ROMA TRE
UNIVERSITÀ DEGLI STUDI

INGEGNERIA
CIVILE, INFORMATICA E DELLE TECNOLOGIE AERONAUTICHE

Algoritmi e Strutture di Dati

Visita in ampiezza di un grafo

m.patrignani

Nota di copyright

- queste slides sono protette dalle leggi sul copyright
- il titolo ed il copyright relativi alle slides (inclusi, ma non limitatamente, immagini, foto, animazioni, video, audio, musica e testo) sono di proprietà degli autori indicati sulla prima pagina
- le slides possono essere riprodotte ed utilizzate liberamente, non a fini di lucro, da università e scuole pubbliche e da istituti pubblici di ricerca
- ogni altro uso o riproduzione è vietata, se non esplicitamente autorizzata per iscritto, a priori, da parte degli autori
- gli autori non si assumono nessuna responsabilità per il contenuto delle slides, che sono comunque soggette a cambiamento
- questa nota di copyright non deve essere mai rimossa e deve essere riportata anche in casi di uso parziale

Contenuto

- Visita in ampiezza di un grafo indiretto
- Grafi e connettività
- Esercizi sulle visite in ampiezza

Algoritmi di visita di un 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
  - alcuni nodi possono essere raggiunti da più nodi adiacenti
    - occorre marcare i nodi per non rischiare di ciclare ad infinito

Uso dei marcatori

- Gli algoritmi di visita dei grafi fanno tutti uso di marcatori
- Un marcatore è un valore associato ad ogni nodo di un grafo
  - per esempio un booleano (TRUE o FALSE) oppure un intero (generalmente 1 o 0)
- Nel caso più generale si può associare ad un nodo un generico intero che viene spesso chiamato “colore” (color) del nodo

Realizzazioni di marcatori

- Se un nodo è identificato da un intero è sufficiente affiancare (o aggiungere) 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
2.  for i = 0 to color.length-1
3.      color[i] = 0     // inizializzo l'array con zero
4.  color[6] = 1         // coloro con 1 il nodo con indice 6
```

- Per verificare se il nodo *i* è marcato eseguiremo:

```
1. if ( color[i] == 1 )          // nodo i marcato
```

Realizzazioni di marcatori

- Se un nodo è identificato da un riferimento ad un oggetto è sufficiente aggiungere alla struttura del nodo un campo intero "color"

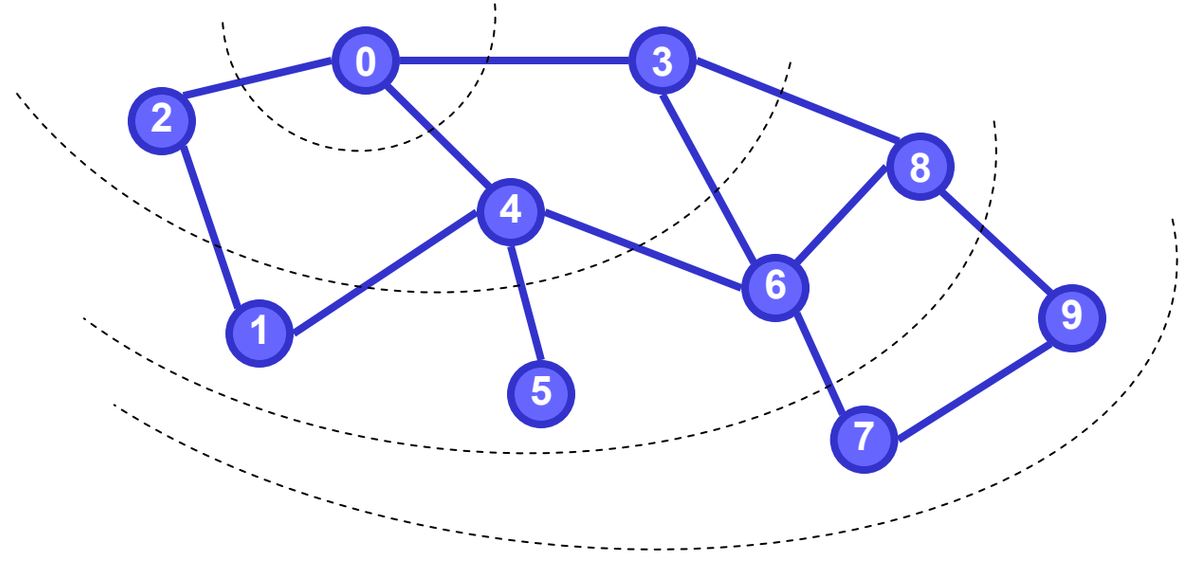
```
1.   x = g.nodi
2.   while x != NULL                  // scorro la lista dei nodi
3.      x.info.color = 0     // inizializzo il colore con zero
4.      x = x.next
```

- Per verificare se il nodo *n* è marcato eseguiremo:

```
1. if ( n.color == 1 )      // nodo n marcato
```

Visita in ampiezza (Breadth-First Search)

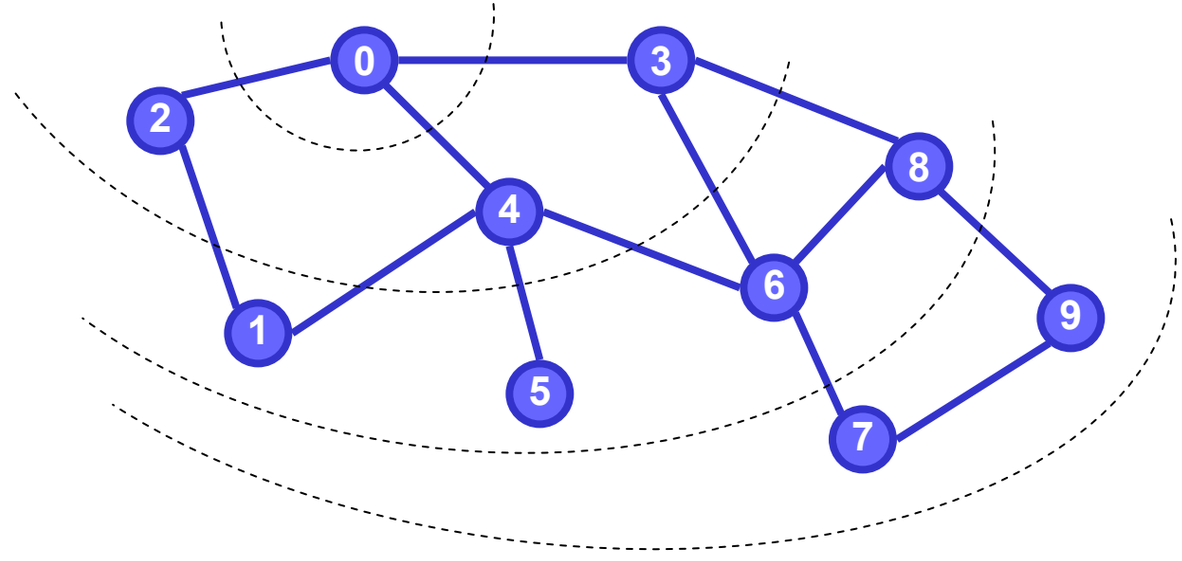
- A partire da un nodo *v* si visitano i nodi raggiungibili da *v* nell'ordine imposto dalla loro distanza
  - prima i più vicini, poi i più lontani



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 marcati e messi in coda
  - per essere sicuri di non metterli in coda due volte li marchiamo appena li mettiamo in coda
- Finché la coda non è vuota
  - estraiamo un nodo dalla coda
  - consideriamo tutti i suoi adiacenti e se sono raggiunti per la prima volta (non sono marcati) li marchiamo e li mettiamo in coda
- L'ordine con cui i nodi sono estratti dalla coda corrisponde ad una visita in ampiezza
  - è lo stesso ordine con cui i nodi sono messi nella coda e marcati

Esempio di visita in ampiezza



esplorati	coda
	0
0	2,4,3
0,2	4,3,1
0,2,4	3,1,5,6
0,2,4,3	1,5,6,8

0,2,4,3,1	5,6,8
0,2,4,3,1,5	6,8
0,2,4,3,1,5,6	8,7
0,2,4,3,1,5,6,8	7,9
0,2,4,3,1,5,6,8,7	9
0,2,4,3,1,5,6,8,7,9	

Procedura BFS (liste di adiacenza)

```
BFS(g,v)          // g.A è un array di liste di adiacenza, v è un indice
1.   for i = 0 to g.A.length-1
2.       color[i] = 0      /* zero = non raggiunto */
3.   q = QUEUE-EMPTY()     /* creo una coda vuota */
4.   color[v] = 1          /* uno = raggiunto e 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 = g.A[u]        /* mi preparo ad esporare gli adiacenti di u */
9.       while x != NULL   /* finché c'è un nodo adiacente */
10.          k = x.info       /* k è l'indice del nodo adiacente a u */
11.          if (color[k] == 0)   /* se k non è stato già raggiunto */
12.              color[k] = 1         /* raggiunto e messo in coda */
13.              ENQUEUE(q,k)
14.          x = x.next
```

Complessità della visita BFS

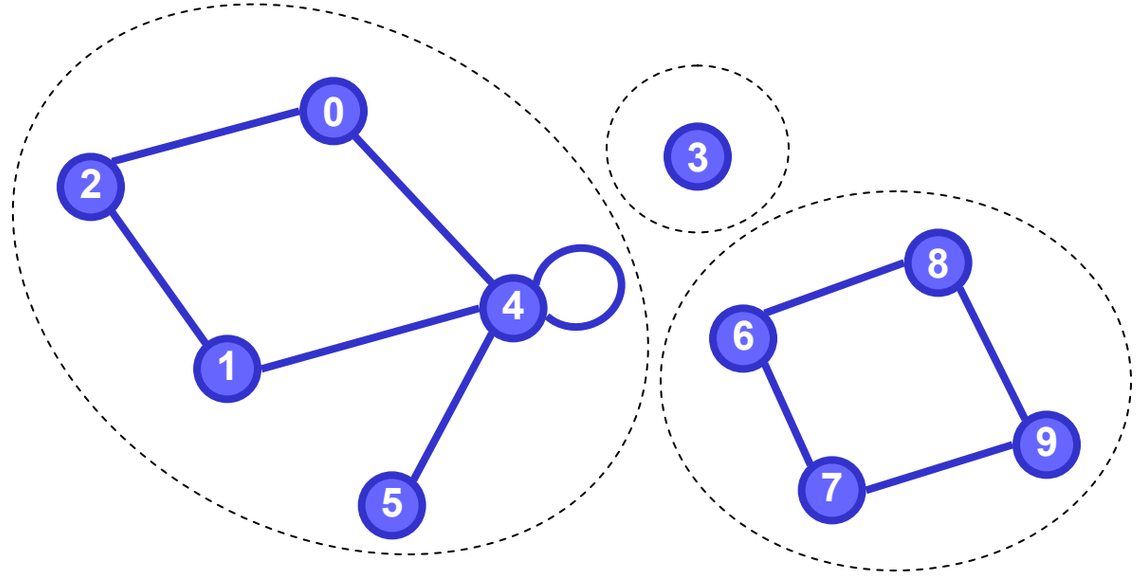
- In una visita in ampiezza
  - ogni nodo è inserito ed estratto dalla coda una sola volta
  - ogni arco (adiacenza) è considerata sia dal nodo di partenza che dal nodo di arrivo
- Dunque la complessità nel caso peggiore è $\Theta(n+m)$

Esercizi sulle visite in ampiezza

1. Scrivi lo pseudocodice della procedura `BFS`(g,v) nel caso in cui il grafo non diretto g sia rappresentato tramite una matrice di adiacenza
2. Scrivi lo pseudocodice della procedura `BFS`(g,v) nel caso in cui il grafo non diretto g sia rappresentato tramite oggetti e riferimenti

Grafi non orientati e connettività

- 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*



Esercizi sulle visite in ampiezza

3. Scrivi lo pseudocodice della procedura `IS_CONNECTED`(g) che restituisce `TRUE` se il grafo è connesso
   - possibile strategia:
     - eseguo una visita a partire da un nodo qualunque
     - se alcuni nodi rimangono non marcati il grafo non è connesso

Visite di un grafo non connesso

- Per eseguire una BFS di un grafo non necessariamente connesso è sufficiente lanciare diverse visite con lo stesso array color
  - per esempio nel caso di grafo rappresentato con liste/matrice di adiacenza il codice potrebbe essere il seguente

```
BFS_non_connesso(g)        /* g è rappresentato tramite liste di adiacenza */
1.   for i = 0 to g.A.length-1
2.        color[i] = 0              /* zero = non raggiunto */
3.   for i = 0 to g.A.length-1
4.        if color[i] == 0          /* se i non ancora visitato… */
5.             BFS(g,i,color)       /* …comincia una BFS da qui */
```

Calcolo delle componenti connesse

4. Scrivi lo pseudocodice della procedura `COMPONENTI_CONNESSE`(g)
   - input: un grafo non diretto g
   - output: il numero delle componenti connesse del grafo g
   - 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 a partire dal nodo non visitato marcando tutti i nodi visitati

Esercizi sulle visite in ampiezza

5. Supponi di disporre di un'implementazione di una tabella hash dai nodi agli interi
   - `new_table`()
     - ritorna una nuova tabella hash vuota
   - `add_pair`(*h*,*n*,*i*)
     - aggiunge alla tabella *h* una coppia formata da un nodo *n* ed un intero *i*
   - `get_value`(*h*,*n*)
     - ritorna il valore associato al nodo *n*

   Scrivi lo pseudocodice della procedura `BFS_order`(*g*,*v*) che restituisce in output una tabella hash `order` dove `get_value`(`order`, *n*) è il numero d'ordine con cui il nodo *n* è stato visitato
   - nel caso in cui il grafo sia rappresentato come una matrice o un array di liste di adiacenza al posto della tabella hash si potrebbe ritornare un semplice array

Esercizi sulle visite in ampiezza

6. Scrivi lo pseudocodice della procedura `DISTANZE`(g,v) che restituisce una tabella hash delle distanze di tutti i nodi dal nodo v
   - i nodi non raggiunti devono avere distanza -1
   - possibile strategia
     - eseguo un visita in ampiezza a partire da v
     - quando un nodo viene marcato, la sua distanza da v è pari alla distanza del nodo da cui è raggiunto più uno
   - nel caso in cui il grafo si rappresentato tramite un array di liste di adiacenza o una matrice di adiacenza la funzione potrebbe restituire un semplice array di interi

Esercizi sulle visite in ampiezza

7. Scrivi lo pseudocodice della funzione `CAMMINO_MINIMO`(g,u,v) che prende in input un grafo g e gli identificatori di due nodi u e v e produce in output la lista 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 di u, cioè il nodo dal quale è stato raggiunto
     - la catena di parent che conduce da u a v è il cammino minimo cercato