

Grafi (IV parte)

Progettazione di Algoritmi a.a. 2023-24

Matricole congrue a 1

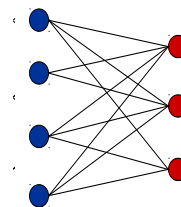
Docente: Annalisa De Bonis

29

29

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'estremità rossa e l'altra blu.
- **Applicazioni.**
 - Scheduling: macchine = rosso, job = blu.



Un grafo bipartito

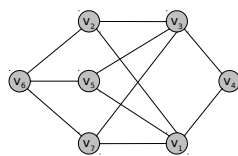
PROGETTAZIONE DI ALGORITMI a.a. 2021-22
A. DE BONIS

30

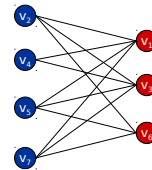
30

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

Se volessimo considerare tutti i possibili modi di colorare i nodi con due colori dovremmo considerare 2^{n-1} possibilità: per ogni sottoinsieme U di V possiamo colorare i nodi di U di rosso e i nodi di $V-U$ di blu $\rightarrow 2^n$ coppie $(U, V-U) \rightarrow 2^n$ modi di colorare i vertici con due colori. Siccome non occorre considerare entrambe le coppie $(U, V-U)$ e $(V-U, U)$ in quanto cio' corrisponderebbe a scambiare i colori dei due insiemi \rightarrow numero colorazioni da esaminare $= 2^n / 2 = 2^{n-1}$

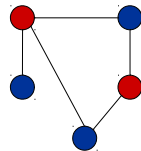
PROGETTAZIONE DI ALGORITMI o.a. 2021-22
A. DE BONIS

31

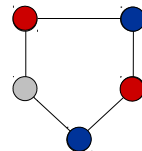
31

Grafi bipartiti

- **Lemma.** Se un grafo G è bipartito, non può contenere un ciclo dispari (formato da un numero dispari di archi)
- In pratica vale l'implicazione: G bipartito \rightarrow nessun ciclo dispari in G
- **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)

PROGETTAZIONE DI ALGORITMI o.a. 2021-22
A. DE BONIS

32

32

Breadth First Search Tree

- Vi ricordate questa proprietà?
- **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.
- Sfatteremo questa proprietà per provare che la BFS, nella versione con i layer, può essere utilizzata per determinare se un grafo è bipartito

G

PROGETTAZIONE DI ALGORITMI a.a. 2021-22
A. DE BONIS

33

Grafì bipartiti

- **Osservazione.** Sia G un grafo connesso e siano L_0, \dots, 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 livello

Caso (i) Caso (ii)

PROGETTAZIONE DI ALGORITMI a.a. 2021-22
A. DE BONIS

34

Grafì 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)

PROGETTAZIONE DI ALGORITMI a.a. 2021-22
A. DE BONIS

35

Grafì 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)

PROGETTAZIONE DI ALGORITMI a.a. 2021-22
A. DE BONIS

36

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 (perché?); altrimenti concludiamo che *G* è bipartito (perché?).
- Tempo: $O(n+m)$

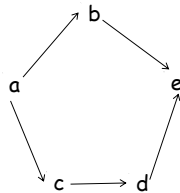
PROGETTAZIONE DI ALGORITMI a.a. 2021-22
A. DE BONIS

37

37

Esercizio

Fornire tutti gli ordinamenti topologici del grafo sottostante



Soluzione. Potremmo esaminare le $5! = 120$ possibili permutazioni...
Ragioniamo: il primo nodo dell'ordinamento non deve avere archi entranti, l'ultimo non deve avere archi uscenti. Gli unici nodi che rispettivamente soddisfano questi requisiti sono *a* ed *e*. Quindi ogni ordinamento topologico deve cominciare con *a* e finire con *e*. In quanti modi possono essere sistemati gli altri nodi? Osserviamo che l'arco (c,d) implica che *c* precede *d* in qualsiasi ordinamento topologico mentre *b* può trovarsi in una qualsiasi posizione tra *a* ed *e*. In totale, ci sono quindi 3 ordinamenti topologici
a b c d e , *a c b d e*, *a c d b e*

38

38

Esercizio 2 Cap 3

- Fornire un algoritmo che, dato un grafo non direzionato G , scopre se G contiene cicli e in caso affermativo produce in output uno dei cicli. L'algoritmo deve avere tempo di esecuzione $O(n+m)$
- **Soluzione 1.** Si esegua una visita BFS sul grafo. Se il grafo non è connesso si eseguono più visite, una per componente connessa. Se al termine gli alberi BFS contengono tutti gli archi di G allora G non contiene cicli. In caso contrario, c'è almeno un arco (x,y) che non fa parte degli alberi BFS. Consideriamo l'albero BFS T in cui si trovano x e y e sia z l'antenato comune più vicino a x e y (LCA di x e y). L'arco (x,y) insieme ai percorsi tra z e x e quello tra z e y forma un ciclo

Come facciamo a trovare lo LCA di x e y in tempo $O(n)$?

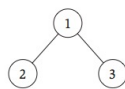
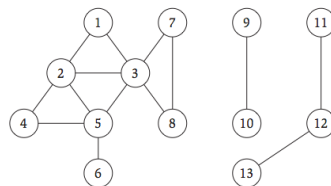
PROGETTAZIONE DI ALGORITMI a.a. 2023-24
A. DE BONIS

39

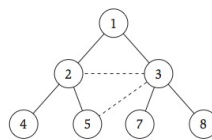
39

Esempio di esecuzione di BFS

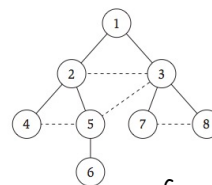
G



a



b



c

Notiamo che $(3,5)$ non appartiene a nessun albero BFS. L'antenato comune più vicino a 3 e 5 è 1 per cui il ciclo restituito è $5,2,1,3,5$ (ciclo pari). Gli archi $(2,3)$, $(4,5)$ e $(7,8)$ ci avrebbero fatto scoprire altri cicli (cicli dispari).

40

40

Esercizio 2 Cap 3

Soluzione 2. Si esegua una visita DFS sul grafo.

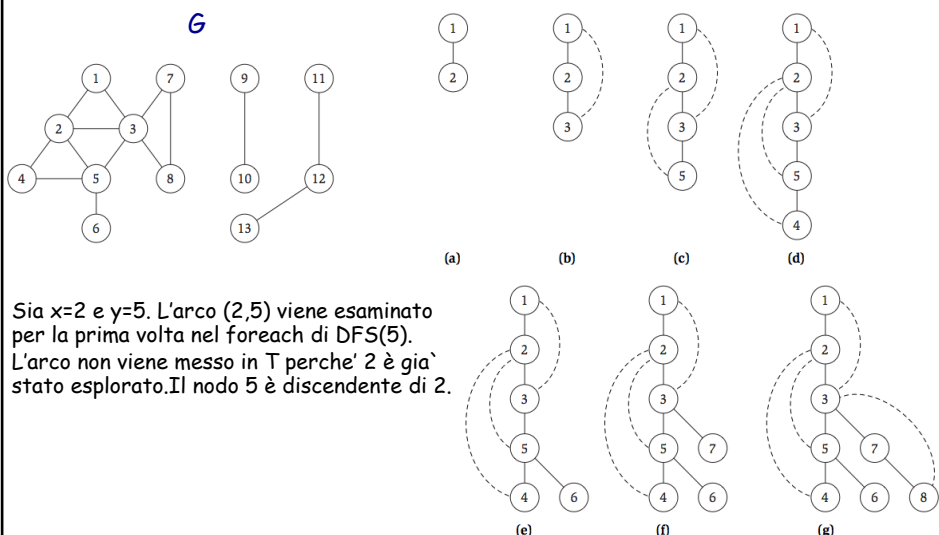
- Se il grafo non è connesso si eseguono più visite, una per componente connessa.
- Se al termine gli alberi DFS contengono tutti gli archi di G allora G non contiene cicli.
- In caso contrario, c'è almeno un arco (x,y) che non fa parte degli alberi DFS.
- Consideriamo l'albero DFS T in cui si trovano x e y . La Proprietà 2 degli alberi DFS ci dice che x e y sono uno discendente dell'altro.
- Se x è stato scoperto prima di y allora y è discendente di x in T (si veda dimostrazione II caso Proprietà 2). Per trovare il ciclo dobbiamo risalire il percorso da y a x aggiungendo man mano i nodi attraversati ad una lista ed infine aggiungendo y alla lista in modo che il ciclo cominci e finisca in y . (tempo $O(n)$). Se invece y è stato scoperto prima di x allora usiamo lo stesso procedimento sopra descritto ma questa risalendo il percorso da y a x e aggiungendo alla fine x alla lista.

PROGETTAZIONE DI ALGORITMI a.a. 2023-24
A. DE BONIS

41

41

Esempio di albero DFS



42

42

Esercizio 2 Cap 3

Come facciamo a capire se x è discendente di y o viceversa?

1 risposta:

Non mi serve capirlo in anticipo. Infatti si potrebbe risalire il percorso da y alla radice fino a che non si arriva ad x . Se questo percorso non passa per x allora si risale il percorso da x alla radice fino a che non si arriva ad y .

In entrambi i casi i vertici incontrati vengono man mano aggiunti in una lista. Se nel risalire il percorso da y alla radice si incontra x allora si aggiunge y alla fine della lista e la si restituisce in output. In caso contrario, questa lista viene scartata e si restituisce la lista con i nodi incontrati risalendo da x ad y e con l'aggiunta di x alla fine.

Tempo: $O(n)$

43

Esercizio 2 Cap 3

Come facciamo a capire se x è discendente di y o viceversa?

2 risposta:

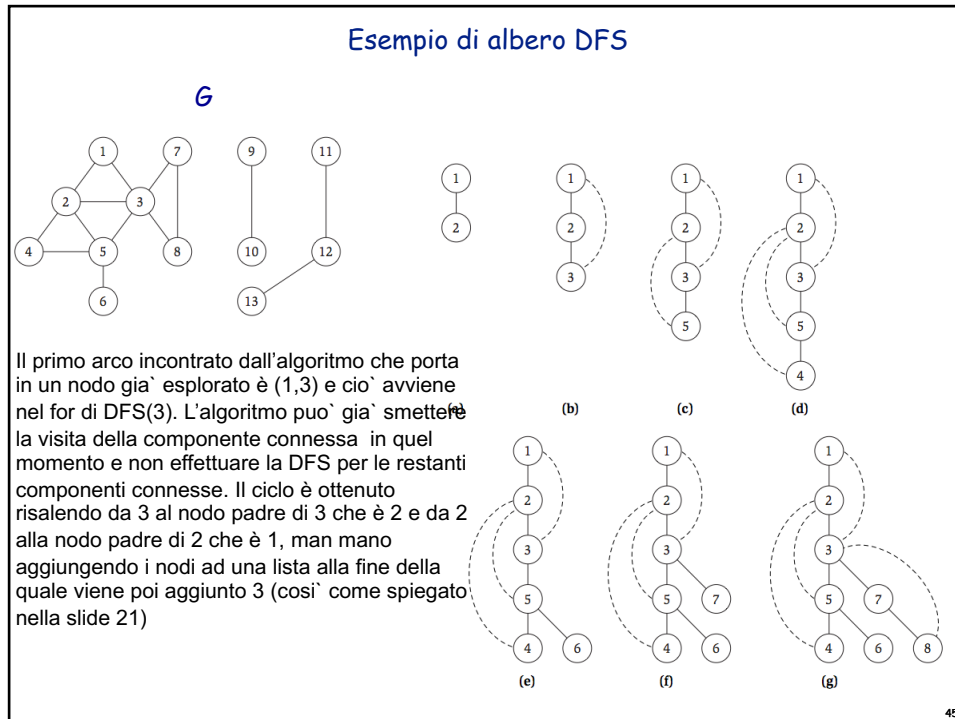
se la prima volta che viene scandito l'arco (x,y) , cio' avviene nel foreach di DFS(y) allora y è discendente di x , altrimenti x è discendente di y .

L'intero algoritmo puo' quindi essere descritto nel seguente modo:

- inizializza tutti i nodi come non esplorati
- fino a quando ci sono nodi non esplorati esegue una versione modificata di DFS su uno dei nodi non ancora esplorati: la versione modificata di DFS restituisce
 - una coppia di nodi u,v tali che u e v compaiono in un ciclo direzionato della componente connessa esplorata dalla DFS e u è discendente di v nell'albero DFS (la coppia puo' essere mantenuta in un lista che ha u come primo elemento e v come secondo elemento);
 - oppure una lista vuota, nel caso in cui la componente connessa esplorata dalla DFS non contenga un ciclo.

Se una delle DFS invocate dall'algoritmo restituisce una coppia di nodi allora l'algoritmo smette di invocare la DFS sui nodi non ancora esplorati e produce in output il ciclo ottenuto come descritto nella slide 21 partendo dal primo vertice della coppia.

44



45

Esercizio 2 Cap 3

Esercizio: scrivere lo pseudocodice della versione modificata di DFS per la soluzione 2.

```

DFSMOD(u):
  ponì explored[u]=true
  for each (u,v) incident to u
    if explored[v]==true and v≠u //c'è un ciclo che contiene u e v
      L ← list with u and v in the first and second positions, respectively
      return L
    if explored[v]==false //il nodo non v è stato già esplorato
      parent[v]=u //equivale a inserire (u,v) nell'albero DFS
      L=DFSMOD(v)
      if L is not empty //la DFS su v ha individuato un ciclo
        return L //restituisce la lista contenente due nodi del ciclo
  end for
  L← empty list
  return L

```

la visita termina non appena incontra un arco che porta in nodo già esplorato diverso da u (primo if) oppure non appena una delle DFS invocate nel for restituisce una lista non vuota (if)

PROGETTAZIONE DI ALGORITMI a.a. 2023-24
A. DE BONIS

46

46

Esercizio 3 Cap. 3

- Modificare l'algoritmo per l'ordinamento topologico di un DAG in modo tale che se il grafo direzionato input non è un DAG l'algoritmo riporta in output un ciclo che fa parte del grafo.

Soluzione. Consideriamo l'algoritmo per l'ordinamento topologico e supponiamo di invocarlo su un grafo G non necessariamente aciclico

- **Caso 1.** Ogni volta che l'algoritmo viene invocato ricorsivamente su un grafo non vuoto, l'insieme S non è vuoto. In questo caso riusciamo ad ottenere un ordinamento topologico perché ogni nodo cancellato v non ha archi entranti che provengono dai nodi che sono ancora attivi e che quindi saranno posizionati nell'ordinamento dopo v . Il Lemma ci dice che se il grafo ha un ordinamento topologico allora il grafo è un DAG.
- **Caso 2.** All'inizio di una certa chiamata ricorsiva su un grafo non vuoto, si ha che S è vuoto. In questo caso il grafo formato dai nodi attivi non è un DAG per il lemma che dice che un DAG ha almeno un nodo senza archi entranti. Il ciclo è ottenuto percorrendo a ritroso gli archi a partire da un qualsiasi nodo attivo v fino a che non incontriamo uno stesso nodo w due volte.

PROGETTAZIONE DI ALGORITMI a.a. 2023-24
A. DE BONIS

Continua nella
prossima slide

47

47

Esercizio 3 Cap. 3

- Basta quindi modificare l'algoritmo in modo che se all'inizio di una chiamata ricorsiva si ha che se G non è vuoto ed S è vuoto allora l'algoritmo sceglie un nodo attivo v e comincia a percorrere gli archi a ritroso a partire da v : si sceglie un arco (x,v) nella lista degli archi entranti in v , poi si sceglie un arco (y,x) nella lista degli archi entranti in x e così via.
 - NB: stiamo supponendo di mantenere per ogni nodo anche la lista degli archi entranti nel nodo
- Ogni volta che viene attraversato un arco (p,q) a ritroso, il nodo p raggiunto viene inserito all'inizio di una lista a doppi puntatori ed etichettato come visitato.
- Se ad un certo punto si raggiunge un nodo w già etichettato come visitato, l'algoritmo interrompe questo percorso all'indietro e cancella dalla lista tutti i nodi a partire dalla fine della lista fino a che incontra per la prima volta w .
- I nodi restanti nella lista formano un ciclo direzionato che comincia e finisce in w .
- Tempo $O(n+m)$ in quanto l'algoritmo per l'ordinamento ha costo $O(n+m)$ e il costo aggiuntivo per trovare il ciclo è $O(n)$.

PROGETTAZIONE DI ALGORITMI a.a. 2023-24
A. DE BONIS

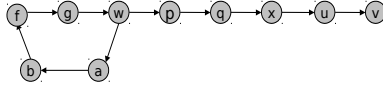
Continua nella
prossima slide

48

48

Esercizio 3 Cap. 3

Esempio di grafo con ciclo



Se cominciamo il cammino a ritroso a partire da v , la lista dei nodi attraversati è $w a b f g w p q x u v$ (aggiungiamo ogni nodo attraversato all'inizio della lista). Non appena incontriamo la seconda occorrenza di w , ci fermiamo e cancelliamo gli ultimi 5 nodi della lista scandendo la lista a partire dalla fine. I nodi che rimangono nella lista formano il ciclo $w a b f g w$.

PROGETTAZIONE DI ALGORITMI a.a. 2023-24
A. DE BONIS

49

49

Esercizio 7 Cap. 3

Dimostrare o confutare la seguente affermazione:

Sia G un grafo non direzionato con un numero n pari di vertici e in cui ogni vertice ha grado almeno $n/2$. G è connesso.

Soluzione:

- L'affermazione è vera.
- Dimostrazione. Immaginiamo di eseguire la BFS su G a partire da un certo nodo u . Dimostreremo che la BFS, a partire da u , raggiunge tutti i nodi di G .
- * Siccome u ha grado almeno $n/2 \rightarrow$ nel livello L_1 ci saranno almeno $n/2$ nodi
- Se tutti i nodi diversi da u sono in L_1 allora il grafo è ovviamente connesso. Supponiamo che esista un vertice v diverso da u che non è in L_1 .
- Siccome v ha grado almeno $n/2$ e non è adiacente ad u (altrimenti v sarebbe in L_1) \rightarrow almeno uno degli archi incidenti su v deve incidere su un vertice che si trova nel livello L_1 .
 - Semplice argomento: se escludiamo u ed i nodi nel livello L_1 , rimangono $n-1-|L_1|$ nodi che per la * sono al più $n-1-n/2 = n/2-1$.
- Quindi ogni vertice v diverso da u e che non è adiacente ad u deve essere adiacente ad un nodo di $L_1 \rightarrow$ ogni nodo che non è ne' in L_0 ne' in L_1 viene inserito nella BFS nel livello $L_2 \rightarrow G$ è connesso

50

Esercizio 6 Cap. 3

Sia G un grafo connesso tale che il DFS tree e il BFS tree di G sono uguali allo stesso albero T . Dimostrare che $G = T$ (cioè non ci sono archi di G che non sono inclusi in T).

Soluzione:

Dimostrazione. Supponiamo per assurdo che esista un arco (x,y) di G che non è in T .

1. T è un BFS tree \rightarrow i livelli di x e y in T differiscono al più di 1
2. T è un DFS tree $\rightarrow x$ è discendente di y in T o y è discendente di x in T
3. per ipotesi assurda (x,y) non è in $T \rightarrow x$ e y non sono in relazione padre figlio.
 - Dalla 2. sappiamo che x e y sono uno discendente dell'altro in T e quindi non possono essere sullo stesso livello. La 1. e la 2. insieme allora implicano che x e y sono in livelli consecutivi e sono uno discendente dell'altro. Ciò è possibile solo se x e y sono in relazione padre figlio.
 - Siamo arrivati a contraddire la 3. e quindi non è possibile che esista un arco di G che non è in T .

51

Individuazione di un ciclo in un grafo direzionato con DFS

Nel caso di un grafo direzionato, imbattersi in un nodo già incontrato (esplorato) non indica che vi sia necessariamente un ciclo.

Nel grafo in basso l'arco (u,v) ci riporta in v che è stato già esplorato ma l'arco (u,v) non fa parte di un ciclo.

Quali archi ci fanno capire che ci sono dei cicli?

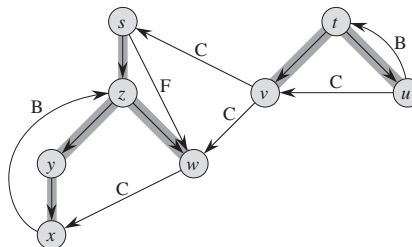
Gli archi (x,z) tali che x è discendente di z nell'albero DFS.

In altre parole x è esplorato prima che si concluda la chiamata $DFS(z)$, come per i nodi x e z in figura.

Per come abbiamo scritto l'implementazione di DFS con lo stack, i nodi lungo il percorso tra z e x sono quelli che si trovano tra x e z (direzione top-bottom) nello stack durante la chiamata ricorsiva su x .

Possiamo aggiungere alla DFS iterativa un array **isinstack** tale che **isinstack[v] = true** se e solo v è nello stack. Settiamo **isinstack[v]** a true quando v viene esplorato e quindi inserito nello stack e lo settiamo di nuovo a false quando facciamo il pop di v dallo stack. Nel foreach di $DFS(x)$, ci accorgiamo che (x,z) ci porta in z e che z è ancora nello stack.

Facciamo quindi i pop dei nodi dallo stack fino a che non arriviamo a z .
Inseriamo via via i nodi estratti in testa ad una lista e alla fine aggiungiamo x in testa.
Nell'esempio la lista conterra' x,z,y,x



52