Algoritmos y estructuras de datos

Grafos

CEIS

Escuela Colombiana de Ingeniería

2024-2

Agenda

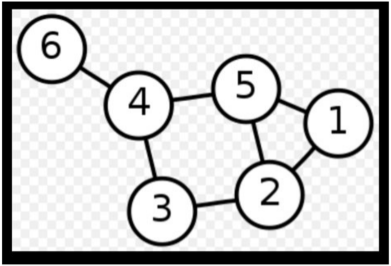
Agenda

Definición

Es una pareja ordenada $G = (V, E)$ donde:

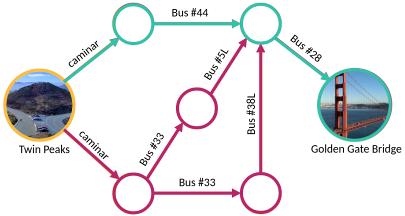
- V es un conjunto de vértices o nodos
- E es un conjunto de aristas o arcos que relacionan estos nodos

El orden del grafo G es su número de vértices $|V|$

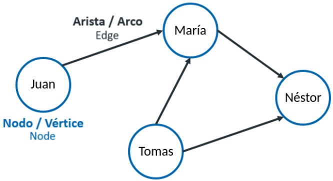


Ejemplos

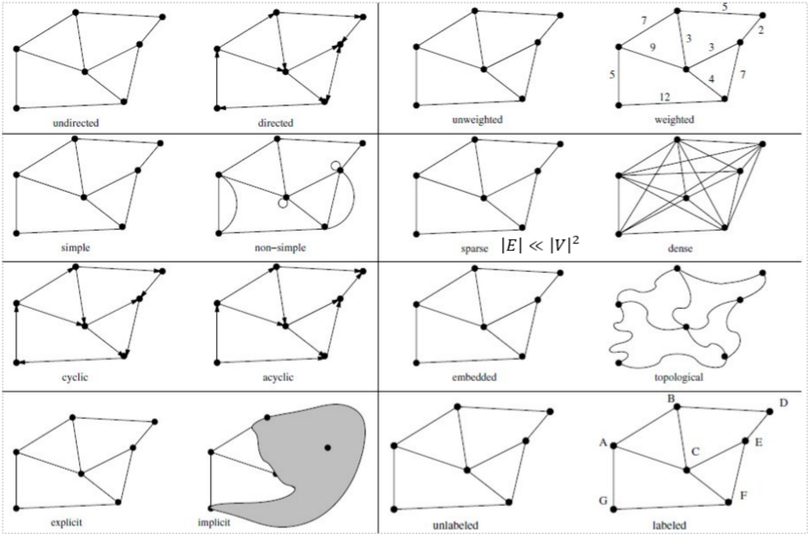
Rutas



Relaciones

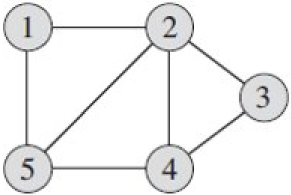


Tipos

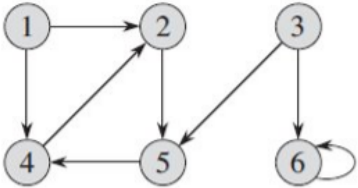

undirected
directed
unweighted
weighted
simple
non-simple
sparse $|E| \ll |V|^2$
dense
cyclic
acyclic
embedded
topological
explicit
implicit
unlabeled
labeled

Tipos

No dirigido

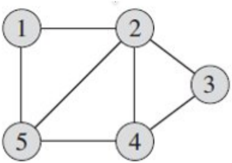


Dirigido

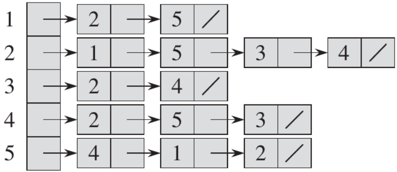


Representación

No dirigido



Lista de Adyacencia

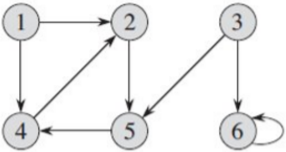


Matriz de Adyacencia

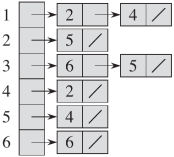
$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

Representación

Dirigido



Lista de Adyacencia



Matriz de Adyacencia

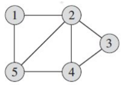
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

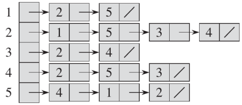
Lista de adyacencia

Consiste en un arreglo, *adj* , de $|V|$ listas

Para cada $u \in V$, $adj(u)$ contine todos los vertices v tales que existe un arco $(u, v) \in E$

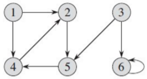
No Dirigido

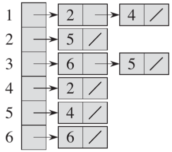




La suma de las longitudes de todas las listas es $2 * |E|$

Dirigido





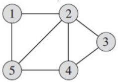
La suma de las longitudes de todas las listas es $|E|$

La cantidad de memoria es $\Phi(V + e)$

Matriz de adyacencia

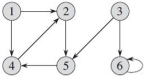
Consiste en una matriz a , de $|V|x|V|$ de $[0,1]$
$a(u,v) = 1$ si existe un arco $(u,v) \in E$
0 de lo contrario

No Dirigido



$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

Dirigido



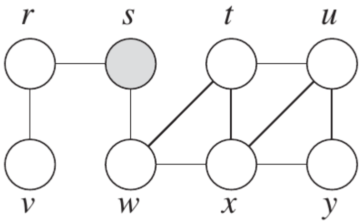
$$
\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}
$$

La cantidad de memoria es $\Phi(V * V)$

Busqueda por anchura

Dado un grafo $G = (V, E)$ y un vértice fuente denominado s, la **BFS** (Breadth-First Search) explora sistemáticamente los arcos de G para descubrir cada vértice que es alcanzable desde s.

Este algoritmo descubre primero todos los vértices a una distancia k desde s antes de descubrir los vértices a una distancia $k + 1$.



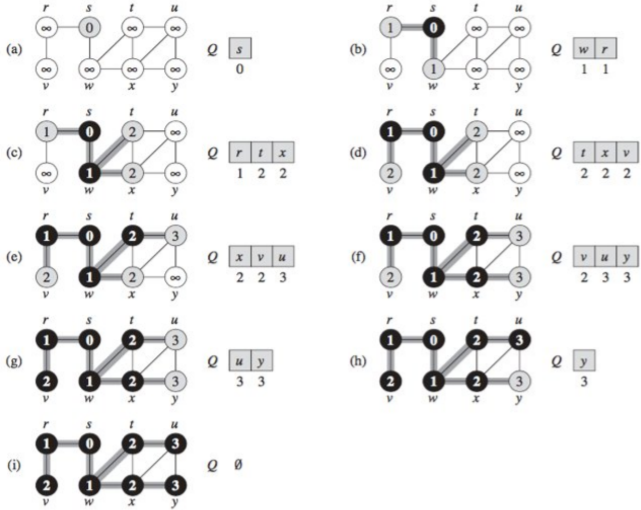
Busqueda por anchura

- Se asume una representación usando la lista de adyacencia.
- El algoritmo usa una cola Q para manejar el conjunto de vértices.
- Si u no tiene predecesor o u no ha sido descubierto entonces $u.\pi = NIL$.

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Busqueda por anchura

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



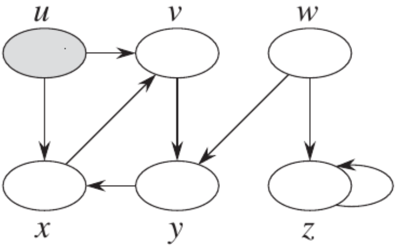
Busqueda por profundidad

La estrategia es buscar cada vez más profundo en el grafo siempre y cuando sea posible.

Su funcionamiento consiste en ir expandiendo los vértices que va localizando, de forma recurrente, en un camino concreto.

Cuando ya no quedan más nodos que visitar en dicho camino, regresa (backtracking), y repite el mismo proceso con cada uno de los hermanos del vértice ya procesado.

Si quedan vertices por exporar, se selecciona uno de ellos y se repite la búsqueda desde esa fuente.



Busqueda por profundidad

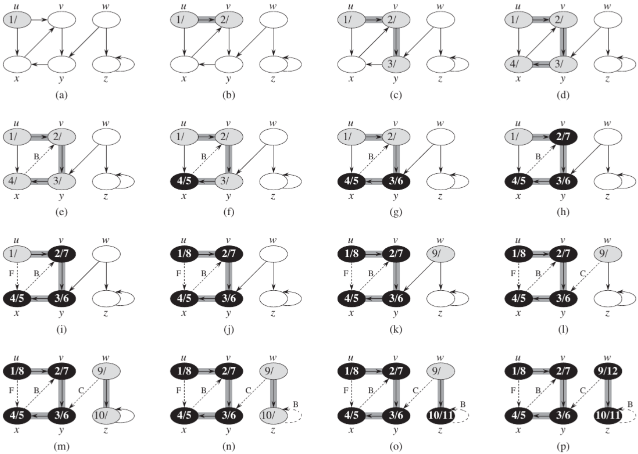
- Se asume una representación usando la lista de adyacencia.
- Los vértices se colorean para indicar su estado
- $v.\pi = u$ si v fue visitado al recorrer la lista de adjacencia del vértice visitado u
- Si v no tiene predecesor o u no ha sido descubierto entonces $u.\pi = NIL$.

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

Busqueda por profundidad

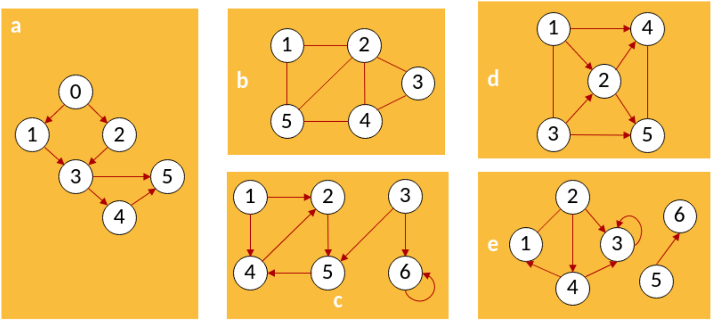
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
 time = time + 1
 u.d = time
 u.color = GRAY
 for each v ∈ G.Adj[u]
     if v.color == WHITE
         v.π = u
         DFS-VISIT(G, v)
 u.color = BLACK
 time = time + 1
 u.f = time
```



Agenda

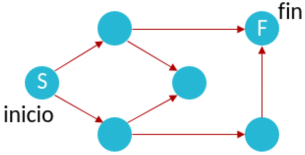
Ejercicios

Represente los siguientes grafos con una lista y con una matriz de adyacencias:

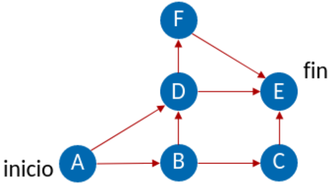


Ejercicios

Ejecute el algoritmo BFS en los siguientes grafos para obtener la respuesta solicitada:



Encuentre la longitud de la ruta más corta de inicio a fin



Encuentre la longitud de la ruta más corta de A a E

Elabore una lista del orden en que las siguientes actividades pueden realizarse (una tarea es dependiente de otra si existe un arco entre ellas)