;
 repetir n-1 veces: {bucle voraz}
  min := infinito;
  para j := 2 hasta n hacer
   si 0 <= Distancia Mínima [j] < min entonces   ⇒ se cumple con j=2        ⇒   se cumple con j=4
    min := Distancia Mínima [j];                   min=4                        min=2
    k := j                                         k=2                          k=4
  T := T U { ( Más Próximo [k], k ) }; ⇒ añadimos (1,4) al árbol
  Distancia Mínima [k] := -1; {añadir k a B} ⇒ lo marcamos como visitado
  para j := 2 hasta n hacer
   si L[j,k] < Distancia Mínima [j] entonces ⇒ se cumple con j=3
    Distancia Mínima [j] := L[j,k];             distancia mín = 2
    Más Próximo [j] := k                        más próximo = 4
fin función
```

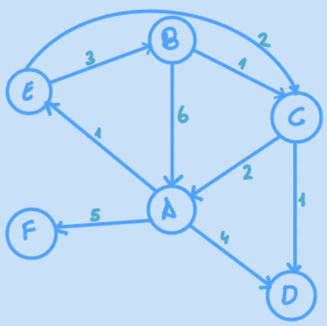


nodo →	2	3	4	5	6
Distancia mínima[nodo] →	4	2	2	2	∞
Más próximo[nodo] →	1	4	1	1	

min = ∞

DIJKSTRA

```
conjunto de nodos menos el primero
función Dijkstra ( L[1..n,1..n] ) : vector[1..n]
  C := {2, 3, ..., n};
  para i := 2 hasta n hacer
distancia mín entre el nodo 1 y el nodo i
    D[i] := L[1,i];   longitud desde el nodo 1 al nodo i
nodo anterior a i en el camino mín
    P[] := 1
  repetir n-2 veces:
      n-2 porque hay n-1 aristas en un grafo pero como quitamos 1 nodo → (n-1)-1 = n-2 (el de origen)
    v := nodo de C que minimiza D[v];   → seleccionamos el nodo más cercano al nodo 1 que aún esté en C básicamente devolvemos la distancia más corta
    C := C-{v};
    para cada w en C hacer
      si D[w] > D[v]+L[v,w] entonces
        D[w] := D[v]+L[v,w];
        P[w] := v
    fin si
  fin repetir;
  devolver D
fin función
```



⇒ Bucle para: inicializamos D con la distancia directa de 1 a i:

* ∞ ⇒ no hay conexión directa

nodo →	B	C	D	E	F
D[nodo] →	∞	∞	4	1	5
P[nodo] →			A	A	A

nodo anterior

⇒ Entramos en repetir:

V = nodo con menor distancia sin visitar = E → D[E]=1

→ Quitamos v del conjunto C porque sabemos que su distancia no va a cambiar → así no lo seleccionamos en la siguiente iteración

→ Vamos al para: → w = vecino del nodo v que está en C

si la D[w] es mayor que la nueva distancia calculada (D[V] + L[V, w])

→ w = C:

D[C] = ∞
D[V] = D[E] = 1
L[V, w] = L[E, C] = 2

} ∞ > 1+2=3 ↳ se cumple:

D[w] = 3
P[w] = V

nodo →	B	C	D	E	F
D[nodo] →	∞	3	4	1	5
P[nodo] →		E	A	A	A

→ w = B

⋮

RECORRIDO EN ANCHURA

```
procedimiento ra (v)
  Crear Cola (C); marca[v] := visitado; Insertar Cola
(v,C);
  mientras no Cola Vacía (C) hacer
    u := Eliminar Cola (C);
    para cada nodo w adyacente a u hacer
      si marca[w] != visitado entonces
        marca[w] := visitado; Insertar Cola (w,C)
fin procedimiento
```