algoritmi e strutture di dati

visite di grafi

m.patrignani

nota di copyright

contenuto

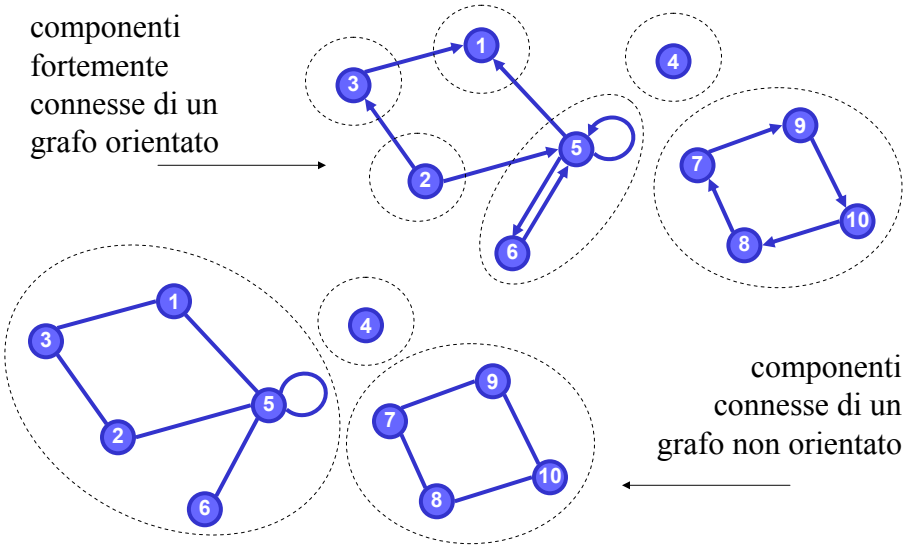
- grafi e connettività
- visita in ampiezza
- visita in profondità

grafi e connettività

- dato un grafo orientato (o diretto) $G(V,E)$
  - un nodo v è *raggiungibile* da un nodo u se esiste un cammino diretto da u a v
  - se ogni coppia di nodi u e v di V sono raggiungibili il grafo è detto *fortemente connesso*
  - un insieme massimale di nodi tra loro tutti raggiungibili è una *componente fortemente connessa*
- dato un grafo non orientato $G(V,E)$
  - un nodo v è *raggiungibile* da un nodo u se esiste un cammino da u a v
  - se per ogni coppia di nodi u e v di V esiste un cammino da u a v il grafo è detto *connesso*
  - la proprietà raggiungibilità tra nodi di un grafo non orientato è una proprietà di equivalenza le cui classi di equivalenza sono chiamate *componenti connesse*

esempi con grafi orientati e non

componenti fortemente connesse di un grafo orientato



componenti connesse di un grafo non orientato

algoritmi di visita di grafo

- lo scopo di questi algoritmi è quello di visitare tutti i nodi raggiungibili a partire da un nodo di partenza
- caratteristiche:
  - i nodi non sono raggiunti in ordine casuale, ma in un ordine determinato dalla forma del grafo
    - diversi algoritmi su grafi sono modifiche di algoritmi di visita
  - non tutti i nodi vengono raggiunti
    - perché il grafo potrebbe avere più componenti connesse o fortemente connesse
  - alcuni nodi possono essere raggiunti da più nodi adiacenti
    - occorre marcare i nodi per non rischiare di ciclare ad infinito

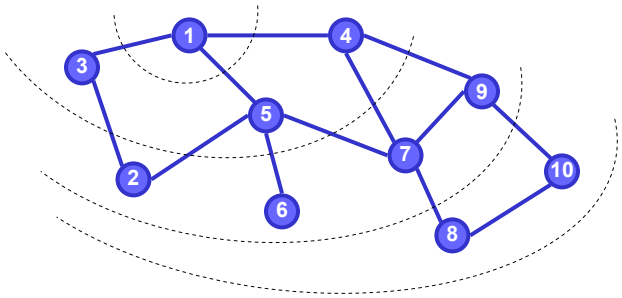
implementazioni di marcatori

- un marcatore è un valore associato ad ogni nodo di un grafo
  - per esempio un booleano TRUE o FALSE
  - talvolta sono anche utilizzati marcatori a più valori
    - per esempio: nero, bianco, grigio
- per marcare i nodi di un grafo è sufficiente abbinare alla struttura del grafo un array di interi con *n* posizioni, dove *n* è il numero dei nodi

```
…
1.  ▷ "color" è un array di interi con n posizioni
4.  for i = 0 to color.length-1
5.      color[i] = 0        ▷ inizializzo l'array con zero
6.  …
7.  color[6] = 1            ▷ coloro il nodo con indice 6
```


visita in ampiezza (breadth-first search)

- a partire da un nodo *v* si visitano i nodi raggiungibili da *v* nell'ordine imposto dalla loro distanza
  - prima i più vicini, poi i più lontani



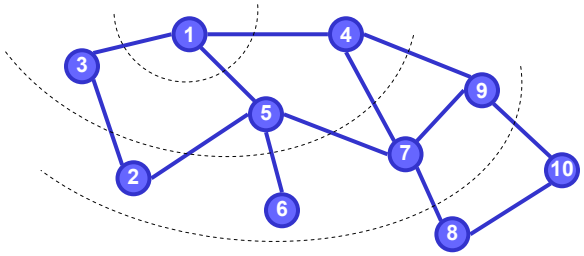
visita in ampiezza

- durante la visita in ampiezza un nodo può essere
  - mai raggiunto
  - raggiunto e le sue adiacenze non ancora esplorate
  - raggiunto e le sue adiacenze già esplorate
- occorre marcare i nodi con tre colori in accordo con i tre stati
  - ○ bianco = 0 = non ancora raggiunto
  - ● grigio = 1 = raggiunto
  - ● nero = 2 = completamente esplorato

strategia per una visita in ampiezza

- facciamo uso di una coda sulla quale è possibile eseguire le operazioni `ENQUEUE` e `DEQUEUE`
- la coda viene inizializzata inserendoci il nodo di partenza
- i nodi raggiunti vengono messi in coda
  - per essere sicuri di non metterli in coda due volte li coloriamo di grigio appena li mettiamo in coda
- finché la coda non è vuota
  - estraiamo un nodo dalla coda
  - lo coloriamo di nero (per segnalare che è stato esplorato)
  - consideriamo tutti i suoi adiacenti e se sono raggiunti per la prima volta (bianchi) li coloriamo di grigio e li mettiamo in coda
- l'ordine con cui i nodi sono estratti dalla coda e colorati di nero corrisponde ad una visita in ampiezza
  - è lo stesso ordine con cui i nodi sono messi nella coda e colorati di grigio

esempio di visita in ampiezza



esplorati (neri)	coda (grigi)
	1
1	3,5,4
1,3	5,4,2
1,3,5	4,2,6,7
1,3,5,4	2,6,7,9

1,3,5,4,2	6,7,9
1,3,5,4,2,6	7,9
1,3,5,4,2,6,7	9,8
1,3,5,4,2,6,7,9	8,10
1,3,5,4,2,6,7,9,8	10
1,3,5,4,2,6,7,9,8,10	

procedura BFS (liste di adiacenza)

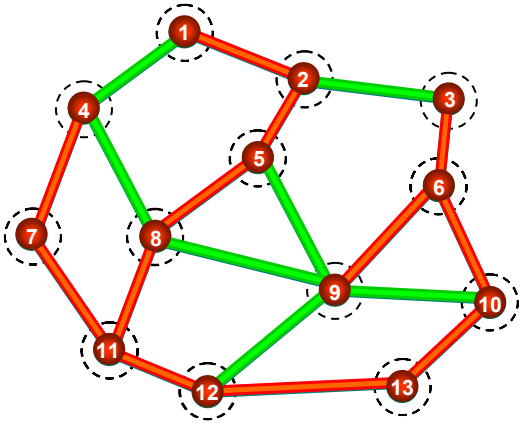
```
BFS(A,v)              ▷ A è un array di liste di adiacenza, v è un indice
1.  for i = 0 to A.length-1
2.      color[i] = 0          ▷ zero = white = non raggiunto
3.  QUEUE-EMPTY(q)            ▷ creo una coda vuota
4.  color[v] = 1              ▷ uno = grigio = raggiunto = messo in coda
5.  ENQUEUE(q,v)              ▷ metto in coda l'indice v
6.  while not QUEUE-VOID(q)   ▷ finché la coda q non è vuota
7.      u = DEQUEUE(q)        ▷ estraggo un indice dalla coda
8.      x = A[u]              ▷ mi preparo ad esporare gli adiacenti di u
9.      while x != NIL        ▷ finché c'è un nodo adiacente
10.         k = x.key         ▷ k è l'indice del nodo adiacente a u
11.         if (color[k] == 0)    ▷ se k è bianco (cioè mai raggiunto)…
12.             color[k] = 1          ▷ …coloro k di grigio…
13.             ENQUEUE(q,k)          ▷ …e lo metto in coda
14.         x = x.next
15.     color[u] = 2          ▷ due = black = visitato (peraltro superfluo)
```


esercizi sulle visite in ampiezza

- scrivi lo pseudocodice della procedura `BFS` nel caso in cui il grafo sia rappresentato tramite una matrice di adiacenza
- scrivi lo pseudocodice della procedura `IS-CONNECTED`(A) che restituisce `TRUE` se il grafo è connesso
  - possibile strategia:
    - eseguo una visita a partire da un nodo qualunque
    - se alcuni nodi rimangono bianchi il grafo non è connesso
- scrivi lo pseudocodice della procedura `DISTANZA`(A,*v*) che restituisce un array delle distanze di tutti i nodi dal nodo con indice *v*
  - possibile strategia
    - eseguo un visita in ampiezza a partire da *v*
    - quando un nodo viene colorato di grigio, la sua distanza da *v* è pari alla distanza del nodo da cui è raggiunto più uno

visita in profondità (depth-first search)

- a partire da un nodo v si percorre il grafo fino a che si trovano nodi non visitati
- quando tutti i vicini sono visitati si torna indietro verso *v* per verificare che non ci siano nodi adiacenti non visitati



procedura DFS (grafo connesso)

- DFS di un grafo connesso a partire da un nodo v

```
DFS(A,v)          ▷ A è un array di liste di adiacenza
1.  for i = 0 to A.length-1
2.      color[i] = 0      ▷ zero = white = non raggiunto
3.  DFS-visit(A,v,color)      ▷ …comincia una DFS da v
```

```
DFS-visit(A,i,color)      ▷ A è un array, i è un indice, color un array
1.  color[i] = 1          ▷ uno = grey = raggiunto
2.  x = A[i]
3.  while x != NIL
4.      v = x.key         ▷ v è l'indice di un nodo adiacente ad i
5.      if color[v] == 0              ▷ se v non ancora visitato…
6.          DFS-visit(A,v,color)      ▷ …continua la visita da v
7.      x = x.next        ▷ passa al prossimo adiacente di i
8.  color[i] = 2          ▷ due = black = tutti gli adiacenti visitati
```


procedura DFS (grafo non connesso)

- per eseguire una DFS di un grafo non necessariamente connesso è sufficiente modificare la procedura esterna

```
DFS(A)            ▷ A è un array di liste di adiacenza
1.  for i = 0 to A.length-1
2.      color[i] = 0      ▷ zero = white = non raggiunto
3.  for i = 0 to A.length-1
4.      if color[i] == 0              ▷ se i non ancora visitato…
5.          DFS-visit(A,i,color)      ▷ …comincia una DFS da qui
```


complessità delle visite BFS e DFS

- in una visita in ampiezza/profondità
  - ogni nodo è inserito ed estratto dalla coda/pila una sola volta
  - ogni arco (adiacenza) è considerata sia dal nodo di partenza che dal nodo di arrivo
- dunque la complessità è $\Theta(n+m)$

esercizi sulle visite di grafi

- scrivi lo pseudocodice della procedura `GRAFO-CONNESSO`(A)
  - input: un grafo non orientato rappresentato da un array A di liste di adiacenza
  - output: `TRUE` se il grafo è connesso, `FALSE` altrimenti
- scrivi lo pseudocodice della procedura `STESSA-COMPONENTE-FORTEMENTE-CONNESSA`(A,u,v)
  - input: un grafo orientato rappresentato da un array A di liste di adiacenza e gli indici di due nodi u e v
  - output: `TRUE` se u e v sono nella stessa componente fortemente connessa, `FALSE` altrimenti

esercizi sulle visite di grafi

- scrivi lo pseudocodice della procedura `COMPONENTI-CONNESSE(`*A*`)`
  - input: un grafo non orientato rappresentato da un array *A* di liste di adiacenza
  - output: il numero delle componenti connesse del grafo
- possibile strategia:
  - pongo il contatore delle componenti connesse a zero
  - finché c'è un nodo non visitato
    - incremento il contatore delle componenti connesse
    - eseguo una visita (qualsiasi) a partire dal nodo non visitato marcando tutti i nodi visitati

esercizi sulle visite di grafi

- scrivi lo pseudocodice della funzione `ALBERO-RICOPRENTE(A,u)`
  - input: un grafo non orientato rappresentato da un array `A` di liste di adiacenza e un nodo `u`
  - output: un array `parent` dove `parent[v]` è l'indice del nodo da cui è stato raggiunto `v` in una DFS a partire dal nodo `u`
    - il parent[u] è convenzionalmente posto uguale a -1
- variante:
  - produci in output un albero `t` (connesso e di grado arbitrario) con le seguenti caratteristiche
    - i nodi di `t` hanno gli stessi indici dei nodi del grafo
    - c'è un arco in `t` diretto da un nodo `x` ad un nodo `y` solo se nella DFS il nodo `y` è stato raggiunto da `x`

esercizi sulle visite di grafi

- scrivi lo pseudocodice della procedura `DFS`(A,v) nel caso in cui il grafo sia rappresentato da una matrice di adiacenza A
- scrivi lo pseudocodice della procedura `DFS`(A,v) che restituisce in output un array `ordine` dove `ordine`[v] è il numero d'ordine con cui il nodo v è stato visitato

esercizi sulle visite di grafi

- scrivi lo pseudocodice della funzione `CAMMINO-MINIMO`(A,u,v) che prende in input un grafo rappresentato da un array A di liste di adiacenza e gli indici di due nodi u e v e produce in output la lista degli indici dei nodi del cammino più corto da u a v
  - possibile strategia
    - eseguo una visita in ampiezza a partire da v e memorizzo per ogni nodo u il parent[u], cioè il nodo dal quale è stato raggiunto
    - la catena di parent che conduce da u a v è il cammino minimo cercato

