APA Modulo 1 Lezione 4

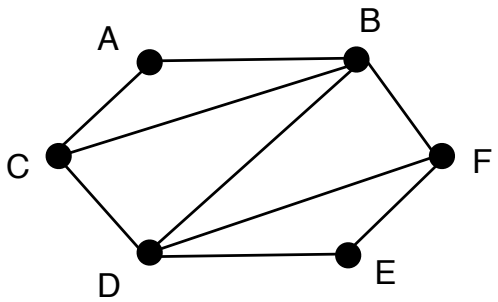
Elena Zucca

16 marzo 2020

Grafi: ripasso definizione

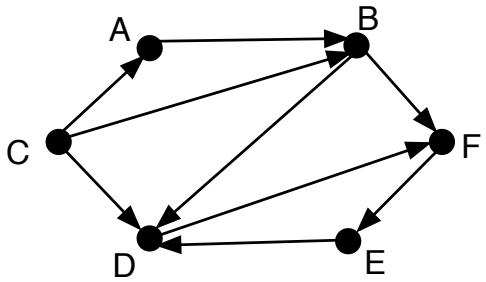
- grafo orientato $G = (V, E)$
- V insieme di nodi o vertici, E insieme di archi (edges)
- ogni arco è una coppia (u, v) di nodi detti estremi dell'arco
- in grafo non orientato archi = coppie non ordinate
  (u, v) e (v, u) denotano lo stesso arco
- cappio = arco da nodo in se stesso, grafo senza cappi = semplice
  (alcune definizioni lo richiedono)

Ripasso terminologia grafi non orientati



- l'arco (A, B) è incidente sui nodi A e B
- i nodi A e B sono adiacenti
- grado $\delta(u)$ = numero di archi incidenti sul nodo u
  per esempio $\delta(B) = 4$

Ripasso terminologia grafi orientati



- l'arco (A, B) è incidente sui nodi A e B, uscente da A, entrante in B
- il nodo B è adiacente ad A, ma A non è adiacente a B
- grado $\delta(u) =$ numero di archi incidenti sul nodo u
- grado uscente (outdegree) $\delta_{out}(u) =$ numero di archi uscenti da u
  per esempio $\delta_{out}(B) = 2$
- grado entrante (indegree) $\delta_{in}(u) =$ numero di archi entranti in u
  per esempio $\delta_{in}(B) = 2$

Alcune ovvie proprietà

Sia $G = (V, E)$, con n nodi ed m archi

- $G = (V, E)$ non orientato:
  - somma gradi dei nodi = doppio del numero archi
    $\sum_{u \in V} \delta(u) = 2m$
  - m = al massimo numero di tutte le coppie non ordinate di nodi
    $n + (n-1) + \ldots + 1 = n(n+1)/2$
    quindi $m = O(n^2)$

Alcune ovvie proprietà

Sia $G = (V, E)$, con n nodi ed m archi

- $G = (V, E)$ orientato:
  - somma gradi uscenti = somma gradi entranti = numero archi
    $\sum_{u \in V} \delta_{out}(u) = \sum_{u \in V} \delta_{in}(u) = m$
    quindi anche in questo caso $\sum_{u \in V} \delta(u) = 2m$
  - m= al massimo numero di tutte le coppie ordinate di nodi n^2
    quindi $m = O(n^2)$
- in genere $m << n^2$
- complessità espressa in funzione di n e m
- se $m \sim n^2$ si dice grafo denso

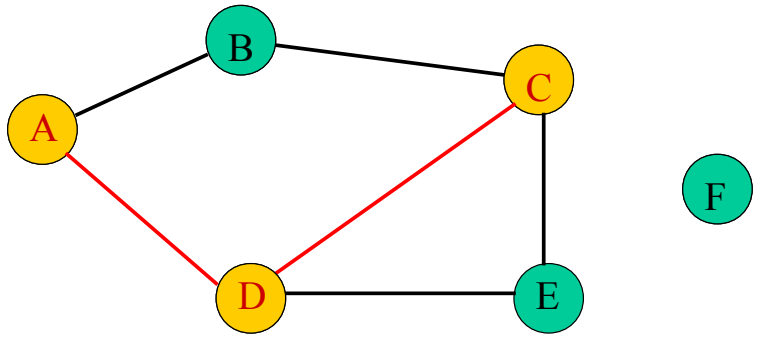
Ripasso terminologia

- cammino (path) = sequenza $u_0, \ldots, u_n$
  con $n \geq 0$ e per ogni $i \in 0..n-1$, (u_i, u_{i+1}) arco
- nel caso grafo non orientato catena
- $n-1$ = numero archi = lunghezza del cammino
- cammino degenere o nullo = un solo nodo (lunghezza 0)
- semplice se nodi distinti tranne al più il primo e l'ultimo
- in grafo orientato:
  ciclo = cammino non nullo con primo nodo = ultimo
- in grafo non orientato: ciclo o circuito = cammino (catena) di lunghezza ≥ 3 con primo nodo = ultimo
- v è raggiungibile da u se esiste un cammino da u a v

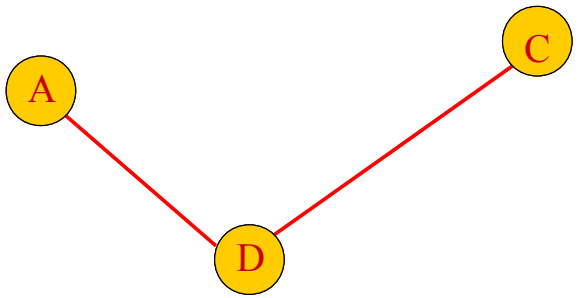
Ripasso terminologia

- G aciclico se non vi sono cicli
- DAG = directed acyclic graph = grafo orientato aciclico
- grafo non orientato connesso se ogni nodo è raggiungibile da ogni altro
- grafo orientato fortemente connesso se ogni nodo è raggiungibile da ogni altro, debolmente connesso se il grafo non orientato corrispondente è connesso
- sottografo di $G = (V, E)$ = grafo ottenuto da G non considerando alcuni archi e/o nodi
- sottografo indotto da $V' \subseteq V$ = sottografo di G con nodi V' e gli archi di G che li connettono

Esempio

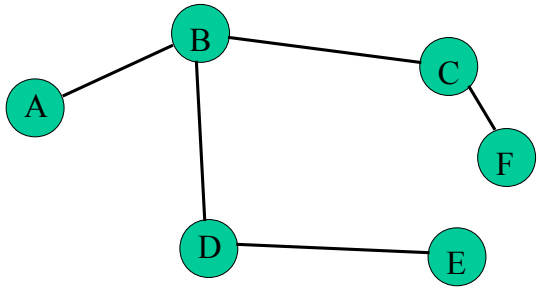


sottografo indotto da A, B, C:

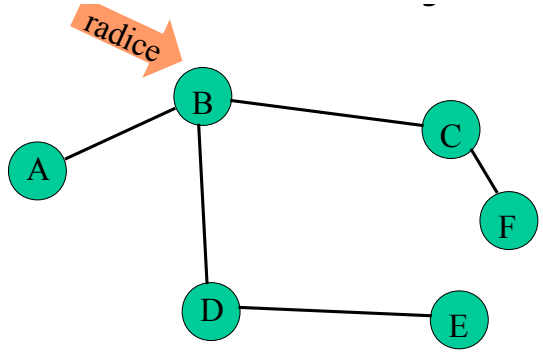


Albero libero

- albero libero= grafo non orientato connesso aciclico



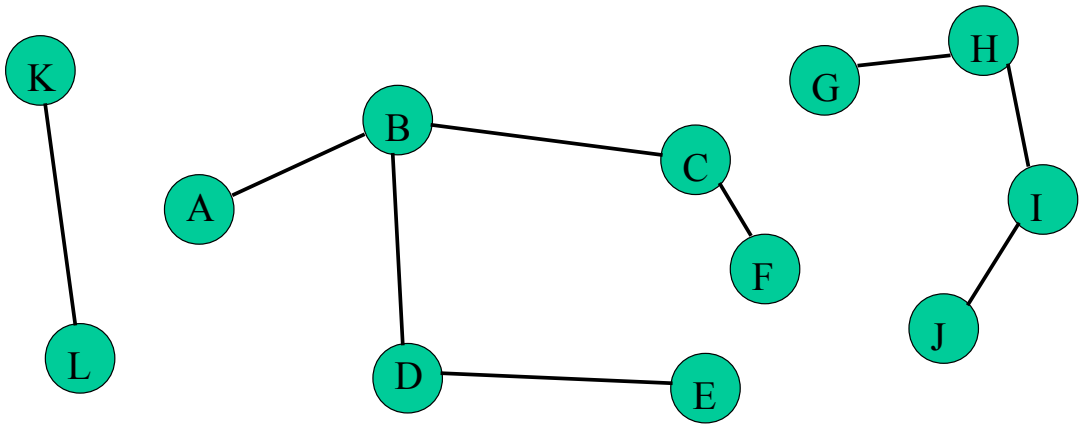
- se si fissa un nodo u come radice, si ottiene un albero nel senso usuale



(ossia, radicato), avente u come radice

- si può pensare di "appendere" il grafo a un qualunque nodo, e si ottiene sempre un albero.

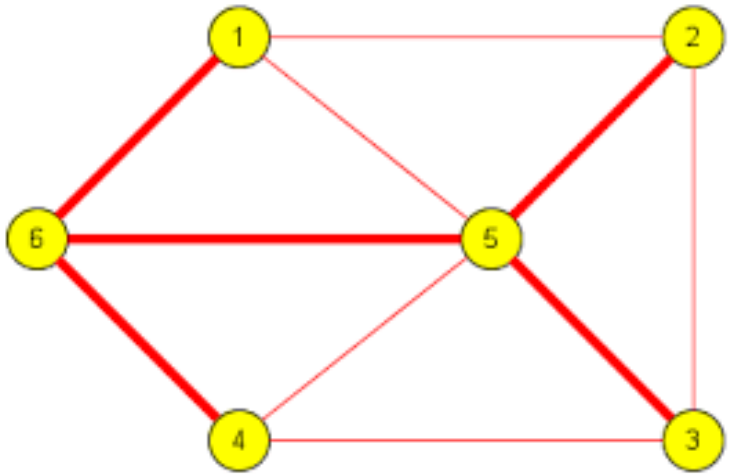
Foresta libera



Albero ricoprente

- dato grafo non orientato e connesso G, un albero ricoprente (spanning tree) di G è un sottografo di G che contiene tutti i nodi ed è un albero libero
- ossia albero che connette tutti i nodi del grafo usando archi del grafo, ha quindi n nodi ed $n - 1$ archi
- se grafo non connesso si ha foresta ricoprente

Esempio



Rappresentazione di grafi: lista di archi

- si memorizza insieme nodi e lista archi
  complessità spaziale $O(n+m)$
- molte operazioni richiedono di scorrere l'intera lista

operazione	**tempo di esecuzione**
grado di un nodo	$O(m)$
nodi adiacenti	$O(m)$
esiste arco	$O(m)$
aggiungi nodo	$O(1)$
aggiungi arco	$O(1)$
elimina nodo	$O(m)$
elimina arco	$O(m)$

Rappresentazione di grafi: liste di adiacenza

- per ogni nodo si memorizza lista nodi adiacenti
  $2m$ per grafo non orientato, m per grafo orientato, complessità spaziale $O(n+m)$
- per grafo non orientato informazione ridondante (coerenza)
- semplice trovare adiacenti di un nodo, operazione spesso cruciale

operazione	**tempo di esecuzione**
grado di u	$O(\delta(u))$
nodi adiacenti a u	$O(\delta(u))$
esiste arco (u, v)	$O(\min(\delta(u), \delta(v)))$
aggiungi nodo	$O(1)$
aggiungi arco	$O(1)$
elimina nodo	$O(m)$
elimina arco (u, v)	$O(\delta(u) + \delta(v))$

Rappresentazione di grafi: matrice di adiacenza

- matrice quadrata M di dimensione n a valori booleani (oppure $0, 1$)
- $M[i, j]$ vero se e solo se esiste l'arco (u_i, u_j)
- complessità spaziale $O(n^2)$
- per grafo non orientato informazione ridondante (matrice simmetrica)
- verifica presenza arco tempo costante, trovare gli adiacenti più costoso (intera riga)

operazione	**tempo di esecuzione**
grado di un nodo	$O(n)$
nodi adiacenti	$O(n)$
esiste arco	$O(1)$
aggiungi nodo	$O(n^2)$ (riallocazione)
aggiungi arco	$O(1)$
elimina nodo	$O(n^2)$ (riallocazione)
elimina arco	$O(1)$

Riassumendo

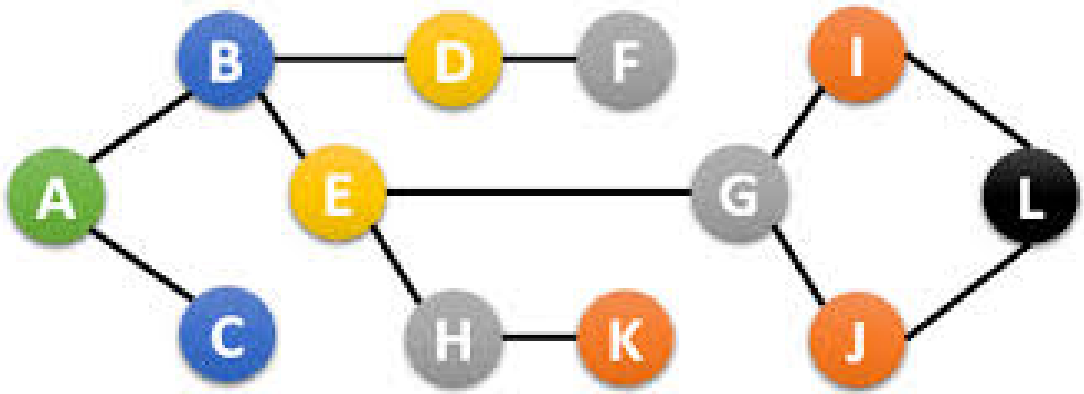
- liste di adiacenza in genere preferibili, in particolare per grafi sparsi ossia con $m << n^2$
- matrice di adiacenza può essere preferibile per grafo denso ($m \sim n^2$) o quando è importante controllare in modo efficiente se esiste un arco tra due vertici
- entrambe le rappresentazioni sono facilmente adattabili ai grafi pesati, ossia dove ogni arco ha un peso (costo) associato

Visite

- algoritmi simili a quelli per gli alberi, ma occorre marcare i nodi
- tradizionalmente marcati come bianco, grigio, e nero:
  - bianco: non ancora toccato
  - grigio: visita iniziata
  - nero: visita conclusa
- visite iterative usano frangia F da cui vengono via via estratti i nodi da visitare
- in visita in ampiezza frangia = coda, in visita in profondità frangia = pila, in algoritmi di Dijkstra e Prim frangia = coda a priorità (heap)
- costruiscono implicitamente albero di visita T (o foresta)

Visita in ampiezza (breadth-first)

convenzione: figli in ordine alfabetico



Breadth-First Search (BFS)

Coda: A $\not{A}BC$ $\not{B}CDE$ $\not{C}DE$ $\not{D}EF$ $\not{E}FGH$
$\not{F}GH$ $\not{G}HIJ$ $\not{H}IJK$ $\not{I}JKL$
$\not{J}KL$ $\not{K}L$ $\not{L}$;

Pseudocodice

```
BFS(G,s) //visita nodi raggiungibili da s
  for each (u nodo in G)
    marca u come bianco;
  Q = coda vuota
  Q.add(s); marca s come grigio;
  while (Q non vuota)
    u = Q.remove() //u non nero
    visita u
    for each ((u,v) arco in G)
          if (v bianco)
            marca v come grigio; Q.add(v);
    marca u come nero
```

- per semplicità visita del sottografo connesso a partire da un nodo
- non è necessario distinguere tra nodi grigi e neri
- costruisce implicitamente albero di visita T, che può essere reso esplicito
- è un albero ricoprente

Pseudocodice con albero di visita

```
BFS(G,s) //visita nodi raggiungibili da s
  for each (u nodo in G)
    marca u come bianco; parent[s] = null
  Q = coda vuota
  Q.add(s); marca s come grigio;
  while (Q non vuota)
    u = Q.remove() //u non nero
    visita u
    for each ((u,v) arco in G)
          if (v bianco)
            marca v come grigio; Q.add(v);
            parent[v]=u
    marca u come nero
```

- i nodi sono tutti quelli raggiungibili da s, ossia tutti gli u tali che parent$[u] \neq$ null, più s stesso
- gli archi sono gli u, v tali che parent$[v]=u$

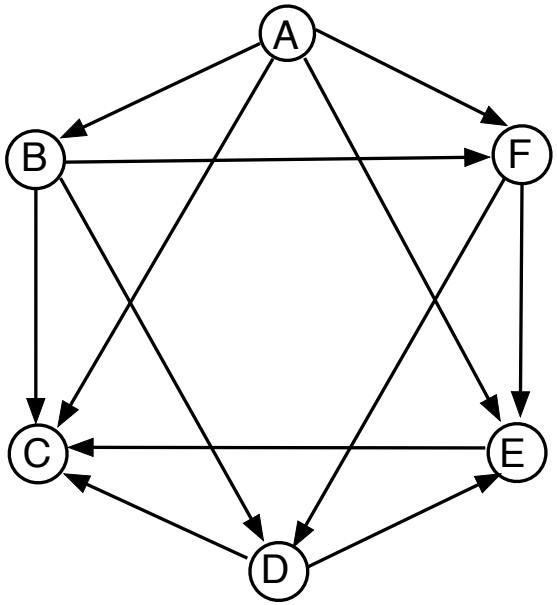
Osservazioni

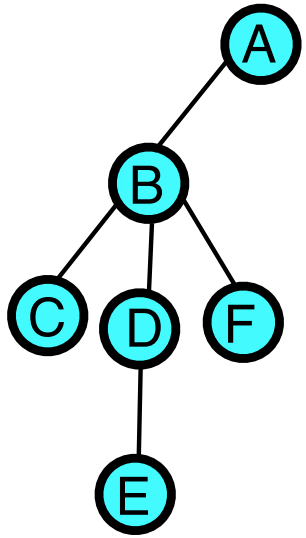
- nodi nell'albero di visita corrente grigi o neri
- la frangia è una coda `Q`, che mantiene l'ordine in cui i nodi sono trovati; ogni nodo entra in coda una volta sola
- invariante del ciclo: nodi grigi $=$ nodi in F
  nodi nell'albero $=$ nodi neri e grigi
- il predecessore (padre) di un nodo viene deciso nel momento in cui il nodo viene incontrato
- la visita calcola la distanza (lunghezza minima di un cammino) dalla radice a ogni nodo

Visita in profondità (depth-first)

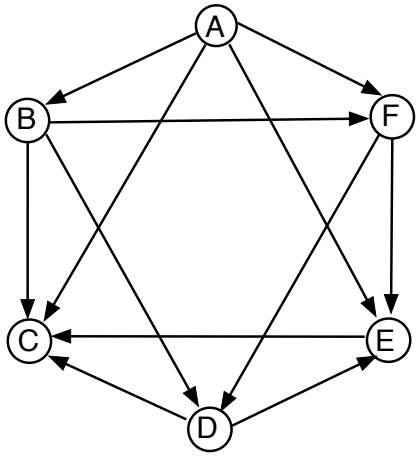
convenzione: figli in ordine alfabetico

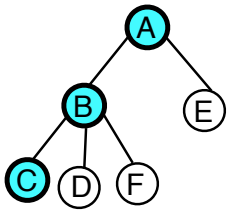
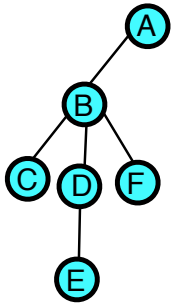
si segue un cammino nel grafo finché possibile

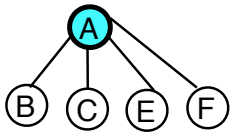
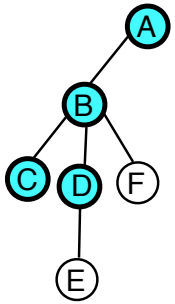


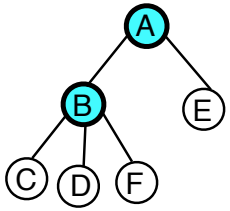
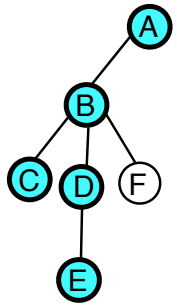


Visita in profondità iterativa con pila



PILA	ALBERO DFS
A	
D F C E F	
C E F	

PILA	ALBERO DFS
B C E F	
E F C E F	
E F	...

PILA	ALBERO DFS
C D F C E F	
F C E F	
F	...

PILA	ALBERO DFS
vuota	...

Pseudocodice

```
DFS(G,s) //visita nodi di G raggiungibili da s
  for each (u nodo in G)
    marca u come bianco; parent[s] = null
  S = pila vuota
  S.push(s); marca s come grigio;
  while (S non vuota)
    u = S.pop()
    if (u non nero)
      visita u
      for each ((u,v) arco in G)
        if (v bianco)
          marca v come grigio; S.push(v); parent[v]= u
        else if (v grigio)
          S.push(v); parent[v]= u //modifica il padre
      marca u come nero
```

Osservazioni

- un nodo può essere inserito nella pila anche se grigio, quindi più volte, nel caso peggiore tante volte quanti sono i suoi archi entranti
- il padre viene modificato ogni volta
- può essere estratto dalla pila un nodo nero

Complessità della visita completa (caso DFS)

- n = numero nodi, m = numero archi
- marcatura e operazioni su coda/pila costanti (no in Dijkstra e Prim)
- marcature dei nodi: $O(n)$
- inserimenti in F e T, modifiche di F e T, estrazioni da F: ogni nodo viene inserito una prima volta in F e T, poi eventualmente F e T vengono aggiornati al più m volte: $O(n+m)$
- esplorazione archi incidenti eseguita per ogni nodo, quindi:
  - lista di archi: $O(n \cdot m)$
  - liste di adiacenza: $O(n+m)$ perché ogni lista di adiacenza viene scandita una volta sola
  - matrice di adiacenza: $O(n^2)$
- complessità totale caso liste di adiacenza: $O(n+m)$

Visita in profondità ricorsiva

- occorre comunque marcare i nodi come visitati
- distinzione grigio/nero non necessaria, evidenzia fine visita
- algoritmo per visita completa (ossia, anche di un grafo non connesso)

```
DFS(G)
  for each (u nodo in G)
    marca u come bianco; parent[u]=null
  for each (u nodo in G)
    if (u bianco) DFS(G,u)
```

Visita a partire da un nodo

```
DFS(G,u)
  //inizio visita
  visita u; marca u come grigio
  for each ((u,v) arco in G)
    if (v bianco)
      parent[v]=u
      DFS(G,v)
  //marca u come nero
  //fine visita
```

viene costruita una foresta DFS