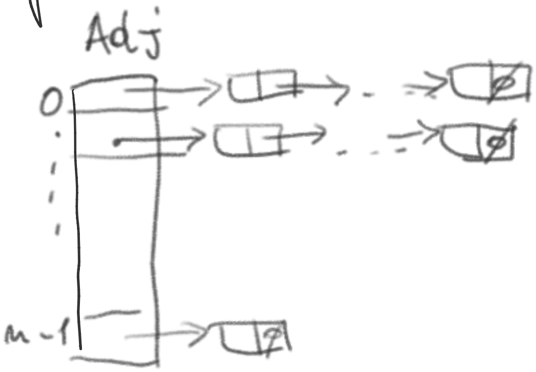
Lezione 7 maggio

Wednesday, May 6, 2020 11:38 AM

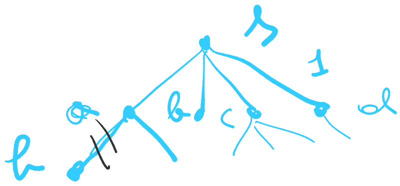
Visita BFS di G a partire dalla sorgente s.

G è rappresentato da Adj array di puntatori



$|V| = n \quad |E| = m$

$\Theta(n+m)$



La procedura BFS usa

- una coda semplice Q FIFO
- un array di bool. raggiunto[0..n-1]
- (un array di bool. inCoda[0..n-1])

```
BFS(s):
for (u=0; u<n; u++) {
    raggiunto[u] = false;
    inCoda[u] = false;
} Enqueue(Q, s);
  InCoda[s] = True;
  while (Q != empty) {
      u = Dequeue(Q);
```

inizializzazione

Spazio $\Theta(n)$

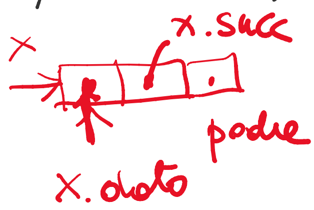
$l_0 + l_1 + \dots + l_{n-1} =$

$v_0 + v_1 + \dots + v_{n-1}$

$2m$

Visita

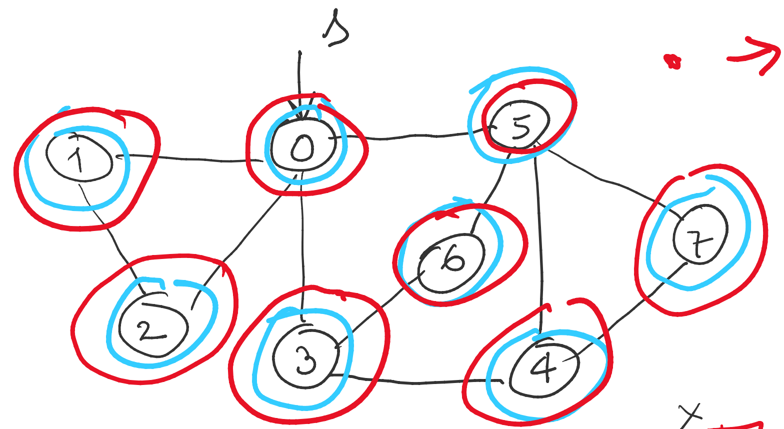
```
u = Dequeue(Q);
raggiunto[u] = True;
for(x = Adj[u]; x != NULL; x = x.succ){
    v = x.dato;
    if (!InCoda[v]){
        Enqueue(Q, v);
        InCoda[v] = true;
    } v.padre = u;
}
}
```



$\Theta(n+m)$

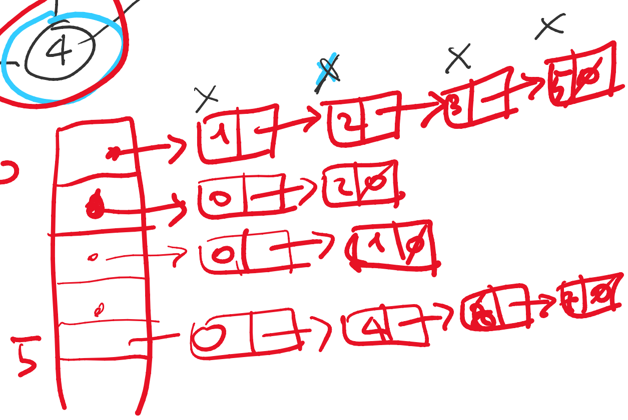
→ in coda

→ visitato



$Q = \{\cancel{0}, \cancel{1}, \cancel{2}, \cancel{3}, \cancel{5}, \cancel{4}, \cancel{6}, \cancel{7}\}$

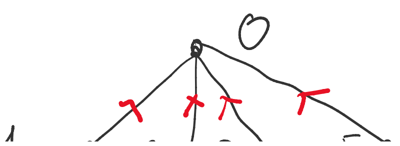
0, 1, 2, 3, 5, 4, 6, 7



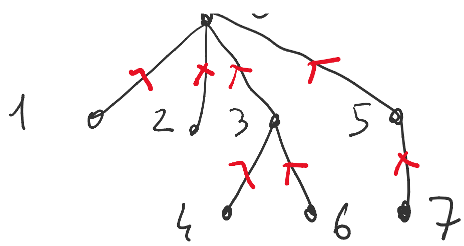
Visita BFS induce albero BFS

È uno "Spanning tree"

"albero di copertura"



BFS tree



BFS
tree

L'albero BFS dà per ogni nodo v il percorso minimo da v a s.

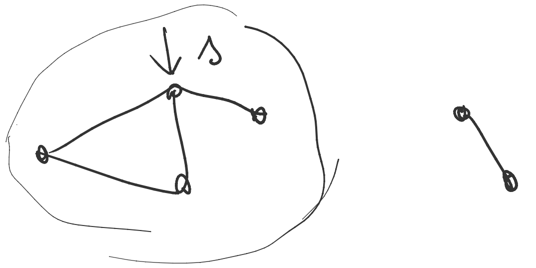
- modificare BFS per calcolare anche la distanza di ciascun v da s
- visitare gli archi di G in ordine BFS

Teo: la distanza minima da v a s in G è uguale alla profondità di v nell'albero BFS.

<u>Raggiungibilità</u> dei nodi

- grafi non orientati
- grafi orientati

G :



- G è connesso?

  tutti i nodi di G sono raggiungibili?

Calcolo del <u>diametro</u> di G

BFS da tutte le possibili sorgenti

$\Theta(n(n+m))$

G grafo dinamico

non esiste la lista di adiacenza

non si conoscono le etichette dei nodi

Adj(u) = funzioni che calcola tutti i link a altri nodi.

Dizionario D

coda Q

```
BFS-explore(s);
Enqueue(Q,s);
Inserisci(D,s);
while (Q!=empty){
    u = Dequeue(Q);
    if Appartiene(D,u)==-1 {
        Inserisci(D,u);
        for (x=Adj(u); x!=NULL; x=x.succ){
            v=x.dato;
            Enqueue(Q,v);
        }
    }
}
```

D è implementato con tabelle hash

D è implementato con tavole hash

BFS $\Theta(n+m)$ al caso medio

D è implementato con AVL

ricerca e ins. $O(\log n)$ caso pessimo

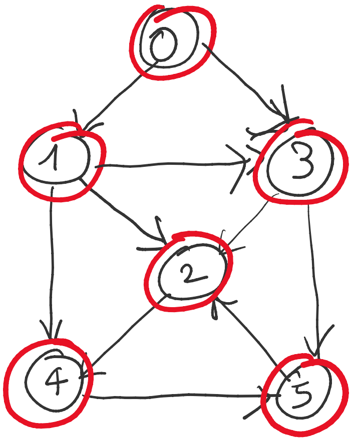
BFS $\Theta((n+m)\log n)$

Visita DFS

Depth First Search (alberi: visita anticipata)

Scansione (G):   Adj: lista di adiacenza
raggiunto

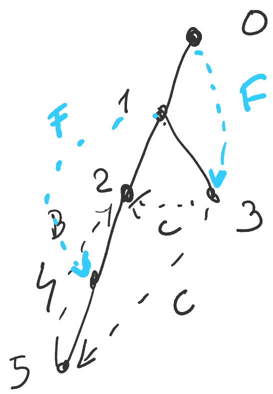
```
for (s=0; s<n; s++)
  raggiunto(s) = false;
for (s=0; s<n; s++)
  if (!raggiunto[s]) DFS(s);

DFS (u):
raggiunto [u] = true;
for (x = Adj[u]; x != NULL; x = x.succ) {
  v = x.dato;
  if (!raggiunto) DFS(v);
}
```



0, 1, 2, 4, 5, 3

DFS(0)
DFS(1)
DFS(2)
DFS(4)
DFS(5)
DFS(3)

Visita DFS induce un albero DFS

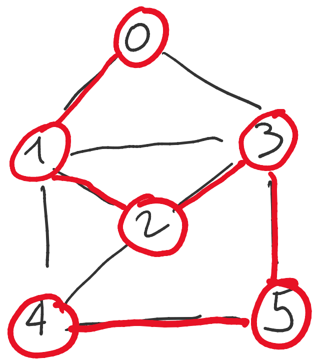


induce anche una classificazione degli archi di G

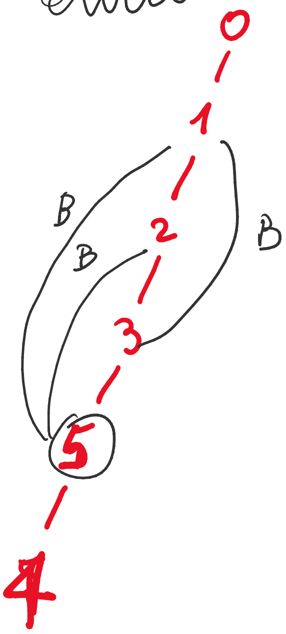
archi dell'albero A
archi all'indietro B — se connette un nodo con un antenato
archi attraversamento C
archi in avanti F — se connette un nodo con un discendente

Se il Grafo è non orientato possiedo
archi dell'albero e archi all'indietro

se il grafo è non orientato possiamo
soltanto archi dell'albero e archi all'indietro



albero DFS



DFS

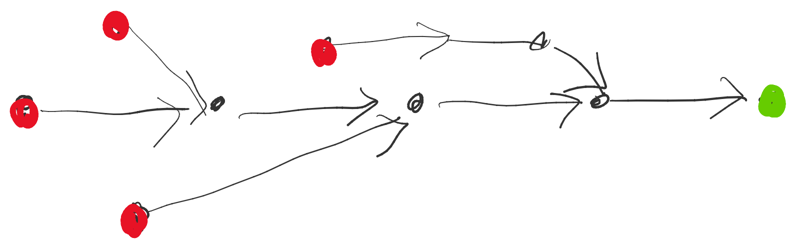
per G non orientato

G : DAG

Directed Acyclic Graph

Ordinamento topologico

DAG



Estendere DFS per classificare gli archi.