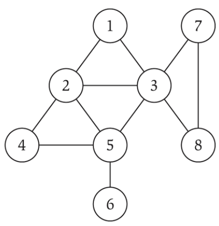
Visite di grafi

Connettività

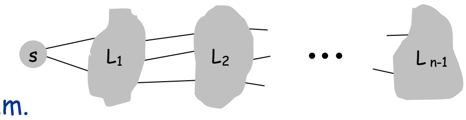
- Problema della connettività tra s e t. Dati due nodi s e t, esiste un percorso tra s e t?
- Problema del percorso più corto tra s e t. Dati due nodi s e t, qual è la lunghezza del percorso più corto tra s e t
- Applicazioni.
  - Attraversamento di un labirinto.
  - Erdős number.
  - Minimo numero di dispositivi che devono essere attraversati dai dati in una rete di comunicazione per andare dalla sorgente alla destinazione
  - Minimo numero di scali in un viaggio aereo



Breadth First Search (visita in ampiezza)

- BFS. Esplora il grafo a partire da una sorgente s muovendosi in tutte le possibile direzioni e visitando i nodi livello per livello (N.B.: il libro li chiama layer e cioè strati).

- I layer sono descritti di seguito

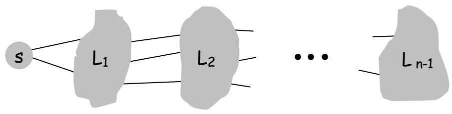


- BFS algorithm.
  - L_0 = { s }.
  - L_1 = tutti i vicini di s.
  - L_2 = tutti i nodi che non appartengono a L_0 o L_1, e che sono uniti da un arco ad un nodo in L_1.
  - L_{i+1} = tutti i nodi che non appartengono agli strati L_0 ,L_1,…,L_i e che sono uniti da un arco ad un nodo in L_i.

Breadth First Search

- distanza tra u e v= lunghezza del percorso piu` corto tra u e v

- Teorema. Per ogni i, L_i consiste di tutti i nodi a distanza i da s. Di conseguenza, c'è un percorso da s a t se e solo t appare in qualche livello.



L_1: livello dei nodi a distanza 1 da s
L_2: livello dei nodi a distanza 2 da s
…
L_{n-1}: livello dei nodi a distanza n-1 da s

Il teorema si puo` dimostrare in modo molto semplice usando l'induzione

Breadth First Search

- Teorema. Per ogni i, L_i consiste di tutti i nodi a distanza i da s. Di conseguenza, c'è un percorso da s a t se e solo t appare in L_i, per un certo $i \in \{0,1,\dots,j\}$.

Dim. per induzione sull'indice del layer:
Base induttiva: per j=0, la tesi è vera perche' $L_0=\{s\}$ e s è l'unico nodo a distanza 0 da se stesso
Passo induttivo: Supponiamo vera la tesi fino ad un certo j e dimostriamo che è vera per j+1. Assumiamo quindi che per i=0,1,…,j, il layer L_i consista di tutti i nodi a distanza i da s.
.
Per def. di L_{j+1}, un nodo u è in L_{j+1} se e solo se valgono le seguenti 1 e 2:

1. u non appartiene a $L_0, L_1, \dots, L_j$
2. u è unito da un arco ad un nodo di L_j

un nodo u soddisfa la 1 e la 2 ←→ la distanza di u da s è j+1
(ricordiamo che stiamo assumendo l'ipotesi induttiva)

dimostrazione nella prossima slide

Breadth First Search

un nodo u soddisfa sia la 1 che la 2 ←→ la distanza di u da s è j+1

dim.

- è immediato vedere che vale →
- vediamo perche':
- per la 2 esiste un arco da un certo nodo z di L_j ad u e siccome per ipotesi induttiva, z è a distanza j da s allora esiste un percorso di lunghezza j+1 da s ad u
- per la 1 il vertice u non è in $L_0 \cup L_1 \cup \dots \cup L_j$ e di conseguenza u è a distanza > j da s (infatti per ipotesi induttiva tutti i nodi a distanza ≤j da s sono in $L_0 \cup L_1 \cup \dots \cup L_j$)

- le affermazioni in rosso implicano che esiste un percorso da s ad u di j+1 archi e questo è il piu` corto possibile → la distanza da u ad s è j+1

Breadth First Search

- dimostriamo che vale ←
- Supponiamo che u sia a distanza j+1 da s e dimostriamo che valgono la 1 e la 2.
- Per ipotesi induttiva la 1 deve essere necessariamente soddisfatta (infatti per ipotesi induttiva $L_0,L_1, \ldots ,L_j$ contengono solo nodi a distanza ≤ j da s)
- Dimostriamo che vale anche la 2. Sia P un percorso di j+1 archi da s ad u. Questo percorso esiste dal momento che la distanza da s di u è proprio j+1. Indichiamo con v il predecessore di u lungo P (P termina con l'arco (v,u)).
- Il sottopercorso P' di P che arriva fino a v ha lunghezza j e di conseguenza la distanza di v da s è ≤ j.
- Facciamo vedere che la distanza di v da s è esattamente j. Se per assurdo questa distanza fosse <j allora esisterebbe un percorso P'' da s a v di lunghezza <j e il percorso da s ad u ottenuto concatenando a P'' l'arco (v,u) avrebbe lunghezza <j+1 contraddicendo l'ipotesi che u fosse a distanza j+1 da s.
- Siccome la distanza di v da s è esattamente j allora per ipotesi induttiva $v \in L_j$ e di conseguenza l'arco (v,u) unisce u ad un nodo in L_j → u soddisfa la 2.

Breadth First Search

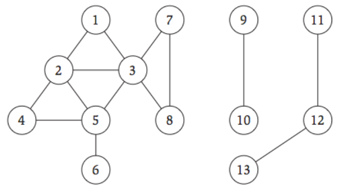
Pseudocodice (schema dell'algoritmo)

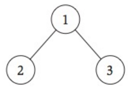
```
BFS(s)
L0={ s }
For(i=0;i≤n-2;i++)
  Li+1= Ø ;
  Foreach nodo u in Li
    Foreach nodo v adiacente ad u
      if( v non appartiene ad L0,…,Li+1)
        Li+1=Li+1 ∪ { v }
      EndIf
    Endforeach
  Endforeach
Endfor
```

- Occorre un modo per capire se un nodo è gia stato visitato in precedenza. Il tempo di esecuzione dipende dal modo scelto, da come è implementato il grafo e da come sono rappresentati gli insiemi L_i che rappresentano i livelli

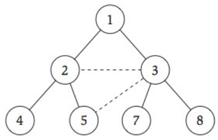
Esempio di esecuzione di BFS

G

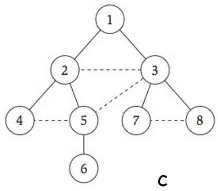




a



b



c

L_0={1}
a. L_1={2,3}
b. L_2={4,5,7,8}
c. L_3={6}

Breadth First Search Tree (Albero BFS)

- Proprietà. L'algoritmo BFS produce un albero che ha come radice la sorgente s e come nodi tutti i nodi del grafo raggiungibili da s.

- L'albero si ottiene in questo modo:
- Consideriamo il momento in cui un vertice v viene scoperto, cioè il momento in cui visitato per la prima volta.
  - Ciò avviene durante l'esame dei vertici adiacenti ad un un certo vertice u di un certo livello L_i (linea 6).
  - In questo momento, oltre ad aggiungere v al livello L_{i+1} (linea 8), aggiungiamo l'arco (u,v) e il nodo v all'albero

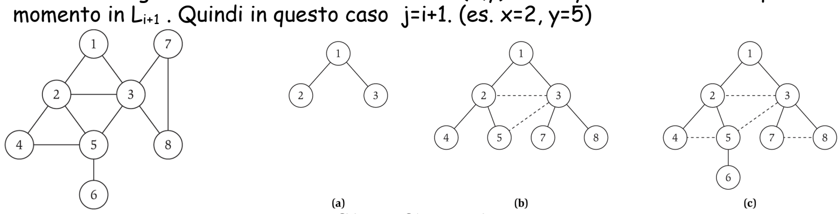
Breadth First Search Tree

- Proprietà. Si consideri un'esecuzione di BFS su $G = (V, E)$, e sia (x, y) un arco di G. I livelli di x e y differiscono di al più di 1.

- Dim. Sia L_i il livello di x ed L_j quello di y. Supponiamo senza perdere di generalità che x venga scoperto prima di y cioè che **$i \le j$**. Consideriamo il momento in cui l'algoritmo esamina gli archi incidenti su x.

- *Caso 1. Il nodo y è stato già scoperto:*
  Siccome per ipotesi y viene scoperto dopo x allora sicuramente y viene inserito
  a) o nel livello i dopo x, se y è adiacente a qualche nodo nel livello i-1 (es. x=2, y=3).
  b) o nel livello i+1, se è adiacente a qualche nodo del livello i esaminato nel **For each** alla linea 5 prima di x). (es. x=3,y=5)
  Quindi in questo caso si ha j= i oppure j=i+1.

- *Caso 2. Il nodo y non è stato ancora scoperto:*
  Siccome tra gli archi incidenti su x c'è anche (x,y) allora y viene inserito in questo momento in L_{i+1} . Quindi in questo caso j=i+1. (es. x=2, y=5)



G (a) (b) (c)

35

Implementazione di BFS con liste di adiacenza e array Discovered

• Ciascun insieme L_i è rappresentato da una lista L[i]
• Usiamo un array di valori booleani Discovered per associare a ciascun nodo il valore vero o falso a seconda che sia già stato scoperto o meno
• Durante l'algoritmo costruiamo anche l'albero BFS

```
1.  BFS(s):
2.  Poni Discovered[s] = true e Discovered[v] = false per tutti gli altri v
3.  Inizializza L[0] in modo che contenga solo s
4.  Poni il contatore dei livelli i = 0
5.  Inizializza il BFS tree T con un albero vuoto
6.  While i<=n-2                        //L[i] è vuota se non ci sono
7.      Inizializza L[i+1] con una lista vuota   //nodi raggiungibili da L[i-1]
8.      Foreach u ∈ L[i]
9.        Foreach arco (u, v) incidente su u
10.         If Discovered[v] = false
11.           Poni Discovered[v] = true
12.           Aggiungi v alla lista L[i+1]
13.           Aggiungi l'arco (u, v) all'albero T
14.         Endif
15.       Endfor
16.     EndFor
17.     i=i+1
18. Endwhile
```


36

Implementazione di BFS per grafo implementato con liste di adiacenza

```
1.  BFS(s):
2.  Poni Discovered[s] = true e Discovered[v] = false per tutti gli altri v      O(n)
3.  Inizializza L[0] in modo che contenga solo s      ┐
4.  Poni il contatore dei livelli i = 0               ├ O(1)
5.  Inizializza il  BFS tree T con un albero vuoto    ┘
6.  While i<=n-2                                 n volte      ┐
7.      Inizializza L[i+1] con una lista vuota   n-1 volte    ┘ O(n)
8.      Foreach u ∈ L[i]   Sul totale di tutte le iterazioni del while, al più n volte
9.        Foreach arco (u, v) incidente su u    Sul totale di tutte le iterazioni del While al più 2m volte.   ┐
10.         If Discovered[v] = false then                                                               │
11.           Poni Discovered[v] = true                                                                 ├ O(m)
12.            Aggiungi v alla lista L[i+1]                                                             │
13.            Aggiungi l'arco (u, v) all'albero T                                                      ┘
14.         Endif
15.       Endfor
16.     Endfor
17.     i=i+1
18. Endwhile
```

Foreach più esterno viene eseguito al più n volte in quanto ogni nodo raggiungibile da s appartiene ad una sola lista L_i
Foreach più interno viene eseguito al piu` $\sum_{u \in V} deg(u) \leq 2m$ volte

Algoritmo è O(n+m)

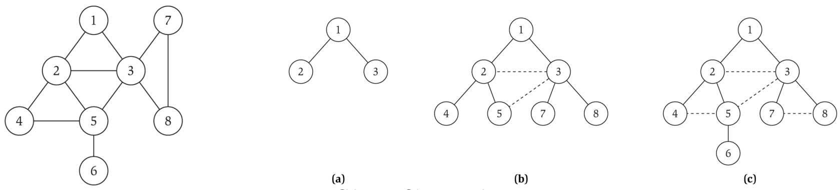
Breadth First Search Tree

- Proprietà. Si consideri un'esecuzione di BFS su $G = (V, E)$, e sia (x, y) un arco di G. I livelli di x e y differiscono di al più di 1.

- Dim. Sia L_i il livello di x ed L_j quello di y. Supponiamo senza perdere di generalità che x venga scoperto prima di y cioè che **$i \leq j$**. Consideriamo il momento in cui l'algoritmo esamina gli archi incidenti su x.

- Caso 1. Il nodo y è stato già scoperto:
  Siccome per ipotesi y viene scoperto dopo x allora sicuramente y viene inserito
  a) o nel livello i dopo x, se y è adiacente a qualche nodo nel livello i-1 (es. x=2, y=3).
  b) o nel livello i+1, se è adiacente a qualche nodo del livello i esaminato nel **For each** alla linea 5 prima di x). Quindi in questo caso si ha j= i oppure j=i+1. (es. x=3,y=5)

- Caso 2. Il nodo y non è stato ancora scoperto:
  Siccome tra gli archi incidenti su x c'è anche (x,y) allora y viene inserito in questo momento in L_{i+1} . Quindi in questo caso j=i+1. (es. x=2, y=5)



G (a) (b) (c)

Implementazione di BFS con coda FIFO

L'algoritmo BFS si presta ad essere implementato con un coda
Ogni volta che viene scoperto un nodo u, il nodo u viene inserito nella coda
Vengono esaminati gli archi incidenti sul nodo al front della coda

```
BFS(s)
Inizializza Q con una coda vuota
Inizializza il  BFS tree T con un albero vuoto
Poni Discovered[s] = true e Discovered[v] = false per tutti gli altri v
Inserisci s in coda a Q con una enqueue
While(Q non è vuota)
   estrai il front di Q con una deque e ponilo in u
   Foreach arco (u,v) incidente su u
    If(Discovered[v]=false)
        poni Discovered[v]= true
        aggiungi v in coda a Q con una enqueue
        aggiungi (u,v) al BFS tree T
  Endif
 Endfor
Endwhile
```

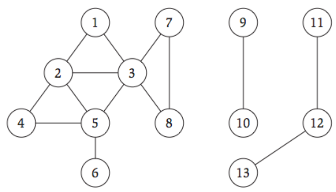
Dimostrare per esercizio che il tempo di esecuzione è O(n+m)
(svolto in classe)

39

39

Esempio di esecuzione di BFS con coda FIFO

G



elementi della coda durante l'esecuzione (front = elemento più a sinistra)

all'inizio Q= 1
dopo I iterazione del while Q= 2 3
dopo II iterazione del while Q = 3 4 5
dopo III iterazione del while Q = 4 5 7 8
dopo IV iterazione del while Q = 5 7 8
dopo V iterazione del while Q = 7 8 6
dopo VI iterazione del while Q = 8 6
dopo VII iterazione del while Q = 6
dopo VIII iterazione del while Q è vuota

- Viene estratta la sorgente e vengono inseriti i nodi del livello 1.
- Poi man mano vengono estratti i nodi del livello 1 ed inseriti quelli del livello 2.
- Quando non ci sono piu` elementi del livello 1, cominciano ad essere estratti i nodi del livello 2 e via via inseriti quelli del livello 3 e così via.

40

40