Grafi (III parte)

Progettazione di Algoritmi a.a. 2019-20

Matricole congrue a 1

Docente: Annalisa De Bonis

1

1

Analisi di DFS implementata mediante uno stack

Assumiamo G rappresentato con liste di adiacenza

DFS(s):

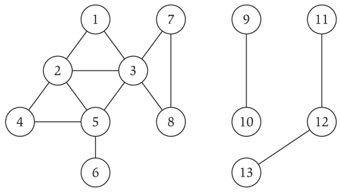
```
1.   Poni Explored[s] = true ed Explored[v] = false per tutti gli altri nodi   O(n)
2.   Inizializza  S con uno stack contenente s   O(1)
3.   While S non è vuoto
4.       Metti in u  il nodo  al top di S
5.       If  c'e` un arco (u, v) incidente su u non ancora esaminato then
6.           If Explored[v] = 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).

2

Componente connessa

- Componente connessa. Sottoinsieme di vertici tale per ciascuna coppia di vertici u e v esiste un percorso tra u e v
- Componente connessa contenente s. Formata da tutti i nodi raggiungibili da s



- Componente connessa contenente il nodo 1 è { 1, 2, 3, 4, 5, 6, 7, 8 }.

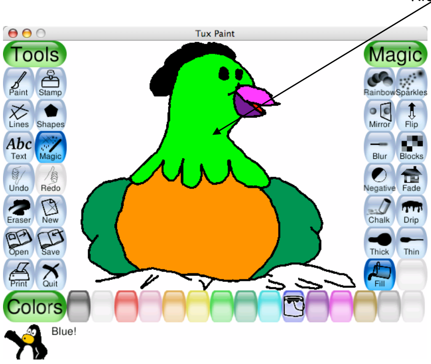
3

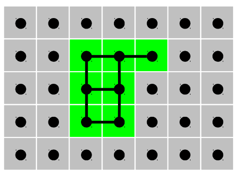
3

Flood Fill

- Flood fill. Data un'immagine, cambia il colore dell'area di pixel vicini di colore verde lime in blu.
  - Nodo: pixel.
  - Arco: due pixel vicini di colore verde lime.
  - Area di pixel vicini di colore verde lime: componente connessa di nodi associati a pixel verde lime.

Ricolora l'area verde lime di blue





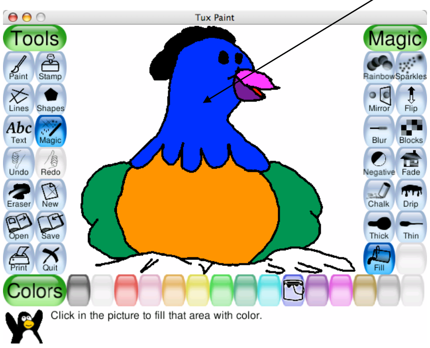
4

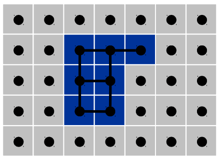
4

Flood Fill

- Flood fill. Data un'immagine, cambia il colore dell'area di pixel vicini di colore verde lime in blu.
  - Nodo: pixel.
  - Arco: due pixel vicini di colore verde lime.
  - Area di pixel vicini: componente connessa di pixel di colore verde lime.

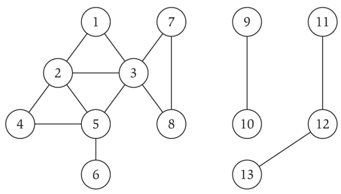
Ricolora l'area verde lime di blue





Componente connessa

- Componente connessa contenente s. Trova tutti i nodi raggiungibili da s
  - Come trovarla. Esegui BFS o DFS utilizzando s come sorgente
- Insieme di tutte le componenti connesse. Trova tutte le componenti connesse
  - Come trovarlo. Fino a quando ci sono nodi che non sono stati scoperti (esplorati), scegli uno di questi nodi ed esegui BFS (o DFS) su u utilizzando questo nodo come sorgente



Esempio: il grafo sottostante ha tre componenti connesse

Insieme di tutte componenti connesse

- Teorema. Per ogni due nodi s e t di un grafo, le loro componenti connesse o sono uguali o disgiunte
- Dim.
- Caso 1. Esiste un percorso tra s e t. In questo caso ogni nodo u raggiungibile da s è anche raggiungibile da t (basta andare da t ad s e da s ad u) e ogni ogni nodo u raggiungibile da t è anche raggiungibile da s (basta andare da s ad t e da t ad u). Ne consegue che un nodo u è nella componente connessa di s **se e solo** se è anche in quella di t e quindi le componenti connesse di s e t sono uguali.
- Caso 2. Non esiste un percorso tra s e t. In questo caso non può esserci un nodo che appartiene sia alla componente connessa di s che a quella di t. Se esistesse un tale nodo v questo sarebbe raggiungibile sia da s che da t e quindi potremmo andare da s a v e poi da v ad t. Ciò contraddice l'ipotesi che non c'è un percorso tra s e t.

Insieme di tutte componenti connesse

- Il teorema precedente implica che le componenti connesse di un grafo sono a due a due disgiunte.
- Algoritmo per trovare l'insieme di tutte le componenti connesse

```
AllComponents(G)
 Per ogni nodo u di G setta discovered[u]=false
 For each node u of G
    If Discovered[u] = false
      BFS(u)
   Endif
 Endfor
```

- BFS modificata in modo tale che nella fase di inizializzazione non vengano settati a False le entrate dell'array Discovered
- Al posto della BFS possiamo usare la DFS e al posto dell'array Discovered l'array Explored

Insieme di tutte componenti connesse: analisi

- Indichiamo con k il numero di componenti connesse
- Indichiamo con n_i e con m_i rispettivamente il numero di nodi e di archi della componente i-esima
- L'esecuzione della visita BFS o DFS sulla componente i-esima richiede tempo $O(n_i+m_i)$
- Il tempo totale richiesto da tutte le visite BFS o DFS e`

$$\sum_{i=1}^{k} O(n_i+m_i)=O(\sum_{i=1}^{k}(n_i+m_i))$$

- Poiche' le componenti sono a due a due disgiunte, si ha che

$$\sum_{i=1}^{k}(n_i+m_i)=n+m$$

- e il tempo totale di esecuzione dell'algoritmo che scopre le componenti connesse e` O(n)+O(n+m)=O(n+m)

9

Insieme di tutte componenti connesse: alcune considerazioni

- Se l'algoritmo utilizza BFS allora BFS deve essere modificata in modo che non resetti a false ogni volta i campi discovered.
- E` possibile modificare AllComponents in modo che assegni a ciascun nodo la componente di cui fa parte. A questo scopo usiamo:
  - contatore delle componenti.
  - array Component t.c. Component[u] = j se u appartiene alla componente j-esima.

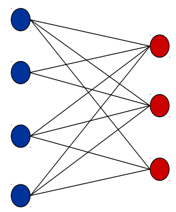
  - Esercizio: modificare lo pseudocodice dell'algoritmo AllComponents in modo che assegni a ciascun nodo la componente di cui fa parte. Ricordatevi che occorre modificare anche l'algoritmo di visita invocato da AllComponents.

10

Grafi bipartiti

- Def. Un grafo non direzionato è **bipartito** se l'insieme di nodi può essere partizionato in due sottoinsiemi X e Y tali che ciascun arco del grafo ha una delle due estremità in X e l'altra in Y
  - Possiamo colorare i nodi con due colori (ad esempio, rosso e blu) in modo tale che ogni arco ha un'estremita rossa e l'altra blu.

- Applicazioni.
  - Matrimoni stabili: uomini = rosso, donna = blu.
  - Scheduling: macchine = rosso, job = blu.

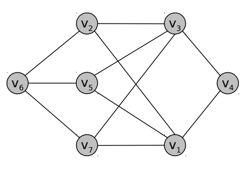


Un grafo bipartito

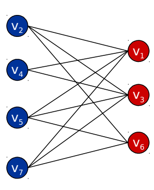
11

Testare se un grafo è bipartito

- Testare se un grafo è bipartito. Dato un grafo G, vogliamo scoprire se è bipartito.
  - Molti problemi su grafi diventano:
    - Più facili se il grafo sottostante è bipartito (matching: sottoinsieme di archi tali che non hanno estremità in comune)
    - Trattabili se il grafo è bipartito (max insieme indipendente)



Un grafo bipartito G

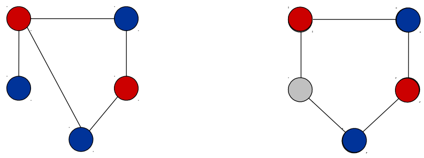


Modo alternativo di disegnare G

12

Grafi bipartiti

- Lemma. Se un grafo *G* è bipartito, non può contenere un ciclo dispari (formato da un numero dispari di archi)
- Dim. Non è possibile colorare di rosso e blu i nodi su un ciclo dispari in modo che ogni arco abbia le estremità di diverso colore.



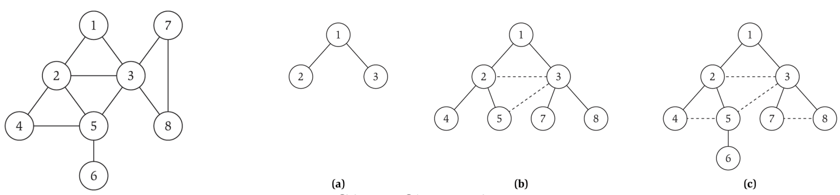
bipartito (2-colorabile)

Non bipartito (non 2-colorabile)

13

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 $i \le j$. 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 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** prima di x). Quindi in questo caso j= i o 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.

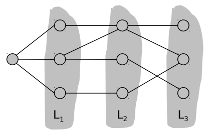


G (a) (b) (c)

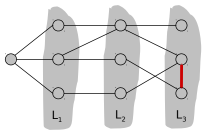
14

Grafi bipartiti

- Osservazione. Sia G un grafo connesso e siano $L_0, \ldots, L_k$ i livelli prodotti da un'esecuzione di BFS a partire dal nodo s. Può avvenire o che si verifichi la (i) o la (ii)
  - (i) Nessun arco di G collega due nodi sullo stesso livello
  - (ii) Un arco di G collega due nodi sullo stesso



Caso (i)

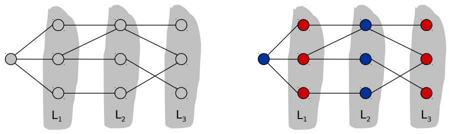


Caso (ii)

15

Grafi Bipartiti

- Nel caso (i) il grafo è bipartito.
- Dim.
- Per la proprietà sulla distanza tra livelli contenenti nodi adiacenti, si ha che due nodi adiacenti o si trovano nello stesso livello o in livelli consecutivi.
- Poiché nel caso (i) non ci sono archi tra nodi di uno stesso livello allora tutti gli archi del grafo collegano nodi in livelli consecutivi.
- Quindi se coloro i livelli di indice dispari di rosso e quelli di indice pari di blu, ho che le estremità di ogni arco sono di colore diverso.

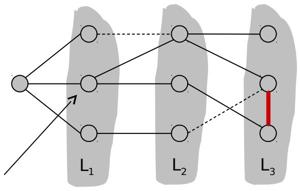


Caso (i)

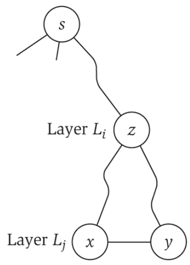
16

Grafi Bipartiti

Nel caso (ii) il grafo non è bipartito.
Dim. Dimostriamo che il grafo contiene un ciclo dispari: supponiamo che esista l'arco (x,y) tra due vertici x e y di L_j. Indichiamo con z l'antenato comune a x e y nell'albero BFS che si trova più vicino a x e y. Sia L_i il livello in cui si trova z. Possiamo ottenere un ciclo dispari del grafo prendendo il percorso seguito dalla BFS da z a x (j-i archi), quello da z a y (j-i archi) e l'arco (x,y). In totale il ciclo contiene 2(j-i)+1 archi.



Antenato comune più vicino (lowest common ancestor)



Caso (ii)

17

Algoritmo che usa BFS per determinare se un grafo è bipartito

Modifichiamo BFS come segue:

- Usiamo un array Color per assegnare i colori ai nodi
- Ogni volta che aggiungiamo un nodo v alla lista L[i+1] poniamo Color[v] uguale a rosso se i+1 è pari e uguale a blu altrimenti
- Alla fine esaminiamo tutti gli archi per vedere se c'è ne è uno con le estremità dello stesso colore . Se c'è concludiamo che G non è bipartito (perche'?); altrimenti concludiamo che G è bipartito (perche'?).

- Tempo: O(n+m)

18