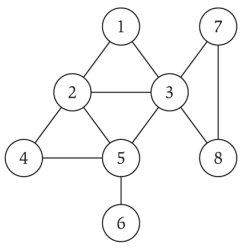
Grafi (I parte)

Progettazione di Algoritmi a.a. 2020-21
Matricole congrue a 1
Docente: Annalisa De Bonis

Grafi non direzionati

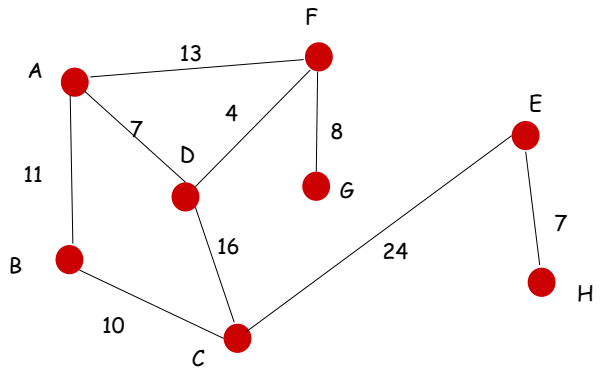
- Grafi non direzionati. $G = (V, E)$
  - V = insieme nodi.
  - E = insieme archi.
  - Esprime le relazioni tra coppie di oggetti.
  - Parametri del grafo: n = |V|, m = |E|.



V = { 1, 2, 3, 4, 5, 6, 7, 8 }
E = { 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6,7-8 }
n = 8
m = 11

Esempio di applicazione

- Archi: strade (a doppio senso di circolazione)
- Nodi: intersezioni tra strade
- Pesi archi: lunghezza in km

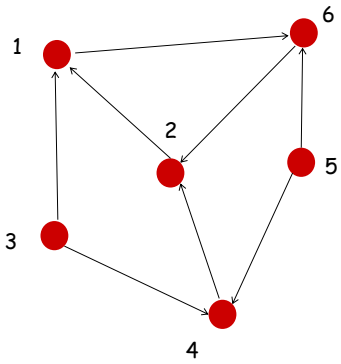


3

3

Grafi direzionati

- Gli archi hanno una direzione
  - L'arco (u,v) è diverso dall'arco (v,u)
  - Si dice che l'arco e=(u,v) lascia u ed entra in v
  - e che u è l'origine dell'arco e v la destinazione dell'arco

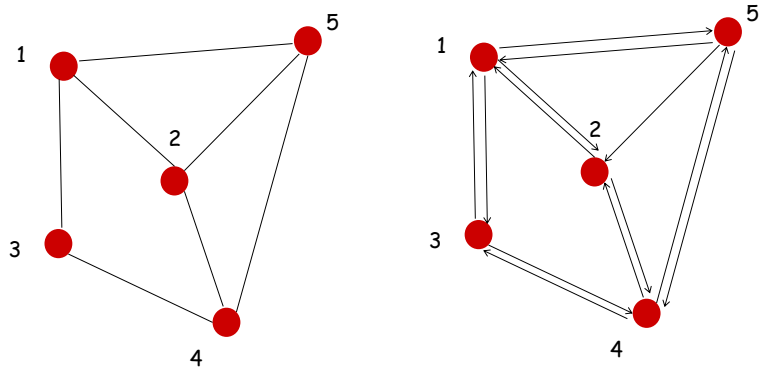


V = { 1, 2, 3, 4, 5, 6 }
E = {1-6, 2-1, 3-1, 3-4, 4-2, 5-4, 5-6, 6-2 }
n = 6
m = 8

4

Grafi direzionati

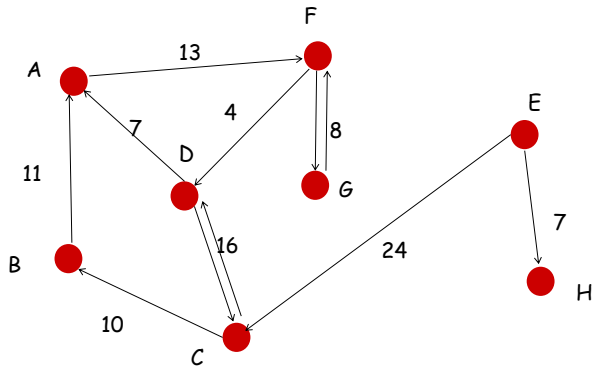
- Grafi non direzionati G = (V, E) possono essere visti come un caso particolare degli archi direzionati in cui per ogni arco (u,v) c'è l'arco di direzione opposta (v,u)



5

Esempio di applicazione

- Archi: strade (a senso unico di circolazione)
- Nodi: intersezioni tra strade
- Pesi archi: lunghezza in km



6

Alcune applicazione dei grafi

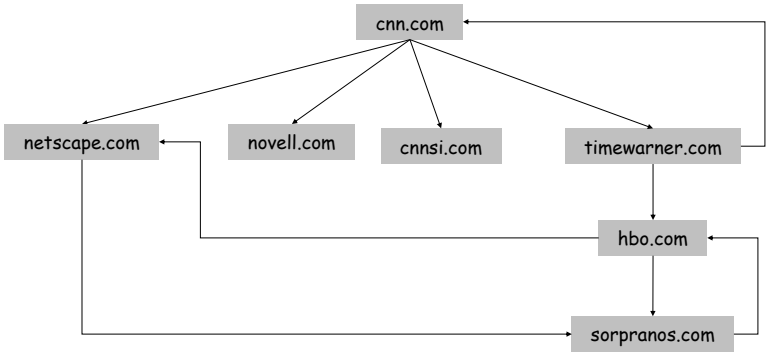
- Rete di amicizia su un social network: ogni utente è un nodo; ogni volta che due utenti diventano amici, si crea un arco del grafo.
- Google maps: i nodi rappresentano città, intersezioni di strade, siti di interesse, ecc. e gli archi rappresentano le connessioni dirette tra i nodi.
  - La rappresentazione mediante un grafo permette di trovare il percorso più corto per andare da un posto all'altro mediante un algoritmo.
- World Wide Web: le pagine web sono i nodi e il link tra due pagine è un arco. Google utilizza questa rappresentazione per esplorare il World Wide Web

7

7

World Wide Web

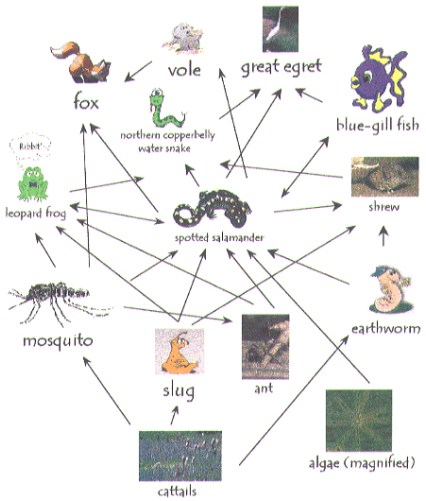
- Web graph.
  - Nodo: pagina web.
  - Edge: hyperlink da una pagina all'altra.



8

Ecological Food Web

- Food web graph.
  - Nodo = specie
  - Arco dalla preda al predatore.



Reference: http://www.twingroves.dètrict96.k12.il.us/Wetlands/Salamander/SalGraphics/salfoodweb.giff

9

Alcune applicazioni dei grafi

Graph	*Nodi*	*Archi*
trasporto	intersezioni di strade	strade
trasporto	aeroporti	voli diretti
comunicazione	computer	cavi di fibra ottica
World Wide Web	web page	hyperlink
rete sociale	persone	relazioni
catena del cibo	specie	predatore-preda
scheduling	task	vincoli di precedenza
circuiti	gate	wire

10

Terminologia

- Consideriamo due nodi u e v di un grafo G connessi dall'arco e = (u,v)

- Si dice che
  - u e v sono adiacenti
  - u e v sono le estremità dell'arco (u,v)
  - l'arco (u,v) incide sui vertici u e v
  - u è un nodo vicino di v
  - v è un nodo vicino di u

- Dato un vertice u di un grafo G
  - grado di u = numero archi incidenti su u
    - è indicato con deg(u)

11

11

Numero di archi di un grafo non direzionato

m = numero di archi di G;

n= numero di nodi di G .

1. La somma di tutti i gradi dei nodi di G è 2m : $\sum_{u \in V} deg(u) = 2m$

Degree (grado)= numero di vicini di u

Dim. Ciascun arco incide su due vertici e quindi viene contato due volte nella sommatoria in alto. L'arco (x,y) è contato sia in deg(x) che in deg(y).

2. Il numero m di archi di un grafo G non direzionato è al più n(n-1)/2 .

Dim. Il numero di coppie **non ordinate** distinte che si possono formare con n nodi è n(n-1)/2 .
Posso scegliere il primo nodo dell'arco in n modi e il secondo in modo che sia diverso dal primo nodo, cioè in n-1 modi. Dimezzo in quanto l'arco (u,v) è uguale all'arco (v,u)

12

12

Numero di archi di un grafo direzionato

m = numero di archi di G;
n= numero di nodi di G

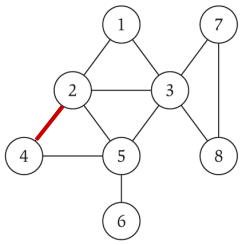
Il numero m di archi di G è al più n^2

Dim. Il numero di coppie **ordinate** distinte che si possono formare con n nodi è n^2 . Posso scegliere il primo nodo dell'arco in n modi e il secondo in altri n modi (se ammettiamo archi con entrambe le estremità uguali).

13

Graph Representation: Adiacenza Matrix

- Matrice di adiacenza. Matrice nxn con A_{uv} = 1 se (u, v) è an arco.
  - Due rappresentazioni di ciascun arco.
  - Spazio proporizionale a n^2.
  - Controllare se (u, v) è un arco richiede tempo $\Theta(1)$.
  - Identificare tutti gli archi richiede tempo $\Theta(n^2)$.

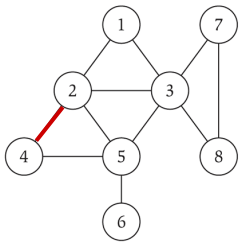


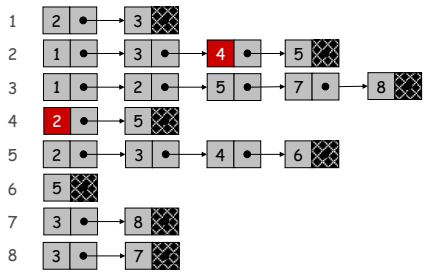
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

14

Rappresentazione di un grafo: liste di adiacenza

- Liste di adiacenza. Array di liste in cui ogni lista è associata ad un nodo.
  - Ad ogni arco corrisponde un elemento della lista.
  - Se esiste l'arco (u,v) allora la lista associata ad u contiene v
  - In un grafo non direzionato l'arco (u,v) corrisponde ad un elemento della lista associata ad u e ad un elemento della lista associata a v
  - Spazio proporzionale a m + n.
  - Controllare se (u, v) è un arco richiede tempo O(deg(u)).
  - Individuare tutti gli archi richiede tempo Θ(m + n).

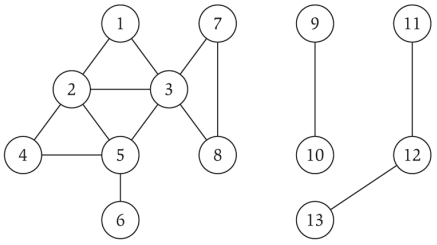




15

Percorsi e connettività

- Def. Un percorso in un grafo non direzionato G = (V, E) è una sequenza P di nodi v_1, v_2, …, v_{k-1}, v_k con la proprietà che ciascuna coppia di vertici consecutivi v_i, v_{i+1} è unita da un arco in E.
- Def. Un percorso è semplice se tutti i nodi sono distinti.
- Def. Un grafo non direzionato è connesso se per ogni coppia di nodi u e v, esiste un percorso tra u e v.

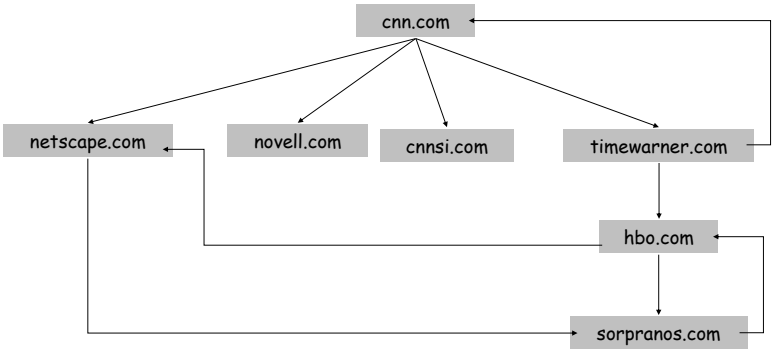


16

Applicazione del concetto di percorso

- **Esempi:**

Web graph. Voglio capire se è possibile, partendo da una pagina web e seguendo gli hyperlink nelle pagine via via attraversate, arrivare ad una determinata pagina



17

Applicazione del concetto di percorso

In alcuni casi può essere interessante scoprire il percorso più corto, cioe` composto dal minimo numero di archi, tra due nodi.

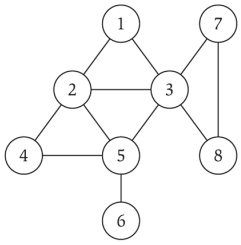
Esempio:

- Grafo : rete di trasporti dove i nodi sono gli aeroporti e gli archi i collegamenti diretti tra aeroporti.
- Voglio arrivare da Napoli a New York facendo il minimo numero di scali.

18

Cicli

- Def. Un ciclo è un percorso $v_1,v_2, \ldots,v_{k-1},v_k$ in cui $v_1 = v_k$, $k > 2$.
- Def. Un ciclo $v_1,v_2, \ldots,v_{k-1},v_1$ è semplice se i primi k-1 nodi del ciclo sono tutti distinti tra di loro



ciclo (semplice) C = 1-2-4-5-3-1

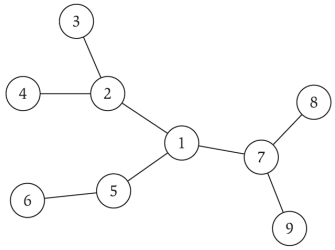
ciclo (non semplice) C' = 1- 3 -7 -8- 3 - 5 - 2- 1

19

Alberi

- Def. Un grafo non direzionato è un albero (tree) se è connesso e non contiene cicli
- Teorema. Sia G un grafo non direzionato con n nodi. Ogni due delle seguenti affermazioni implica la restante affermazione.
  - 1 e 2 ⟹ 3 ; 1 e 3 ⟹ 2 ; 2 e 3 ⟹ 1

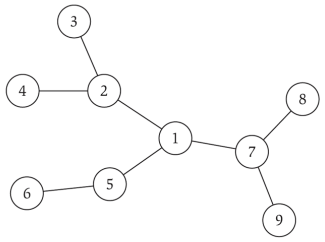
1. G è connesso.
2. G non contiene cicli
3. G ha n-1 archi.



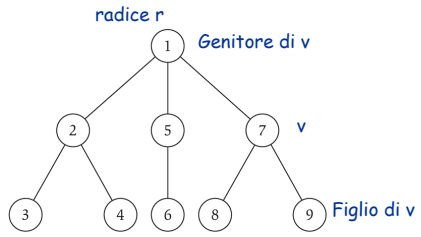
20

Alberi con radice

- Albero con radice. Dato un albero T, si sceglie un nodo radice r e si considerano gli archi di T come orientati a partire da r
- Dato un nodo v di T si dice
  - Genitore di v: il nodo che w precede v lungo il percorso da r a v (v viene detto figlio di w)
  - Antenato di v: un qualsiasi nodo w lungo il percorso che va da r a v (v viene detto discendente di w)
- Foglia: nodo senza discendenti



Un albero

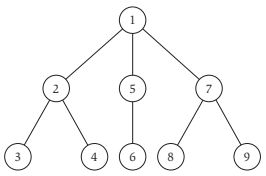


Lo stesso albero con radice 1

21

Alberi con radice

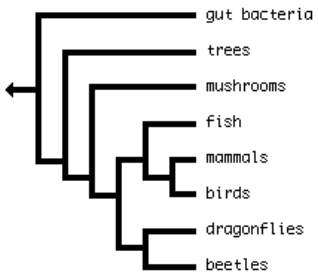
- Scegliere un nodo come radice, rende più semplice dimostrare la seguente affermazione
- se G è connesso e non contiene cicli, in altre parole, se G è un albero allora il numero di archi è n-1 (dove n è il numero di nodi).
- Dim.
- Per ogni nodo diverso dalla radice c'e` un arco distinto che lo connette al proprio padre →numero di archi ≥ n-1
- Per ogni arco c'è un nodo distinto (non radice) che è congiunto al padre da quell'arco → numero di archi ≤ n-1
- le due diseguaglianze → numero di archi= n-1



22

Importanza degli alberi: rappresentano strutture gerarchiche

- Alberi filogenetici. Descrivono la storia evolutiva delle specie animali.

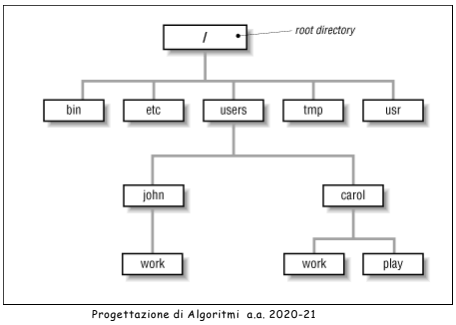


La filogenesi afferma l'esistenza di una specie ancestrale che diede origine a mammiferi e uccelli ma non alle altre specie rappresentate nell'albero (cioè, mammiferi e uccelli condividono un antenato che non è comune ad altre specie nell'albero). La filogenesi afferma inoltre che tutti gli animali discendono da un antenato non condiviso con i funghi, gli alberi e i batteri, e così via.

23

Importanza degli alberi: rappresentano strutture gerarchiche

- File system. Un file system tipicamente consiste di file organizzati in gruppi chiamati directory.
  - Una directory può contenere file e altre directory,
  - Un file system gerarchico è organizzato secondo una struttura gerarchica ad albero con radice
    - nodi interni: directory
    - foglie: file

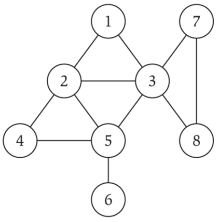


24

Visite di grafi

Connettività

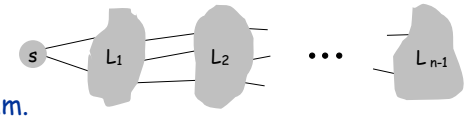
- Problema della connettività tra s e t. Dati due nodi s e t, esiste un percorso tra s e t?
- Problema del percorso più corto tra s e t. Dati due nodi s e t, qual è la lunghezza del percorso più corto tra s e t
- Applicazioni.
  - Attraversamento di un labirinto.
  - Erdős number.
  - Minimo numero di 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



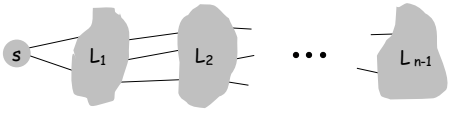
- BFS algorithm.
  - L_0 = { s }.
  - L_1 = tutti i vicini di s.
  - L_2 = tutti i nodi che non appartengono a L_0 or L_1, e che sono uniti da un arco ad un nodo in L_1.
  - L_{i+1} = tutti i nodi che non appartengono agli strati precedenti e che sono uniti da un arco ad un nodo in L_i.

27

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. 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

28

Breadth First Search

Pseudocodice (schema dell'algoritmo)

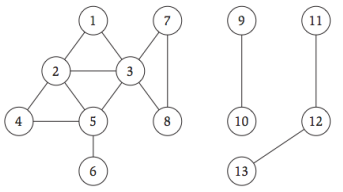
1. BFS(s)
2. L_0={ s }
3. **For**(i=0;i≤n-2;i++)
4. L_{i+1}= Ø ;
5. **Foreach** nodo u in L_i
6. **Foreach** nodo v adiacente ad u
7. **if(** v non appartiene ad $L_0,\ldots,L_{i+1}$)
8. $L_{i+1}=L_{i+1} \cup$ { v }
9. **EndIf**
10. **Endforeach**
11. **Endforeach**
12. **Endfor**

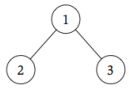
- Occore un modo per capire se un nodo è gia stato visitato in precedenza. Il tempo di esecuzione dipende dal modo scelto, da come è implementato il grafo e da come sono rappresentati gli insiemi L_i che rappresentano i livelli

29

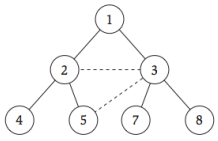
Esempio di esecuzione di BFS

G

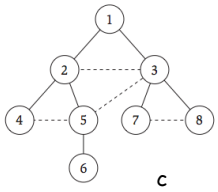




a



b



c

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

30

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

31

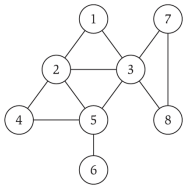
Breadth First Search Tree

- 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.

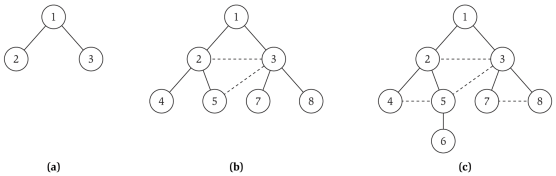
- Dim. Sia L_i il livello di x ed L_j quello di y. Supponiamo senza perdere di generalità che x venga scoperto prima di y cioè che **$i \leq j$**. Consideriamo il momento in cui l'algoritmo esamina gli archi incidenti su x.

- 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). (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)

32

Implementazione di BFS per grafo implementato con liste di adiacenza

- 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
```

33

33

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
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
7.      Inizializza L[i+1] con una lista vuota
8.      Foreach u ∈ L[i]
9.        Foreach arco (u, v) incidente su u
10.       If Discovered[v] = false then
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
```

O(n)

O(1)

n volte

n-1 volte

O(n)

Sul totale di tutte le iterazioni del while, al più n volte

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

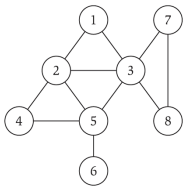
Algoritmo è O(n+m)

34

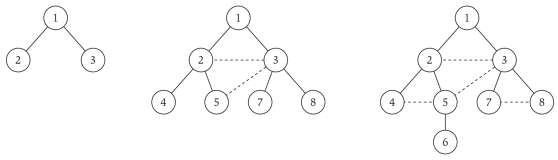
34

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)

35

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

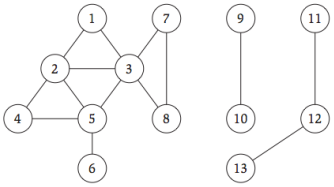
```
BFS(s)
1.   Inizializza Q con una coda vuota
2.   Inizializza il  BFS tree T con un albero vuoto
3.   Poni Discovered[s] = true e Discovered[v] = false per tutti gli altri v
4.   Inserisci s in coda a Q con una enqueue
5.   While(Q non è vuota)
6.     estrai il front di Q con una deque e ponilo in 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)

36

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.

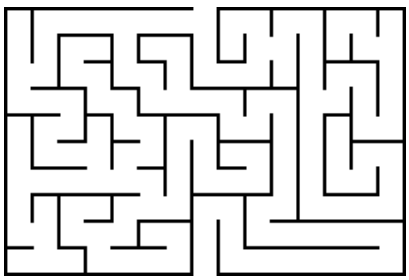
37

37

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.



38

38

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.

39

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

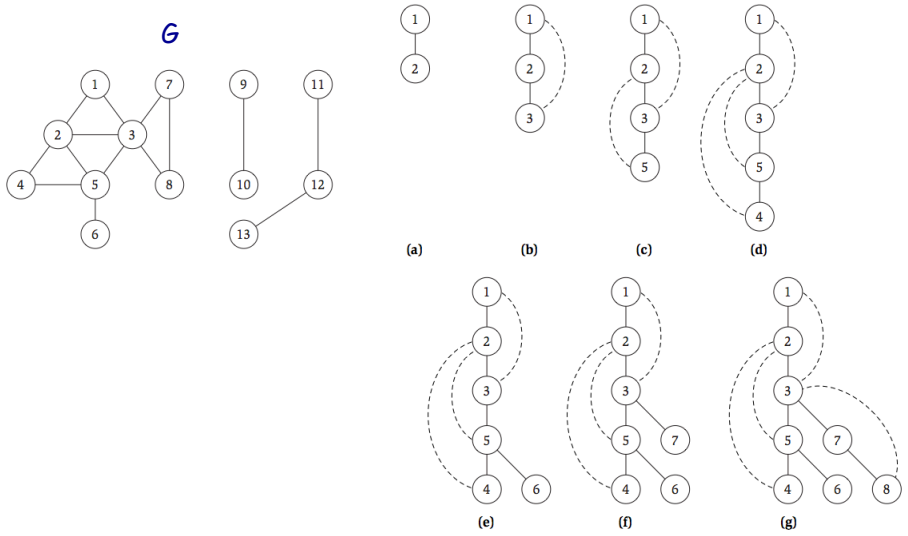
Analisi: (assumendo G rappresentato con liste di adiacenza)

- 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

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

40

Esempio di esecuzione di DFS



41

41

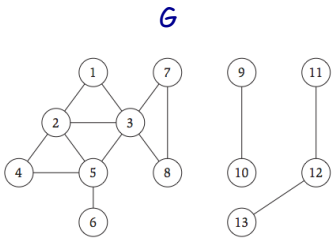
Depth First Search Tree (Albero DFS)

- 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.

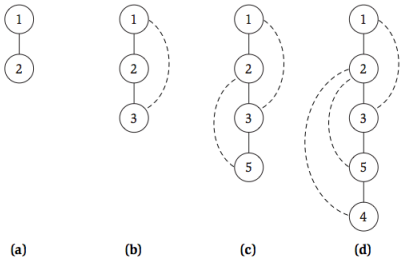
- L'albero si ottiene in questo modo:
  - Consideriamo il momento in cui viene invocata DFS(v)
  - Ciò avviene durante l'esecuzione di DFS(u) per un certo nodo u. In particolare durante l'esame dell'arco (u,v) nella chiamata DFS(u).
  - In questo momento, aggiungiamo l'arco (u,v) e il nodo v all'albero

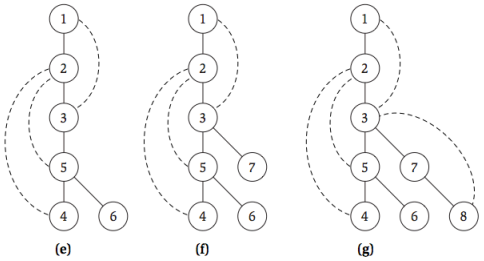
42

Esempio di albero DFS



Gli archi non tratteggiati fanno parte del DFS tree.





43

43

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.

Dim. Proprietà 1.

- Sia x un nodo esplorato tra l'inizio e la fine della chiamata DFS(u)
- La dimostrazione è per induzione sul numero m di chiamate ricorsive iniziate dopo l'inizio di DFS(u) e non ancora terminate **all'inizio** di DFS(x) (esclusa DFS(u) e inclusa DFS(x)).
- **Base. m=1** → Una sola chiamata cominciata dopo l'inizio di DFS(u) e che si deve ancora concludere nel momento in cui esploriamo x → questa chiamata è proprio DFS(x) → DFS(x) è invocata nel foreach di DFS(u) e in questo caso x diventa figlio di u →la proprieta` è soddisfatta.
- **Passo induttivo.** Supponiamo vera la proprieta` per m-1>=1 e dimostriamo che è vera per m. m>=2 → DFS(x) non è invocata nel foreach di DFS(u)
- Notiamo che dopo aver marcato come esplorato u, DFS(u) non fa altro che invocare ricorsivamente la DFS sui nodi adiacenti ad u non ancora esplorati. Nel caso che stiamo considerando x non è tra questi nodi. Cio` vuol dire che x sara` marcato come esplorato da una DFS innescata da una delle DFS invocate nel foreach di DFS(u). Sia DFS(z) la DFS che innesca DFS(x). In altre parole x≠z e x è marcato come esplorato tra l'inizio e la fine di DFS(z). Il numero di chiamate iniziate dopo l'inizio di DFS(z) e non ancora terminate quando inizia DFS(x) è pari a m-1 per cui possiamo applicare l'ipotesi induttiva.
- Per ipotesi induttiva x e discendente di z. Siccome z è figlio di u allora x è anch'esso discendente di u.

44

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.

45

Implementazione di DFS mediante uno stack

```
DFS(s):
1.    Poni Explored[s] = true ed Explored[v] = false per tutti gli altri nodi
2.    Inizializza  S con uno stack contenente s
3.    While S non è vuoto
4.        Metti in u  il nodo  al top di S
5.        If  c'e` un arco (u, v) incidente su u non ancora esaminato then
6.            If Explored[v] = false then
7.              Poni Explored[v] = true
8.              Inserisci v al top di S
9.            Endif
10.       Else  // tutti gli archi incidenti su u sono stati esaminati
11.           Rimuovi il top di S
12.       Endif
13.   Endwhile
```

- Per implementare la linea 6 in modo efficiente possiamo mantenere per ogni vertice u un puntatore al nodo della lista di adiacenza di u corrispondente al prossimo arco (u,v) da scandire.
- Si noti che un nodo u rimane nello stack fino a che non vengono scanditi tutti gli archi incidenti su u.

46

Analisi di DFS implementata mediante uno stack

Assumiamo G rappresentato con liste di adiacenza

DFS(s):

1. Poni Explored[s] = true ed Explored[v] = false per tutti gli altri nodi — $O(n)$
2. Inizializza S con uno stack contenente s — $O(1)$
3. **While** S non è vuoto
4. Metti in u il nodo al top di S
5. **If** c'e` un arco (u, v) incidente su u non ancora esaminato **then**
6. **If** Explored[v] = false **then**
7. Poni Explored[v] = true
8. Inserisci v al top di S
9. **Endif**
10. **Else** // tutti gli archi incidenti su u sono stati esaminati
11. Rimuovi il top di S
12. **Endif**
13. **Endwhile** — **O(n+m)**

- Analisi linee 3-13: Il while viene iterato deg(v) volte per ogni nodo v esplorato (e di conseguenza presente in S): ogni volta che v viene a trovarsi al top dello stack viene esaminato uno dei suoi archi non ancora esaminati. Quindi in totale il while e` iterato un numero di volte pari alla somma dei gradi dei nodi esplorati che e` al più 2m
  - Se manteniamo traccia del prossimo arco da scandire (vedi slide precedente), la linea 6 richiede tempo O(1). Di conseguenza il corpo del while richiede O(1) per ogni iterazione → tempo totale per tutte le iterazioni O(m).

47