Grafi (II parte)

Progettazione di Algoritmi a.a. 2019-20

Matricole congrue a 1

Docente: Annalisa De Bonis

1

1

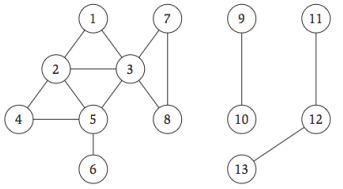
Breadth First Search

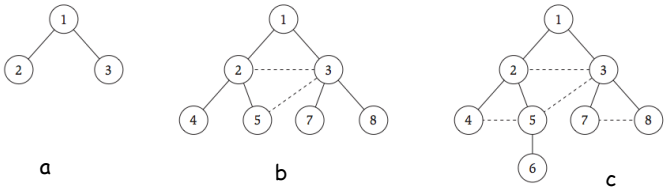
1. BFS(s)
2. L_0={ s }
3 For(i=0;i≤n-2;i++)
4. L_{i+1}= Ø ;
5. Foreach nodo u in L_i
6. Foreach nodo v adiacente ad u
7. if(v non appartiene ad $L_0,\ldots,L_{i+1}$)
8. $L_{i+1}=L_{i+1}\cup$ { v }
9. EndIf
10. Endforeach
11. Endforeach
12. Endfor

2

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}

3

3

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 è 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

4

Breadth First Search Tree (Albero BFS)

- Def. Profondità di un nodo u di un albero T
  - 0 se u è la radice di T
  - 1 + profondità del padre di u, altrimenti

- Proprietà. Si consideri un'esecuzione di BFS su $G = (V, E)$. I nodi inseriti nel livello Li hanno profondità i nell'albero BFS

5

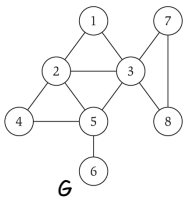
Breadth First Search Tree

- 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. Cio` implica che le profondità di x e y nell'albero BFS differiscono al più di 1.

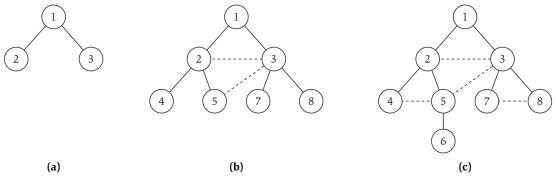
- Dim. Sia L_i il livello di x ed L_j quello di y. Supponiamo senza perdere di generalità che x venga scoperto prima di y cioè che i≤j. Consideriamo il momento in cui l'algoritmo esamina gli archi incidenti su x.

  - Caso 1. Il nodo y è stato già scoperto:
    Siccome per ipotesi y viene scoperto dopo x allora sicuramente y viene inserito o nel livello i dopo x (se y è adiacente a qualche nodo nel livello i-1) o nel livello i+1 (se 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.

  - 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.



G



(a) (b) (c)

6

Implementazione di BFS con tempo $O(n+m)$

- Grafo rappresentato con liste di adiacenza
- Ciascun insieme L_i è rappresentato da una lista L[i]
- Usiamo un array Discovered di valori booleani 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

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

7

7

Implementazione di BFS con tempo $O(n+m)$

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

8-14 eseguite, sul totale di tutte le iterazioni del While, al più 2m volte.

O(m)

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

Algoritmo è $O(n+m)$

8

8

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)
1.  Inizializza Q con una coda vuota
2.  Inizializza il  BFS tree T con un albero vuoto
3.  Poni Discovered[s] = true e Discovered[v] = false per tutti gli altri v
4.  Inserisci s in coda a Q con una enqueue
5.  While(Q non è vuota)
6.     estrai il front di Q con una deque e ponilo in u
7.     Foreach arco (u,v) incidente su u
8.      If(Discovered[v]=false)
9.         poni Discovered[v]= true
10.        aggiungi v in coda a Q con una enqueue
11.        aggiungi (u,v) al BFS tree T
12.   Endif
13.  Endfor
14.  Endwhile
```

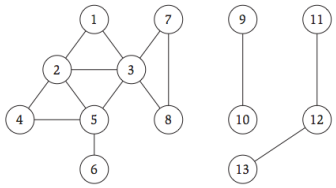
Dimostrare per esercizio che il tempo di esecuzione è $O(n+m)$ (svolto in classe)

9

9

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

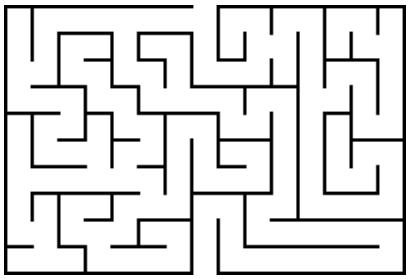
- Viene estratta la sorgente e vengono inseriti i nodi del livello 1.
- Poi man mano vengono estratti i nodi del livello 1 ed inseriti quelli del livello 2.
- Quando non ci sono piu` elementi del livello 1, cominciano ad essere estratti i nodi del livello 2 e via via inseriti quelli del livello 3 e così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

10

10

Depth first search (visita in profondità)

- La visita in profondità riproduce il comportamento di una persona che esplora un labirinto di camere interconnesse
  - La persona parte dalla prima camera (nodo s) e si sposta in una delle camere accessibili dalla prima (nodo adiacente ad s), di lì si sposta in una delle camere accessibili dalla seconda camera visitate e così via fino a quando raggiunge una camera da cui non è possibile accedere a nessuna altra camera non ancora visitata. A questo punto torna nella camera precedentemente visitata e di lì prova a raggiungere nuove camere.



11

11

Depth first search (visita in profondità)

- La visita DFS parte dalla sorgente s e si spinge in profondità fino a che non è più possibile raggiungere nuovi nodi.
  - La visita parte da s, segue uno degli archi uscenti da s ed esplora il vertice v a cui porta l'arco.
  - Una volta in v, se c'è un arco uscente da v che porta in un vertice w non ancora esplorato allora l'algoritmo esplora w
  - Uno volta in w segue uno degli archi uscenti da w e così via fino a che non arriva in un nodo del quale sono già stati esplorati tutti i vicini.
  - A questo punto l'algoritmo fa backtrack (torna indietro) fino a che torna in un vertice a partire dal quale può visitare un vertice non ancora esplorato in precedenza.

12

Depth first search: pseudocodice

```
DFS(u):
  Mark u as "Explored" and add u to R
  For each edge (u,v) incident to u
    If v is not marked "Explored" then
      Recursively invoke DFS(v)
    Endif
  Endfor
```

R = insieme dei vertici raggiunti

Analisi: (assumendo G rappresentato con liste di adiacenza)

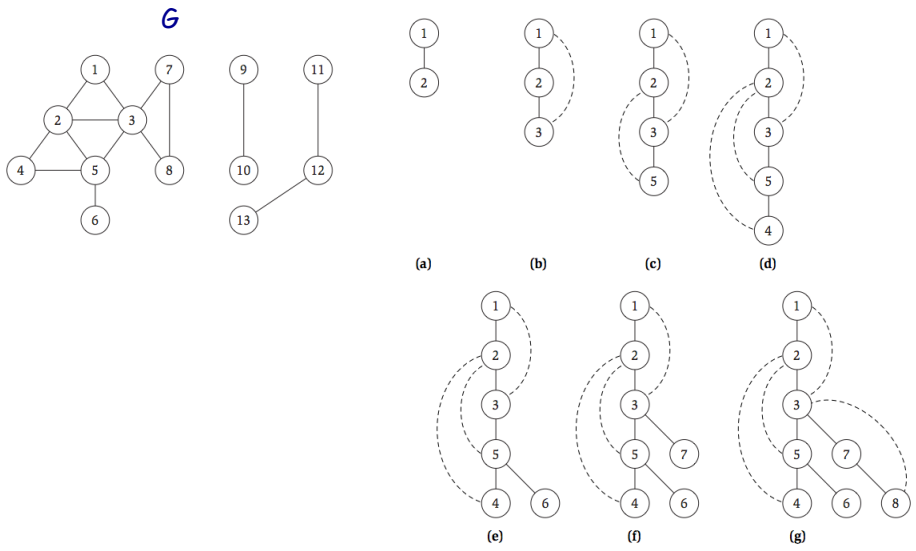
- Se ignoriamo il tempo delle chiamate ricorsive al suo interno, ciascuna visita ricorsiva richiede tempo O(1+deg(u)) : O(1) per marcare u e aggiungerlo ad R e O(deg(u)) per eseguire il for.
- Se inizialmente invochiamo DFS su un nodo s, allora DFS viene invocata ricorsivamente su tutti i nodi raggiungibili a partire da s. Il costo totale è quindi al più

$$\sum_{u \in V} O(1 + deg(u)) = O(\sum_{u \in V} 1 + \sum_{u \in V} deg(u)) = O(n + m)$$

13

13

Esempio di esecuzione di DFS



14

14

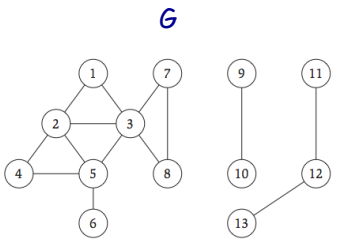
Depth First Search Tree (Albero DFS)

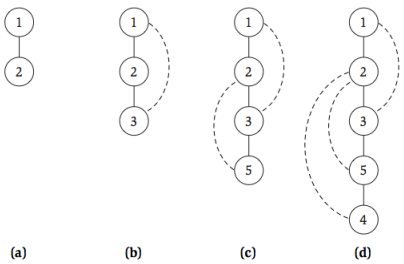
- Proprietà. L'algoritmo DFS produce un albero che ha come radice la sorgente s e come nodi tutti i nodi del grafo raggiungibili da s.

- L'albero si ottiene in questo modo:
  - Consideriamo il momento in cui viene invocata DFS(v)
  - Ciò avviene durante l'esecuzione di DFS(u) per un certo nodo u. In particolare durante l'esame dell'arco (u,v) nella chiamata DFS(u).
  - In questo momento, aggiungiamo l'arco (u,v) e il nodo v all'albero

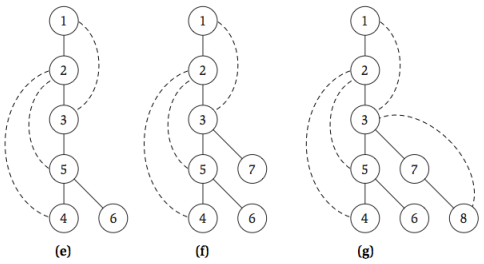
15

Esempio di albero DFS





Gli archi non tratteggiati fanno parte del DFS tree.



16

Albero DFS

- Proprietà 1. Per una data chiamata ricorsiva DFS(u), tutti i nodi che vengono etichettati come "Esplorati" tra l'inizio e la fine della chiamata DFS(u), sono discendenti di u nell'albero DFS.

- Proprietà 2. Sia T un albero DFS e siano x e y due nodi di T collegati dall'arco (x,y) in G. Si ha che x e y sono l'uno antenato dell'altro in T.

Dim. Proprietà 2

- **Caso (x,y) e` in T.** In questo caso la proprietà e` ovviamente soddisfatta.
- **Caso (x,y) non e` in T.** Supponiamo senza perdere di generalità che DFS(x) venga invocata prima di DFS(y). Ciò vuol dire che quando viene invocata DFS(x), y non è ancora etichettato come "Esplorato".
- La chiamata DFS(x) esamina l'arco (x,y) e per ipotesi non inserisce (x,y) in T. Ciò si verifica solo se y è già stato etichettato come "Esplorato". Siccome y non era etichettato come "Esplorato" all'inizio di DFS(x) vuol dire è stato esplorato tra l'inizio e la fine della chiamata DFS(x). La proprietà 1 implica che y è discendente di x.