Grafos dirigidos

Grafos dirigidos, notas

- Grado: Cantidad de aristas de un nodo
- Un árbol con raíz es un grafo dirigido
- Grafos acíclicos dirigidos: Grafos sin ciclos
- Grafos bipartitos: grafos con nodos de distintos grupos, ninguna arista conecta a dos nodos del mismo grupo

Para comprobar si un nodo está conectado con otro lo podes ver como m[i][j]==1, ciendo “1” el número de aristas que une a i con j

Una de las manera de ver las uniones de los grafos es como si fuera una tabla hash, en donde tenes cada nodo (celda) y dentro de cada uno hay una lista que son sus uniones, por ejemplo:

- Lista 1: 2, 3, 4
- Lista 2: 3, 4
- Lista 3: 1, 4
- Lista 4: 2

Longitud de un camino: Por cuantos nodo pasa dicho camino, ejemplo, si va de 1 a 1, la longitud es 0, si va de 1 a 3, de forma 1 → 2 → 4, la longitud es 2

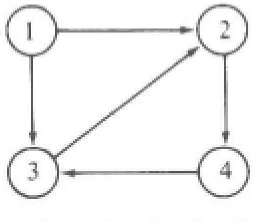


Fig. 6.1. Grafo dirigido.

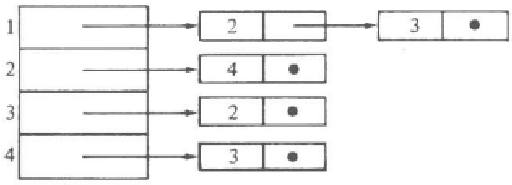


Fig. 6.4. Representación con lista de adyacencia para el grafo dirigido de la figura 6.1

Operaciones de grafos dirigidos:

1. PRIMERO(v), que devuelve el índice del primer vértice adyacente a v. Se devuelve un vértice nulo Λ si no existe ningún vértice adyacente a v.
2. SIGUIENTE(v, i), que devuelve el índice posterior al índice i de los vértices adyacentes a v. Se devuelve Λ si i es el último vértice de los vértices adyacentes a v.
3. VERTICE(v, i), que devuelve el vértice cuyo índice i está entre los vértices adyacentes a v.

Dijkstra Algoritm:

El peso de una arista es RANDOM, el peso de ir de 1⟶2 puede ser 1, como puede ser también 55 (mientras sea ≥0) y no tiene porque ser igual para todas las aristas.

Relaxation:

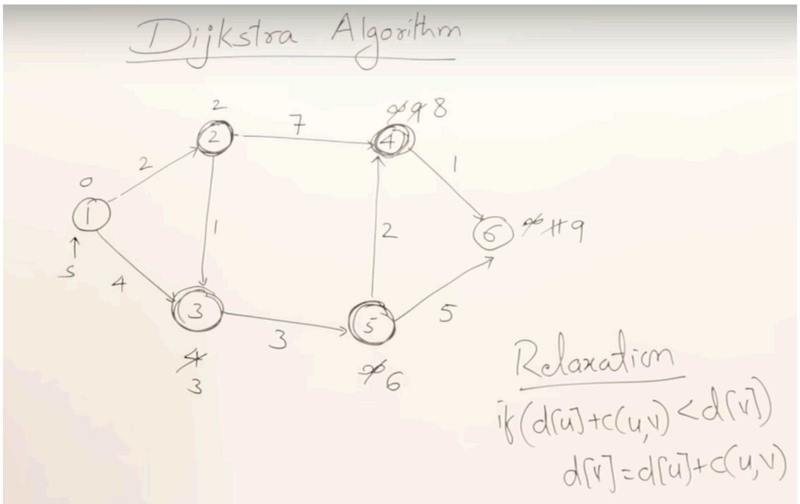
if (d(u) + d(u,v) < d(v)):

    d(v) = d(u) + d(u,v)

(siendo d(v) la distancia desde el nodo actual hasta el nodo buscado, sin que tenga un acceso directo)

Cuando no hay camino directo, se dice que la distancia d(v) es infinita, y luego se calcula enserio

Para ver el camino más corto de un nodo hasta otro nodo v es necesario ir viendo nodo a nodo cual tiene una distancia más corta, como se hizo en este video que se fue eliminando y reemplazando a medida que encontraba distancias más cortas:

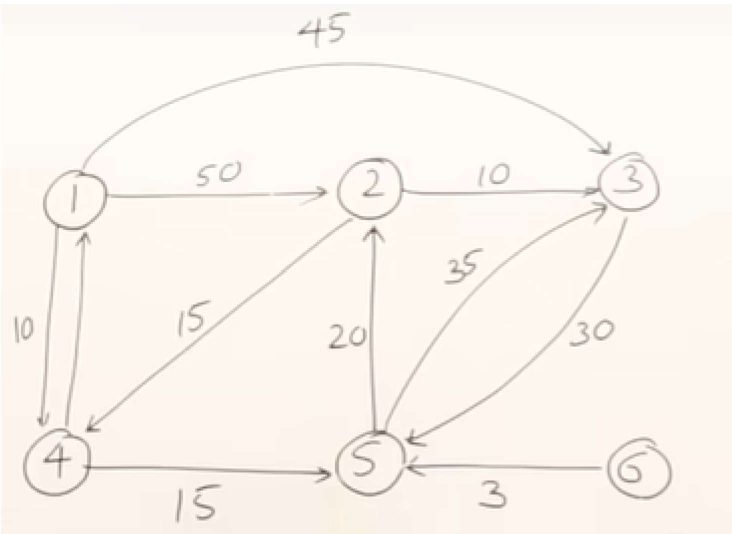


n = número de vertices(V)

Tiempo de ejecución: O(n), siendo más específico: O(V)

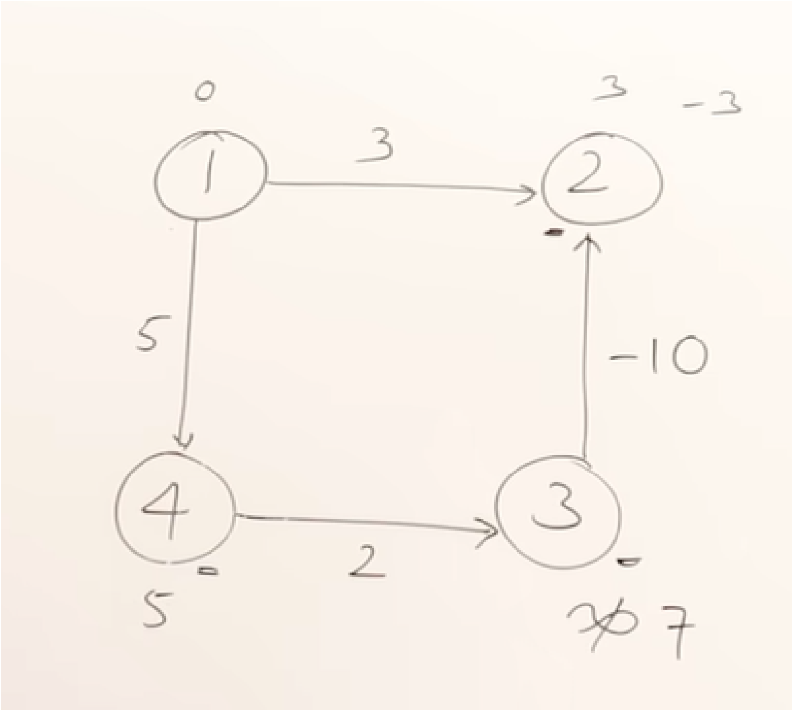
Peor caso: O(n^2).

Si no hay ningún nodo que conecte al final, la distancia seguirá siendo infinito, como en el siguiente caso:



En este caso la d(v) con v siendo 6, será infinito

A veces este algoritmo puede fallar con casos en donde el peso o distancia sea negativo, como el siguiente:



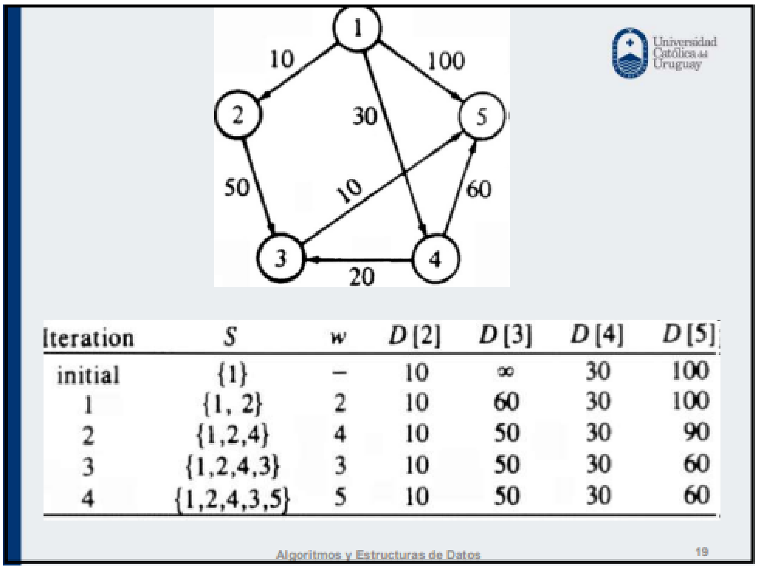
En donde siguiendo con el algoritmo la distancia más corta daría 3, cuando la mejor opción es hacer todo el camino largo para terminar con una distancia de -3. Aunque esto no es siempre así, ya que si en su lugar era -3, seguía andando bien.

```
El algoritmo de Dijkstra Algoritmos y Estructuras de Datos 17
Función Dijkstra
COM
Inicializar S, D
S = {1};
para i = 2 a n hacer D[i] = C[1,i] //(el valor inicial, infinito si
//no hay camino directo)
   Mientras V <> S hacer
      Elegir w perteneciente a V-S, tal que la distancia D[w)] sea un mínimo
      Agregar w a S
      ParaCada v perteneciente a V-S hacer
         D[v] = min (D[v], D[w]+ costo(w,v)
   FinMientras;
FIN {Dijkstra}
```

```
procedure Dijkstra (con caminos)
COM
```

```
Inicializar S, D, P. S = {1};
para i = 2 a n hacer D[i] = C[1,i] (el valor inicial, infinito si no hay camino direct
   Mientras V <> S hacer
      Elegir w perteneciente a V-S, tal que la distancia D[w)] sea un mínimo
      Agregar w a S
         Para cada v perteneciente a V-S hacer
            Si D[w]+ costo(w,v) < D[v] entonces
               D[v] = D[w] + costo(w,v) y P[v] = w
            finsi
         Fin para cada
   FinMientras;
FIN {Dijkstra}
```

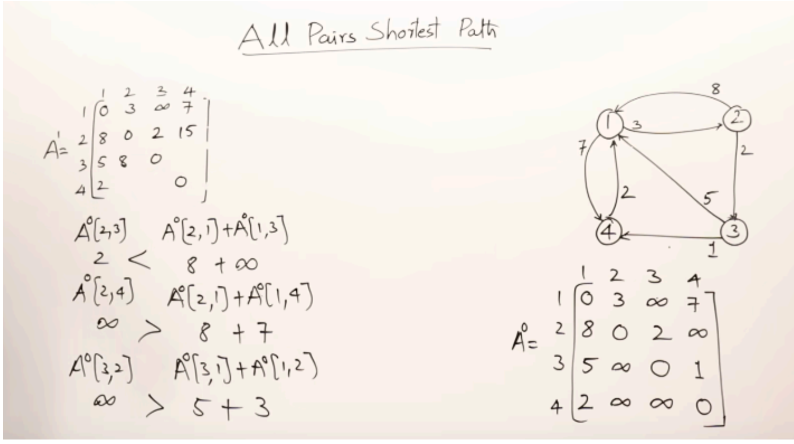
Por pasos en donde vas viendo los casos que ya tenes:

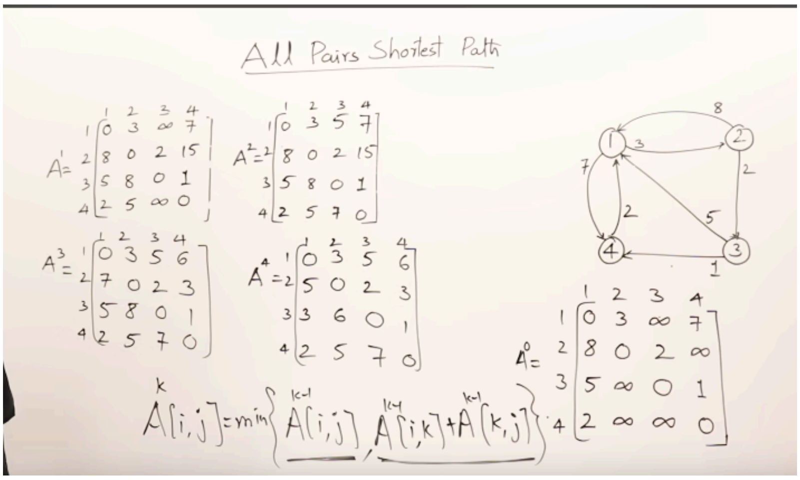


Iteration	S	w	D[2]	D[3]	D[4]	D[5]
initial	{1}	–	10	∞	30	100
1	{1, 2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

S: conjunto de vértices cuya distancia más corta al origen es conocida. Al principio S sólo contiene el origen

Método Floyd-Marshall:





Orden: O(n^2)

Peor caso: O(n^3)

```
Función Floyd (A : array[1..n,1..n] of real;
C : array[1..n,1..n] of real); 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;
```

```
Función Floyd (var A : array[1..n,1..n] of real;
C : array[1..n,1..n] of real);
i, j, k : integer;
COM
for i:= 1 to n do
   for j:= 1 to n do
      A[i,j]:= C[i,j]; P[i, j] := 0 ;
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]; P[i, j] := k ;
END;
```

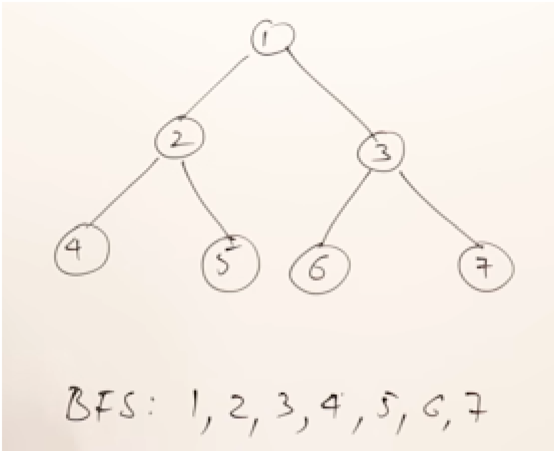
```
procedure Warshall (A : array[1..n,1..n] of boolean;
C : array[1..n,1..n] of boolean);
i, j, k : integer;
COM
for i:= 1 to n do
   for j:= 1 to n do
      A[i,j]:= C[i,j];
for k:= 1 to n do
   for i:= 1 to n do
      for j:= 1 to n do
         if A[i,j] = false
            then A[i,j]:= A[i,k] and A[k,j];
FIN;
```

BFS y DFS:

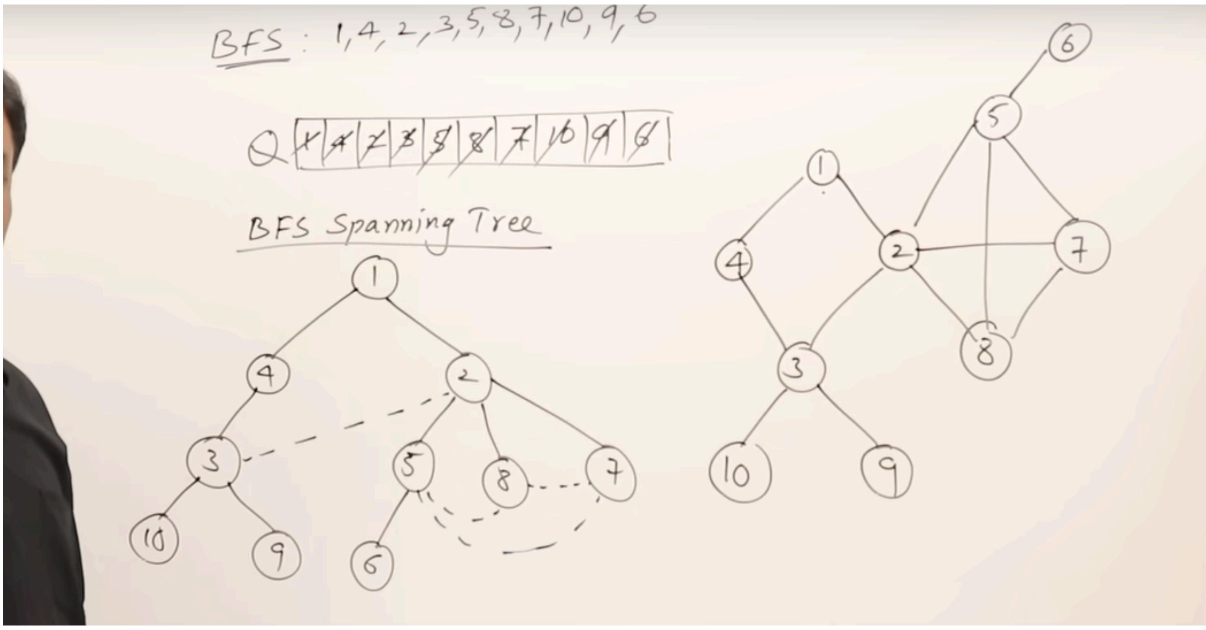
BFS (Breadth First Search):

Búsqueda en amplitud en español

El orden en el que se anotan los nodos es el mismo que el de niveles de un árbol



Ejemplo más desarrollado:

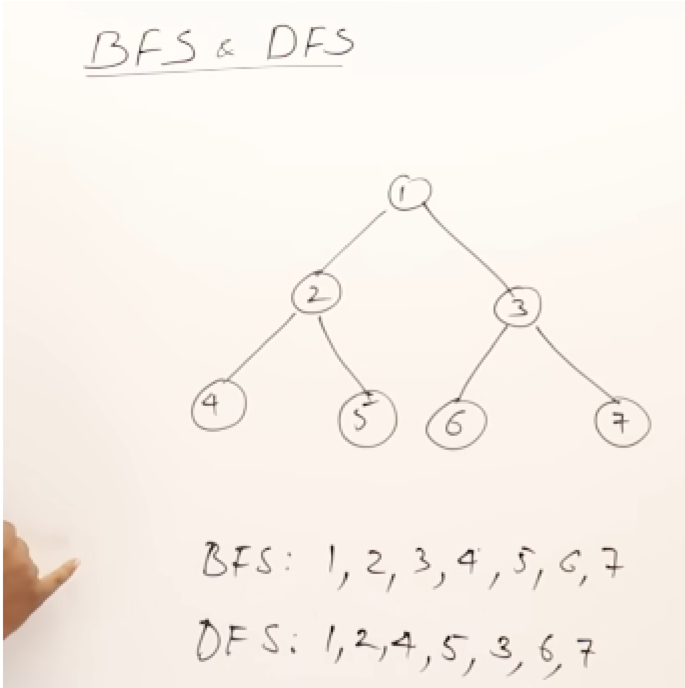


No importa con cual nodo arranques, importa que cuando vas a hacerlo, anotes todos los nodos adyacentes del que estas viendo, y marques bien las conexiones. Cross edges (arcos cruzados), son los vínculos con líneas punteadas.

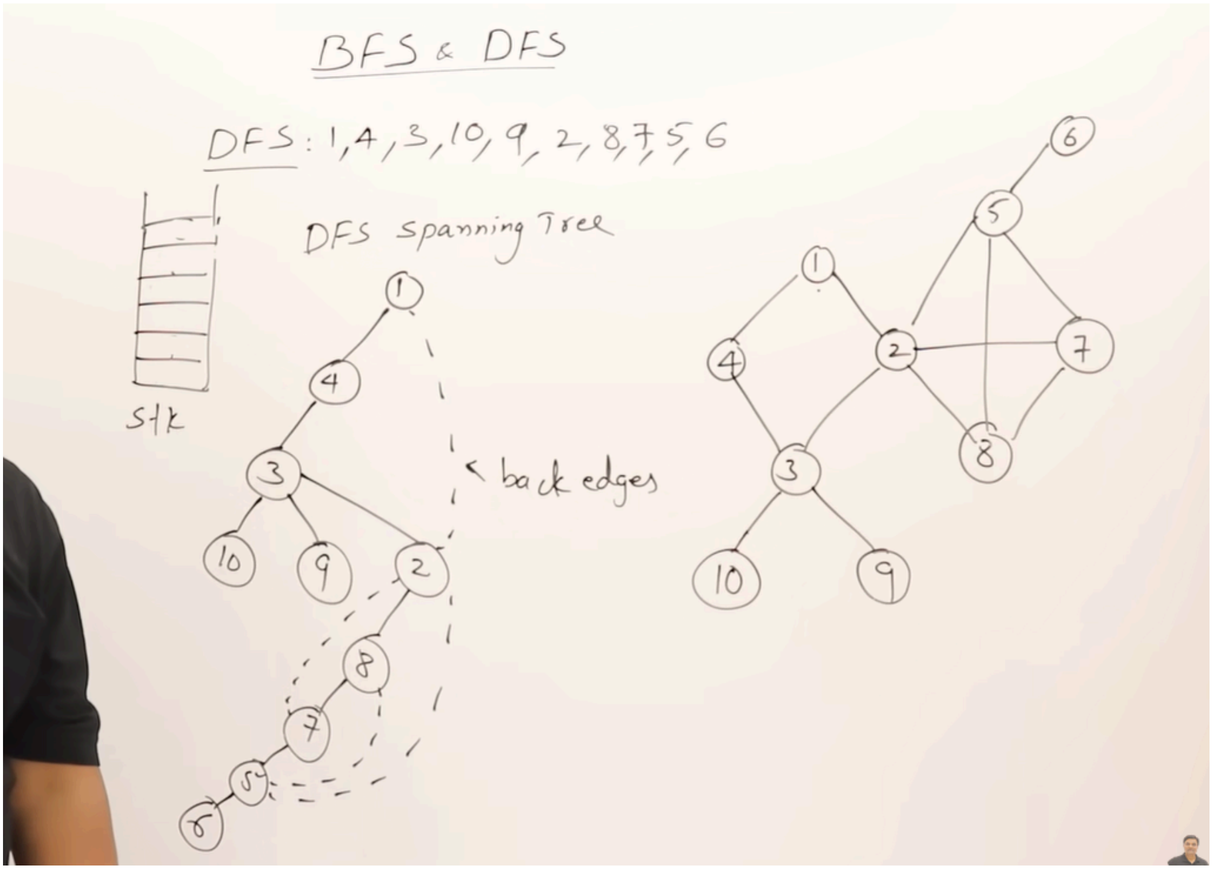
DFS (Depth First Search):

Búsqueda en profundidad en español

El orden en preorden



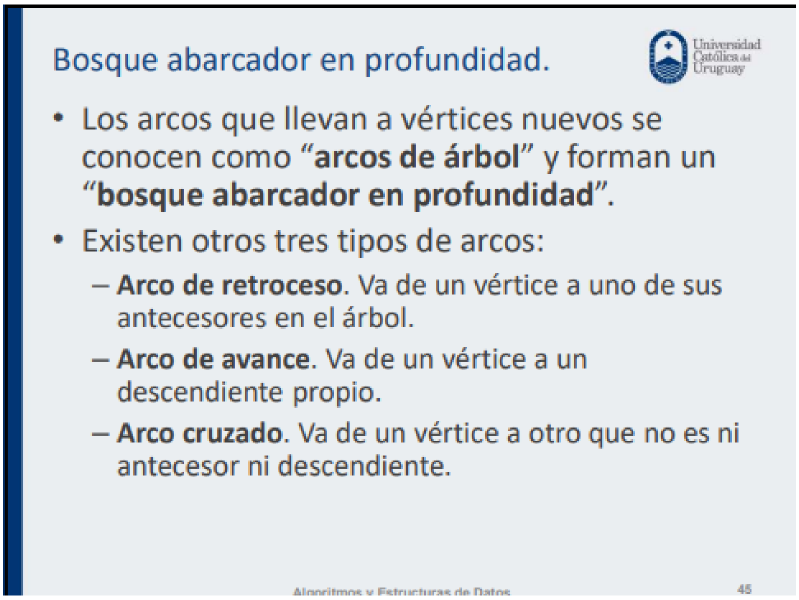
Mejor explicado:

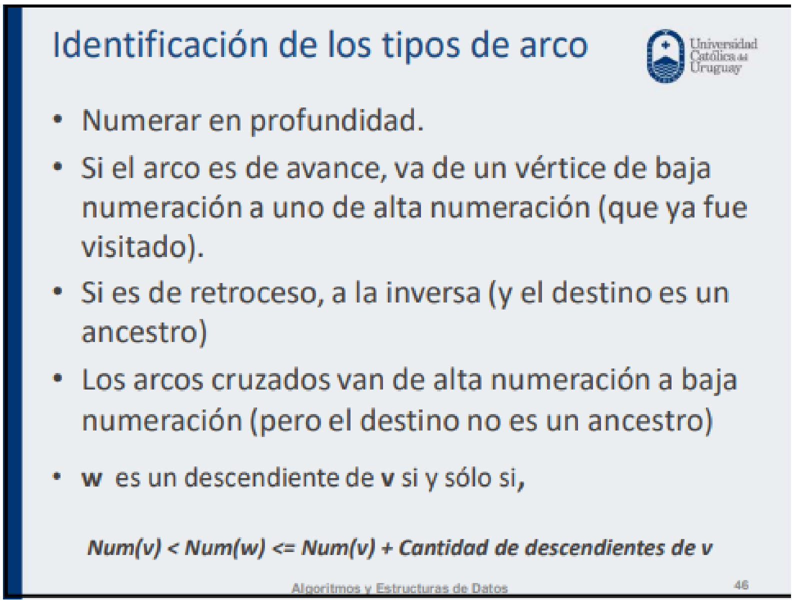


Literalmente es recorrer en preorden. Se puede hacer partiendo del hijo derecho también, pero lo importante es que siga ese método de empezar yendo a por los hijos y luego de que no hayan más ir hacia arriba. Back edges son los arcos de retroceso.

Representación	Complejidad BFS/DFS
Lista de adyacencia	**O(n + m) (V + A)**
Matriz de adyacencia	**O(n^2)**

```
métodoTvertice.bpf ();
w : Tvertice;
COM
(1) Visitar();
Para cada adyacente w hacer
(2) Si no(w.visitado()) entonces
   (3) w.bpf()
      Fin Si
Fin para cada
FIN {bpf}
```

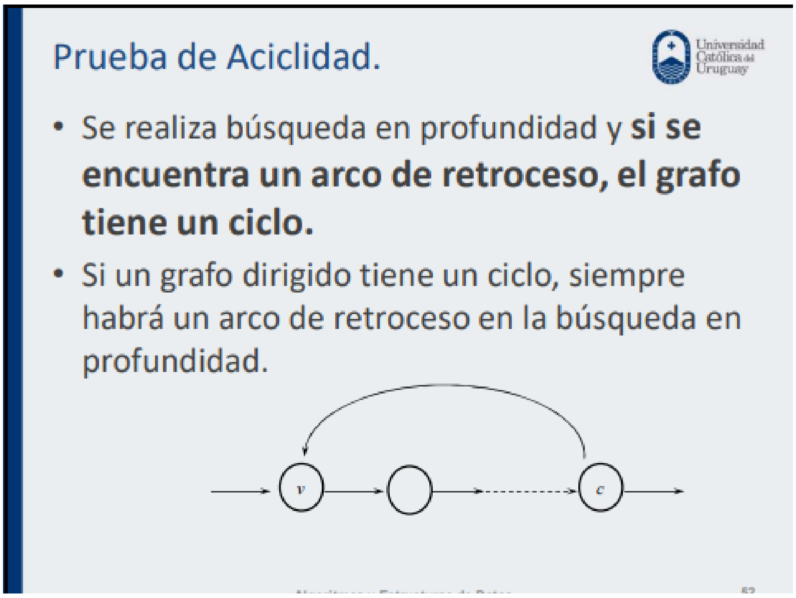





```
método Camino (Destino : Tvértice;
ElCamino: TCamino);
w : Tvértice;
COM
Visitar(); Agregar(“this”, ElCamino)
   Para cada adyacente w
      Si w = Destino entonces
         Guardar(ElCamino+Destino)
      Sino
         Si no(visitado(w)) entonces
            Camino(w,Destino,ElCamino)
         Fin si
      Fin si
   Fin para cada
Quitar(“this”, ElCamino)
FIN
```

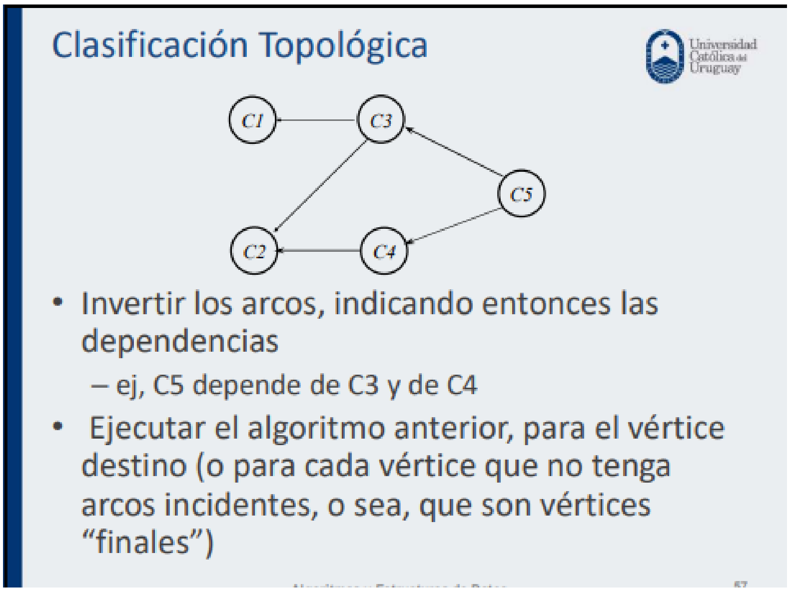
Excentricidad:

La máxima distancia posible entre un nodo v y el resto de nodos, siempre y cuando estemos hablando de los caminos más cortos que toma v hasta dichos nodos.



Clasificación topológica:

```
procedure ClasificacionTopologica ();
w : Tvertice;
COM
(1) Visitar();
   (2) Para cada adyacente w hacer
      (3) Si no(w.visitado()) entonces
               w.ClasificacionTopologica()
            Fin Si
         Fin para cada
         imprimir (); //agregar "this" al principio de la lista de
previas....
FIN; {ClasificacionTopologica}
```



Componente fuertemente convexo (CFC):

Cuando hay un bucle y hay un ida y vuelta entre ciertos nodos de un grafo, ejemplo:

```
A → B → C → A     (forma un ciclo → componente fuerte)
D → E           (pero E no puede volver a D → no están en el mismo component
```

Componentes fuertes:

- {A,B,C} → todos se alcanzan mutuamente
- {D}
- {E}