POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Gli algoritmi di visita dei grafi

Gianpiero Cabodi e Paolo Camurati

Algoritmi di visita

Visita di un grafo G=(V, E):

- a partire da un vertice dato, seguendo gli archi con una certa strategia, elencare i vertici incontrati, eventualmente aggiungendo altre informazioni.

Algoritmi:

- in profondità (depth-first search, DFS)
- in ampiezza (breadth-first search, BFS).

Visita in profondità (versione base)

Dato un grafo (connesso o non connesso), a partire da un vertice s, visita **tutti** i vertici del grafo (raggiungibili da s e non).

Principio base della visita in profondità: espandere l'ultimo vertice scoperto che ha ancora vertici non ancora scoperti adiacenti.

Scoperta di un vertice: prima volta che si incontra nella visita (discesa ricorsiva, visita in pre-order). Il vettore `pre` gioca il ruolo del vettore `mark` nel Calcolo Combinatorio, ma contiene i tempi crescenti di scoperta.

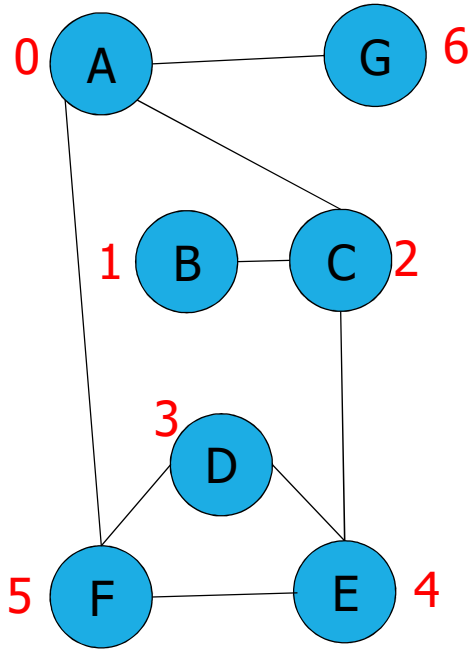
Algoritmo

wrapper

- `GRAPHsimpleDfs`: funzione che, a partire da un vertice dato, visita tutti i vertici di un grafo, richiamando la procedura ricorsiva `SimpleDfsR`. Termina quando tutti i vertici sono stati visitati.
- `SimpleDfsR`: funzione che visita in profondità a partire da un vertice `id` identificato con un arco fittizio `EDGEcreate(id,id)` utile in fase di visualizzazione. Termina quando ha visitato in profondità tutti i nodi raggiungibili da `id`.

B: alcuni autori chiamano visita in profondità la sola `SimpleDfsR`.

Esempio



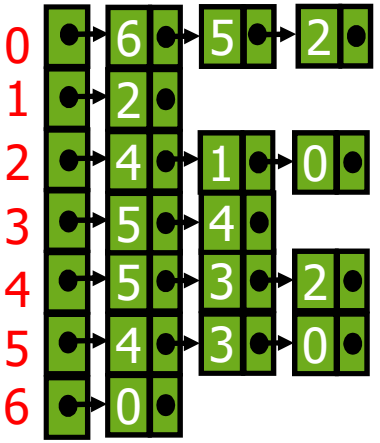
in.txt

```
A C
A F
A G
B C
C E
D E
D F
F E
```

ST

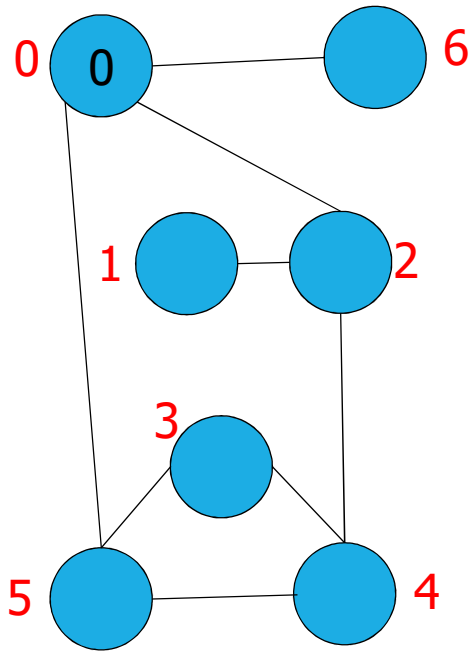
0	A
1	B
2	C
3	D
4	E
5	F
6	G

Lista delle adiacenze

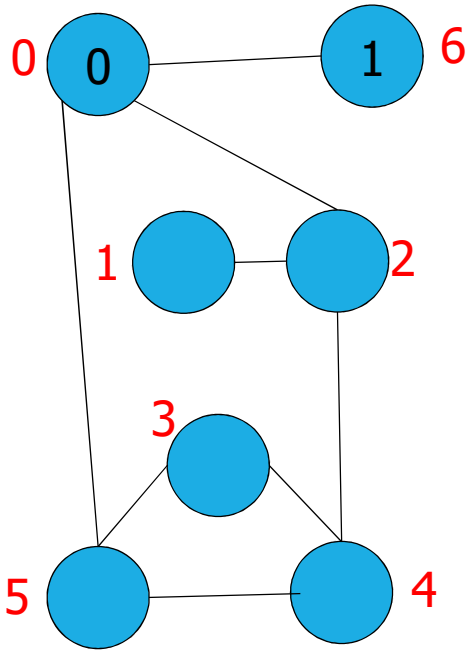


Esempio

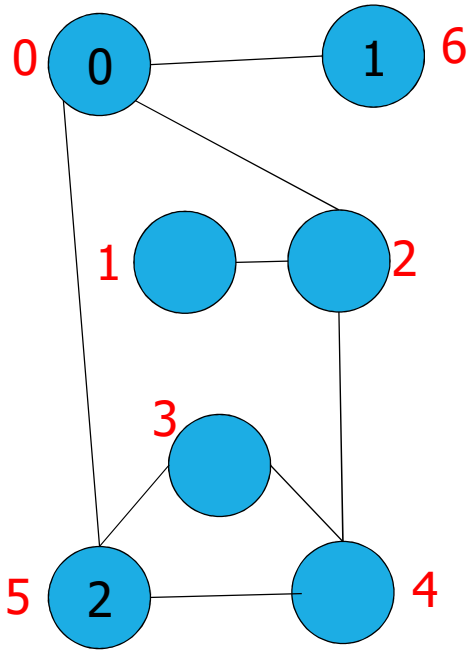
cnt = 0

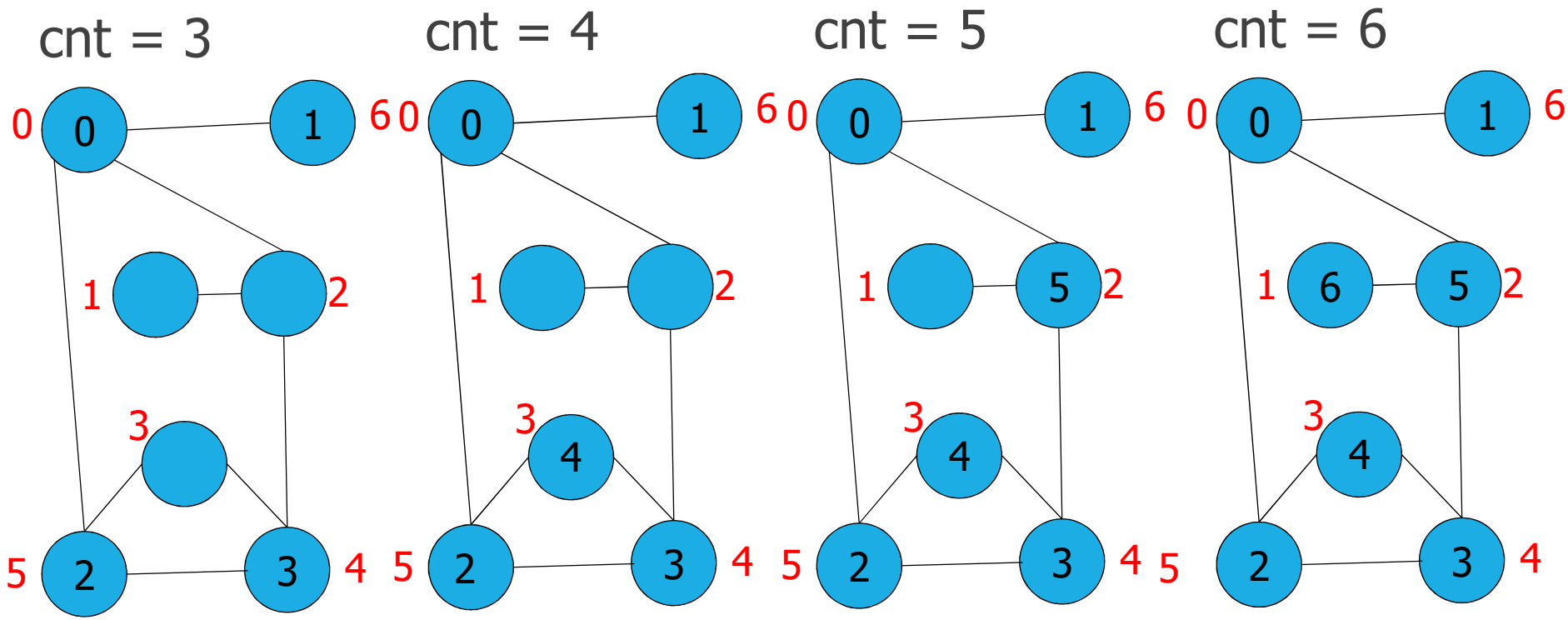


cnt = 1



cnt = 2




cnt = 3
cnt = 4
cnt = 5
cnt = 6
0
1
2
3
4
5
6

Strutture dati

- grafo non pesato come lista delle adiacenze
- vettore `pre[i]` dove per ciascun vertice si registra il tempo di scoperta (numerazione in preordine dei vertici)
- contatore `cnt` per tempi di scoperta

`cnt` e `*pre` sono locali alla funzione `GRAPHsimpleDfs` e passati by reference alla funzione ricorsiva `SimpleDfsR`.

```
void GRAPHsimpleDfs(Graph G, int id) {
  int v, cnt=0, *pre;
  pre = malloc(G->V * sizeof(int));
  if ((pre == NULL)) return;
  for (v=0;v<G->V;v++) pre[v]=-1;

  simpleDfsR(G, EDGEcreate(id,id), &cnt, pre);

for (v=0; v < G->V; v++)
    if (pre[v]== -1)
      simpleDfsR(G, EDGEcreate(v,v), &cnt, pre);

  printf("discovery time labels \n");
  for (v=0; v < G->V; v++)
    printf("vertex %s : %d \n", STsearchByIndex(G->tab, v), pre[v]);

}
```

visita a partire da id

visita dei nodi non ancora visitati

```
static void simpleDfsR(Graph G, Edge e, int *cnt, int *pre) {
  link t; int w = e.w;
  pre[w] = (*cnt)++;
  for (t = G->ladj[w]; t != G->z; t = t->next)
    if (pre[t->v] == -1)
      simpleDfsR(G, EDGEcreate(w, t->v), cnt, pre);
}
```

terminazione implicit
della ricorsione

Visita in profondità (versione estesa)

Estensione:

- nodi etichettati con etichetta tempo di scoperta / tempo di completamento
- foresta di alberi della visita in profondità, memorizzata in un vettore.

Scoperta di un vertice: prima volta che si incontra nella visita (discesa ricorsiva, visita in pre-order), vettore `pre[i]`.
Completamento: fine dell'elaborazione del vertice (uscita dalla ricorsione, visita in post-order) vettore `post[i]`.

Scoperta/Completamento: tempo discreto che avanza mediante contatore `time`. Avanzamento quando si scopre o si completa.

Identificazione del padre nella visita in profondità: vettore `st[i]`.

Algoritmo

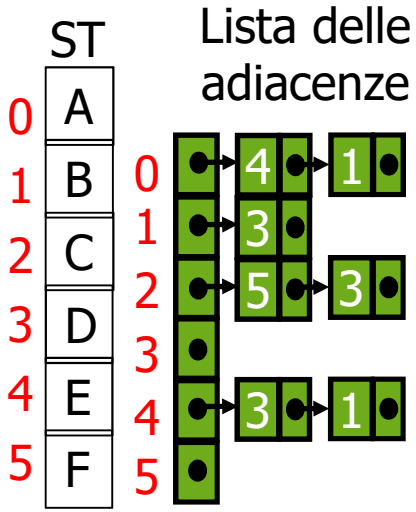
wrapper

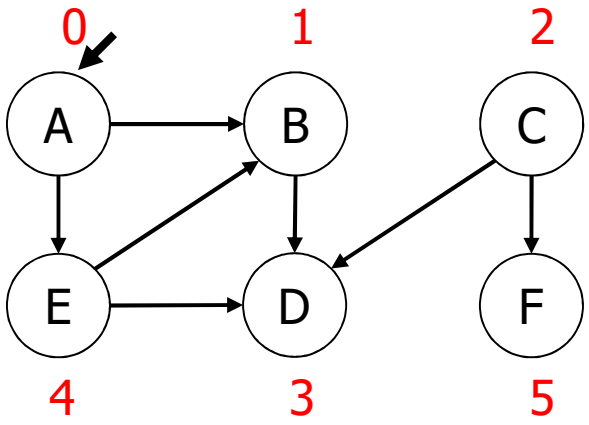
- `GRAPHextendedDfs`: funzione che, a partire da un vertice dato, visita tutti i vertici di un grafo, richiamando la procedura ricorsiva `ExtendedDfsR`. Termina quando tutti i vertici sono stati visitati.
- `ExtendedDfsR`: funzione che visita in profondità a partire da un vertice `id` identificato con un arco fittizio `EDGEcreate(id,id)` utile in fase di visualizzazione. Termina quando ha visitato in profondità tutti i nodi raggiungibili da `id`.

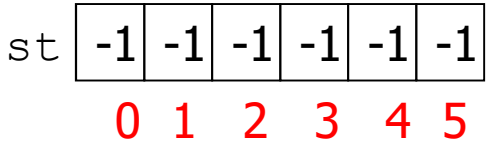
B: alcuni autori chiamano visita in profondità la sola `ExtendedDfsR`.

Esempio

`time` =0 `st`





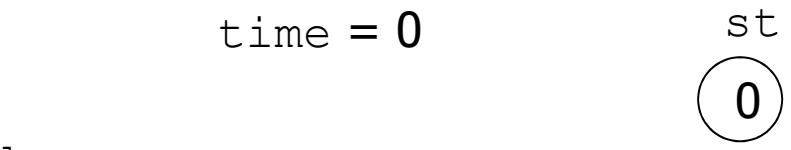


Convenzione grafica:

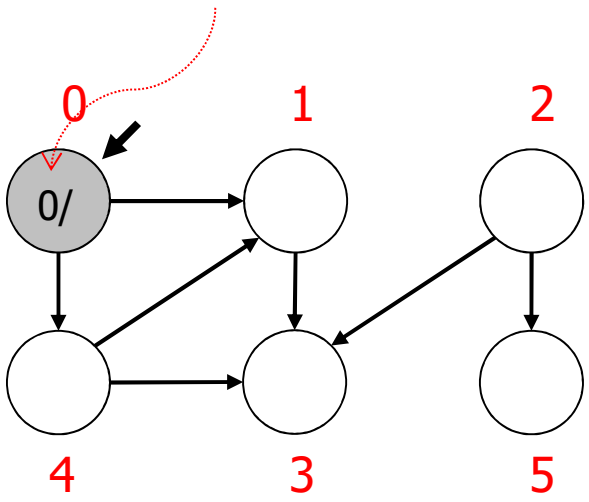
- nodo bianco: non ancora scoperto
- nodo grigio: scoperto ma non terminato
- nodo nero: terminato

`time` = 0

`st`



`pre[i]/post[i]`

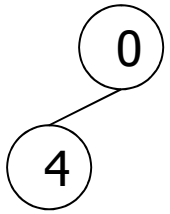


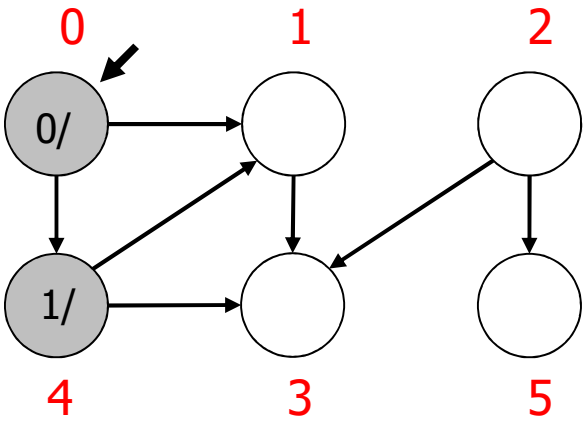
`st`

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

time = 1

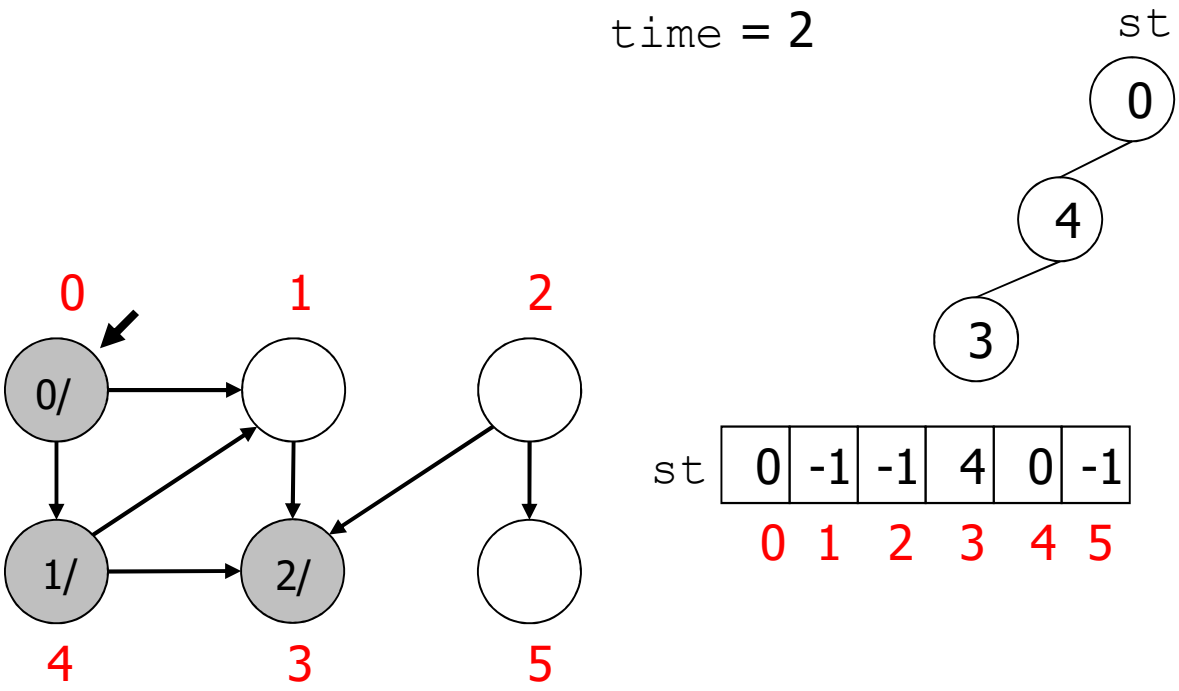
st

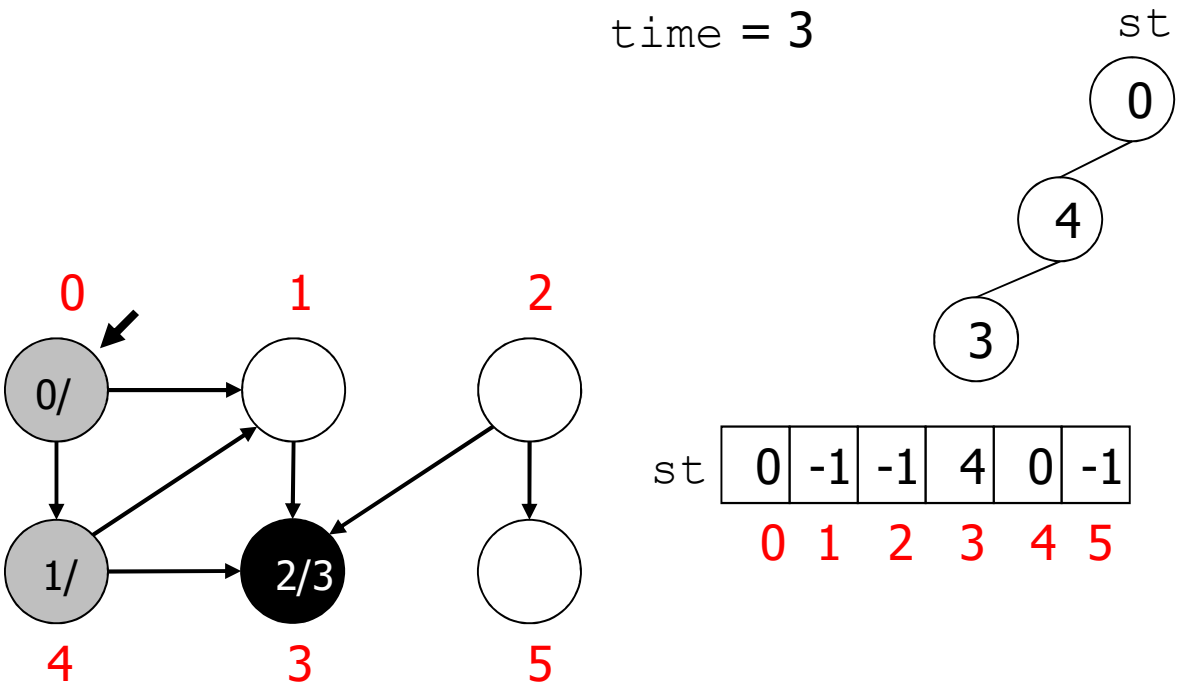


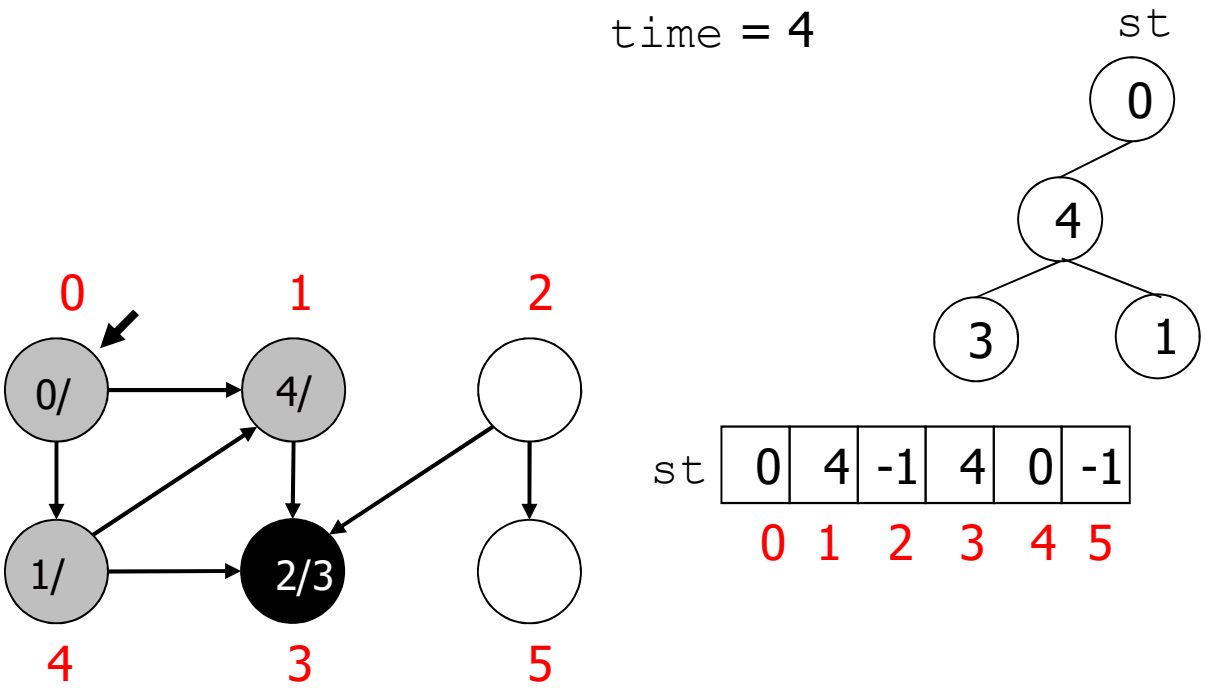


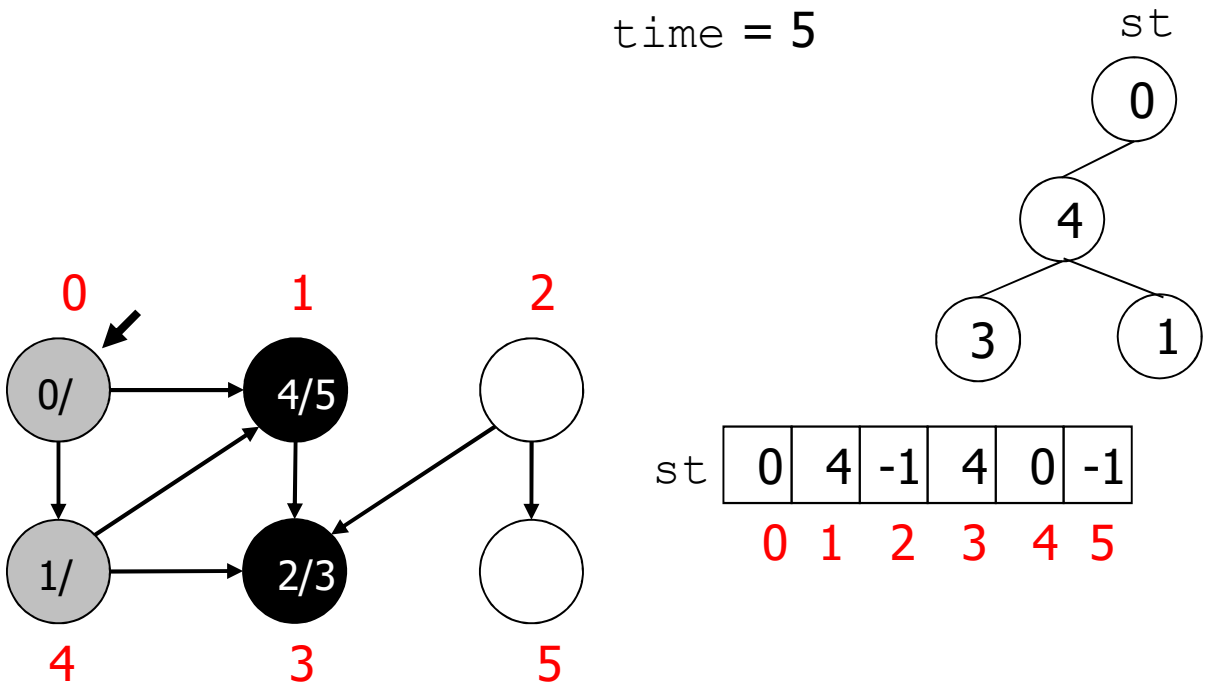
st

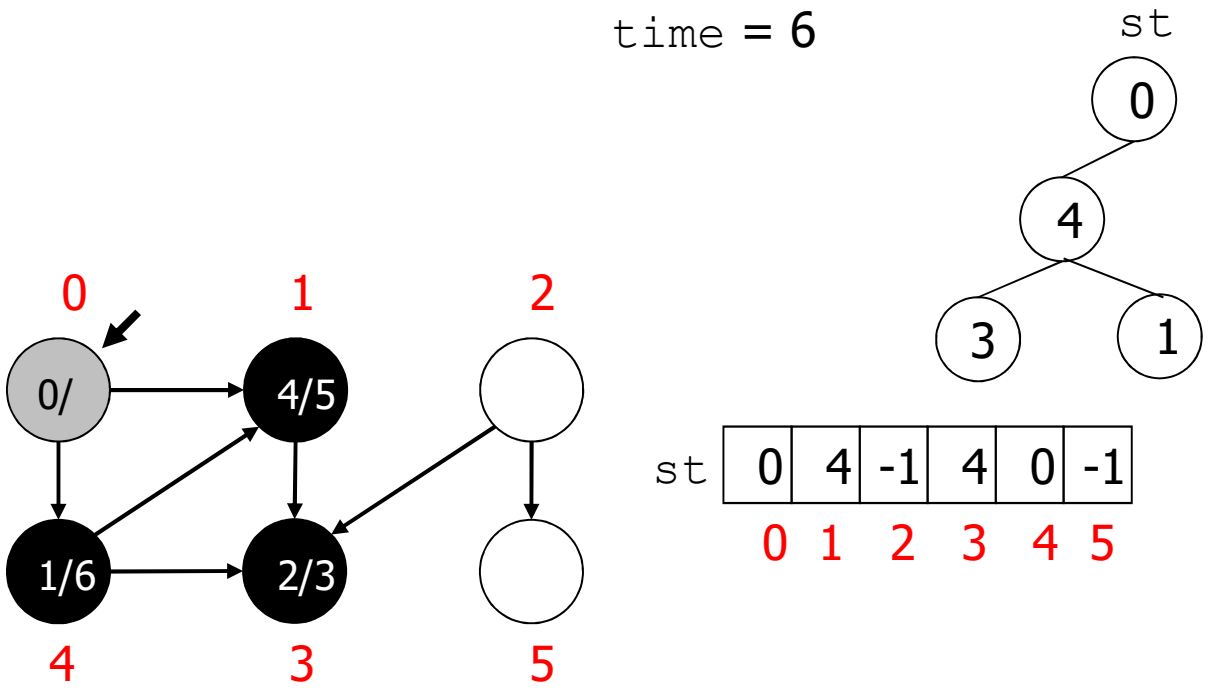
0	-1	-1	-1	0	-1
0	1	2	3	4	5

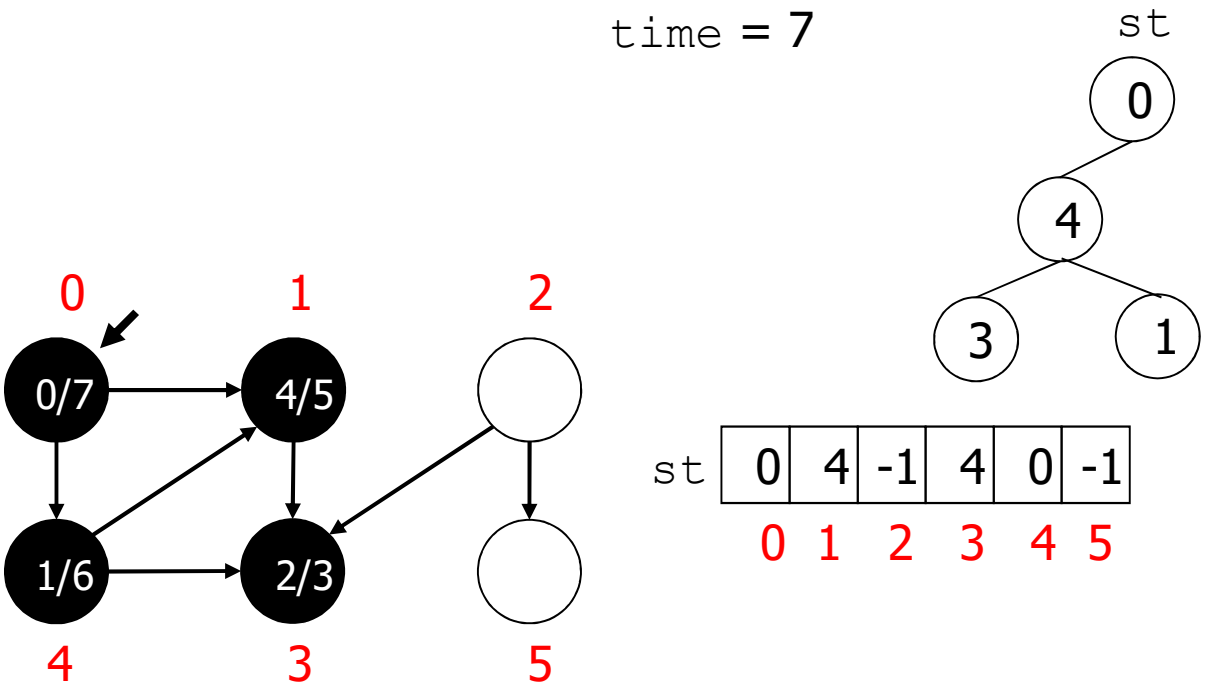

time = 2
st
0
4
3
0
1
2
0/
1/
2/
4
3
5
st
0 -1 -1 4 0 -1
0 1 2 3 4 5

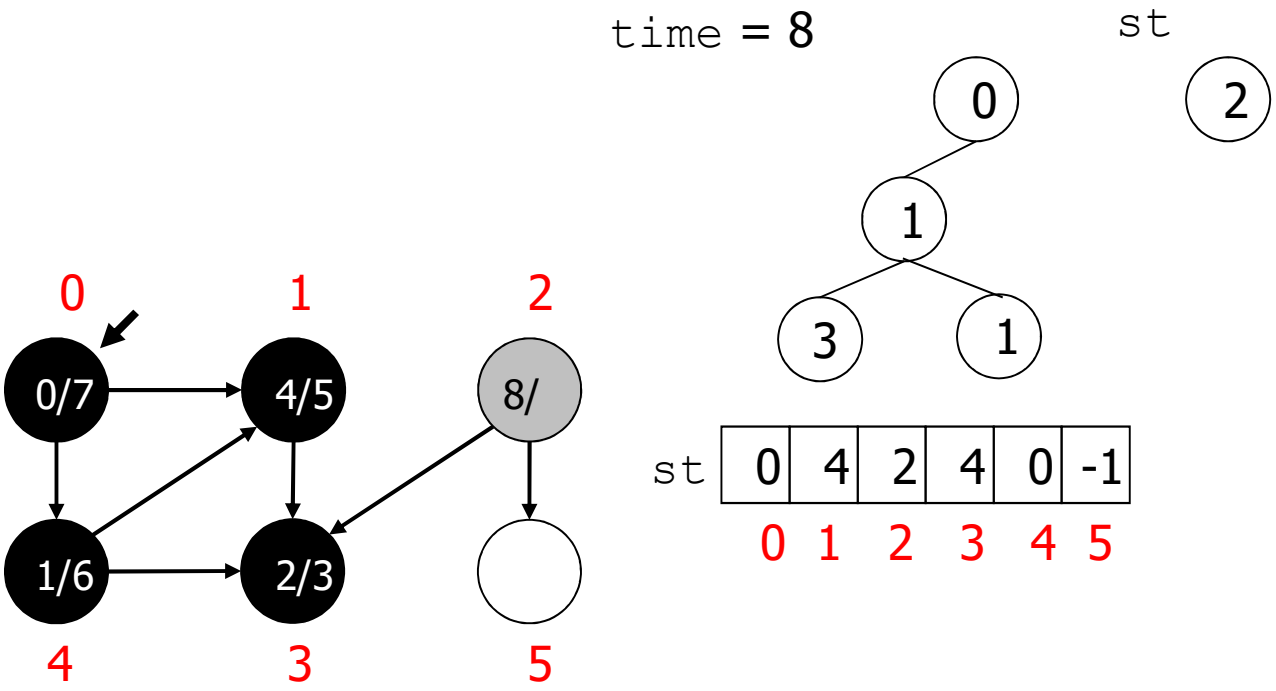

time = 3
st
0
4
3
0
1
2
0/
1/
2/3
4
3
5
st
0 -1 -1 4 0 -1
0 1 2 3 4 5

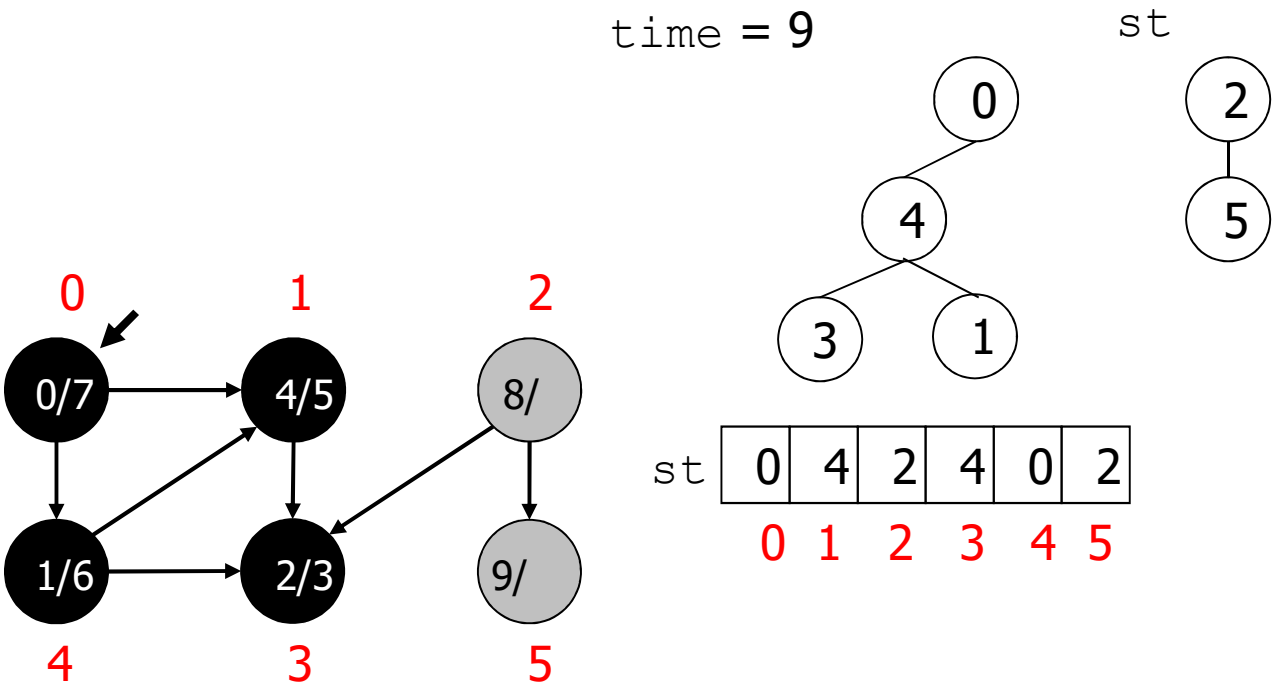

time = 4
st
0
4
3
1
0
1
2
0/
4/
1/
2/3
4
3
5
st
0
4
-1
4
0
-1
0 1 2 3 4 5

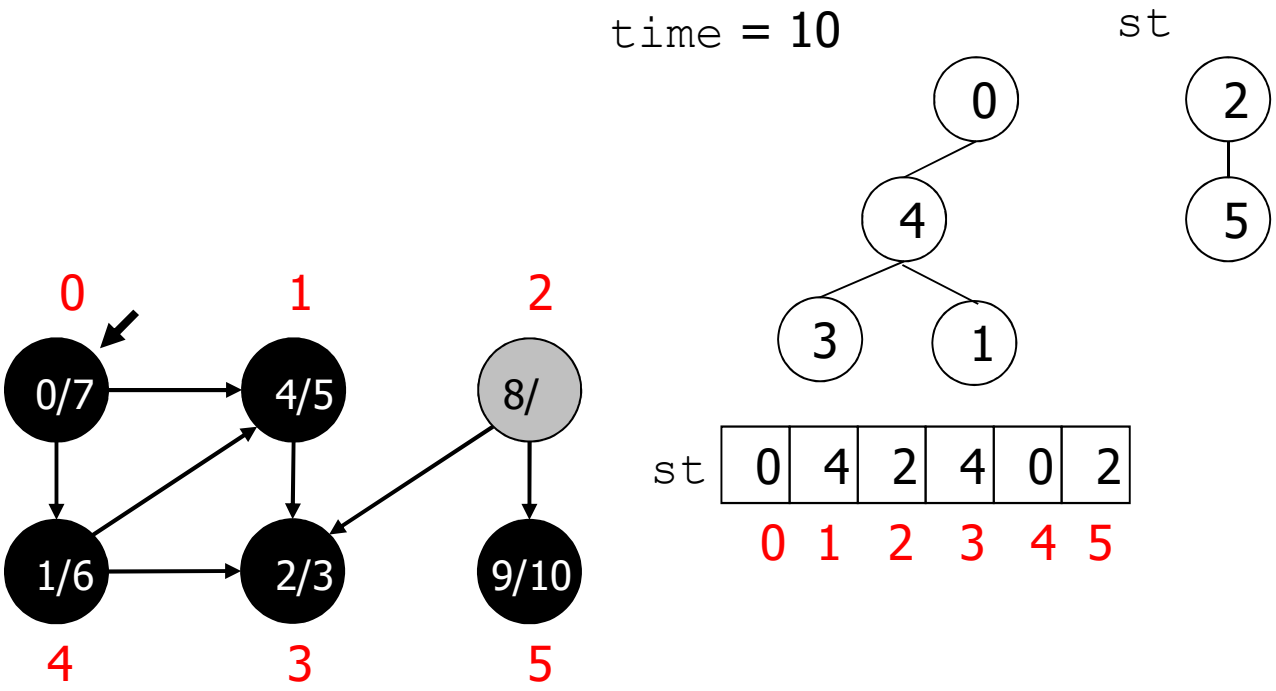

time = 5
st
0
4
3
1
0
1
2
0/
4/5
1/
2/3
4
3
5
st
0 4 -1 4 0 -1
0 1 2 3 4 5

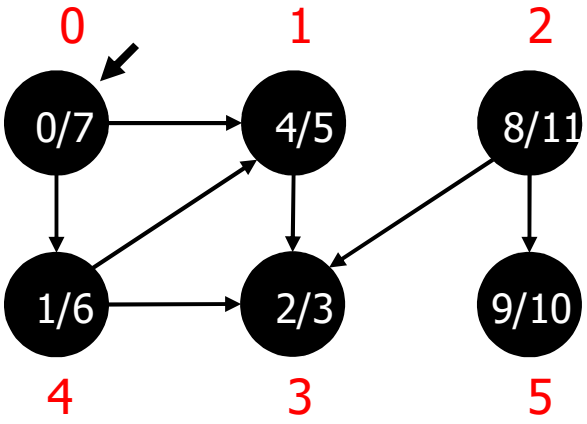

time = 6
st
0
4
3
1
0
1
2
0/
4/5
1/6
2/3
4
3
5
st
0
4
-1
4
0
-1
0 1 2 3 4 5


time = 7
st
0
4
3
1
0
1
2
0/7
4/5
1/6
2/3
4
3
5
st
0
4
-1
4
0
-1
0 1 2 3 4 5


time = 8
st
0
2
1
3
1
0
1
2
0/7
4/5
8/
1/6
2/3
4
3
5
st
0
4
2
4
0
-1
0
1
2
3
4
5

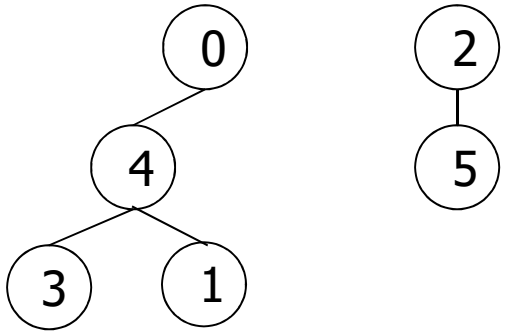

time = 9
st
0
2
4
5
3
1
0
1
2
0/7
4/5
8/
1/6
2/3
9/
4
3
5
st
0 4 2 4 0 2
0 1 2 3 4 5


time = 10
st
0
4
3
1
2
5
0
1
2
0/7
4/5
8/
1/6
2/3
9/10
4
3
5
st
0 4 2 4 0 2
0 1 2 3 4 5



time = 11

st



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

Strutture dati

- grafo non pesato come lista delle adiacenze
- vettori dove per ciascun vertice:
  - si registra il tempo di scoperta (numerazione in preordine dei vertici) `pre[i]`
  - si registra il tempo di completamento (numerazione in postordine dei vertici) `post[i]`
  - si registra il padre per la costruzione della foresta degli alberi della visita in profondità: `st[i]`
- contatore `time` per tempi di scoperta/completamento

`time, *pre, *post` e `*st` sono locali alla funzione `GRAPHextendedDfs` e passati by reference alla funzione ricorsiva `ExtendedDfsR`.

```
void GRAPHextendedDfs(Graph G, int id) {
  int v, time=0, *pre, *post, *st;
  pre/post/st = malloc(G->V * sizeof(int));
  for (v=0;v<G->V;v++) {
    pre[v]=-1; post[v]=-1; st[v]=-1;}
  extendedDfsR(G, EDGEcreate(id,id), &time, pre, post, st);
  for (v=0; v < G->V; v++)
    if (pre[v]==-1)
      extendedDfsR(G,EDGEcreate(v,v),&time,pre,post,st);
  printf("discovery/endprocessing time labels \n");
  for (v=0; v < G->V; v++)
    printf("%s:%d/%d\n",STsearchByIndex(G->tab,v),pre[v],post[v]);
  printf("resulting DFS tree \n");
  for (v=0; v < G->V; v++)
     printf("%s's parent: %s \n",STsearchByIndex (G->tab, v),
             STsearchByIndex (G->tab, st[v]));
}
```

```
static void ExtendedDfsR(Graph G, Edge e, int *time, int *pre,
                         int *post, int *st) {
  link t;
  int w = e.w;
  st[e.w] = e.v;
  pre[w] = (*time)++;
  for (t = G->ladj[w]; t != G->z; t = t->next)
    if (pre[t->v] == -1)
      ExtendedDfsR(G, EDGEcreate(w, t->v), time, pre, post, st);
  post[w] = (*time)++;
}
```

terminazione implicita della ricorsione

Visita in profondità (versione completa)

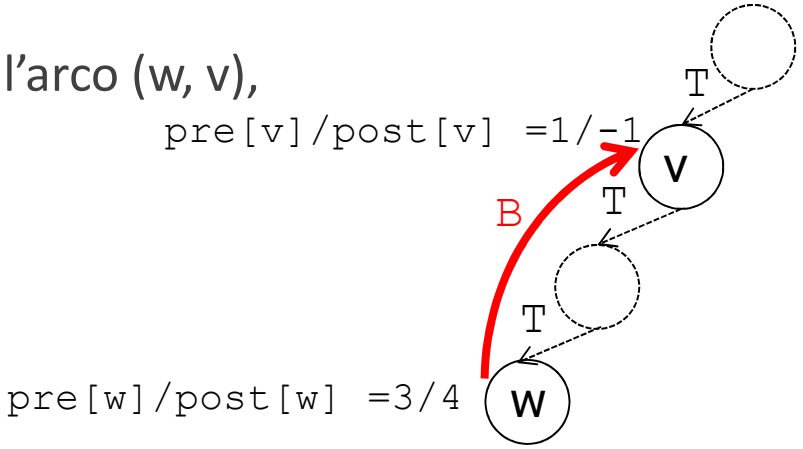
Si etichetta ogni arco:

- grafi orientati: T(tree), B(backward), F(forward), C(cross)
- grafi non orientati: T(tree), B(backward)

Classificazione degli archi

Grafo orientato:

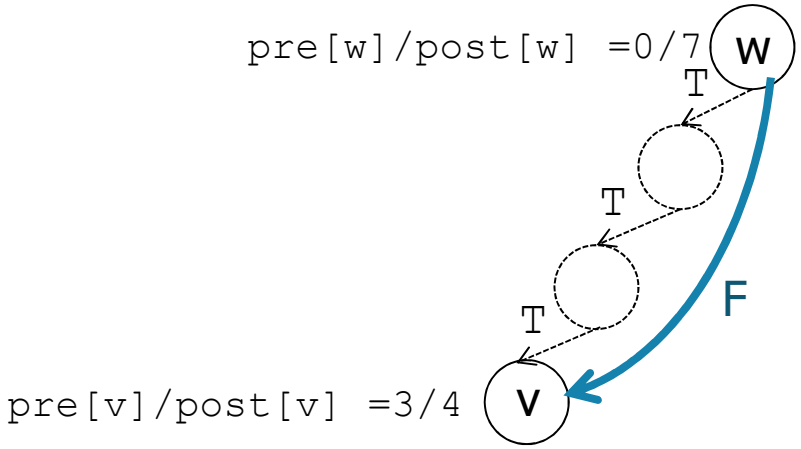
- **T**: archi dell'albero della visita in profondità
- **B**: connettono un vertice w ad un suo antenato v nell'albero:
  tempo di fine elaborazione di v **sarà** > tempo di fine elaborazione di w.
  Equivale a testare se, quando scopro l'arco (w, v),
  `post[v] == -1`



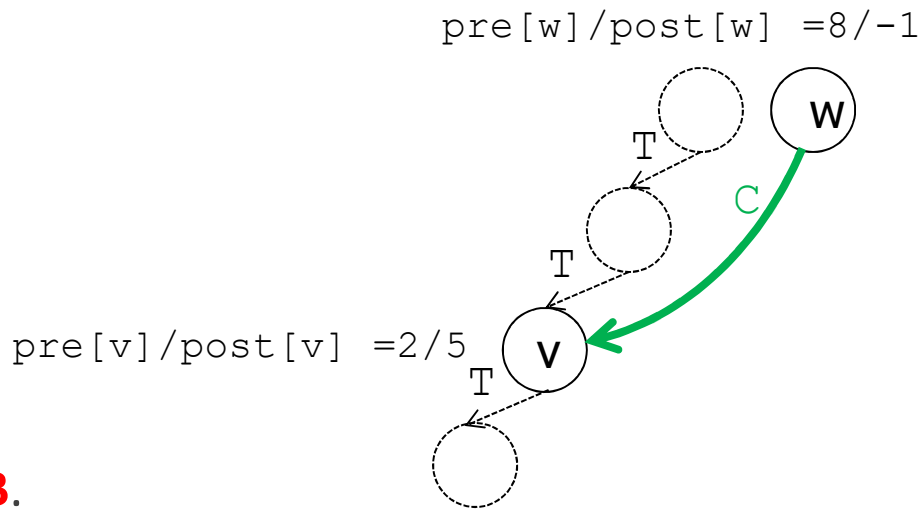
- **F**: connettono un vertice w ad un suo discendente v nell'albero:

  tempo di scoperta di v **è** > tempo di scoperta di v quando scopro l'arco (w, v)

  `pre[v] > pre[w]`



- **C**: archi rimanenti, per cui
  tempo di scoperta di w **è** > tempo di scoperta di v quando scopro l'arco (w, v)
  `pre[w] > pre[v]`



Grafo non orientato: solo archi **T** e **B**.

Algoritmo

wrapper

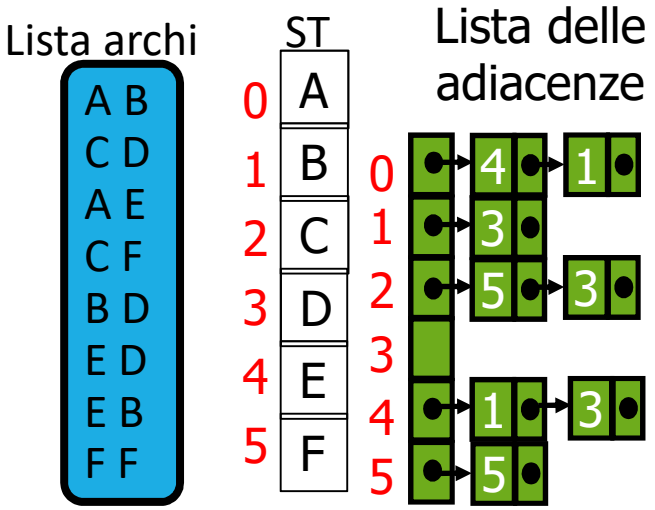
- `GRAPHdfs`: funzione che, a partire da un vertice dato, visita tutti i vertici di un grafo, richiamando la procedura ricorsiva `dfsR`. Termina quando tutti i vertici sono stati visitati.
- `dfsR`: funzione che visita in profondità a partire da un vertice `id` identificato con un arco fittizio `EDGEcreate(id,id)` utile in fase di visualizzazione. Termina quando ha visitato in profondità tutti i nodi raggiungibili da `id`.

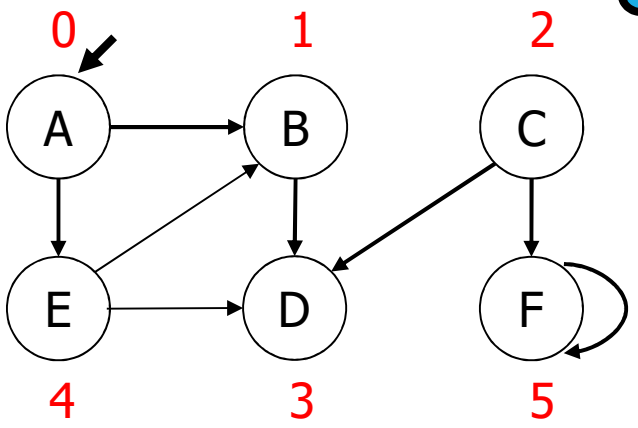
NB: alcuni autori chiamano visita in profondità la sola `dfsR`.

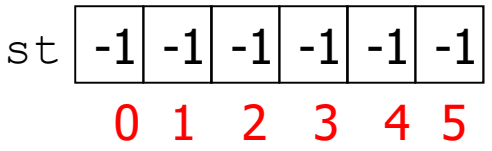
Strutture dati: come nella versione estesa. Codice della `GRAPHdfs` identico a quello della `GRAPHextendedDfs`.

Esempio

`time` = 0 `st`

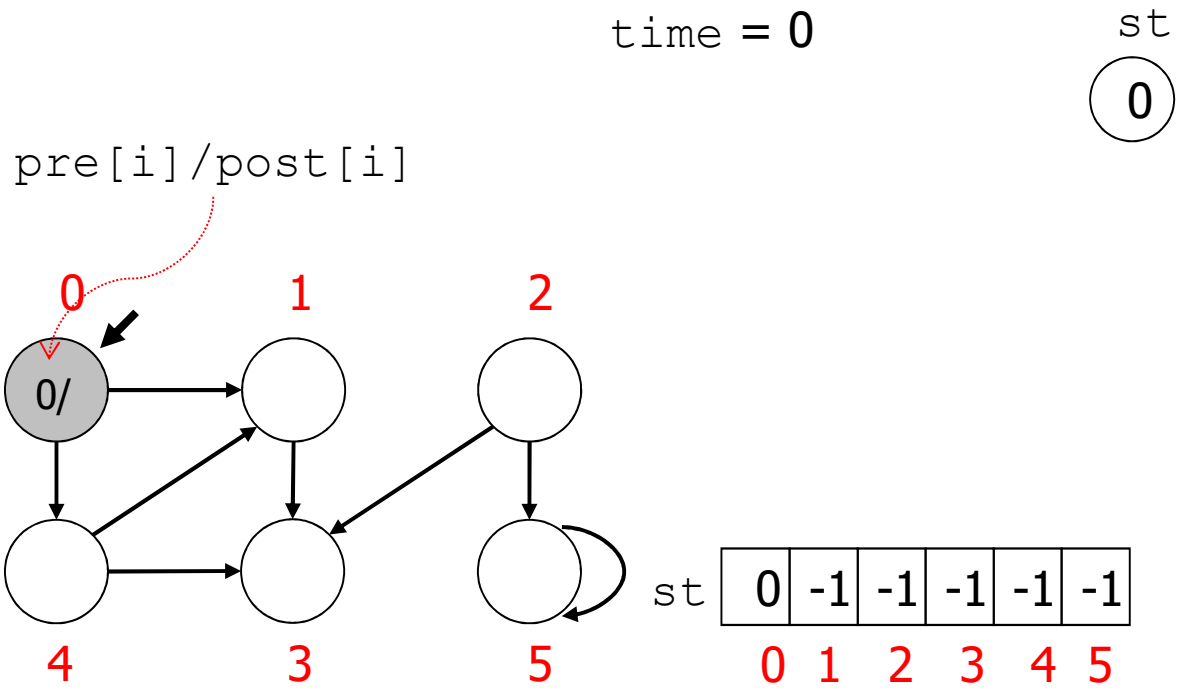






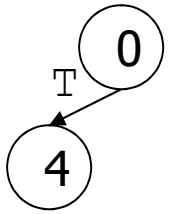
time = 0

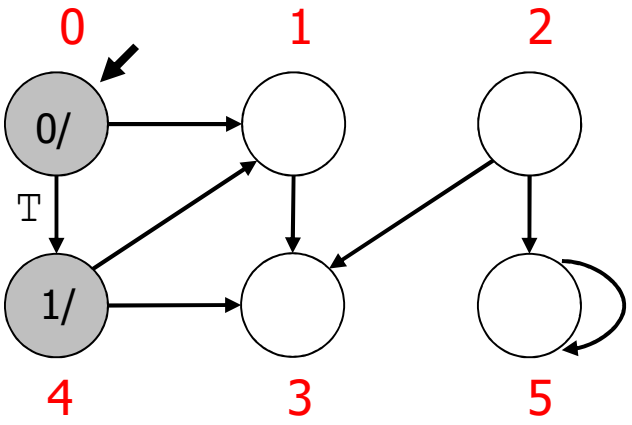
st



time = 1

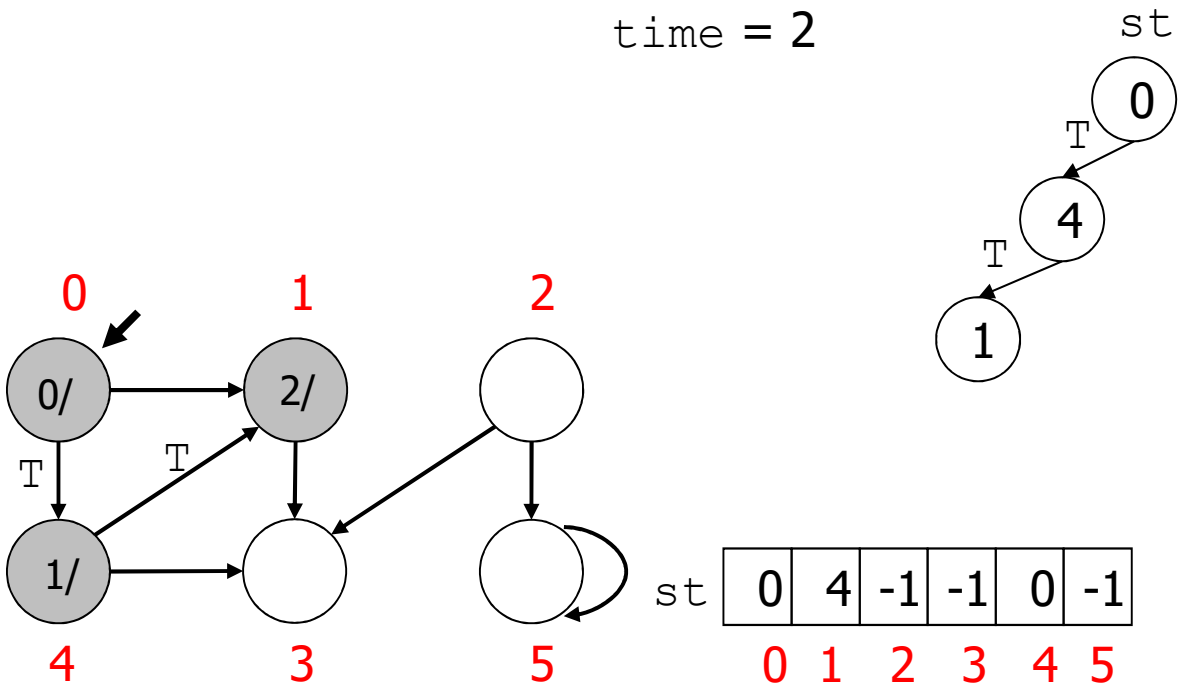
st

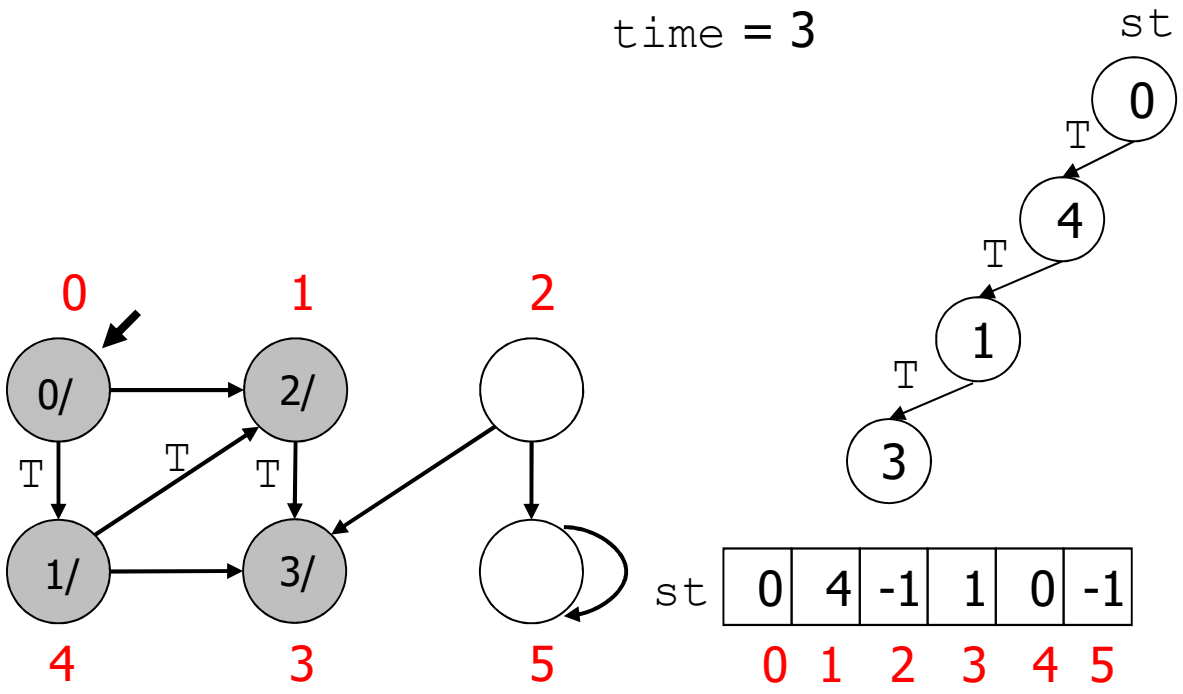


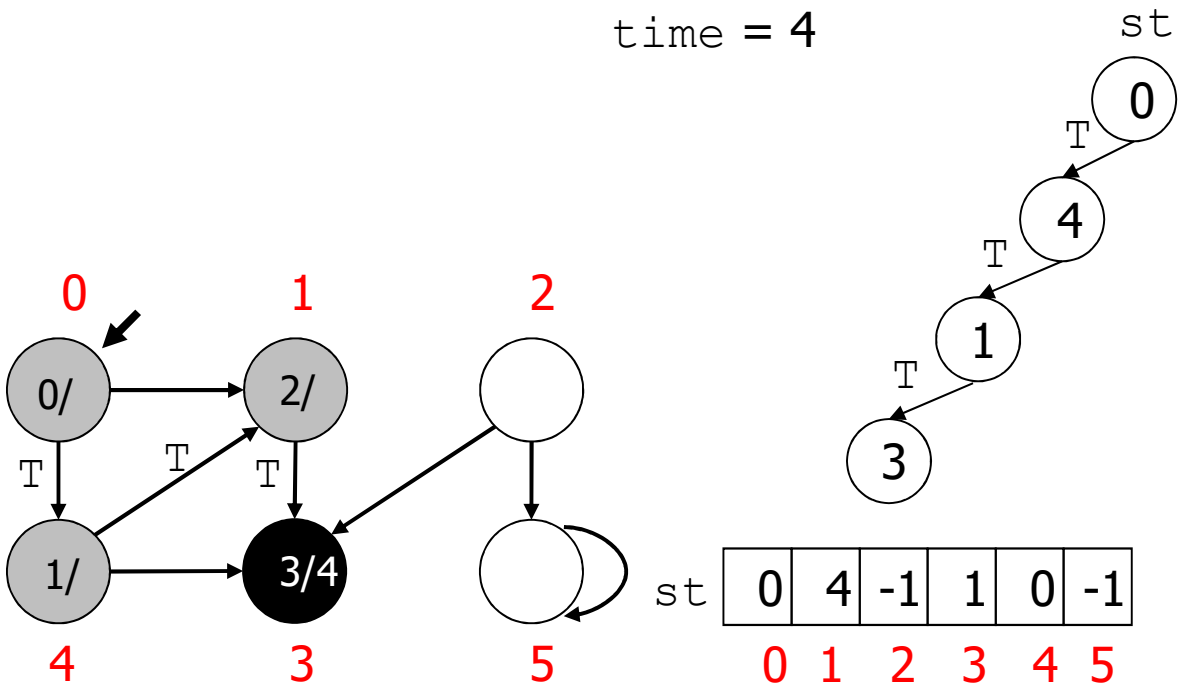


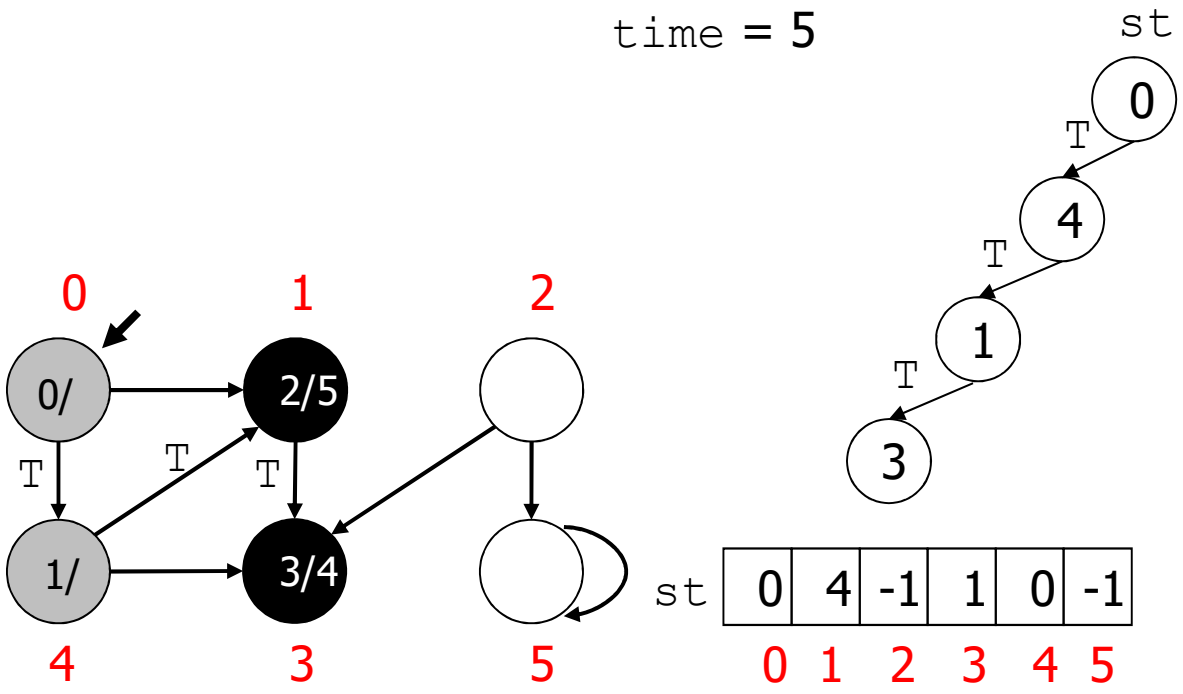
st

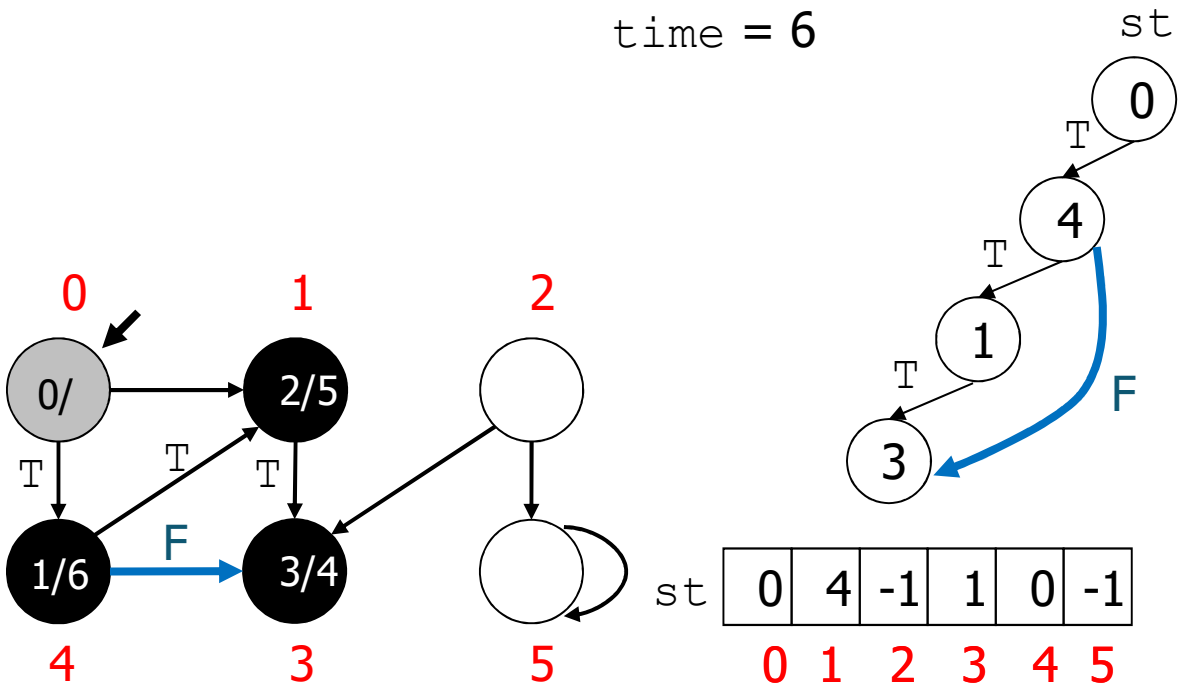
0	-1	-1	-1	0	-1
0	1	2	3	4	5

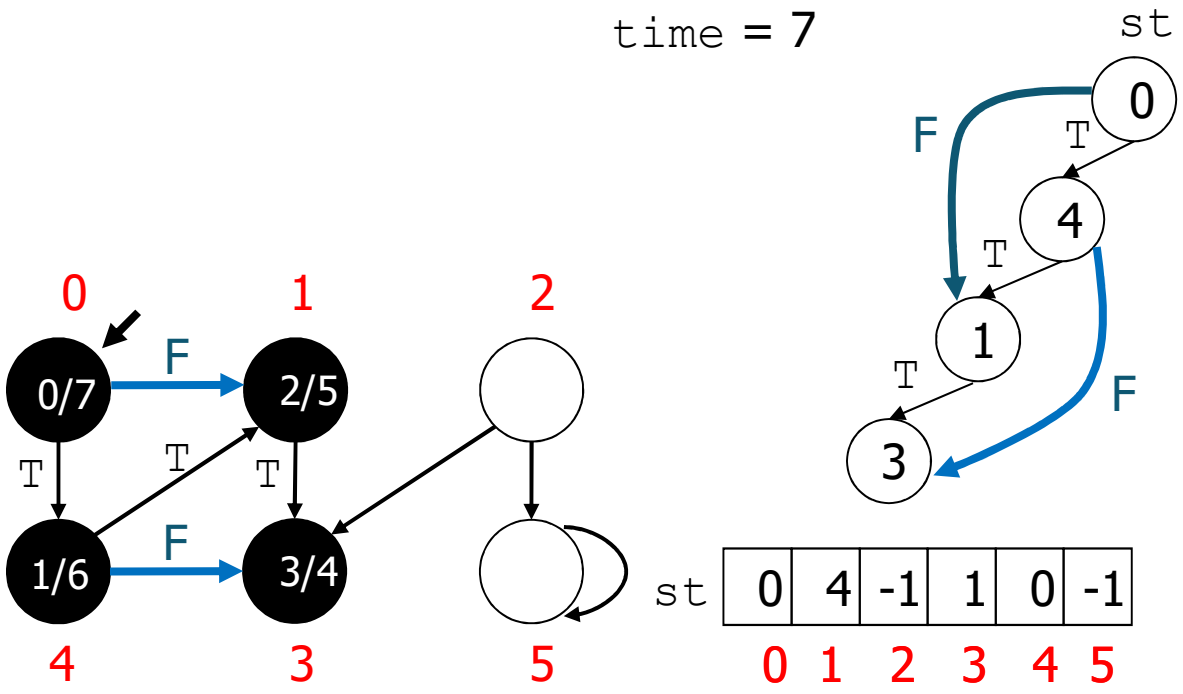

time = 2
st
0
T
4
T
1
0
1
2
0/
2/
T
T
1/
4
3
5
st
0
4
-1
-1
0
-1
0 1 2 3 4 5

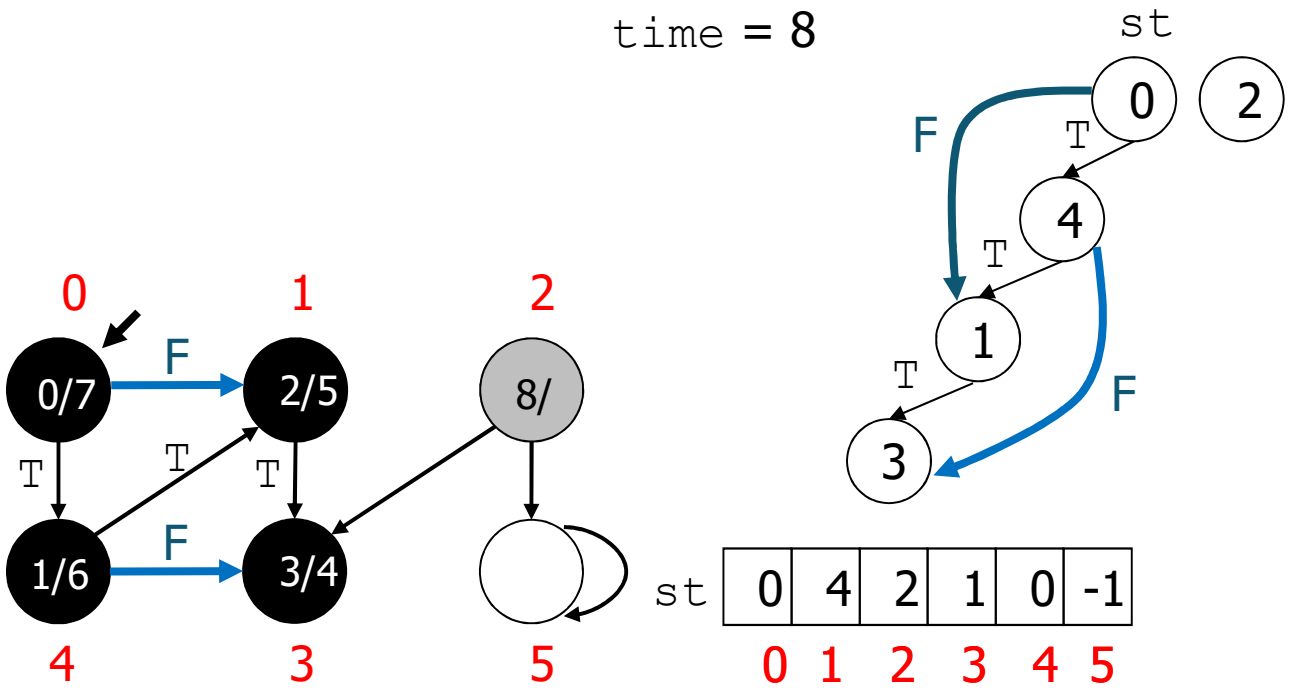

time = 3
st
0
T
4
T
1
T
3
0
1
2
0/
2/
1/
3/
T
T
T
4
3
5
st
0
4
-1
1
0
-1
0 1 2 3 4 5

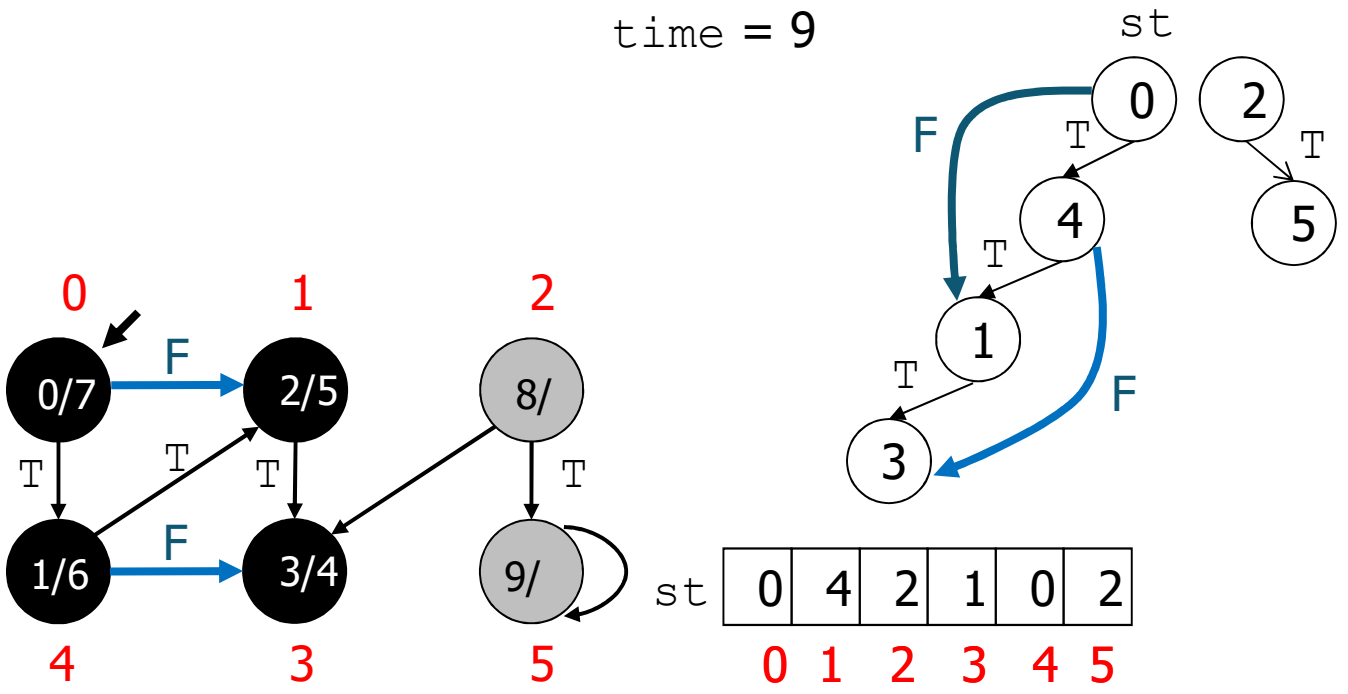

time = 4
st
0
T
4
T
1
T
3
0
1
2
0/
2/
T
T
T
1/
3/4
4
3
5
st
0
4
-1
1
0
-1
0 1 2 3 4 5

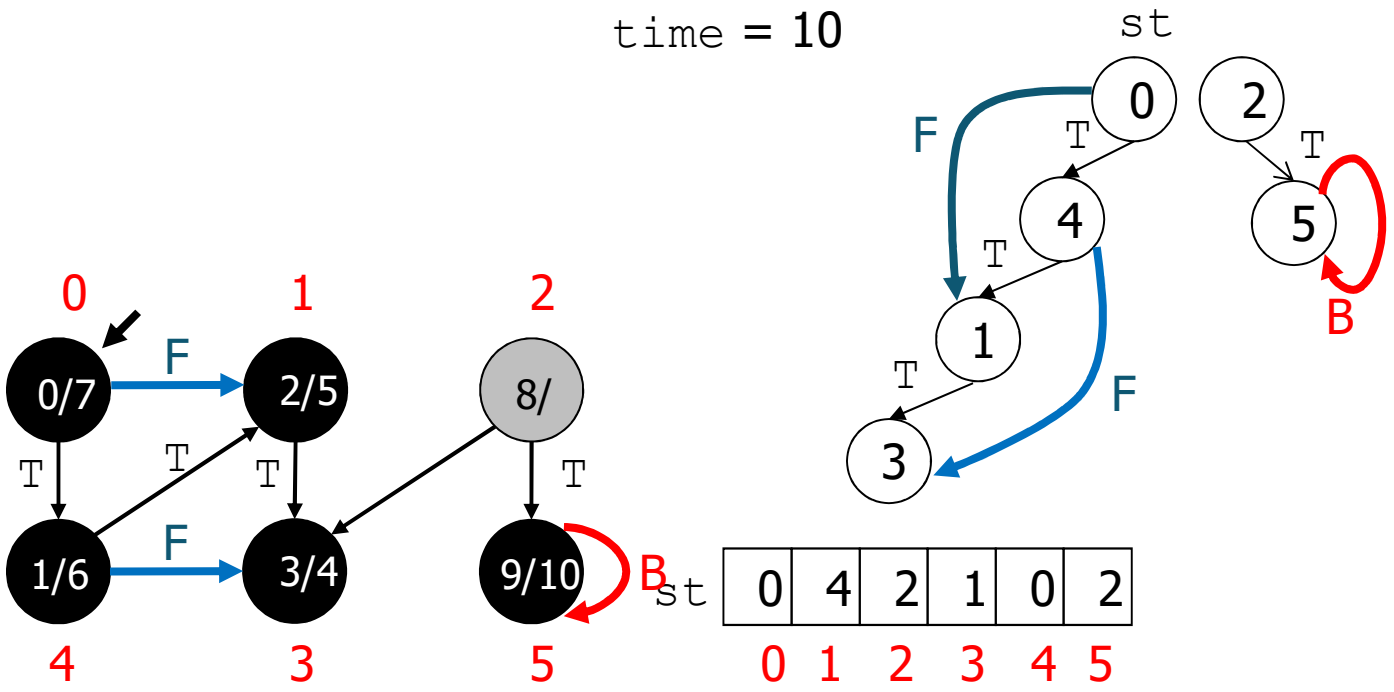

time = 5
st
0
T
4
T
1
T
3
0
1
2
0/
2/5
1/
3/4
T
T
T
4
3
5
st
0
4
-1
1
0
-1
0 1 2 3 4 5

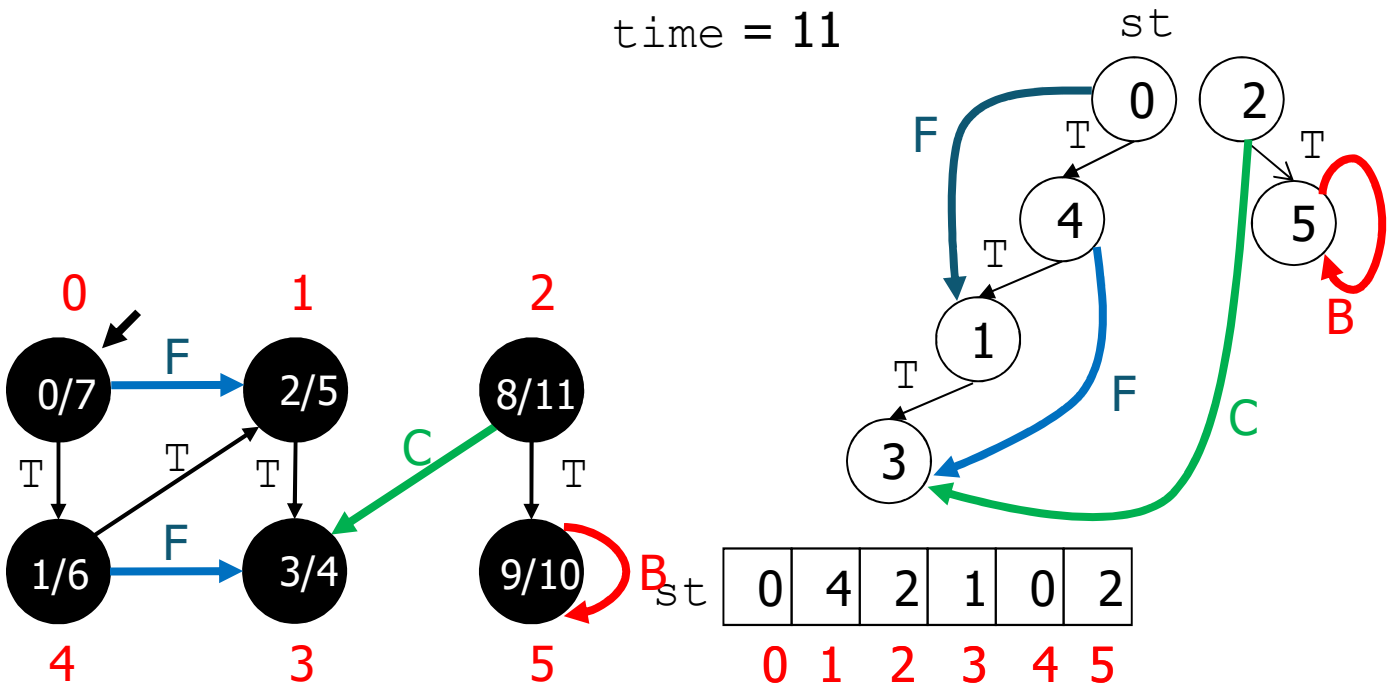

time = 6
st
0
T
4
T
1
T
3
F
0
1
2
0/
2/5
1/6
3/4
T
T
T
F
4
3
5
st
0
4
-1
1
0
-1
0 1 2 3 4 5


time = 7
st
0
F
T
4
T
1
T
F
3
0
1
2
0/7
F
2/5
T
T
T
1/6
F
3/4
4
3
5
st
0
4
-1
1
0
-1
0 1 2 3 4 5


time = 8
st
0
2
F
T
4
T
1
T
F
3
0
1
2
0/7
F
2/5
8/
T
T
T
1/6
F
3/4
4
3
5
st
0
4
2
1
0
-1
0 1 2 3 4 5

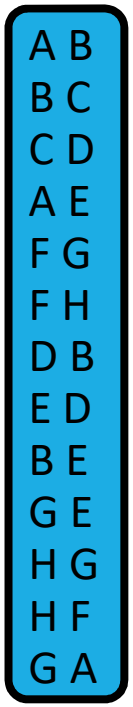

time = 9
st
0
2
F
T
T
4
5
T
1
T
F
3
0
1
2
0/7
2/5
8/
T
T
T
T
F
1/6
3/4
9/
4
3
5
st
0
4
2
1
0
2
0 1 2 3 4 5

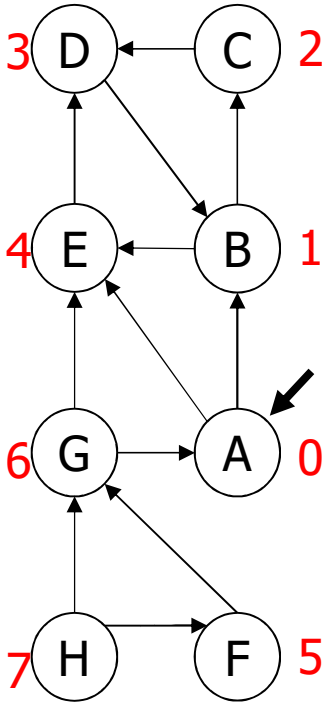

time = 10
st
0
2
F
T
T
4
5
T
B
1
T
F
3
0
1
2
0/7
2/5
8/
F
T
T
T
T
1/6
F
3/4
9/10
B
4
3
5
st
0
4
2
1
0
2
0 1 2 3 4 5

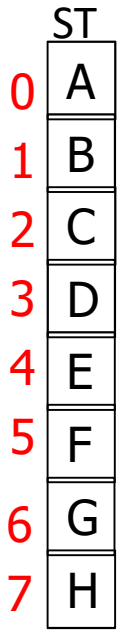

time = 11
st
0
2
F
T
T
4
5
T
B
1
T
F
C
3
0
1
2
0/7
2/5
8/11
T
T
T
C
T
1/6
F
3/4
9/10
B
4
3
5
st
0
4
2
1
0
2
0 1 2 3 4 5

Esempio

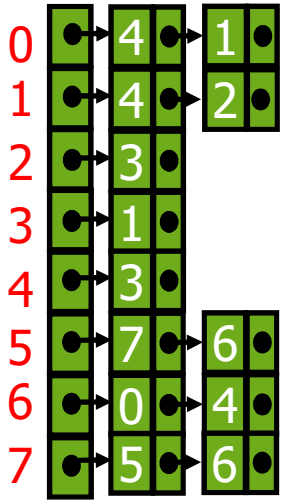
Lista archi

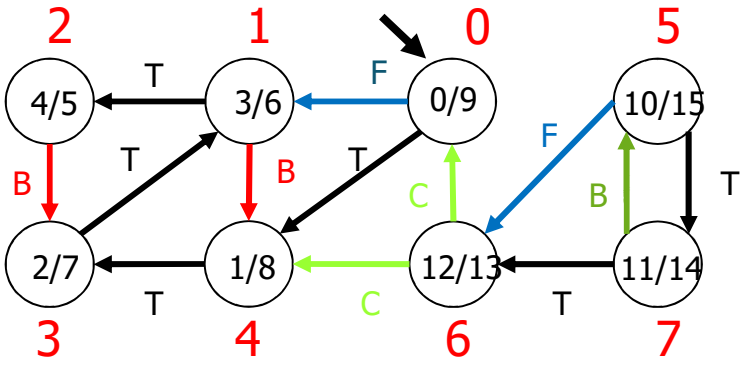


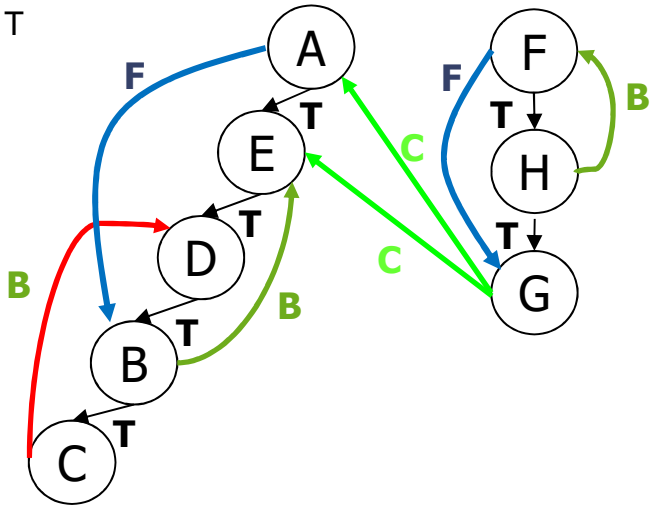




Lista delle adiacenze




2
1
0
5
4/5
3/6
0/9
10/15
2/7
1/8
12/13
11/14
3
4
6
7
T
F
B
C


A
E
D
B
C
F
H
G
F
B
C
T

```
void dfsR(Graph G, Edge e, int *time,
          int *pre, int *post, int *st){
  link t;
  int v, w = e.w;
  Edge x;
  if (e.v != e.w)
    printf("(%s, %s):  T \n",STsearchByIndex(G->tab, e.v),
                            STsearchByIndex(G->tab, e.w));
  st[e.w] = e.v;
  pre[w] = (*time)++;
  for (t = G->ladj[w]; t != G->z; t = t->next)
    if (pre[t->v] == -1)
      dfsR(G, EDGEcreate(w, t->v), time, pre, post, st);
    else {
      v = t->v;
      x = EDGEcreate(w, v);
```

escludi arco fittizio

terminazione implicita della ricorsione

test per non considerare gli archi 2 volte

grafi non orientati

```
    if (pre[w] < pre[v])
      printf("(%s, %s): B\n", STsearchByIndex(G->tab, x.v),
                 STsearchByIndex(G->tab,x.w)) ;
    if (post[v] == -1)
      printf("(%s, %s): B\n", STsearchByIndex(G->tab, x.v),
                STsearchByIndex(G->tab, x.w));
    else
      if (pre[v] > pre[w])
        printf("(%s,%s):F\n",STsearchByIndex(G->tab, x.v),
                  STsearchByIndex(G->tab, x.w));
      else
        printf("(%s,%s):C\n",STsearchByIndex(G->tab, x.v),
                  STsearchByIndex(G->tab, x.w));
  }
  post[w] = (*time)++;
}
```

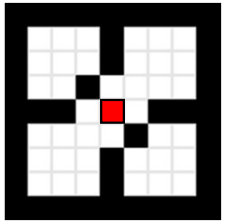
grafi orientati

Complessità (lista adiacenze)

- Inizializzazione $\Theta(|V|)$
- visita ricorsiva da u $\Theta(|E|)$
- $T(n) = \Theta(|V|+|E|)$.
- con la matrice delle adiacenze: $T(n) = \Theta(|V|^2)$.

Applicazione: flood fill

- Scopo: colorare un'intera area di pixel connessi con lo stesso colore (Bucket Tool)
- DFS a partire dal pixel sorgente (seed), terminazione quando si incontra una frontiera (boundary):



http://en.wikipedia.org
Sedgewick, Wayne, Algorithms Part I & II, www.coursera.org

Visita in ampiezza

A partire da un vertice s:

- determina tutti i vertici raggiungibili da s, quindi non visita necessariamente tutti i vertici a differenza della DFS
- calcola la distanza minima da s di tutti i vertici da esso raggiungibili.
- genera un albero della visita in ampiezza.

Ampiezza: espande tutta la frontiera tra vertici già scoperti/non ancora scoperti.

Principi base

Scoperta di un vertice: prima volta che si incontra nella visita raggiungendolo percorrendo un arco.

Il vettore `pre[v]` registra il tempo di scoperta di `v`.

Dato un vertice `v`, il vettore `st[v]` registra il padre di `v` nell'albero della visita in ampiezza.

Non appena il vertice viene scoperto, si registra il tempo di scoperta e il padre nell'albero, concludendo così l'elaborazione del vertice stesso.

Strutture dati

- grafo non pesato come matrice delle adiacenze
- coda `Q` di archi `e=(v,w)`
- vettore `st` dei padri nell'albero di visita in ampiezza
- vettore `pre` dei tempi di scoperta dei vertici
- contatore `time` del tempo

`time, *pre` e `*st` sono locali alla funzione `GRAPHbfs` e passati by reference alla funzione `bfs`.

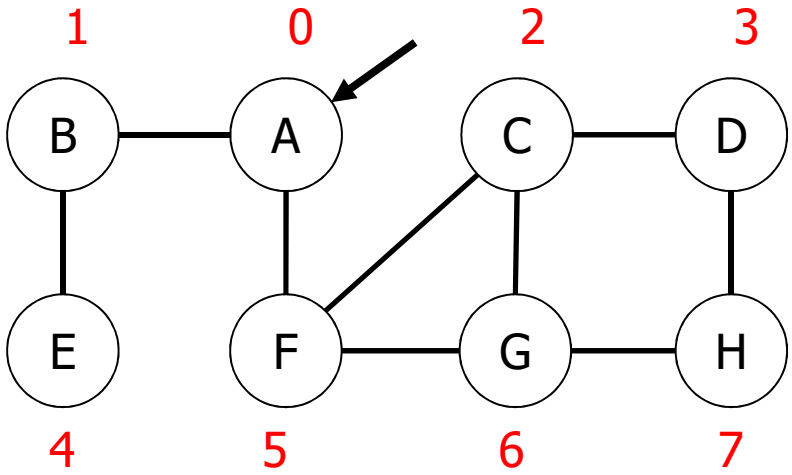
wrapper

Algoritmo

- metti l'arco fittizio di partenza `e=(id,id)` nella coda
- ripeti fintanto che la coda non si svuota
  - estrai dalla coda un arco `e=(v,w)`
  - se `e.w` è un vertice non ancora scoperto (`pre[e.w]==-1`)
    - indica che `e.v` è padre di `e.w` nella BFS (`st[e.w]=e.v`)
    - marca `e.w` come scoperto al tempo `time` (`pre[e.w]=time++`) e incrementa il tempo
    - trova i vertici non ancora scoperti `x` adiacenti a `e.w` e metti in coda tutti gli archi che connettono `e.w` ad essi.

`bfs`: funzione che visita in ampiezza a partire da un vertice di partenza `id`.

Esempio



Lista archi

A	B
C	D
B	E
C	F
F	G
D	H
A	F
C	G
G	H

ST

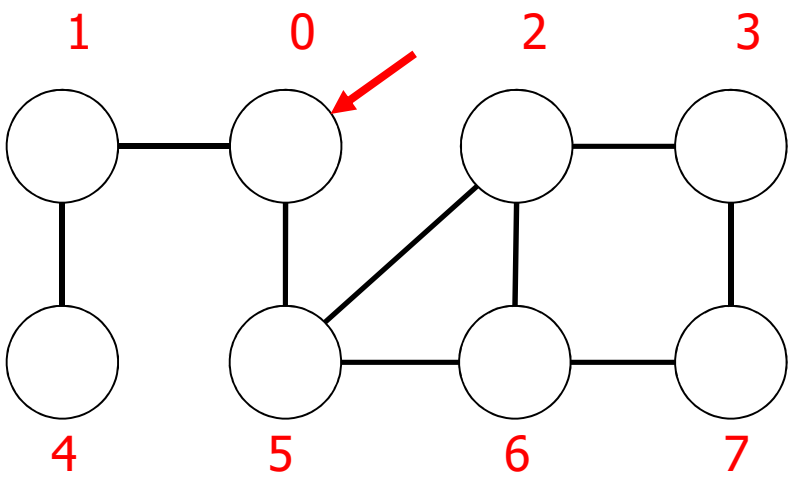
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

st

0	1	2	3	4	5	6	7
-1	-1	-1	-1	-1	-1	-1	-1

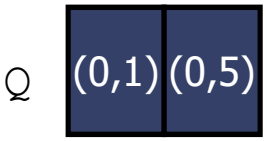
Q (0,0)

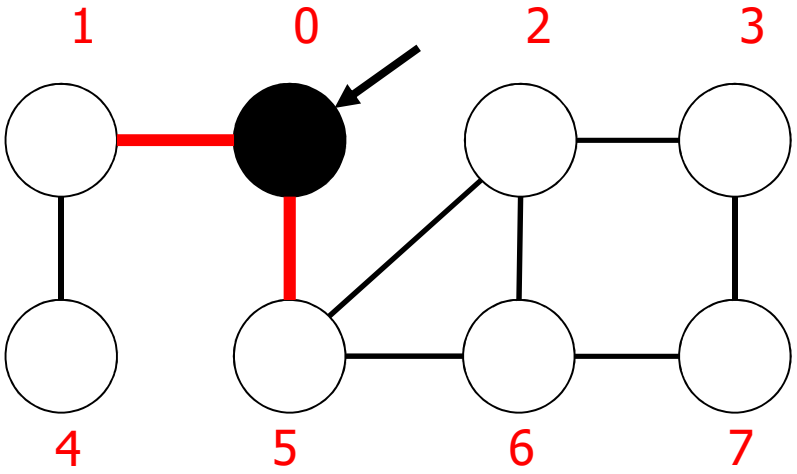
st



st

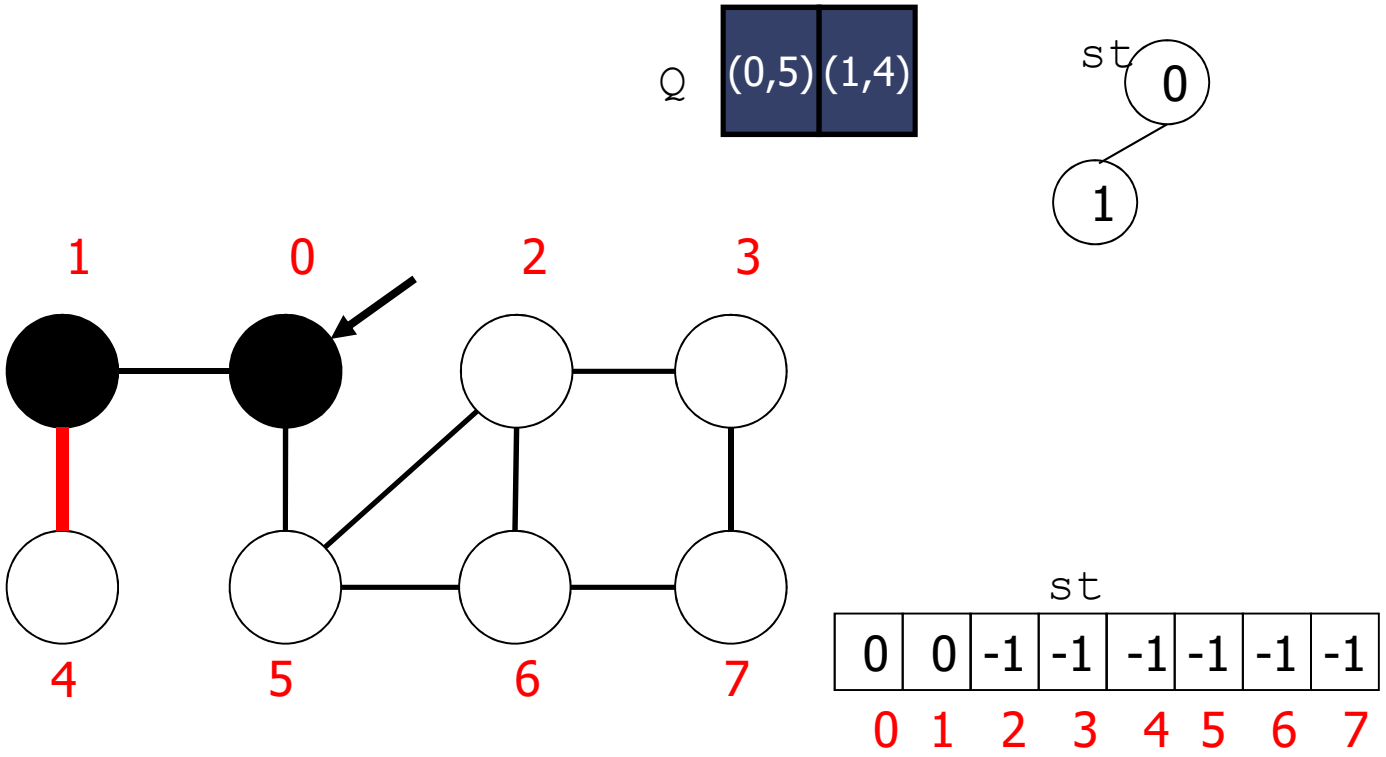
-1	-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6	7

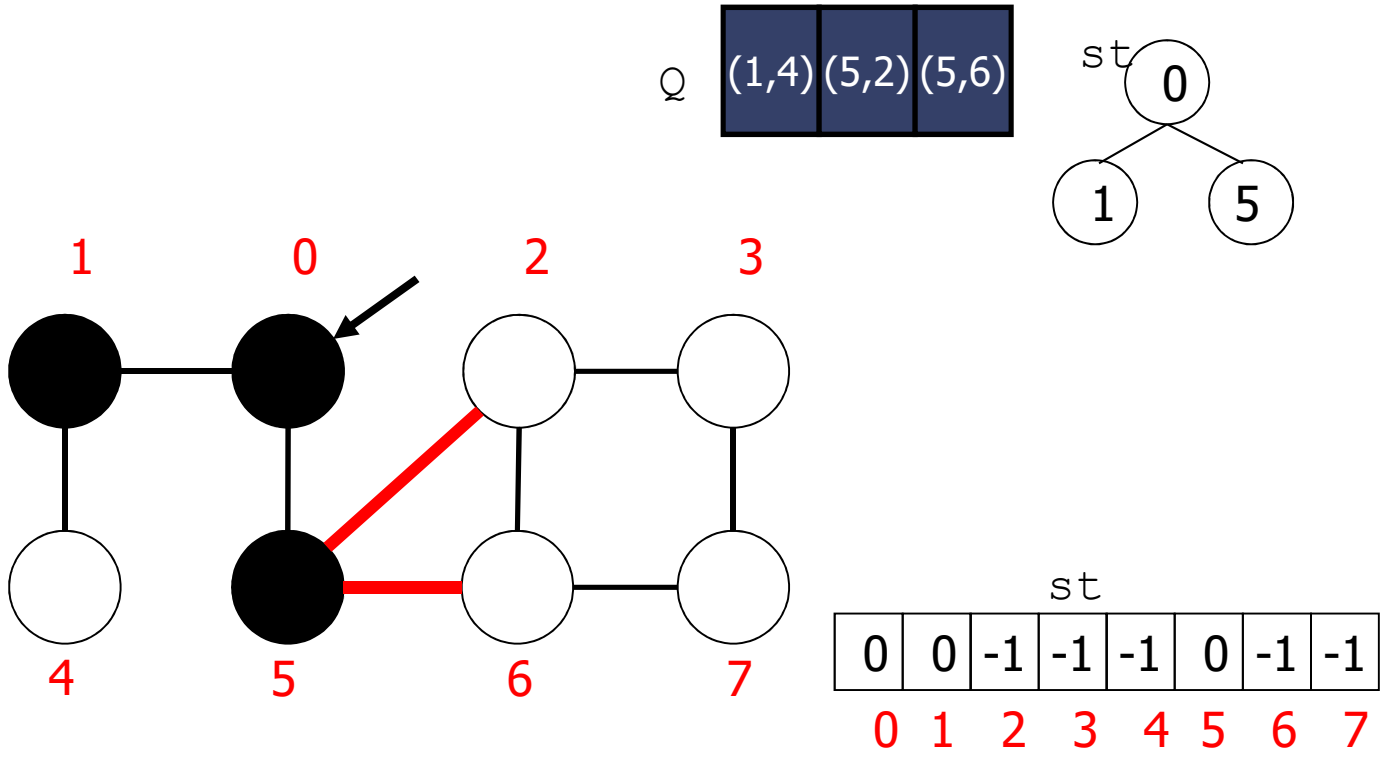


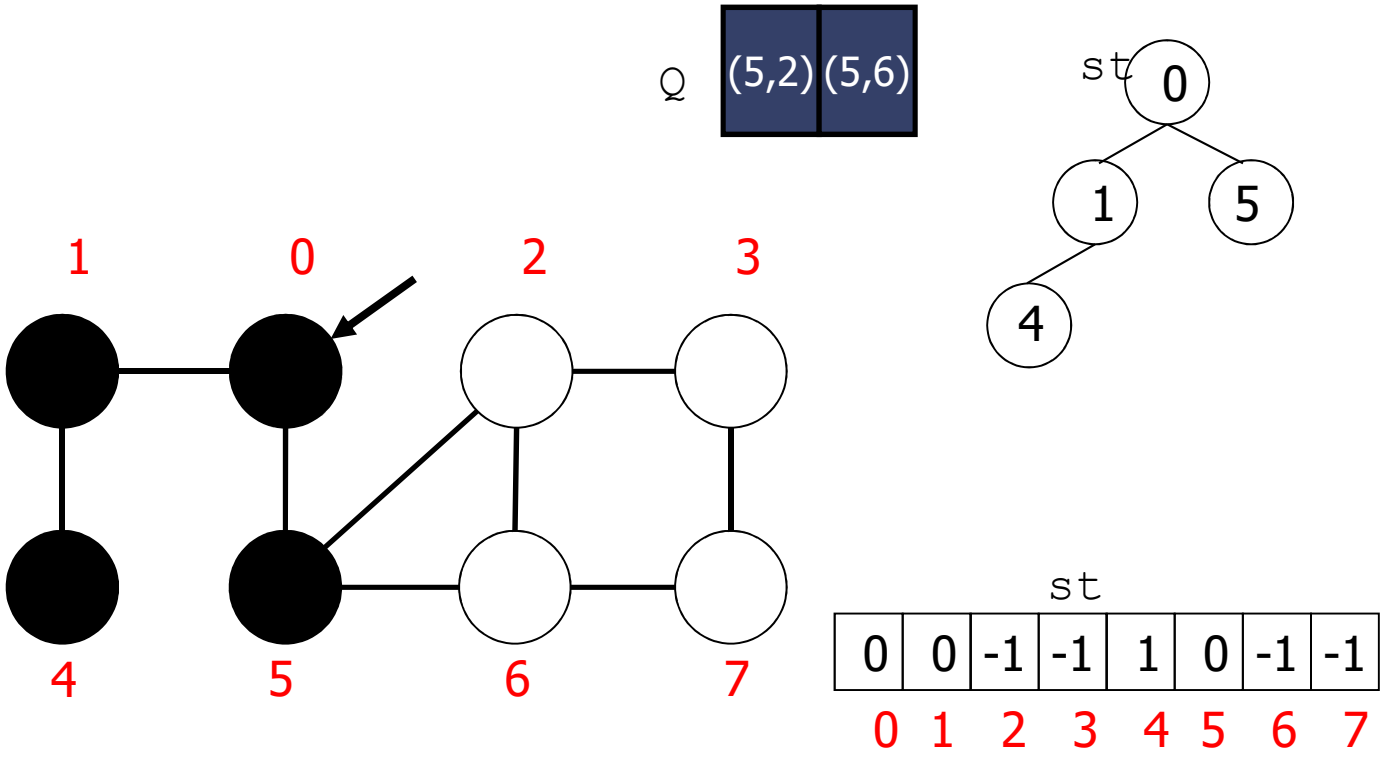


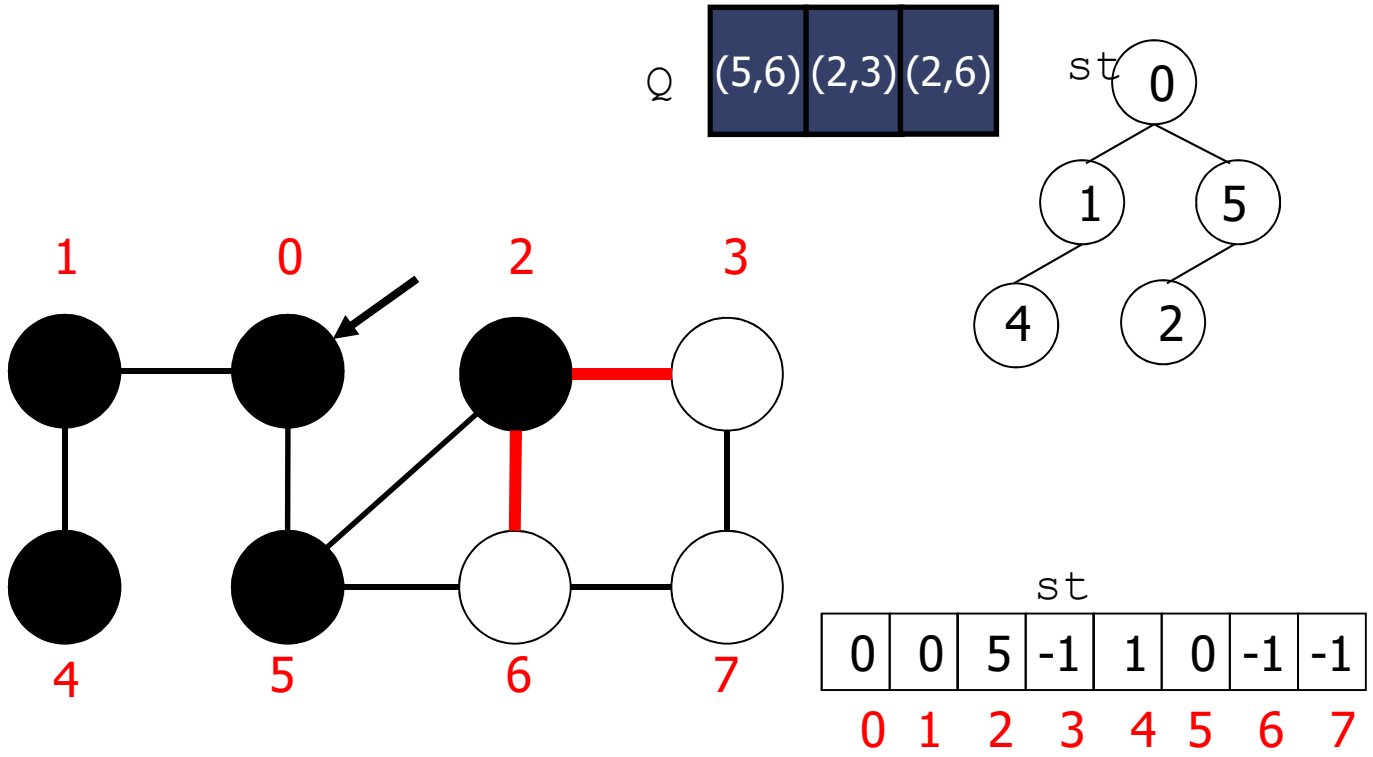
st

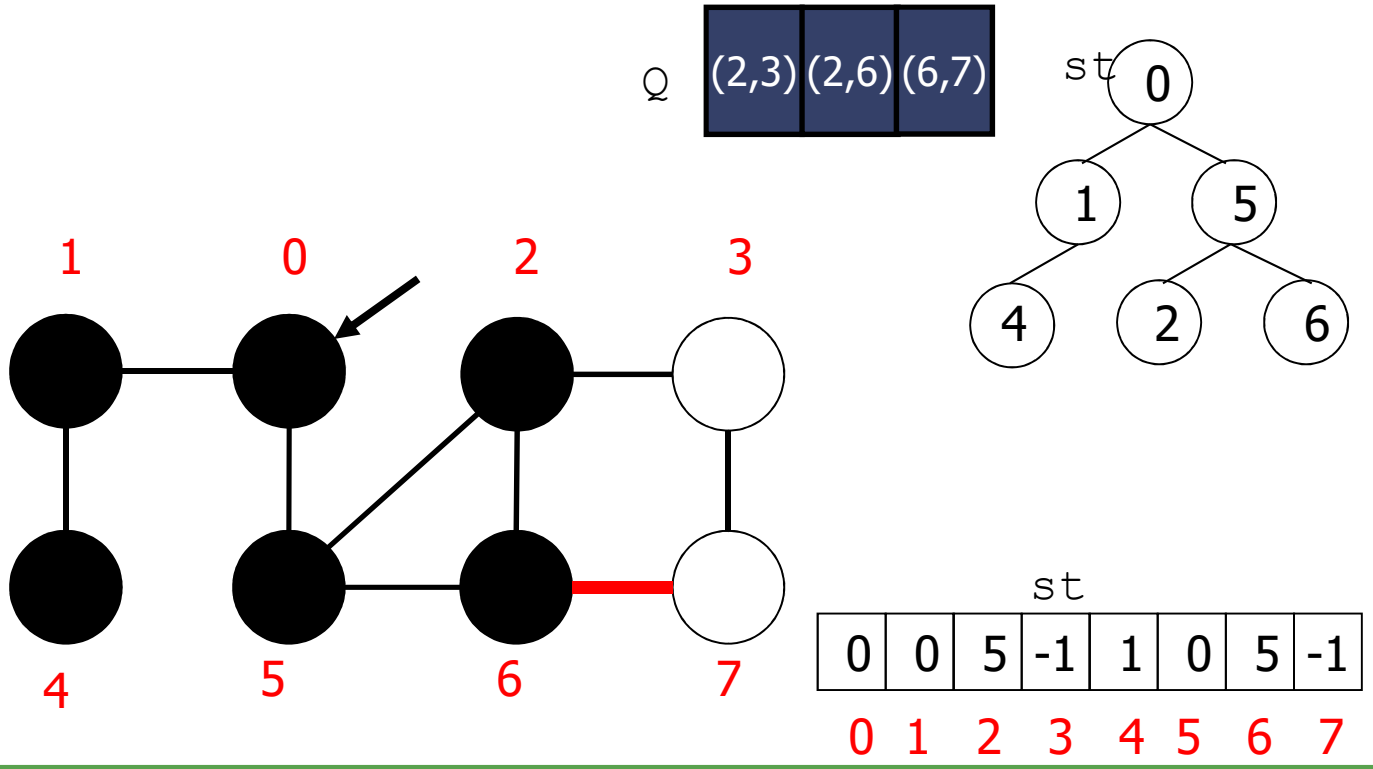
0	-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6	7

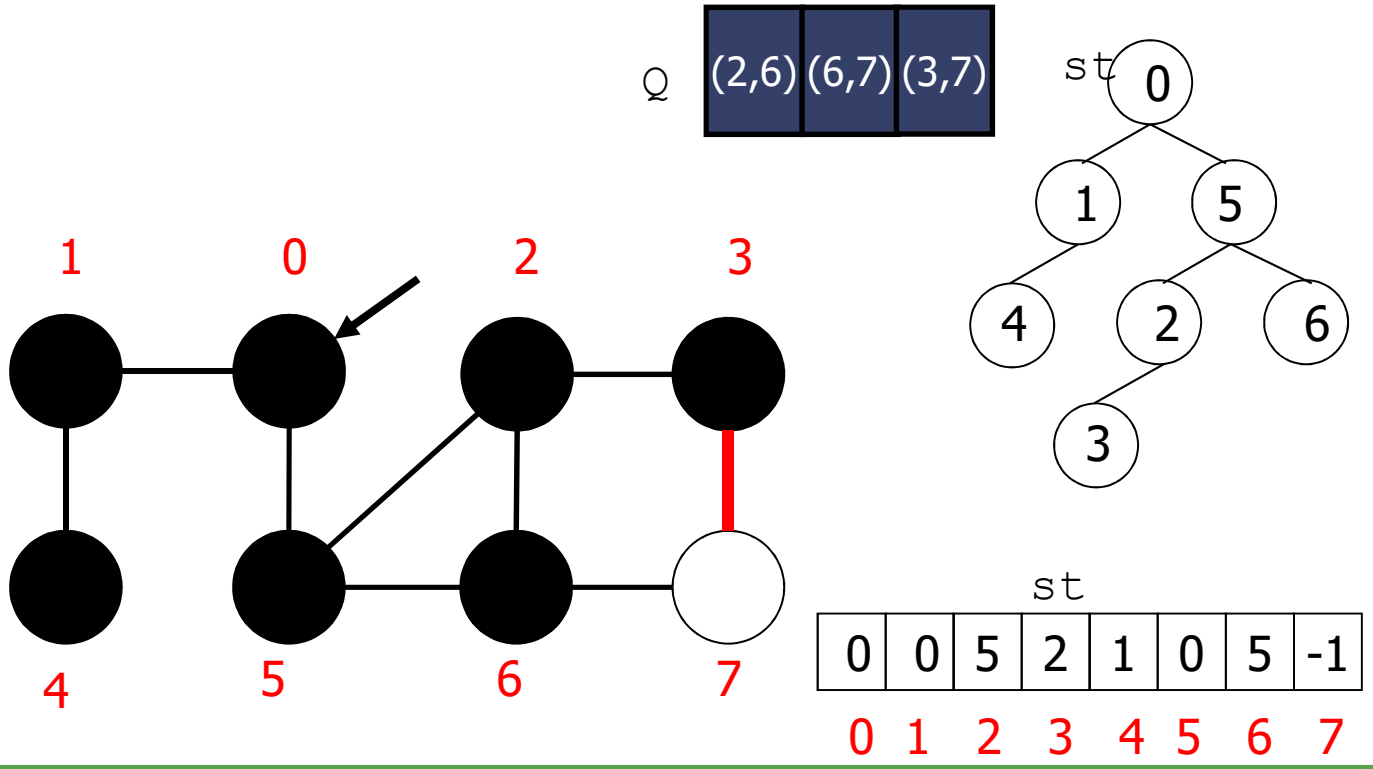

Q
(0,5)
(1,4)
st
0
1
1
0
2
3
4
5
6
7
st
0 0 -1 -1 -1 -1 -1 -1
0 1 2 3 4 5 6 7

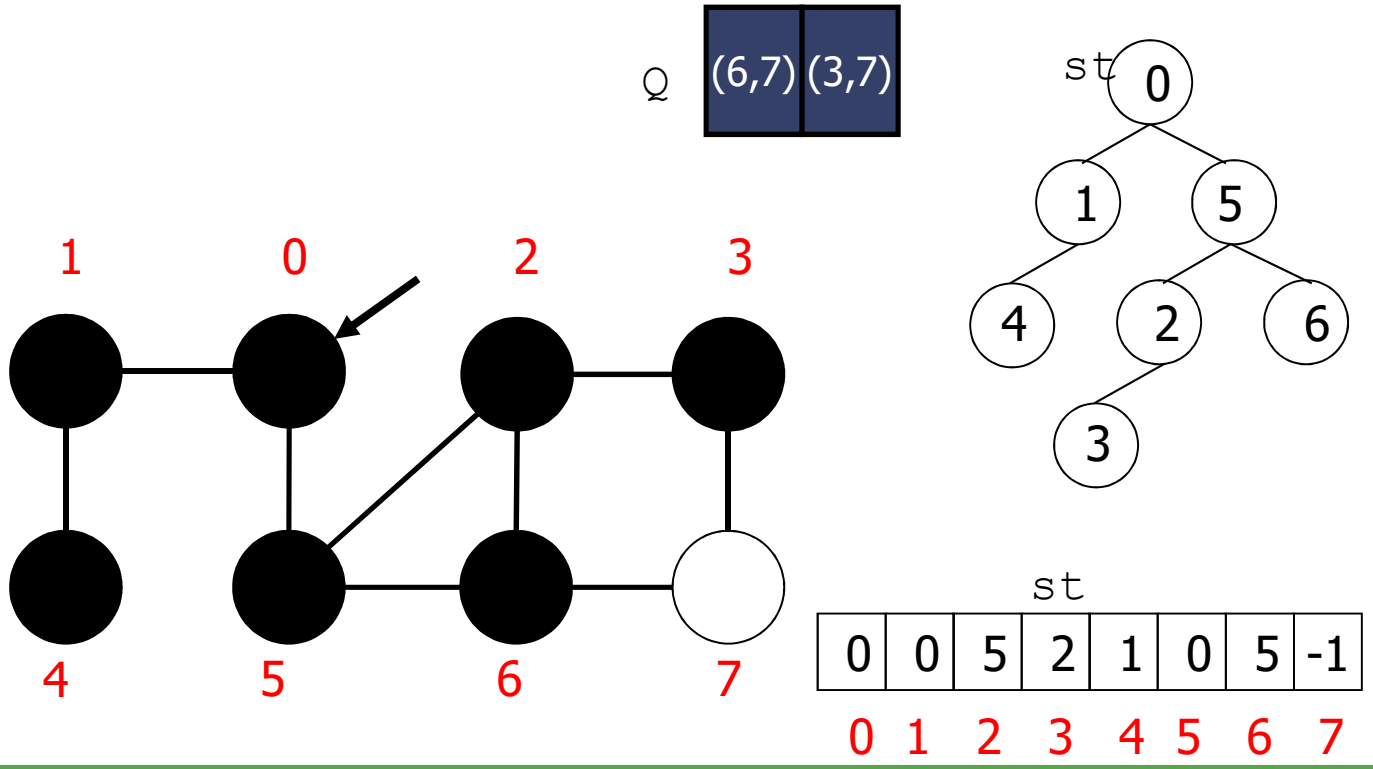

Q
(1,4)
(5,2)
(5,6)
st
0
1
5
1
0
2
3
4
5
6
7
st
0 0 -1 -1 -1 0 -1 -1
0 1 2 3 4 5 6 7

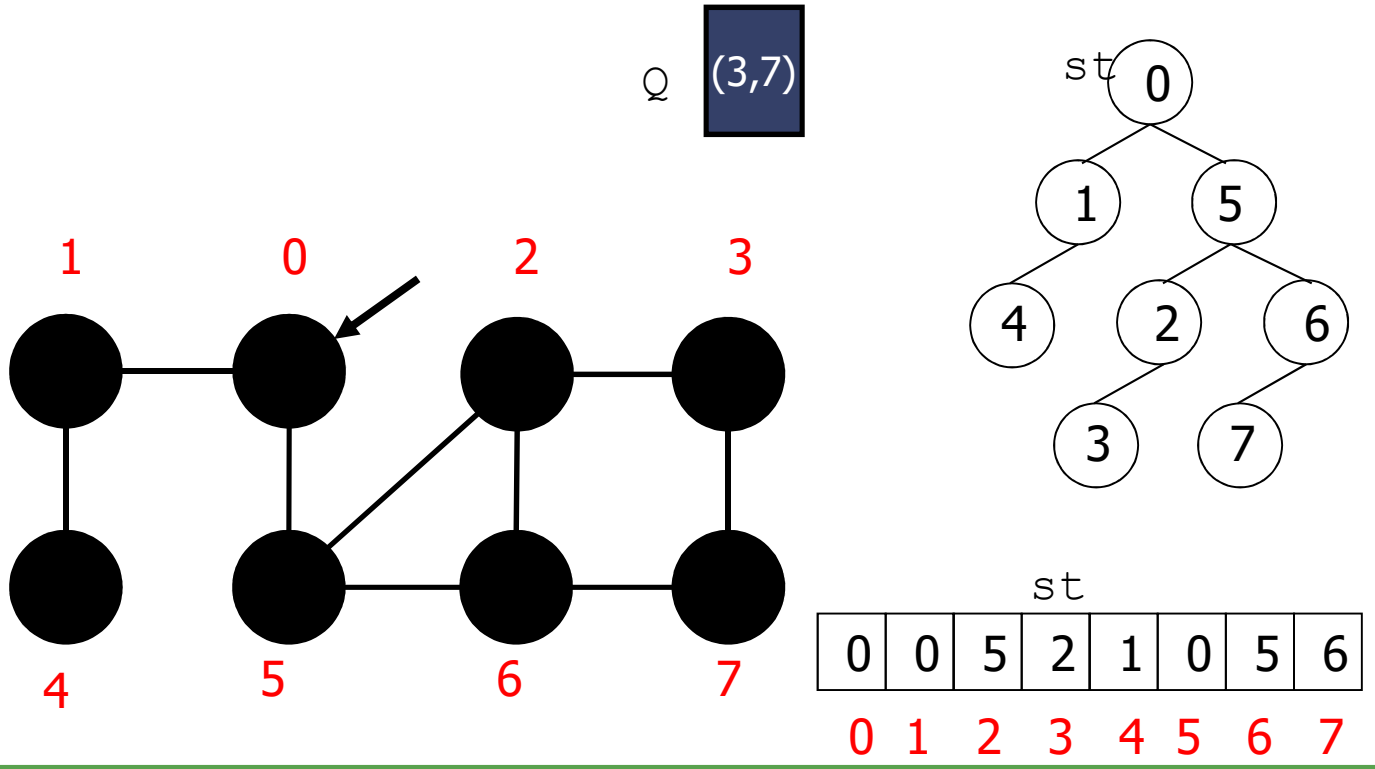

Q
(5,2)
(5,6)
st
0
1
5
4
1
0
2
3
4
5
6
7
st
0 0 -1 -1 1 0 -1 -1
0 1 2 3 4 5 6 7

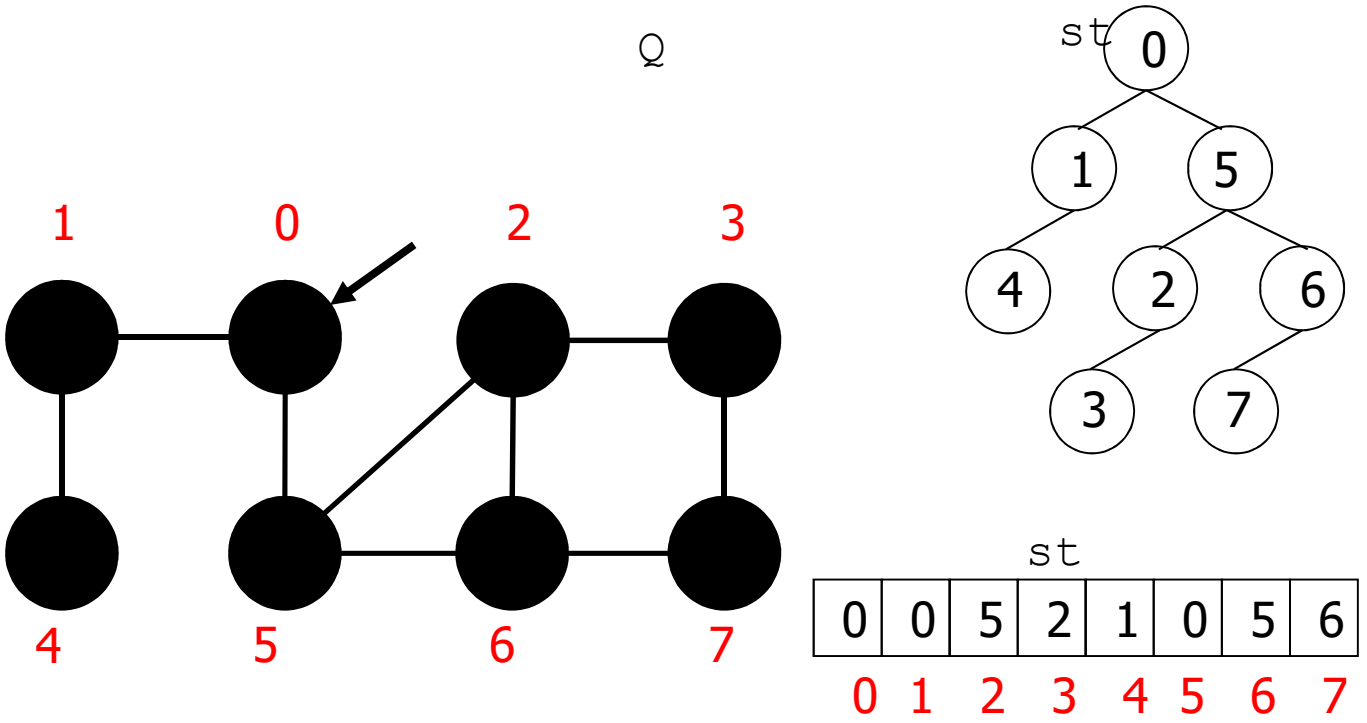

Q
(5,6)
(2,3)
(2,6)
st
0
1
5
4
2
1
0
2
3
4
5
6
7
st
0 0 5 -1 1 0 -1 -1
0 1 2 3 4 5 6 7


Q
(2,3)
(2,6)
(6,7)
st
0
1
5
4
2
6
1
0
2
3
4
5
6
7
st
0 0 5 -1 1 0 5 -1
0 1 2 3 4 5 6 7


Q
(2,6)
(6,7)
(3,7)
st
0
1
5
4
2
6
3
1
0
2
3
4
5
6
7
st
0 0 5 2 1 0 5 -1
0 1 2 3 4 5 6 7


Q
(6,7)
(3,7)
st
0
1
5
4
2
6
3
1
0
2
3
4
5
6
7
st
0 0 5 2 1 0 5 -1
0 1 2 3 4 5 6 7


Q
(3,7)
st
0
1
5
4
2
6
3
7
1
0
2
3
4
5
6
7
st
0 0 5 2 1 0 5 6
0 1 2 3 4 5 6 7


Q
st
0
1
5
4
2
6
3
7
1
0
2
3
4
5
6
7
st
0 0 5 2 1 0 5 6
0 1 2 3 4 5 6 7

```
void GRAPHbfs(Graph G, int id) {
  int v, time=0, *pre, *st, *dist;
  /* allocazione di pre, st e dist */
  for (v=0; v < G->V; v++) {
    pre[v] = -1; st[v] = -1; dist[v] = INT_MAX;
  }
  bfs(G, EDGEcreate(id,id), &time, pre, st, dist);
  printf("\n Resulting BFS tree \n");
  for (v=0; v < G->V; v++)
    if (st[v] != -1)
      printf("%s's parent is:%s\n",STsearchByIndex(G->tab, v),
             STsearchByIndex(G->tab, st[v]));
  printf("\n Levelizing \n");
  for (v=0; v < G->V; v++)
    if (st[v] != -1)
      printf("%s: %d \n",STsearchByIndex(G->tab,v),dist[v]);
}
```

```
void bfs(Graph G, Edge e, int *time, int *pre, int *st,
         int *dist) {
  int x;
  Q q = Qinit();
  Qput(q, e);
  dist[e.v]=-1;
  while (!Qempty(q))
    if (pre[(e = Qget(q)).w] == -1) {
      pre[e.w] = (*time)++;
      st[e.w] = e.v;
      dist[e.w] = dist[e.v]+1;
      for (x = 0; x < G->V; x++)
        if (G->madj[e.w][x] == 1)
          if (pre[x] == -1)
            Qput(q, EDGEcreate(e.w, x));
    }
}
```

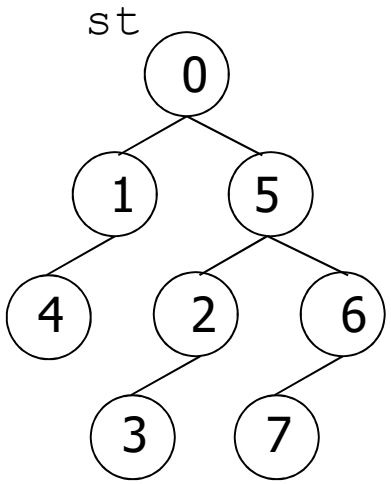
matrice delle adiacenze

Complessità

- Operazioni sulla coda
- Scansione della matrice delle adiacenze T(n) = $\Theta(|V|^2)$.
- Con la lista delle adiacenze: T(n) = $O(|V|+|E|)$.

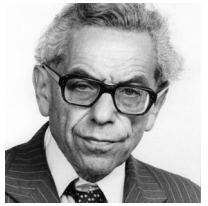
Proprietà

Cammini minimi: la visita in ampiezza determina la minima distanza tra s e ogni vertice raggiungibile da esso.

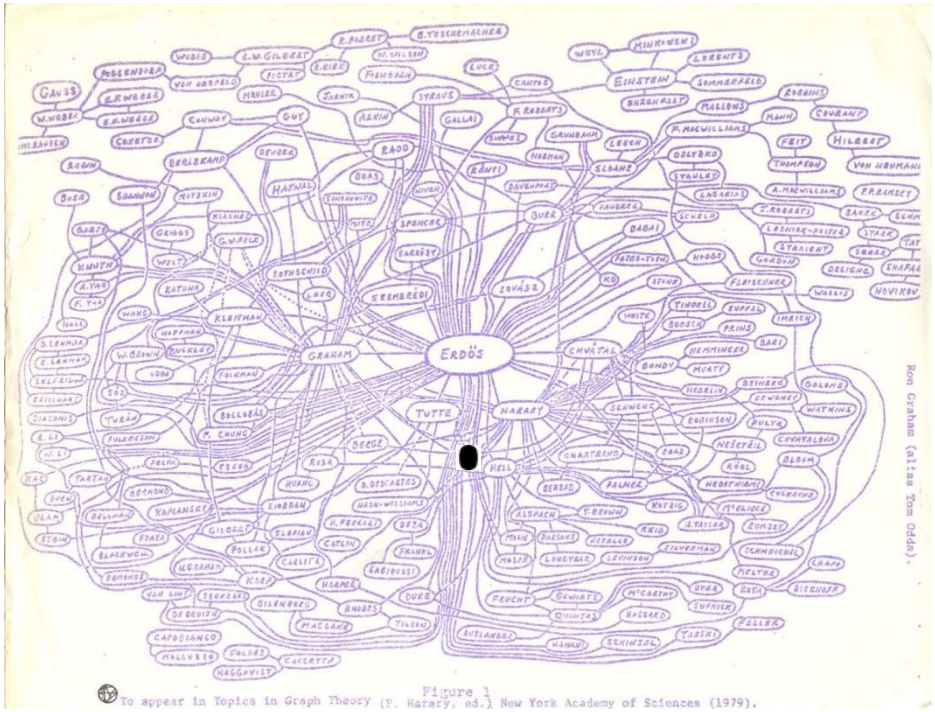


Cammino minimo da 0 a 3:
0, 5, 2, 3
lunghezza = 3

Applicazione: I numeri di Erdős



- Paul Erdős (1913-1996): matematico ungherese «itinerante»: pubblicazioni con moltissimi coautori
- Grafo non orientato:
  - vertici: matematici
  - arco: unisce 2 matematici che hanno una pubblicazione in comune
- numero di Erdős: distanza minima di ogni matematico da Erdős
- BFS

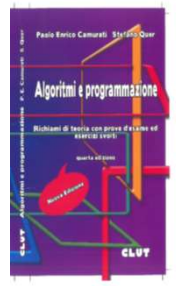


http://www.oakland.edu/upload/images/Erdos%20Number%20Project/cgraph.jpg
Sedgewick, Wayne, Algorithms Part I & II, www.coursera.org

Riferimenti

- Visita in profondità:
  - Sedgewick Part 5 18.2, 18.3, 18.4
  - Cormen 23.3
- Visita in ampiezza:
  - Sedgewick Part 5 18.7
  - Cormen 23.2
- Numero di Erdős:
  - Bertossi 9.5.2

Esercizi di teoria



- 10. Visite dei grafi e applicazioni
  - 10.2 Visita in ampiezza
  - 10.3 Visita in profondità