

Grafi (III parte)

Progettazione di Algoritmi a.a. 2020-21

Matricole congrue a 1

Docente: Annalisa De Bonis

1

1

Visita di grafi direzionati

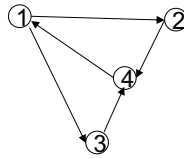
- **Raggiungibilità con direzione.** Dato un nodo s , trova tutti i nodi raggiungibili da s .
- **Il problema del più corto percorso diretto da s a t .** Dati due nodi s e t , qual è la lunghezza del percorso più corto da s a t ?
- **Visita di un grafo.** Le visite BFS e DFS si estendono naturalmente ai grafi direzionati.
 - Quando si esaminano gli archi incidenti su un certo vertice u , si considerano solo quelli uscenti da u .
- **Web crawler.** Comincia dalla pagina web s . Trova tutte le pagine raggiungibili a partire da s , sia direttamente che indirettamente.

2

2

Connettività forte

- **Def.** I nodi u e v sono **mutualmente raggiungibili** se c'è un percorso da u a v e anche un percorso da v a u .
- **Def.** Un grafo in cui ogni coppia di nodi è mutualmente raggiungibile si dice **fortemente connesso**

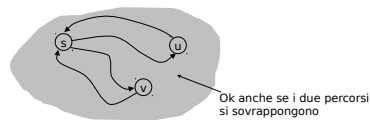


PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

3

Connettività forte

- **Lemma.** Sia s un qualsiasi nodo di un grafo direzionato G . G è fortemente connesso se e solo se ogni nodo è raggiungibile da s ed s è raggiungibile da ogni nodo.
- **Dim.** \Rightarrow Segue dalla definizione.
- **Dim.** \Leftarrow Un percorso da u a v si ottiene concatenando il percorso da u ad s con il percorso da s a v . Un percorso da v ad u si ottiene concatenando il percorso da v ad s con il percorso da s ad u .



PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

4

Algoritmo per la connettività forte

Teorema. Si può determinare se G è fortemente connesso in tempo $O(m + n)$.

Dim.

- Prendi un qualsiasi nodo s .
- Esegui la BFS con sorgente s in G .
- Crea il grafo G^{rev} invertendo la direzione di ogni arco in G .
- Esegui la BFS con sorgente s in G^{rev} .
- Restituisci true se e solo se tutti i nodi di G vengono raggiunti in entrambe le esecuzioni della BFS.
- La correttezza segue dal lemma precedente.
 - La prima esecuzione trova i percorsi da s a tutti gli altri nodi
 - La seconda esecuzione trova i percorsi da tutti gli altri nodi ad s perchè avendo invertito gli archi un percorso da s a u è di fatto un percorso da u ad s nel grafo di partenza.

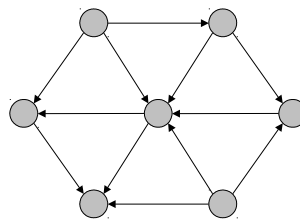
PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

5

5

Grafi direzionati aciclici (DAG)

- **Def.** Un **DAG** è un grafo direzionato che non contiene cicli direzionati
- Possono essere usati per esprimere vincoli di precedenza o dipendenza: l'arco (v_i, v_j) indica che v_i deve precedere v_j o che v_j dipende da v_i
- Infatti generalmente i grafi usati per esprimere i suddetti vincoli sono privi di cicli
- **Esempio.** Vincoli di precedenza: grafo delle propedeuticità degli esami



Un DAG G

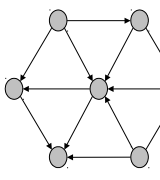
PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

6

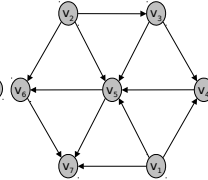
6

Ordine topologico

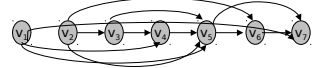
- Def.** Un **ordinamento topologico** di un grafo direzionato $G = (V, E)$ è un'etichettatura dei suoi nodi v_1, v_2, \dots, v_n tale che se G contiene l'arco (v_i, v_j) si ha $i < j$. Detto in un altro modo, un **ordinamento topologico di G** è un **ordinamento dei nodi di G** tale che se c'è l'arco (u, w) in G , allora il vertice u precede il vertice w nell'ordinamento (tutti gli archi puntano in avanti nell'ordinamento).
- Esempio.** Nel caso in cui un grafo direzionato G rappresenti le propedeuticità degli esami, un ordinamento topologico indica un possibile ordine in cui gli esami possono essere sostenuti dallo studente.



Un DAG G



Un ordinamento topologico di G



Un modo diverso di ridisegnare G in modo da evidenziare l'ordinamento topologico di G

PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

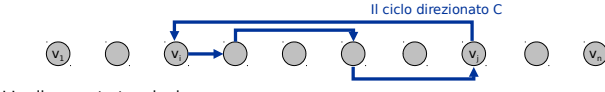
7

DAG e ordinamento topologico

Lemma. Se un grafo direzionato G ha un ordinamento topologico allora G è un DAG.

Dim. (per assurdo)

- Supponiamo che G sia un grafo direzionato e che abbia un ordinamento v_1, \dots, v_n . Supponiamo per assurdo che G non sia un DAG ovvero che abbia un ciclo direzionato C . Vediamo cosa accade.
- Consideriamo i nodi che appartengono a C e tra questi sia v_i quello con indice più piccolo e sia v_j il vertice che precede v_i nel ciclo C . Ciò ovviamente implica che (v_j, v_i) è un arco.
- Siccome (v_j, v_i) è un arco e v_1, \dots, v_n è un ordinamento topologico allora, deve essere $j < i$.
- $j < i$ è impossibile in quanto abbiamo scelto v_i come il vertice di indice più piccolo in C e di conseguenza vale $i < j$. Siamo arrivati ad un assurdo. Cioè una contraddizione al fatto che G contiene un ciclo.



Il ciclo direzionato C

L'ordinamento topologico v_1, \dots, v_n

PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

8

DAG e ordinamento topologico

- Abbiamo visto che se G ha un ordinamento topologico allora G è un DAG.
- **Domanda.** è vera anche l'implicazione inversa? Cioè dato un DAG, è sempre possibile trovare un suo ordinamento topologico?
- E se sì, come trovarlo?

PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

9

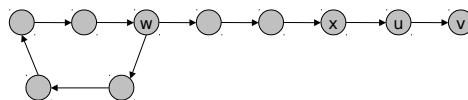
9

DAG e ordinamento topologico

Lemma. Se G è un DAG allora G ha un nodo senza archi entranti

Dim. (per assurdo)

- Supponiamo che G sia un DAG e che ogni nodo di G abbia almeno un arco entrante. Vediamo cosa succede.
- Prendiamo un qualsiasi nodo v e cominciamo a seguire gli archi in senso contrario alla loro direzione a partire da v . Possiamo farlo perchè ogni nodo ha un arco entrante: v ha un arco entrante (u,v) , il nodo u ha un arco entrante (x,u) e così via.
- Possiamo continuare in questo modo per quante volte vogliamo. Immaginiamo di farlo per n o più volte. Così facendo attraversiamo a ritroso almeno n archi e di conseguenza passiamo per almeno $n+1$ vertici. Ciò vuol dire che c'è un vertice w che viene incontrato almeno due volte e quindi deve esistere un ciclo direzionato C che comincia e finisce in w



10

10

DAG e ordinamento topologico

Lemma. Se G è un DAG, G ha un ordinamento topologico.

Dim. (induzione su n)

- **Caso base:** vero banalmente se $n = 1$.
- **Passo induttivo:** supponiamo asserto del lemma vero per DAG con $n \geq 1$ nodi
- Dato un DAG con $n+1 > 1$ nodi, prendiamo un nodo v senza archi entranti (abbiamo dimostrato che un tale nodo deve esistere).
- $G - \{v\}$ è un DAG, in quanto cancellare un nodo non introduce cicli nel grafo.
- Poiché $G - \{v\}$ è un DAG con n nodi allora, per ipotesi induttiva, $G - \{v\}$ ha un ordinamento topologico.
- Consideriamo l'ordinamento dei nodi di G che si ottiene mettendo v all'inizio dell'ordinamento e aggiungendo gli altri nodi nell'ordine in cui appaiono nell'ordinamento topologico di $G - \{v\}$.
- Siccome v non ha archi entranti tutti i suoi archi sono archi uscenti e ovviamente puntano verso nodi di $G - \{v\}$. Quello che si ottiene è un ordinamento topologico (tutti gli archi puntano in avanti).

PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

11

Algoritmo per l'ordinamento topologico

- La dimostrazione per induzione che abbiamo appena visto suggerisce un algoritmo ricorsivo per trovare l'ordinamento topologico di un DAG.

G : DAG

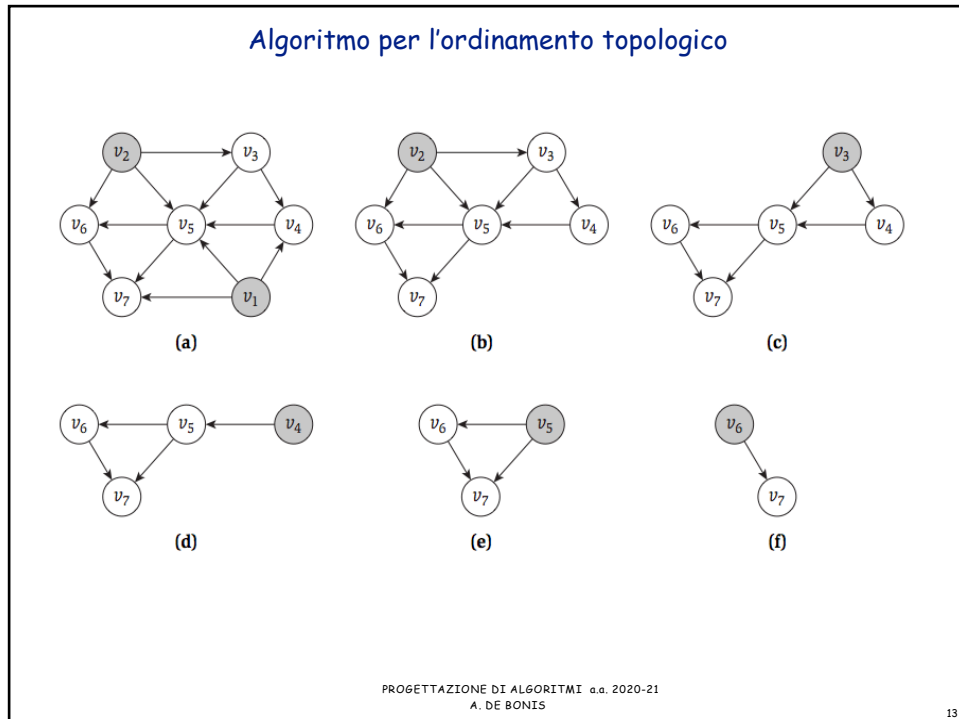
```

TopologicalOrder(G)
  if esiste nodo v senza archi entranti
    cancella v da G in modo da ottenere G-{v}
    L=TopologicalOrder(G-{v})
    aggiungi v all'inizio di L
    return L
  endif
  else //siccome G è un DAG, l'else è eseguito solo se G vuoto
    return lista vuota
  
```

PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

12

12



13

Algoritmo per l'ordinamento topologico : analisi dell'algoritmo

- 1) Trovare un nodo senza archi entranti nell'if richiede $O(n)$ se per ogni nodo viene memorizzato il numero di archi entranti
- 2) Cancellare un nodo v da G richiede tempo proporzionale al numero di archi uscenti da v che è **al più** $\deg(v)$

- Se consideriamo tutte le n chiamate ricorsive il tempo è $O(n^2)$ per 1) e $O(m)$ per 2). Quindi il tempo di esecuzione è $O(n^2+m)=O(n^2)$

```

TopologicalOrder(G)
  if esiste nodo v senza archi entranti
    cancella v da G in modo da ottenere G-{v}
    L=TopologicalOrder(G-{v})
    aggiungi v all'inizio di L
    return L
  endif
  else
    return lista vuota

```

PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

14

Algoritmo per l'ordinamento topologico : analisi dell'algoritmo

- Possiamo anche scrivere la relazione di ricorrenza

$$T(n) \leq \begin{cases} c & \text{per } n=1 \\ T(n-1)+c'n & \text{per } n>1 \end{cases}$$

Lavoro ad ogni chiamata ricorsiva è $O(n+\text{deg}(v))=O(n)$, dove v è il nodo rimosso da G

che ha soluzione $T(n)=O(n^2)$

Metodo iterativo

$$T(n) \leq T(n-1)+c'n \leq T(n-2)+c'(n-1)+c'n \leq T(n-3)+c'(n-2)+c'(n-1)+c'n \leq \dots \leq$$

$$T(1) + c'2 + \dots + c'(n-1) + c'n \leq c + c'2 + \dots + c'(n-1) + nc' = c + c'n(n+1)/2 - c' = O(n^2)$$

Metodo di sostituzione. Ipotizziamo $T(n) \leq Cn^2$ per $n \geq n_0$, dove C ed n_0 sono costanti positive da determinare. Dimostriamo che la nostra intuizione è corretta utilizzando l'induzione.

Base induzione: $T(1) \leq c \leq 1^2C$ se $C \geq c$

Passo induttivo.

$T(n) \leq T(n-1)+c'n \leq C(n-1)^2+c'n = Cn^2+C-2Cn+c'n$ l'ultimo membro è $\leq Cn^2$ se $C \geq c'$

Basta prendere $C = \max\{c, c'\}$ e $n_0=1$

PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

15

15

Algoritmo per l'ordinamento topologico con informazioni aggiuntive

- Il bound $O(n^2)$ non è molto buono se il grafo è sparso, cioè se il numero di archi è molto più piccolo di n^2
- Possiamo ottenere un bound migliore?
 - Per ottenere un bound migliore occorre usare un modo efficiente per individuare un nodo senza archi entranti ad ogni chiamata ricorsiva
 - Si procede nel modo seguente:
 - Un nodo si dice attivo se non è stato ancora cancellato
 - Occorre mantenere le seguenti informazioni:
 - per ciascun vertice attivo w
 - $\text{count}[w]$ = numero di archi entranti in w provenienti da nodi attivi.
 - S = insieme dei nodi attivi che non hanno archi entranti provenienti da altri nodi attivi.

PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

16

16

Algoritmo per l'ordinamento topologico con informazioni aggiuntive: analisi

Teorema. L'algoritmo trova l'ordinamento topologico di un DAG in tempo $O(m + n)$.

Dim.

- **Inizializzazione.** Richiede tempo $O(m + n)$ in quanto
 - I valori di $\text{count}[w]$ vengono inizializzati scandendo tutti gli archi e incrementando $\text{count}[w]$ per ogni arco entrante in w basta scandire tutti gli archi una sola volta \rightarrow tempo $O(m)$
 - Se per ogni nodo viene memorizzato il numero di archi entranti \rightarrow tempo $O(n)$
 - Inizialmente tutti i nodi sono attivi per cui S consiste dei nodi di G senza archi entranti ed è sufficiente esaminare $\text{count}[w]$ per tutti i nodi w una sola volta per inizializzare $S \rightarrow$ tempo $O(n)$

- **Aggiornamento.** Per trovare il nodo v da cancellare basta prendere un nodo da S . Per cancellare v occorre
 1. Cancellare v da S e da G . Cancellarlo da G costa $\text{deg}(v)$. Possiamo rappresentare S mediante una lista. Se cancelliamo ogni volta da S il primo nodo della lista \rightarrow tempo $O(1)$ (anche in una lista a puntatori singoli)
 2. Decrementare $\text{count}[w]$ per ogni arco (v,w) . Se $\text{count}[w]=0$ allora occorre aggiungere w a $S \rightarrow$ tempo $O(\text{deg}(v))$.

I passi 1. e 2. vengono eseguiti una volta per ogni vertice tutti gli aggiornamenti

vengono fatti in $\sum_{u \in V} O(1) + \sum_{u \in V} O(\text{deg}(u)) = O(n) + O(m) = O(n + m)$

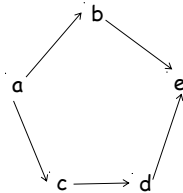
PROGETTAZIONE DI ALGORITMI a.a. 2020-21
A. DE BONIS

17

17

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

$a b c d e$, $a c b d e$, $a c d b e$

18

18

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.** 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 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. 2020-21
A. DE BONIS

19

19

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. 2020-21
A. DE BONIS

Continua nella
prossima slide

20

20

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. 2020-21
A. DE BONIS

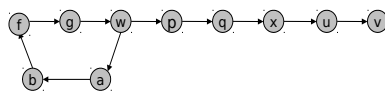
Continua nella
prossima slide

21

21

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. 2020-21
A. DE BONIS

22

22

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 né in L_0 né in L_1 viene inserito nella BFS nel livello $L_2 \rightarrow G$ è connesso

23

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 .

24