Grafi: nozioni elementari

Fulvio CORNO - Matteo SONZA REORDA
Dip. Automatica e Informatica
Politecnico di Torino

Sommario

- Definizioni
- Rappresentazione dei grafi
- Algoritmi di visita
- Esempi in C

Sommario

- Definizioni
- Rappresentazione dei grafi
- Algoritmi di visita
- Esempi in C

Grafo orientato

Un grafo *orientato* (o *diretto* o *di-grafo*) G è una coppia (V,E), dove

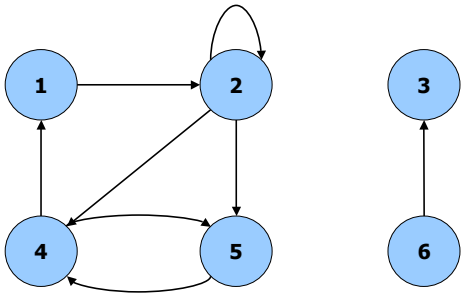
- V è l'insieme (finito) dei *vertici*
- E è l'insieme (finito) degli *archi*, che corrisponde ad una relazione binaria su V.

Rappresentazione

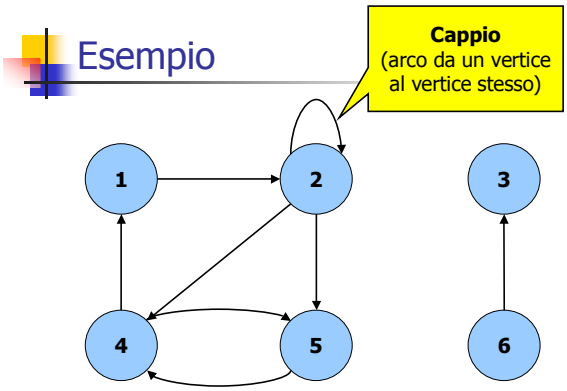
I grafi orientati si rappresentano spesso disegnando

- I vertici sotto forma di cerchi
- Gli archi sotto forma di frecce.

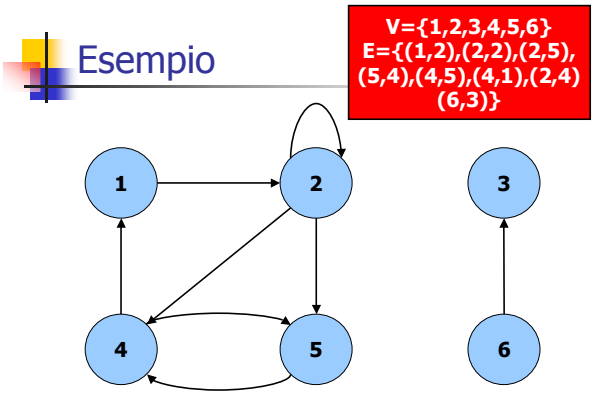
Esempio



Esempio



Esempio



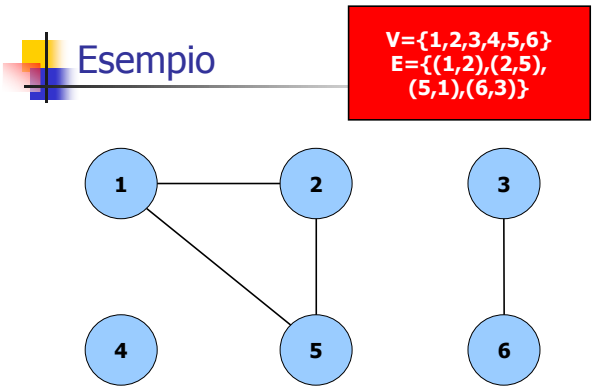
Grafo non orientato

Un grafo non orientato è ancora una coppia G=(V,E), ma l'insieme E è costituito in questo caso da coppie non ordinate di vertici.

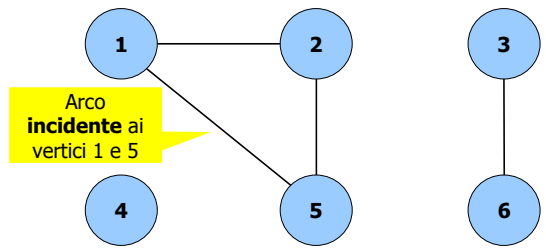
Nei grafi non orientati gli archi sono usualmente rappresentati con linee.

Nei grafi non orientati non sono ammessi cappi.

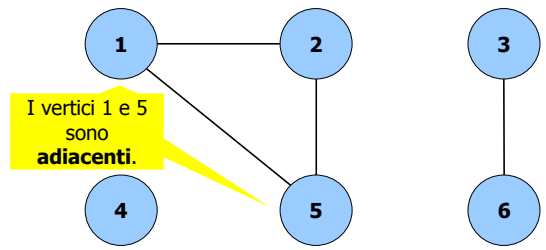
Esempio



Esempio



Esempio



Grado

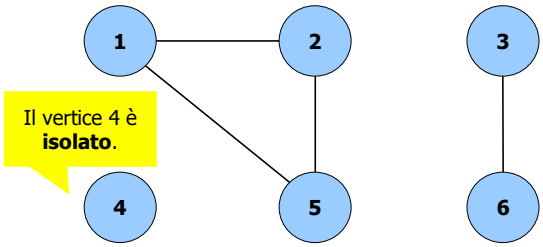
In un grafo non orientato il grado di un vertice è il numero di archi incidenti.

In un grafo orientato:

- Il grado entrante è il numero di archi entranti
- Il grado uscente è il numero di archi uscenti
- Il grado è la somma del grado entrante e del grado uscente.

Un vertice con grado 0 si dice isolato.

Esempio



Cammini

Un cammino da un vertice u ad un vertice u' in un grafo G=(V,E) è una sequenza di vertici $(v_0,v_1,v_2,\ldots,v_k)$ tale che $u=v_0$ e $u'=v_k$, con $(v_{i-1},v_i)\in E$ per $i=1,2,\ldots,k$.

k è la lunghezza del cammino.

Se esiste un cammino da u a u' si dice che u' è raggiungibile da u.

Un cammino è semplice se tutti i vertici che lo compongono sono distinti.

Cicli

Un ciclo è un cammino in cui $v_0=v_k$.

Un cappio è un ciclo di lunghezza 1.

Un grafo senza cicli si dice aciclico.

Raggiungibilità

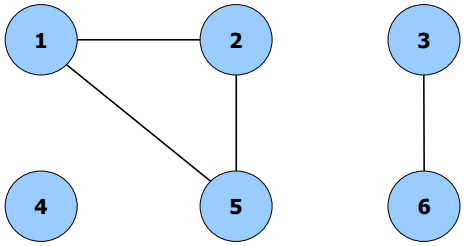
Un grafo non orientato è connesso se per ogni coppia di vertici esiste un cammino che la collega.

I sottografi connessi di dimensione massima si dicono componenti connesse.

Un grafo connesso è composto da una sola componente connessa.

Esempio

Il grafo ha 3 componenti connesse: {1,2,5}, {3,6}, {4}



Raggiungibilità

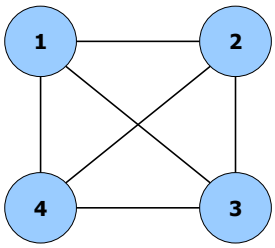
Un grafo orientato è fortemente connesso se per ogni coppia ordinata di vertici (u,u') esiste un cammino che collega u a u'.

Grafi completi

Un grafo si dice completo se data una qualsiasi coppia di vertici, questi sono adiacenti.

Esempio

Il grafo è completo



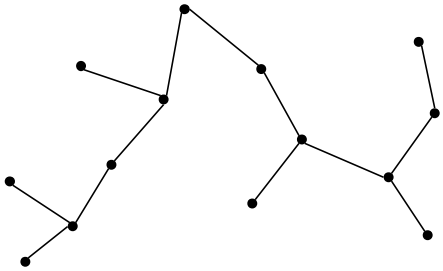
Alberi e foreste

Un grafo non orientato aciclico si dice foresta.

Un grafo non orientato aciclico connesso si dice albero.

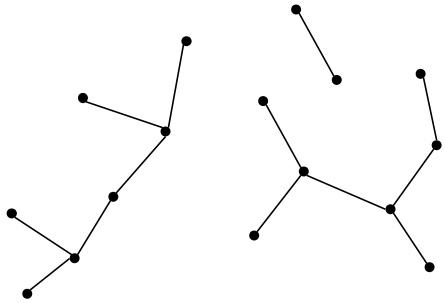
Esempio

Il grafo è un albero.



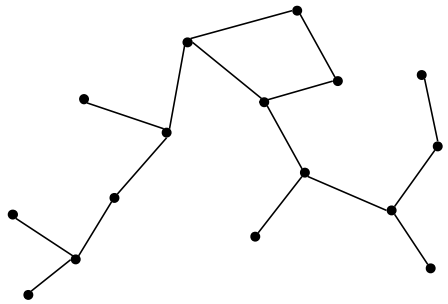
Esempio

Il grafo è una foresta.



Esempio

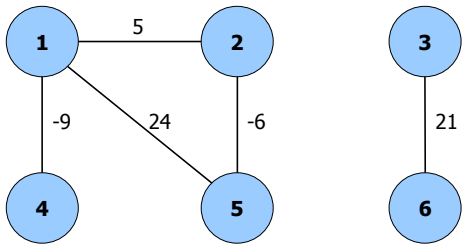
Il grafo non è né un albero né una foresta, in quanto contiene un ciclo.



Grafi pesati

Nei grafi pesati ad ogni arco è associato un peso.

Esempio



Esempi di applicazione

I grafi trovano applicazione in molti settori.
Si riportano nel seguito due esempi:

- Connessioni telefoniche
- Flow chart.

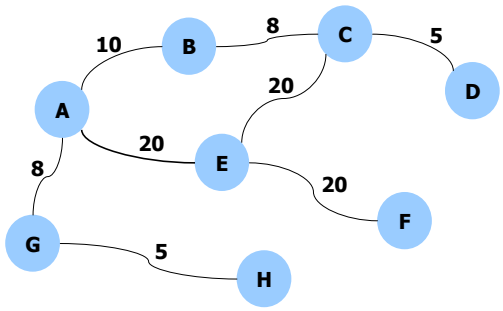
Connessioni telefoniche (1)

Una società telefonica gestisce un certo numero di centrali (a cui sono collegati gli utenti), collegate tra loro da canali dotati ciascuno di una certa banda.

Per rappresentare in ciascun istante la situazione, tenendo conto di eventuali guasti a canali e centrali, si può utilizzare un grafo non orientato pesato.

Il peso di ciascun arco corrisponde alla sua capacità (o banda).

Connessioni telefoniche (2)



Connessioni telefoniche (3)

Problemi:

- Il grafo è connesso? Se non lo è, ci saranno delle connessioni non possibili.
- Quali sono i canali *critici*, ossia tali per cui la mancanza del corrispondente arco rende il grafo non connesso?
- Quali sono le centrali *critiche*, ossia tali per cui la mancanza del corrispondente vertice (e degli archi incidenti) rende il grafo non connesso?

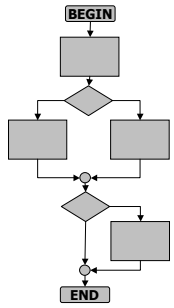
Flow chart (1)

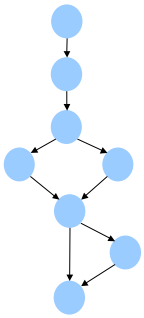
Nell'ingegneria del software è frequente l'uso di programmi per l'analisi del codice.

Tali programmi:

- Estraggono dal codice il flow chart
- Lo rappresentano sotto forma di un grafo orientato: i nodi sono i blocchi di istruzioni o le istruzioni condizionali
- Analizzano il comportamento del programma considerando tutti i possibili cammini nel grafo.

Flow chart (2)





Sommario

- Definizioni
- Rappresentazione dei grafi
- Algoritmi di visita
- Esempi in C

Modi di rappresentazione

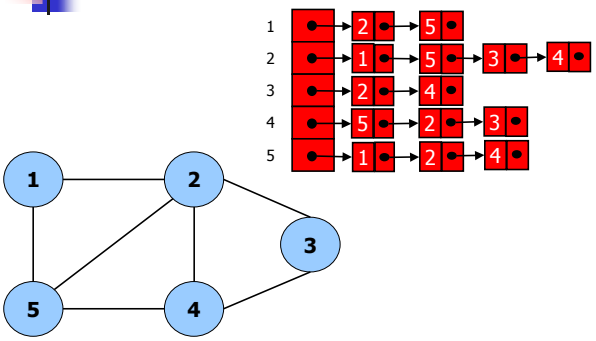
Esistono due modi principali per rappresentare un grafo G=(V,E):

- Tramite liste di adiacenza
- Tramite matrice di adiacenza.

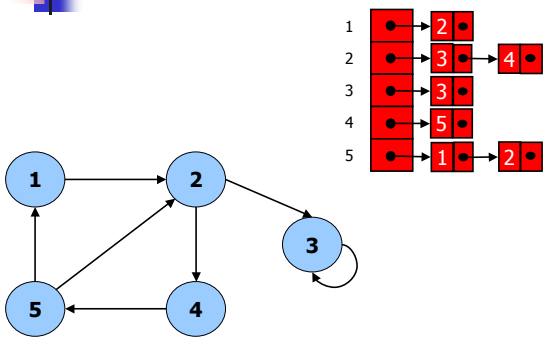
Liste di adiacenza

Dato un grafo G=(V,E), esso può essere rappresentato tramite un vettore A, il cui elemento A[i] contiene il puntatore alla lista dei vertici adiacenti al vertice i.

Esempio



Esempio



Uso della memoria

Se il grafo è orientato, il numero di elementi presenti nelle liste è pari a |E|.

Se il grafo è non orientato, il numero di elementi presenti nelle liste è pari a 2|E|.

La quantità di memoria richiesta è quindi O(max(V,E)).

Limiti

Il principale limite della rappresentazione tramite liste di adiacenza consiste nella difficoltà nel verificare se esiste un arco tra il vertice u ed il vertice u'.

In tal caso è necessario scandire l'intera lista dei vertici adiacenti a u (o a u').

Matrice di adiacenza

Consiste in una matrice di dimensione $|V|\times|V|$. L'elemento a_{ij} vale

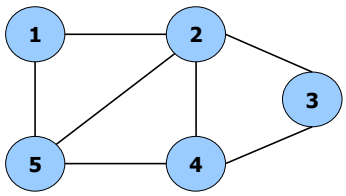
- 1 se $(i,j)\in E$
- 0 altrimenti.

Nel caso di grafi non orientati la matrice è simmetrica.

Nel caso di grafi pesati l'elemento a_{ij}, qualora non sia nullo, assume il valore del peso dell'arco.

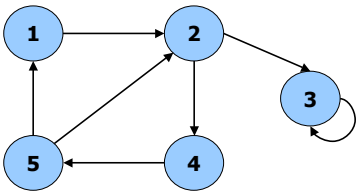
Esempio

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



Esempio

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	0	0	0	1
5	1	1	0	0	0



Confronto

Rispetto alla rappresentazione tramite liste di adiacenza, quella tramite matrice di adiacenza:

- Richiede più memoria (a meno che il grafo non sia particolarmente denso)
- Permette un accesso più efficiente alla topologia del grafo.

Sommario

- Definizioni
- Rappresentazione dei grafi
- Algoritmi di visita
- Esempi in C

Algoritmi di visita

L'operazione corrispondente ad esplorare un grafo a partire da un vertice dato (denominato *sorgente*) toccando tutti i vertici da questo raggiungibili si definisce visita.

Esistono due principali algoritmi di visita:

- Visita in ampiezza
- Visita in profondità.

Visita in ampiezza

La visita in ampiezza (o *breadth-first search*, BFS) consiste nel visitare i vertici del grafo in livelli:

1. l=0, S_l={sorgente}
2. Si visitano tutti i vertici S_{l+1} non ancora visitati e adiacenti ad almeno un vertice in S_l
3. l=l+1
4. Si ripete da 2 sino a che S_l è vuoto.

Albero BFS

La visita in ampiezza produce un albero BFS, nel quale

- La radice è il vertice sorgente
- I vertici sono quelli del grafo
- Gli archi sono un sottoinsieme di quelli del grafo.

Pseudo-codice

```
BFS(G, s)
 1  for ogni vertice u ∈ V[G] – {s}
 2      do color[u] ← WHITE
 3         d[u] ← ∞
 4         π[u] ← NIL
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← {s}
 9  while Q ≠ ∅
10      do u ← head[Q]
11         for ogni v ∈ Adj[u]
12             do if color[v] = WHITE
13                then color[v] ← GRAY
14                     d[v] ← d[u] + 1
15                     π[v] ← u
16                     ENQUEUE(Q, v)
17         DEQUEUE(Q)
18         color[u] ← BLACK
```

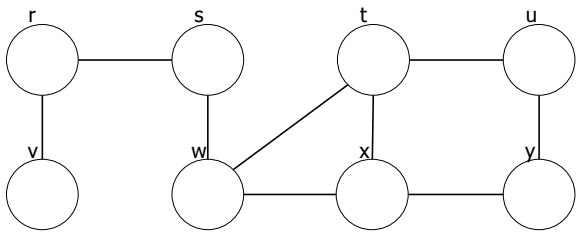

Nota

La colorazione dei vertici segue il seguente criterio:

- Inizialmente tutti i vertici sono bianchi
- Un vertice è colorato in grigio quando viene raggiunto per la prima volta
- Un vertice è colorato in nero quando tutti i vertici ad esso adiacenti e non ancora visitati sono stati inseriti nella coda.

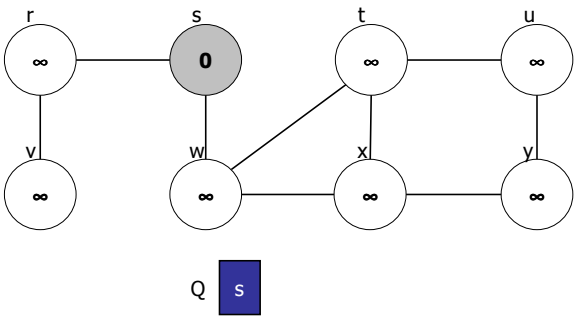
Esempio (0)

Vertice sorgente: s



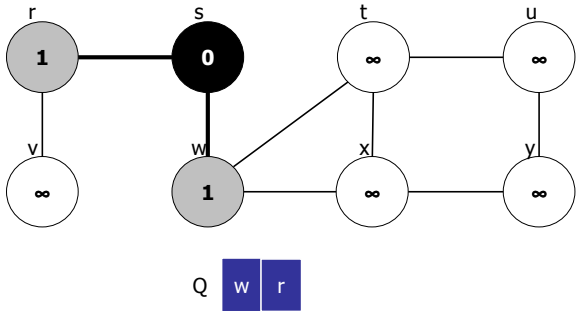
Esempio (1)

Inizializzazione



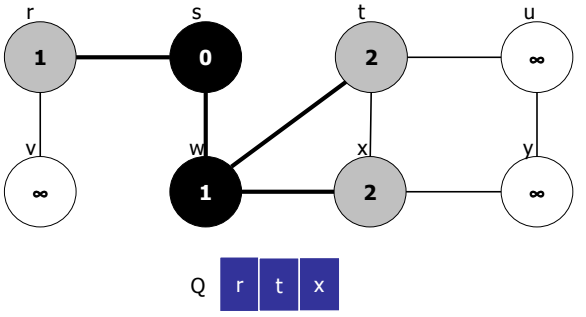
Esempio (2)

- **Estrae s da Q**
- **Metti in Q e colora in grigio i vertici adiacenti a s**
- **Colora s in nero**



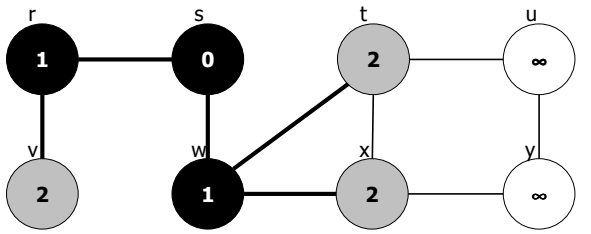
Esempio (3)

- **Estrae w da Q**
- **Metti in Q e colora in grigio i vertici adiacenti a w**
- **Colora w in nero**



Esempio (4)

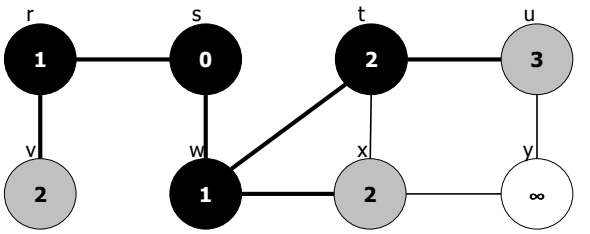
- Estrae r da Q
- Metti in Q e colora in grigio i vertici adiacenti a r
- Colora r in nero



Q t x v

Esempio (5)

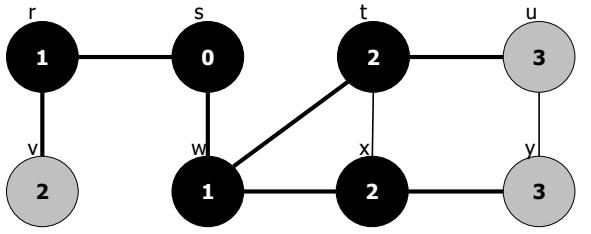
- Estrae t da Q
- Metti in Q e colora in grigio i vertici adiacenti a t
- Colora t in nero



Q x v u

Esempio (6)

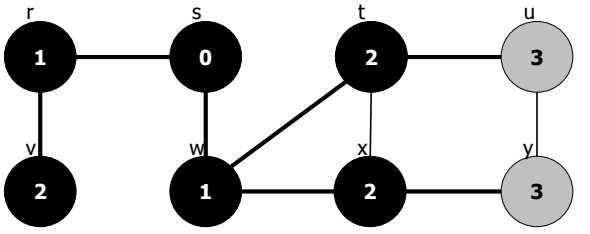
- Estrae x da Q
- Metti in Q e colora in grigio i vertici adiacenti a x
- Colora x in nero



Q v u y

Esempio (7)

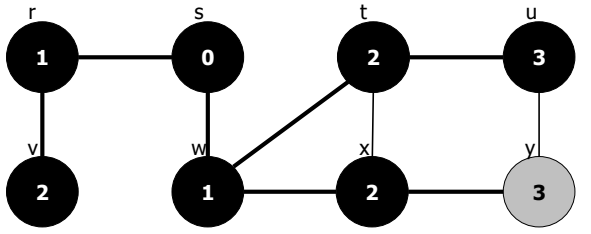
- Estrae v da Q
- Metti in Q e colora in grigio i vertici adiacenti a v
- Colora v in nero



Q u y

Esempio (8)

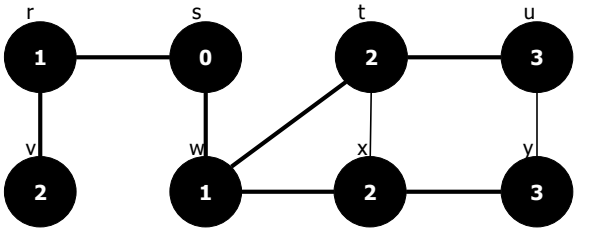
- Estrae u da Q
- Metti in Q e colora in grigio i vertici adiacenti a u
- Colora u in nero



Q y

Esempio (9)

- Estrae y da Q
- Metti in Q e colora in grigio i vertici adiacenti a y
- Colora y in nero



Q

Complessità

Il tempo totale richiesto dalla procedura BFS è O(V+E).

Cammini minimi

Dati due vertici s e v in un grafo non pesato, il numero minimo di archi su un cammino da s a v si definisce distanza sul cammino minimo.

La procedura BFS calcola le distanze sul cammino minimo per ogni vertice di un grafo (rispetto ad un vertice sorgente).

Visita in profondità

La visita in profondità (Depth-First Search, o DFS) segue un approccio opposto a BFS.

Ad ogni passo si visita un vertice adiacente all'ultimo vertice visitato.

Quando non ne esistono, si ritorna indietro all'ultimo vertice visitato che abbia vertici adiacenti non visitati, e li si visita.

La procedura di DFS è normalmente implementata attraverso una procedura recursiva.

Pseudo-codice (1)

```
DFS(G)
1  for ogni vertice u ∈ V[G]
2      do color[u] ← WHITE
3         π[u] ← NIL
4  time ← 0
5  for ogni vertice u ∈ V[G]
6      do if color[u] = WHITE
7         then DFS-VISIT(u)
```


Pseudo-codice (2)

```
DFS-VISIT(u)
1  color[u] ← GRAY              ▷ Il vertice bianco u è stato appena scoperto
2  d[u] ← time ← time + 1
3  for ogni v ∈ Adj[u]          ▷ Si esplora l'arco (u, v)
4      do if color[v] = WHITE
5         then π[v] ← u
6              DFS-VISIT(v)
7  color[u] ← BLACK             ▷ Si rende u nero: la sua visita è finita.
8  f[u] ← time ← time + 1
```

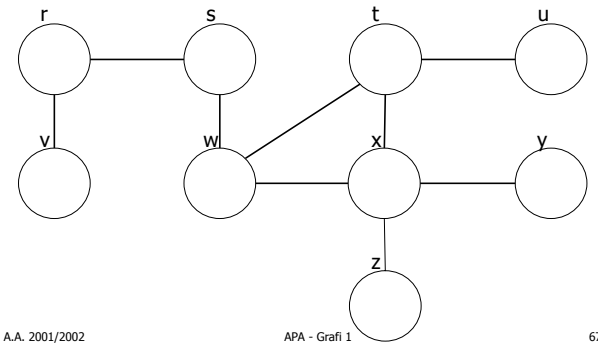

Nota

La colorazione dei vertici segue il seguente criterio:

- Inizialmente tutti i vertici sono bianchi
- Un vertice è colorato in grigio quando viene raggiunto per la prima volta
- Un vertice è colorato in nero quando tutti i vertici ad esso adiacenti sono stati visitati.

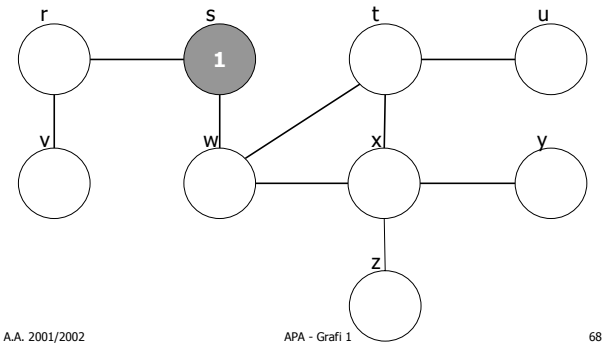
Esempio (0)

Vertice sorgente: *s*



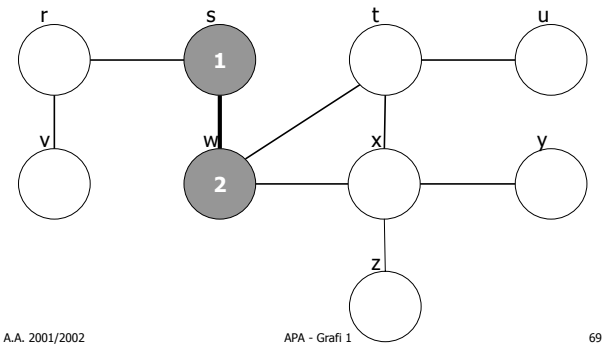
Esempio (1)

Visita s



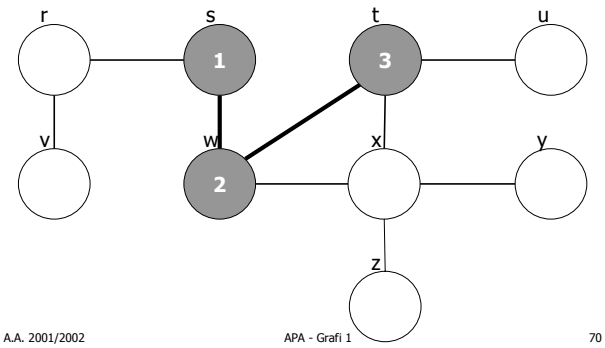
Esempio (2)

Visita w



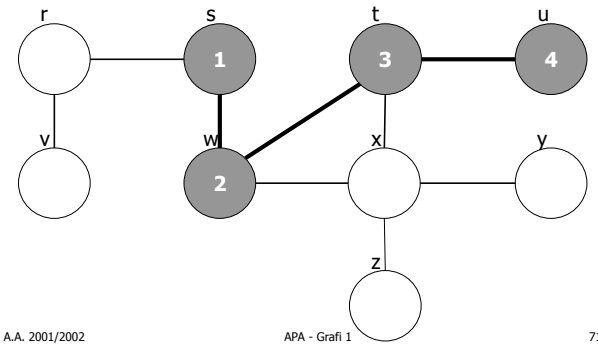
Esempio (3)

Visita t



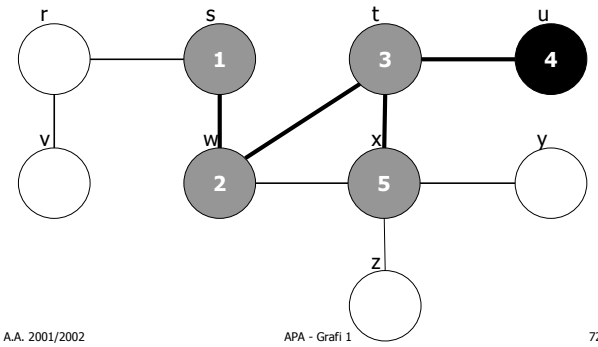
Esempio (4)

Visita u



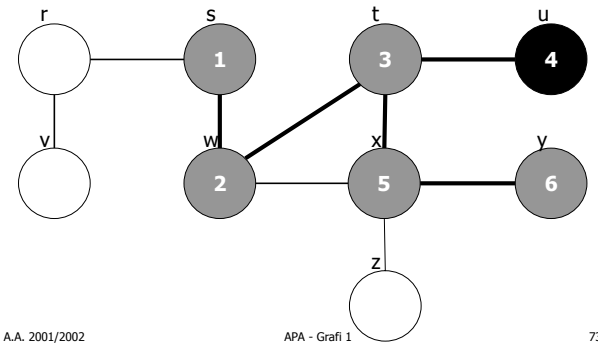
Esempio (5)

Visita x



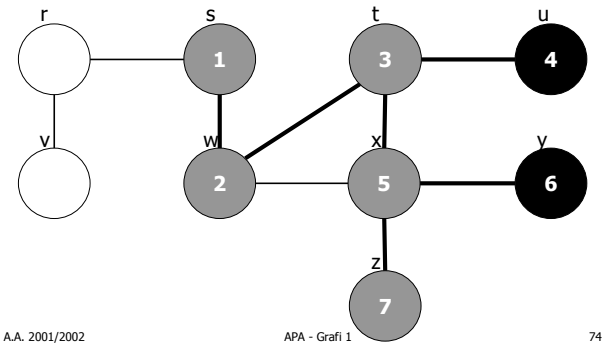
Esempio (6)

Visita y



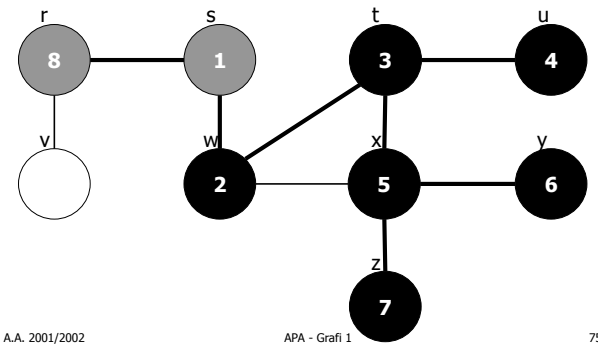
Esempio (7)

Visita z



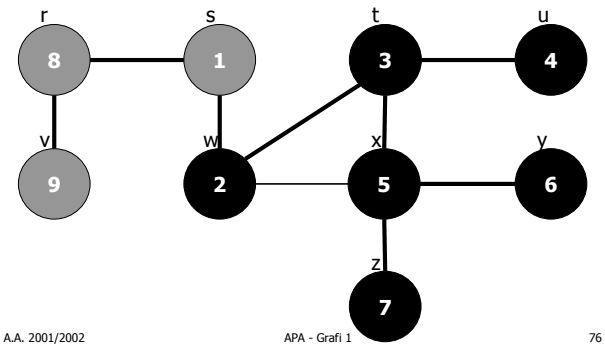
Esempio (8)

Visita r



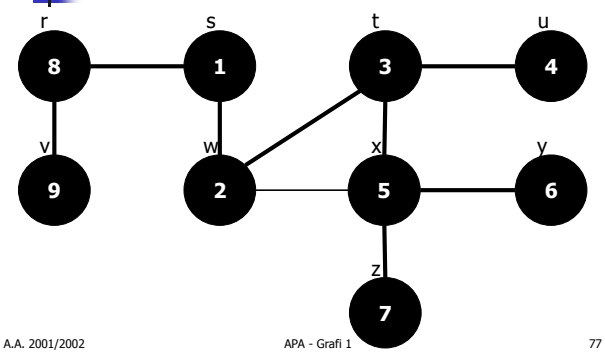
Esempio (9)

Visita v



Esempio (10)

Concludi la visita



Complessità

Il tempo di esecuzione della procedura DFS è $\Theta(V+E)$.

Foresta DFS

La procedura DFS costruisce una foresta DFS, composta da uno o più alberi DFS.

Gli archi che compongono la foresta sono detti archi dell'albero.

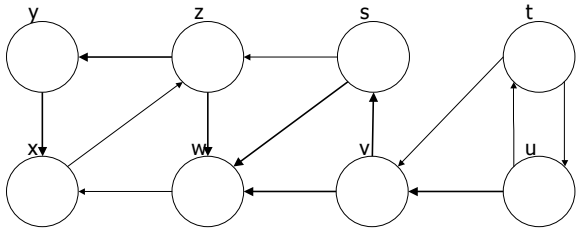
Classificazione degli archi

In un grafo orientato, gli archi possono essere raggruppati in 4 categorie:

- Archi dell'albero (T)
- Archi all'indietro (B): non sono archi T e connettono un vertice ad un suo antenato
- Archi in avanti (F): non sono archi T e connettono un vertice ad un suo discendente
- Archi di attraversamento (C): gli archi rimanenti.

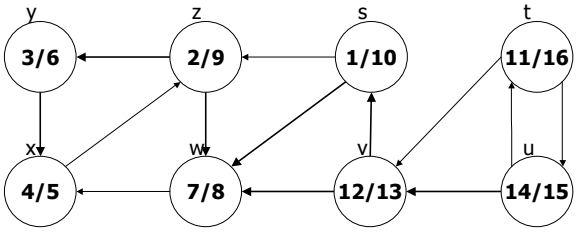
Esempio (0)

Vertice sorgente: *s*



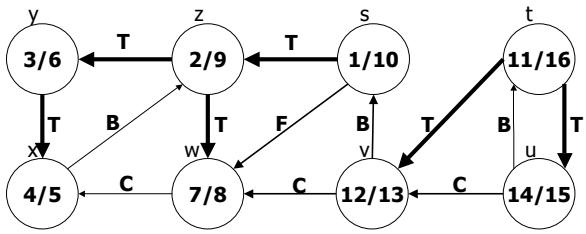
Esempio (1)

Visita in profondità



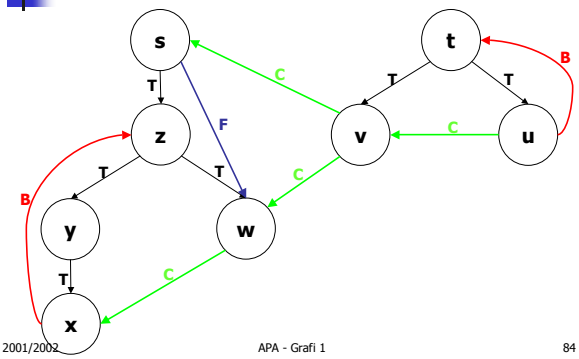
Esempio (2)

Classificazione degli archi



Esempio (3)

Classificazione degli archi



Classificazione degli archi

La procedura DFS può essere facilmente modificata in modo da classificare gli archi.

Ogni volta che si incontra un arco (u,v) si considera il colore del vertice v in quel momento:

- Se è bianco l'arco è un arco dell'albero
- Se è grigio l'arco è un arco all'indietro
- Se è nero l'arco è un arco in avanti (se d[u]<d[v]) o un arco di attraversamento (se d[u]>d[v]).

Classificazione degli archi

In un grafo non orientato non esistono archi in avanti o archi di attraversamento.

Sommario

- Definizioni
- Rappresentazione dei grafi
- Algoritmi di visita
- Esempi in C

Lettura da file

Si supponga di voler leggere da un file la descrizione di un grafo e di volerla memorizzare in una lista di adiacenza.

Occorre definire

- Il formato del file
- Il formato della rappresentazione in memoria.

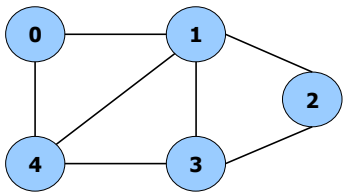
Formato del file

La prima linea contiene il numero dei vertici *n*.

Ciascuno dei successivi *n* blocchi di righe contiene (un'informazione per riga):

- Una riga contenente i caratteri "*k" dove k è il numero del vertice cui si riferisce il blocco
- Il numero dei vertici adiacenti al vertice k
- La lista dei vertici adiacenti.

Esempio




```
5
*0
2
1
4
*
4
0
4
3
2
```

```
*2
2
1
3
*3
3
2
3
4
*4
3
0
1
3
```

Formato interno

Il formato interno è basato su

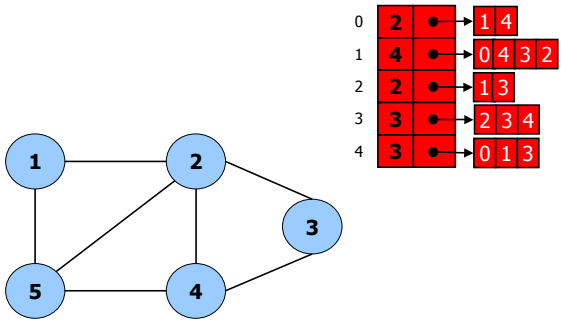
- Un vettore di n puntatori di tipo VERTEX:

```
struct vertex{
     int  nadj;
     int  *adjlist;
}VERTEX;
```

- Ogni elemento del vettore punta ad un vettore di interi di lunghezza pari al numero di vertici adiacenti al vertice k, e contenente i loro indici.

Tutti i vettori sono allocati dinamicamente all'atto della lettura del file.

Esempio



Grafi.h

```
typedef struct vertex *VERTEXP;
struct vertex{
  int  nadj;
  int  *adjlist;
}VERTEX;
```


Readgrf (1)

```
#include <stdio.h>
#include "grafi.h"
VERTEXP graph;
int     nvertex;
int readgrf(char name[])
{  FILE   *fin;
   int     nadj, n, i, j;
   if( (fin=fopen( name, "r")) == NULL)
   {  printf( "Errore in apertura file %s\n", name);
      return(0);  }
   fscanf( fin, "%d\n", &nvertex);
   if( (graph = (VERTEXP)malloc( nvertex * sizeof( VERTEX))) == NULL)
   {  printf( "Errore in allocazione graph\n");
      return( 0);  }
```


Readgrf (2)

```
   for( i=0; i<nvertex; i++)
   {  fscanf( fin, "*%d\n", &n);          /* numero del vertice */
      if( n != i)
      {  printf( "Errore nell'ordine dei vertici (%d)\n", n);
         return( 0); }
      fscanf( fin, "%d\n", &nadj);
      graph[i].nadj = nadj;
      if( nadj != 0)
      {  if( (graph[i].adjlist = (int *)malloc( nadj * sizeof( int))) == NULL)
         {  printf( "Errore in allocazione adjlist vertice %d\n", i);
            return( 0); }
         for( j=0; j<nadj; j++)
            fscanf( fin, "%d\n", &(graph[i].adjlist[j]));
      }
   }
```


Readgrf (3)

```
   fclose( fin);
   return( 1);
}
```

Visita in ampiezza

La procedura che segue fa uso di una coda fifo che implementa due operazioni:

- Inserimento
- Estrazione: ritorna –1 se la coda è vuota.

Inoltre, anziché marcare con un colore ogni vertice, la procedura usa un vettore *visited* precedentemente allocato.

bfs

```
void bfs( int root)
{   int     vertex, i;
    vertex = root;
    visited[vertex] = 1;
    while( vertex != -1)
    {   visit( vertex);
        for( i=0; i<graph[vertex].nadj; i++)
        {   if( visited[graph[vertex].adjlist[i]] == 0)
        {   insert( graph[vertex].adjlist[i]);
                visited[graph[vertex].adjlist[i]] = 1;
        }
        }
        vertex = extract();
    }
}
```


Visita in profondità

Anche la procedura che segue usa il vettore *visited* che deve essere stato precedentemente definito ed allocato.

dfs

```
void dfs(int root)
{  int       i;
   visited[root] = 1;
   visit(root);
   for(i=0; i<graph[root].nadj; i++)
   {  if(visited[graph[root].adjlist[i]] == 0)
      dfv(graph[root].adjlist[i]);
   }
}
```