GRAFI

I grafi sono una struttura matematica fondamentale: servono a rappresentare un'infinità di tipi di problemi.

I grafi sono strutture composte di due tipi di oggetti:

- *vertici*/nodi/siti
- *archi*/collegamenti/spigoli (edges,links)

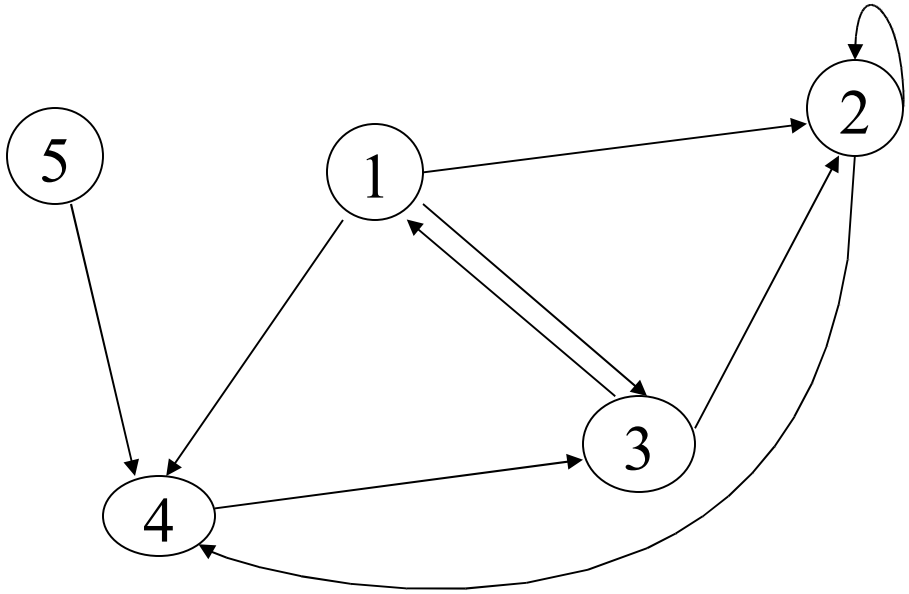
Grafi Orientati (o diretti)

Un grafo orientato G `e una coppia (**V**, **E**) dove

- **V** e' un insieme non vuoto,
- **E** e' una *relazione* su **V**, ossia un insieme di coppie *ordinate* di elementi di **V**

Gli elementi di **V** sono i *vertici*, gli elementi di **E** sono gli *archi*

Esempio



V = {1,2,3,4,5}

E = {(1,2), (1,3),(1,4),(2,2),(2,4),(3,1),(3,2),(4,3),(5,4)}

Il Grafo ha 5 vertici e 9 archi

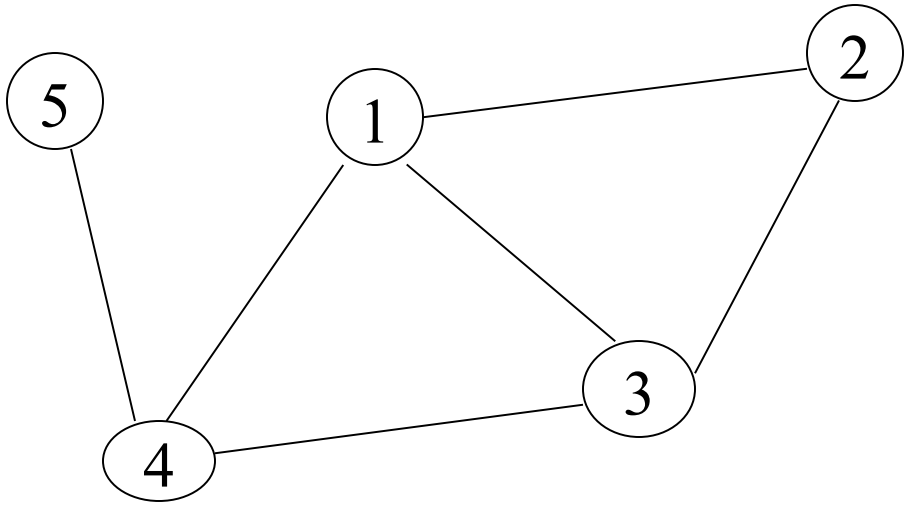
Grafi Non-Orientati

Un grafo non-orientato G `e una coppia (**V**, **E**) dove

- **V** e' un insieme non vuoto,
- **E** e' un insieme di coppie *non-ordinate* di elementi di **V**

- Coppia non ordinata: {u,v} = {v,u}

Esempio



V = {1,2,3,4,5}

E = {{1,2}, {1,3},{2,3},{1,4},{4,3},{5,4}}

Il Grafo ha 5 vertici e 6 archi

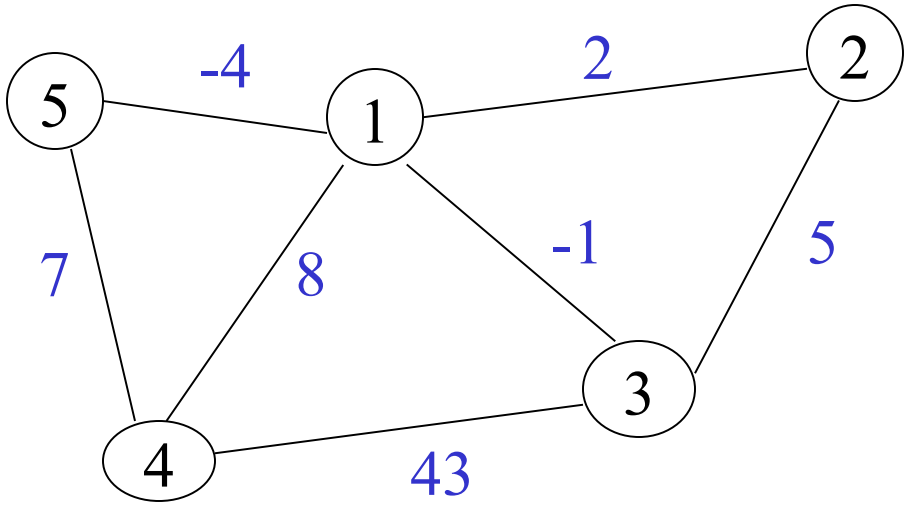
Grafi pesati

Sia nel caso orientato che in quello non orientato agli *archi* di un grafo può essere associata un'informazione, sovente un valore numerico detto *costo* o *peso*.

I grafi in cui gli archi hanno un peso sono detti *grafi pesati*

Un grafo pesato può essere indicato con G = (V,E,w) dove ad esempio w: E $\longrightarrow$ **R** funzione peso

Esempio



V = {1,2,3,4,5}

E = {{1,2}, {1,3},{1,4},{2,3},{4,3},{5,4},{5,1}}

Terminologia (per i grafi orientati)

- In un grafo orientato G = (V,E):

  un vertice **v** e' *adiacente* a **u** se c'è un arco

  (u,v) $\in$ E.

  Diciamo che l'arco **(u,v)** esce da **u** ed entra in **v**.

- Un *cammino di lunghezza* **k** da **u** a **u'** e'una sequenza di vertici $\mathbf{v_0}$=**u**, $\mathbf{v_1}$,…,$\mathbf{v_k}$ = **u'**, tali che

  $(\mathbf{v_{i-1}}, \mathbf{v_i})$ $\in$ E per i =1,…,k

  Il cammino è *semplice* se tutti i $\mathbf{v_i}$ sono distinti

Terminologia (segue)

- Diciamo che u e' *raggiungibile* da u' se c'e' un cammino da u' a u.
- Un cammino $\langle v_0, v_1, \ldots, v_k \rangle$ forma un *ciclo* se

  $v_0 = v_k$ e $k > 0$

  Il ciclo è *semplice* se $v_1, \ldots, v_k$ sono distinti

Un grafo orientato aciclico (=senza cicli) e'chiamato **DAG** (directed acyclic graph)

In un grafo *non-orientato* si danno le stesse definizioni con le ovvie modifiche:

- un vertice **u** e' adiacente a **v** se c'è un arco

  $\{u,v\} \in E$.

- Un cammino $\langle v_0, v_1, \ldots, v_k \rangle$ forma un *ciclo* se

  $v_0 = v_k$ e $k > 1$

Altre nozioni:

- un grafo *non-orientato* e' *connesso* se ogni coppia di vertici e' mutuamente raggiungibile.
- un grado *orientato* è *fortemente connesso* se ogni coppia di vertici e' mutuamente raggiungibile

Una *componente connessa* (*fortemente connessa*) di un grafo è un insieme ***massimale*** di vertici mutuamente raggiungibili

Proprietà:

un grafo non-orientato connesso è ***aciclico***

se e solo se

- vi e' esatamente un cammino tra ogni coppia di vertici
- ha esattamente |V| -1 archi dove |V| è il numero di vertici

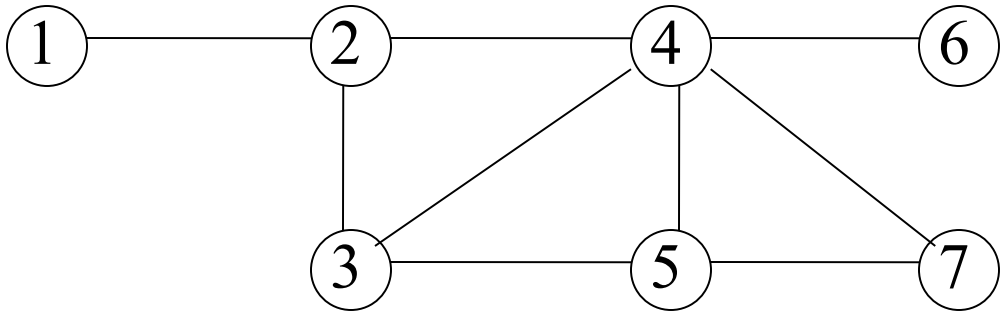
Un grafo non orientato connesso e aciclico è chiamato **albero** (libero)

Rappresentazione di grafi non pesati

Modi principali di rappresentazione:

Matrice di adiacenza: Il grafo è rappresentato da una matrice V x V i cui elementi assumono valori binari (ad esempio 0 e 1). L'elemento in posizione (i, j) è uguale a 1 se e solo se il vertice j è adiacente al vertice i.

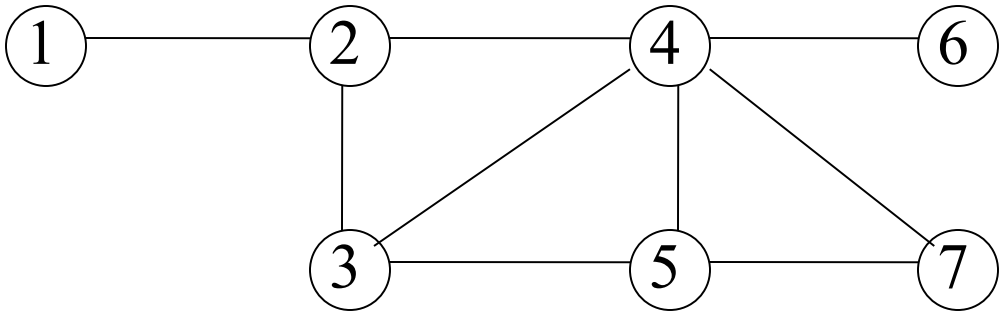
Liste di adiacenza: Il grafo è rappresentato da un vettore di liste, ognuna delle quali contenente i vertici adiacenti al vertice cui la lista è associata.

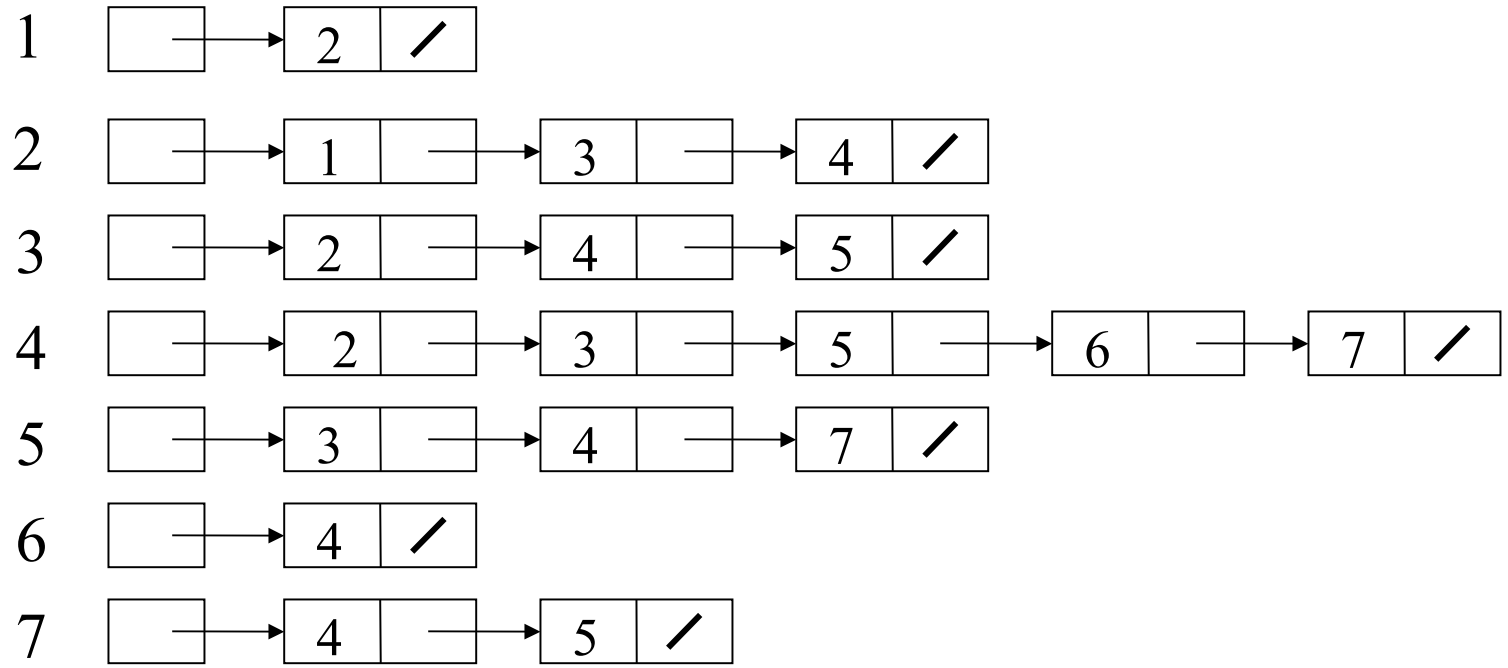


$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Spazio richiesto: $\mathbf{O(V^2)}$

Adatta per grafi *densi*, in cui |E| è dell'ordine di $|V|^2$

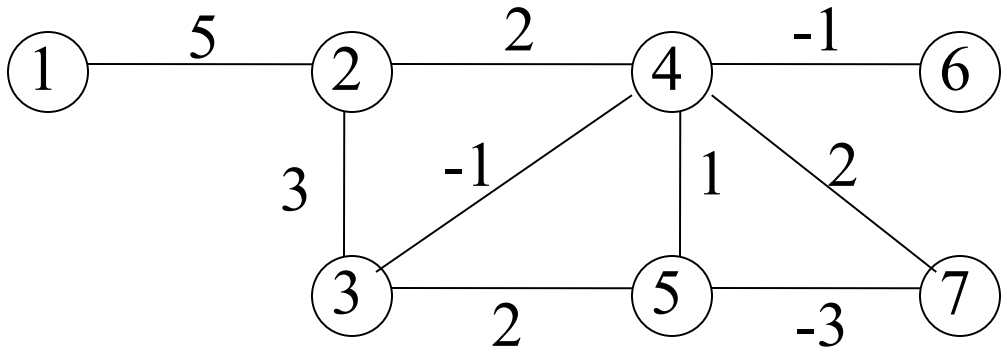




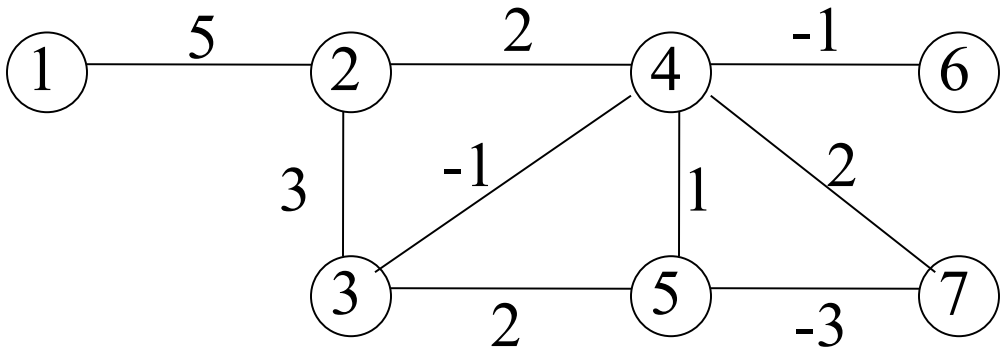
Spazio richiesto: **O (V + E)**

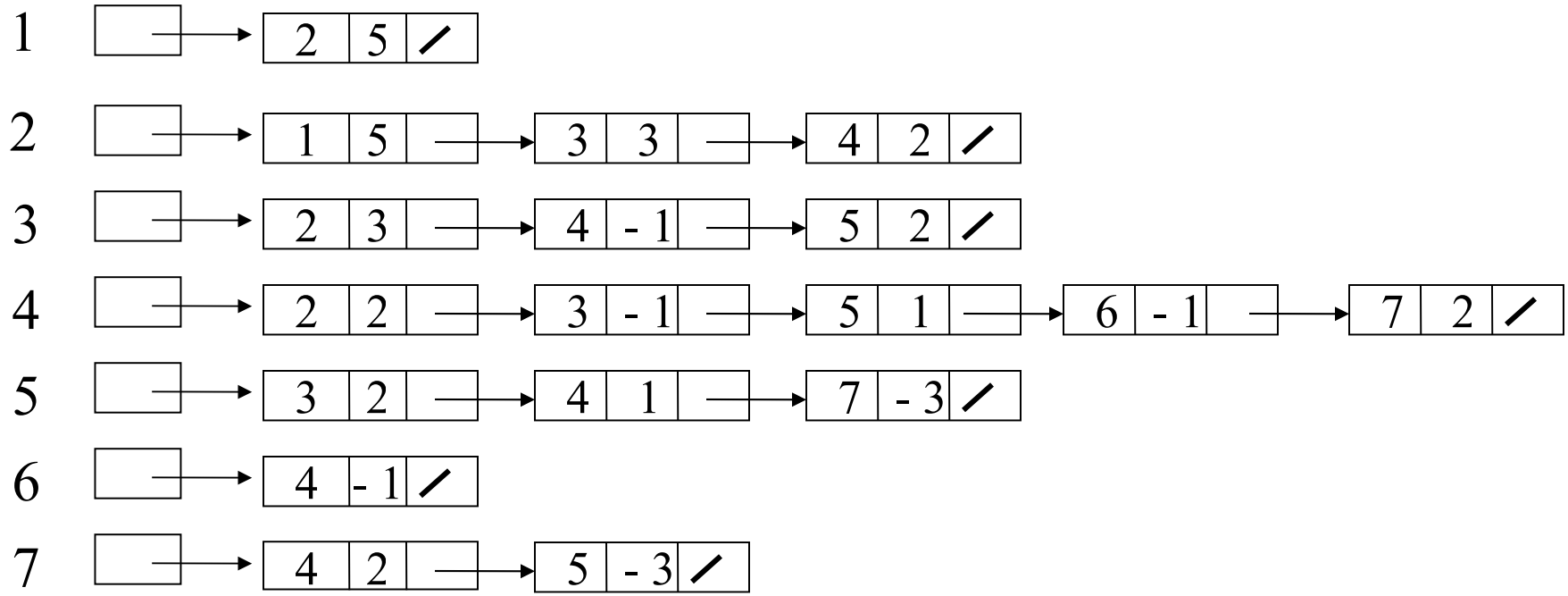
E' adatta per grafi *sparsi*, in cui |E| è molto minore di $|V|^2$

Se il grafo è pesato:



$$A = \begin{pmatrix} 0 & 5 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 3 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & -1 & 2 & 0 & 0 \\ 0 & 2 & -1 & 0 & 1 & -1 & 2 \\ 0 & 0 & 2 & 1 & 0 & 0 & -3 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -3 & 0 & 0 \end{pmatrix}$$


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


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

Algoritmi di visita

Scopo: visitare tutti i vertici di un grafo

Realizzazione: Dividiamo l'insieme dei vertici in tre insiemi colorati in modo diverso

Bianco: Vertici non ancora visitati o "scoperti"

Grigio: Vertici visitati i cui adiacenti non sono ancora stati tutti scoperti, quindi vertici utili per continuare la visita

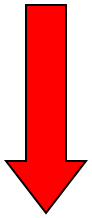
Nero: Vertici visitati e non più utili per continuare a scoprire altri vertici (i loro adiacenti sono stati tutti già scoperti)

Consideriamo il seguente algoritmo:

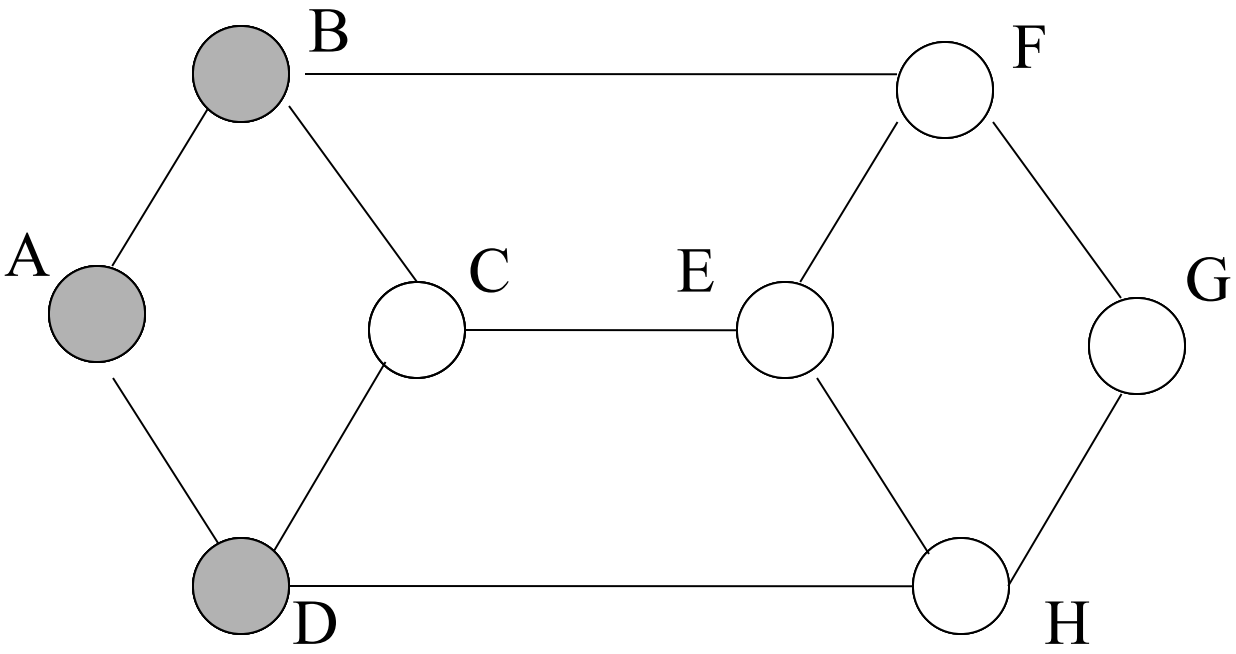
```
VISITA (G, s)
    D ←make_empty
    color [s] ←gray
    add (D, s)
    while (not_empty (D) )do {
   u ←first (D)
   if (c'è v bianco ∈ ADJ [u])
            {   color [v] ←gray
              π  [v] ←u
                   add (D, v)
            } else  {    color [u] ←black
                    remove_first (D)
          }
   }
```

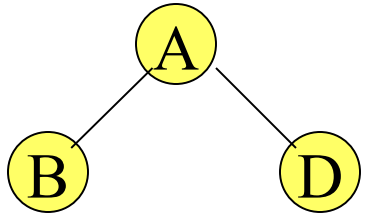
La struttura dati D è una

CODA

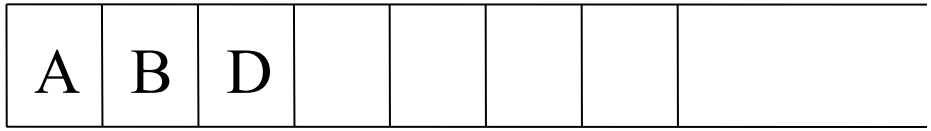


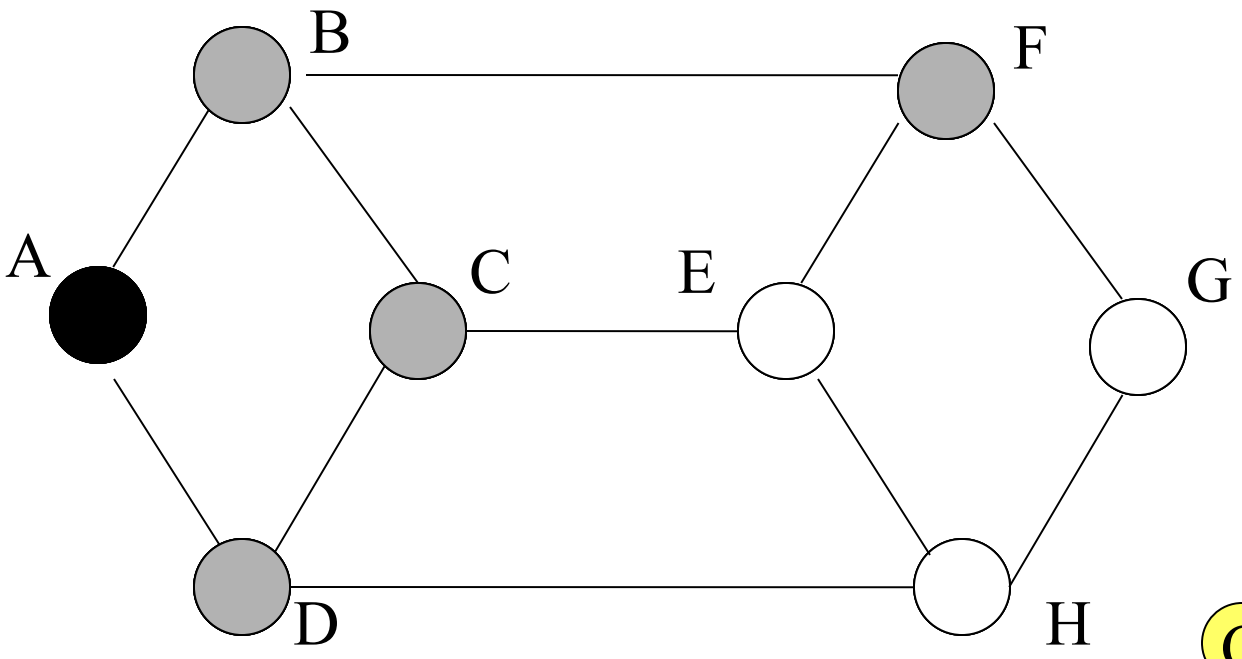
Visita in ampiezza o Breadth-First-Search (BFS)

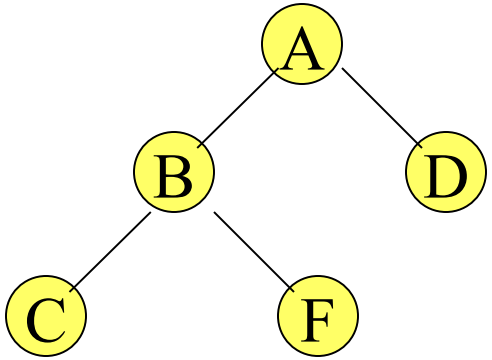




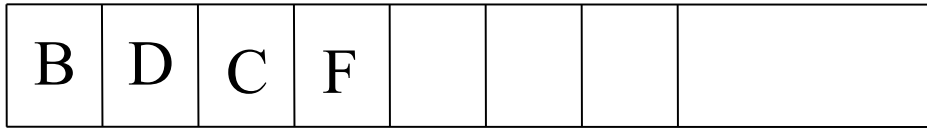
Q

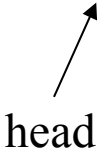


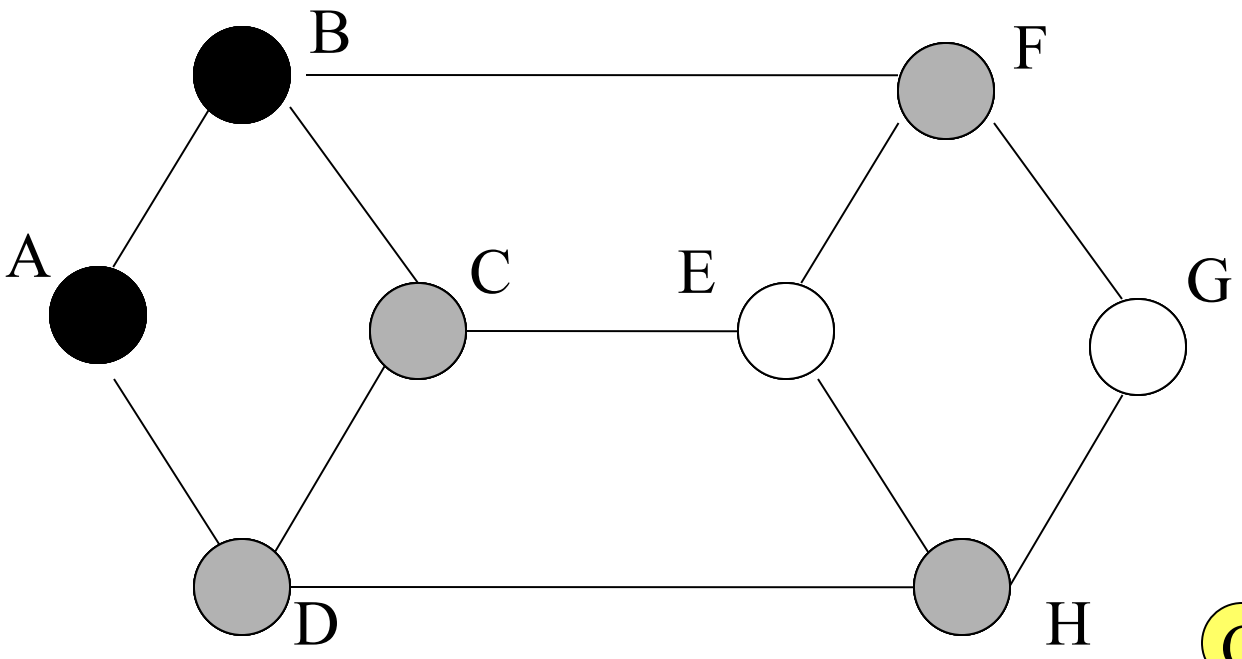


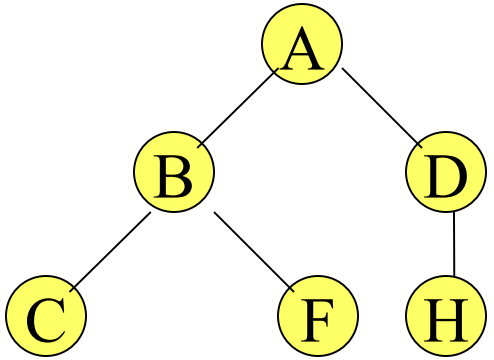


Q

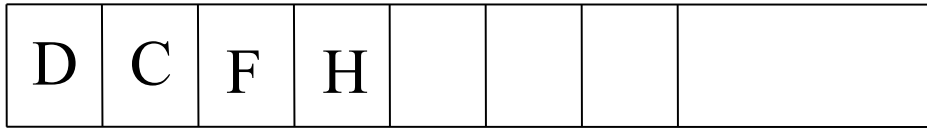


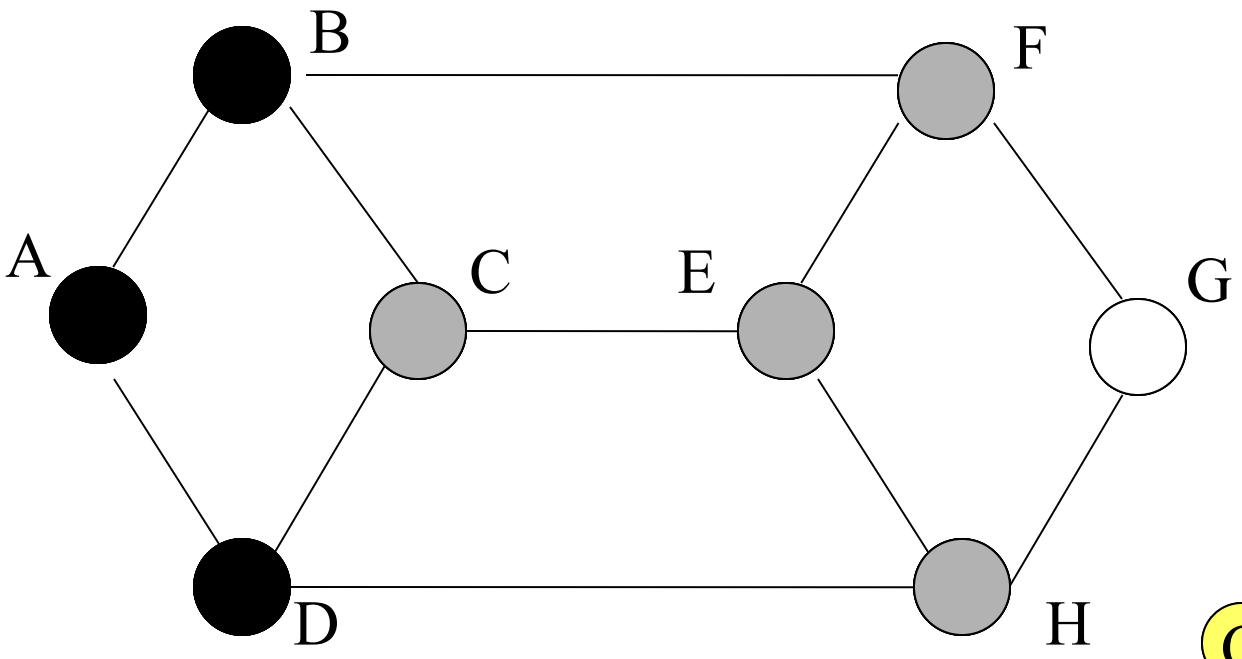


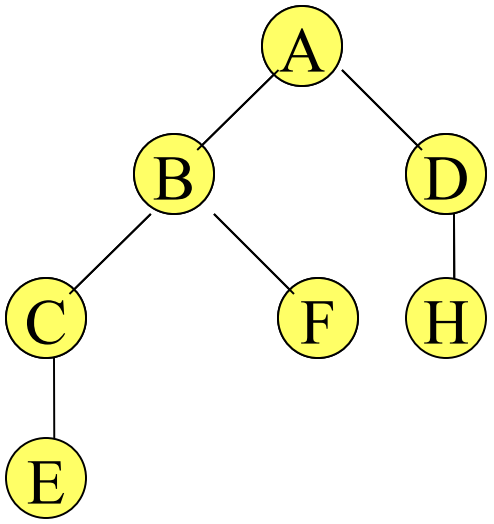




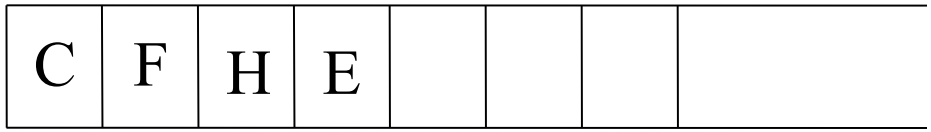
Q

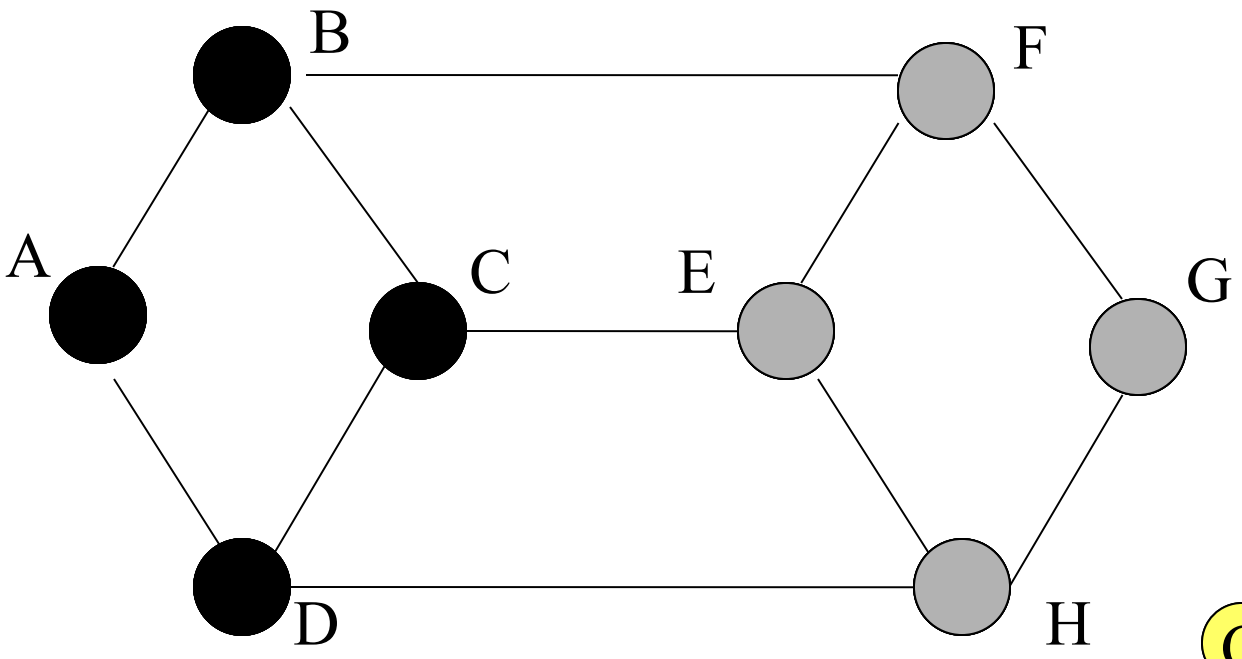


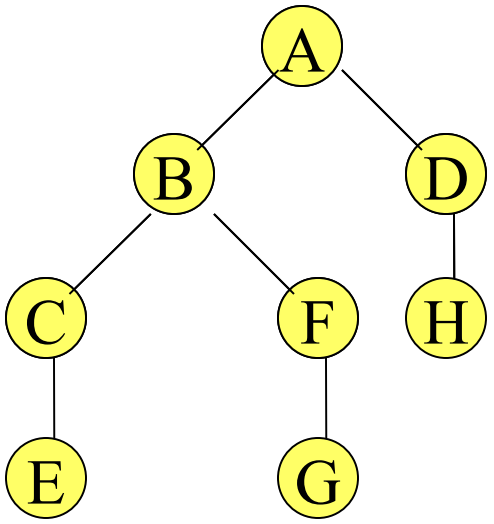




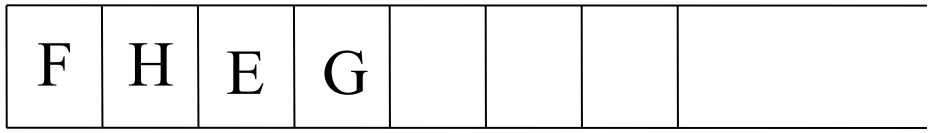
Q

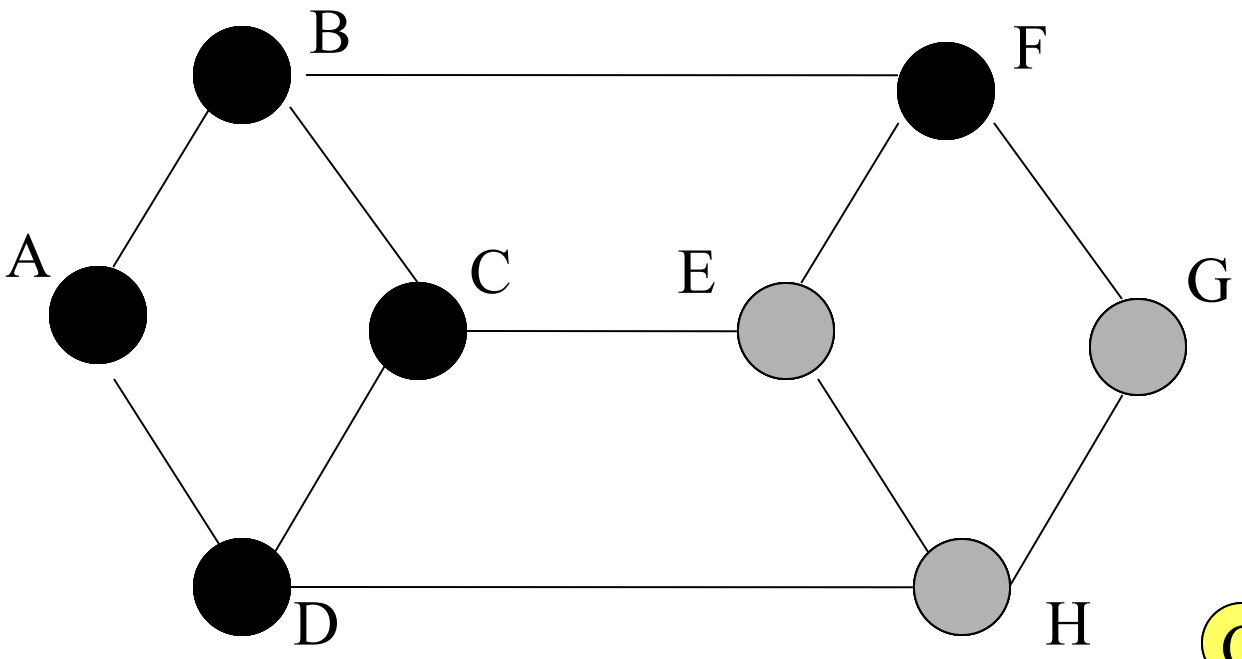


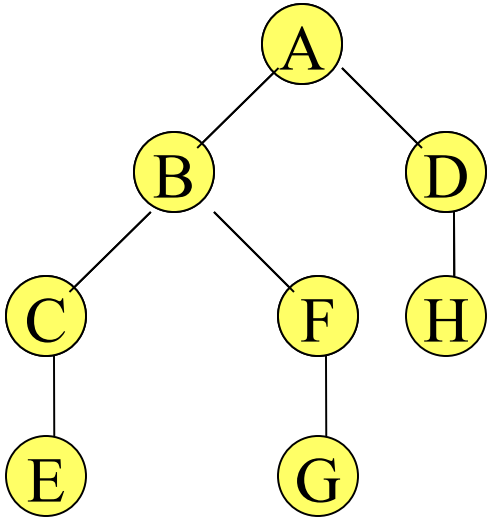




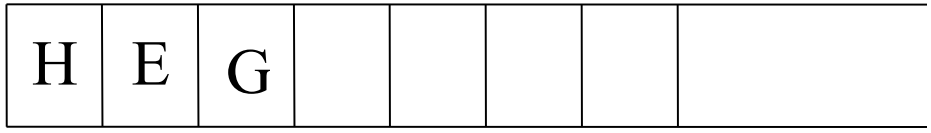
Q

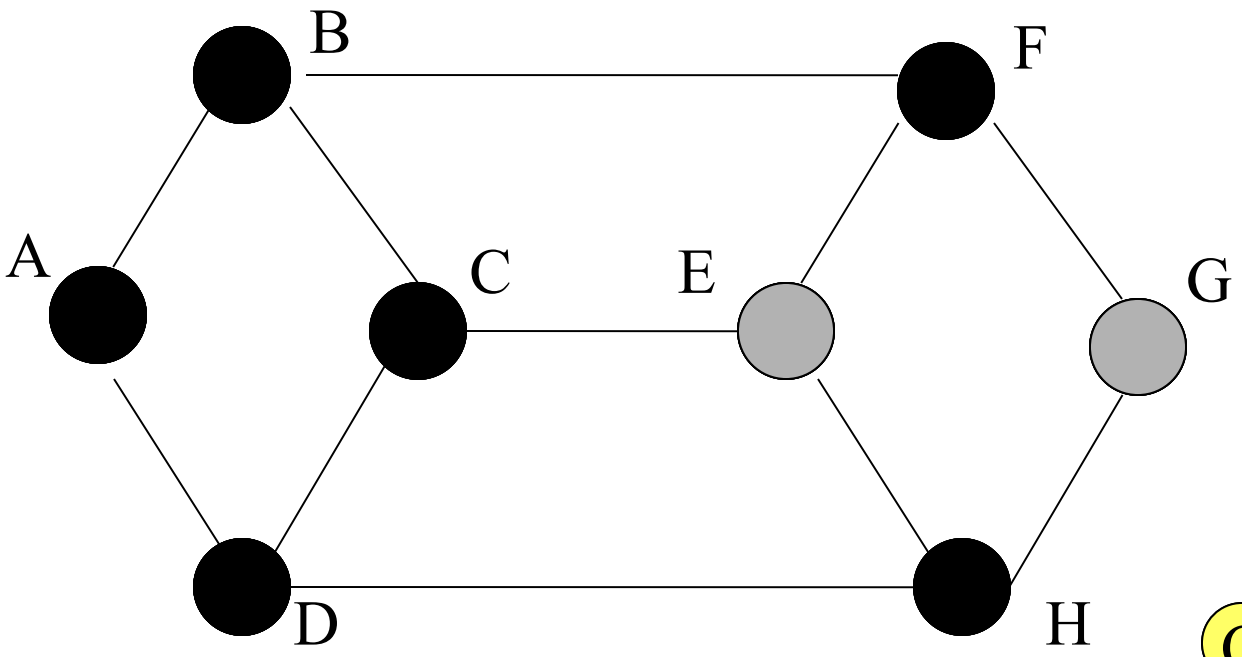


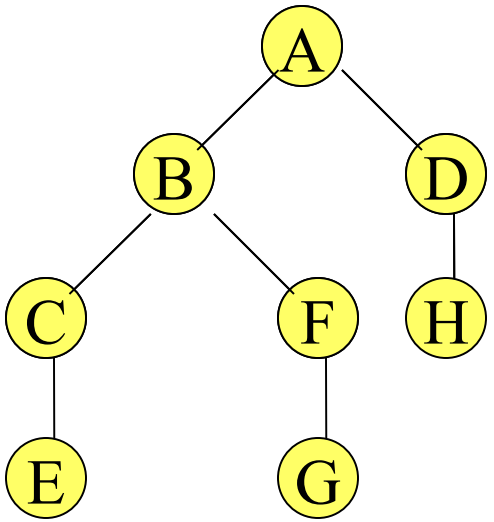




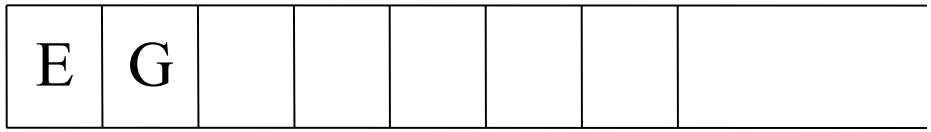
Q

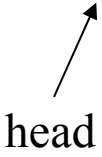


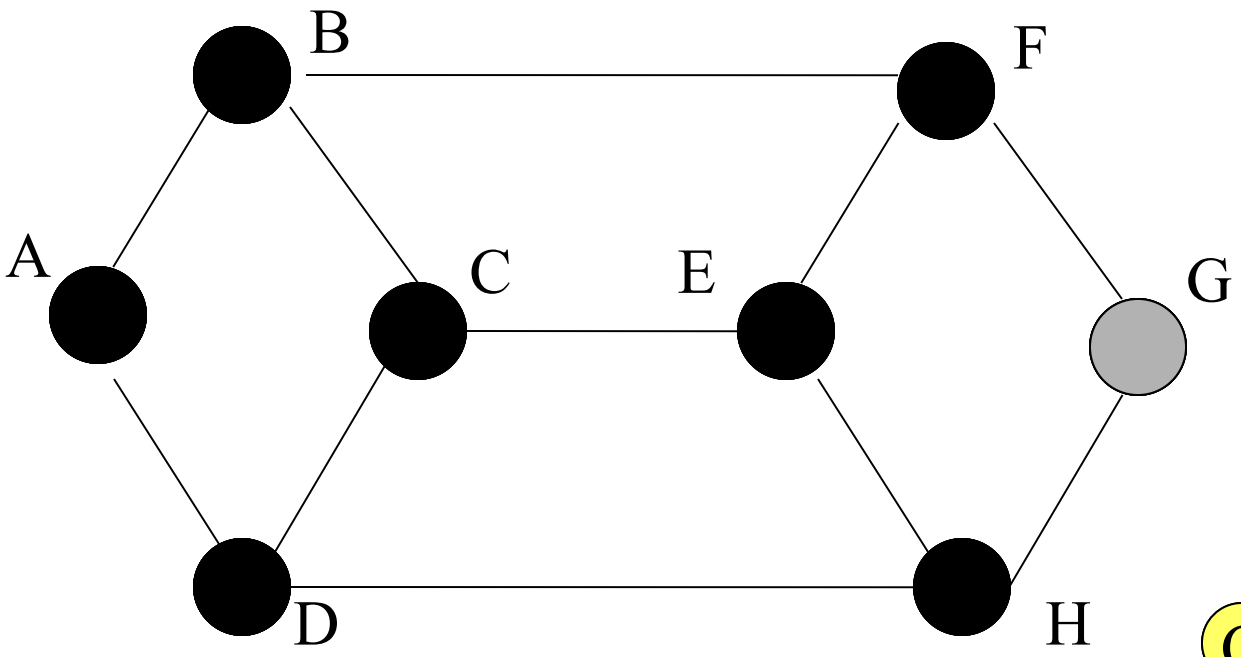


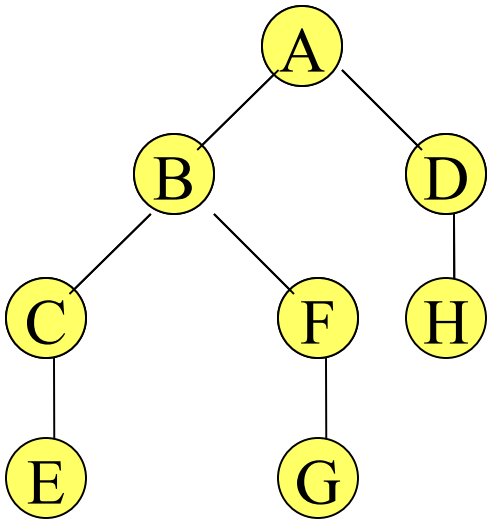


Q

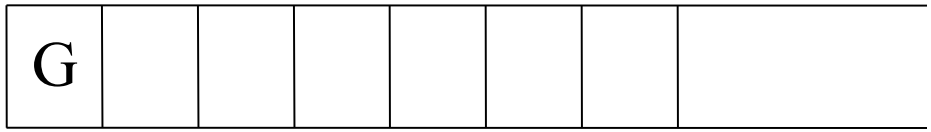


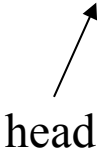


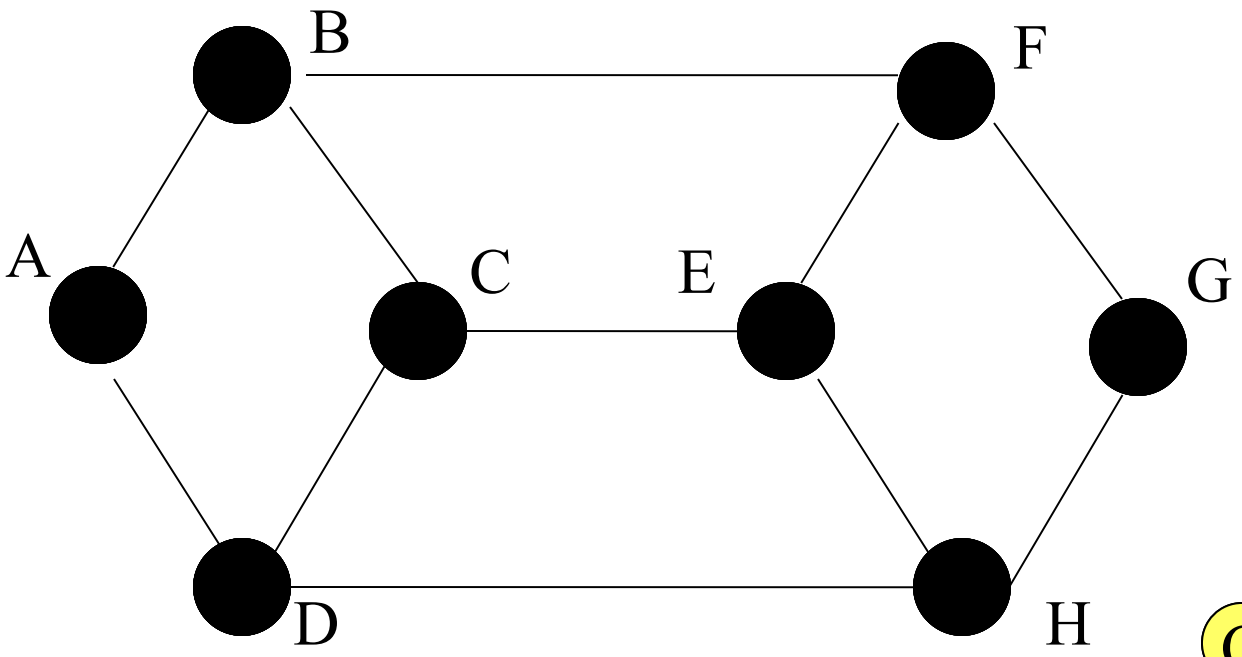


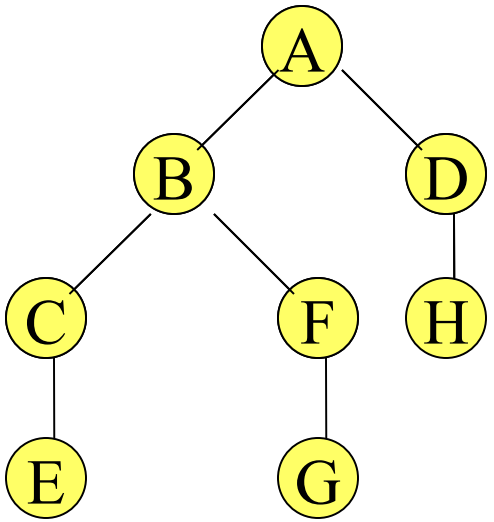


Q

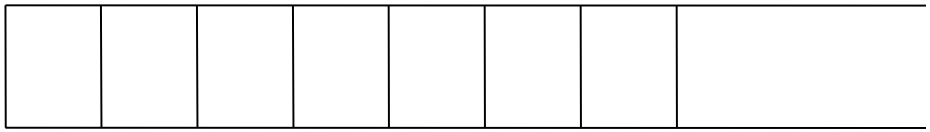








Q



head

BFS-VISITA (G, s)

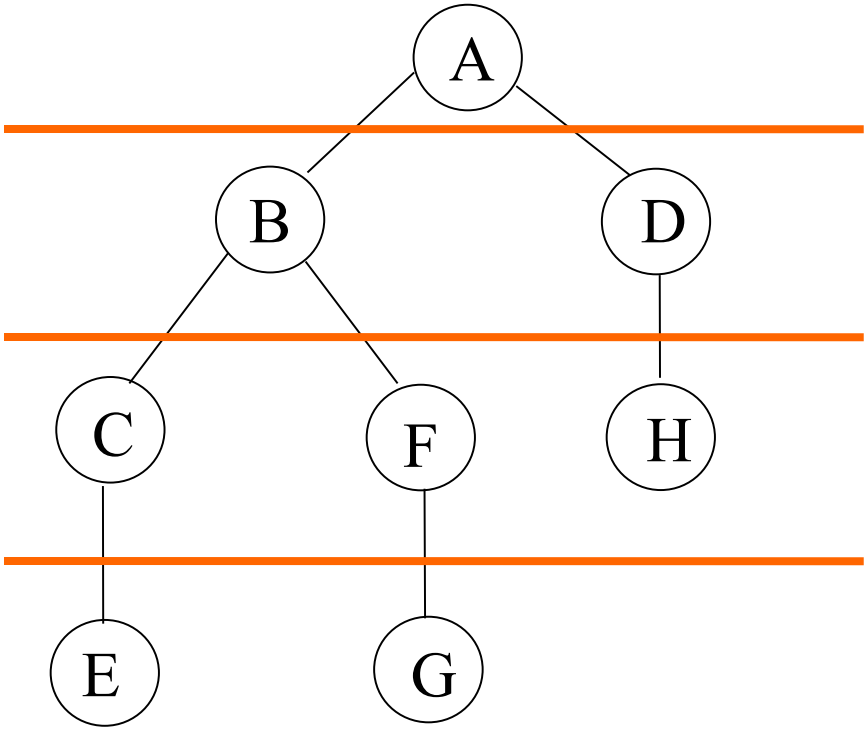
Q ← make_empty
color [s] ← gray
enqueue (Q, s)
while not_empty (Q) **do**
u ← head (Q)

if c'è v bianco ∈ ADJ [u]
then color [v] ← gray
π [v] ← u
enqueue (Q, v)
else color [u] ← black
dequeue (Q)

Poichè la head della coda non cambia finché ci sono adiacenti bianchi, l'algoritmo può essere modificato nel modo seguente:

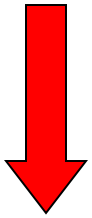
while c'è v ∈ ADJ [u]
non considerato **do**
if color [v] = white
then color [v] ← gray
π [v] ← u
enqueue (Q, v)
color [u] ← black
dequeue (Q)

N.B. Nella visita in ampiezza Breadth First Search l'albero viene costruito a livelli

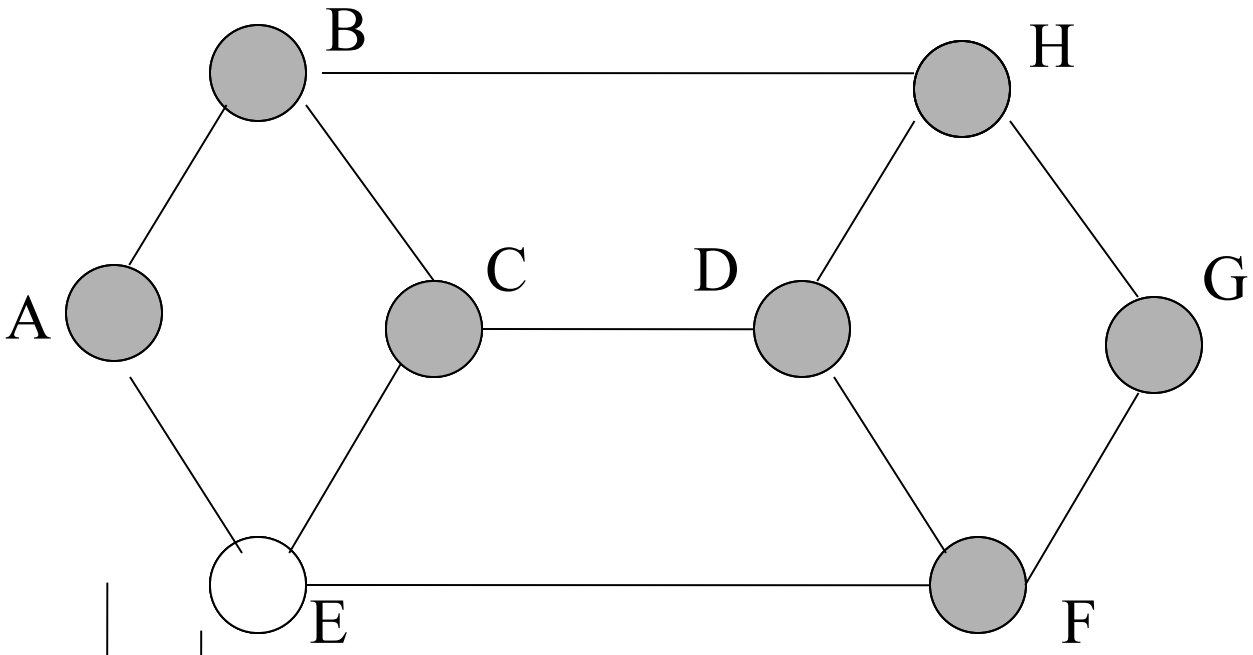


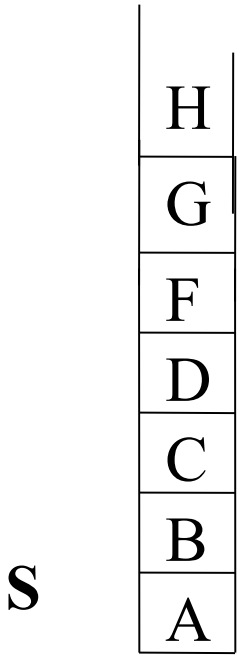
La struttura dati D è uno

STACK

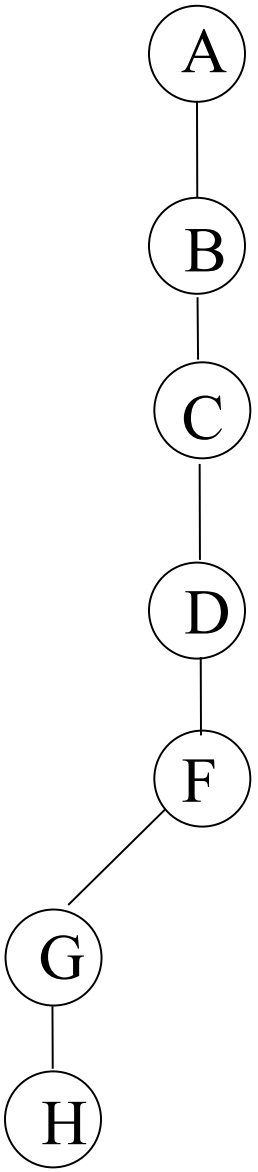


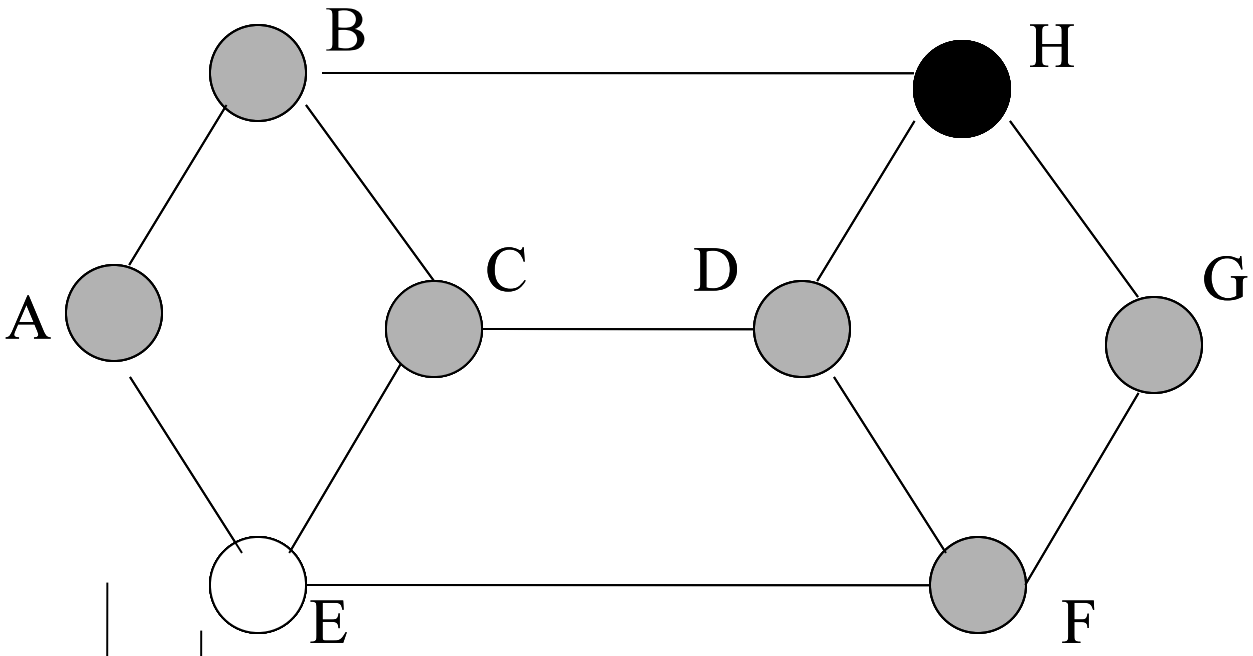
Visita in profondità o Depth-First-Search (DFS)



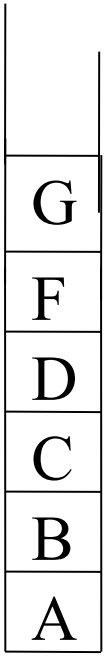


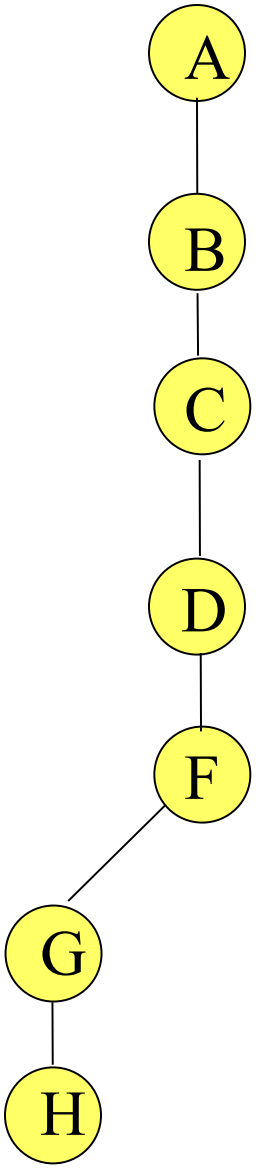
S

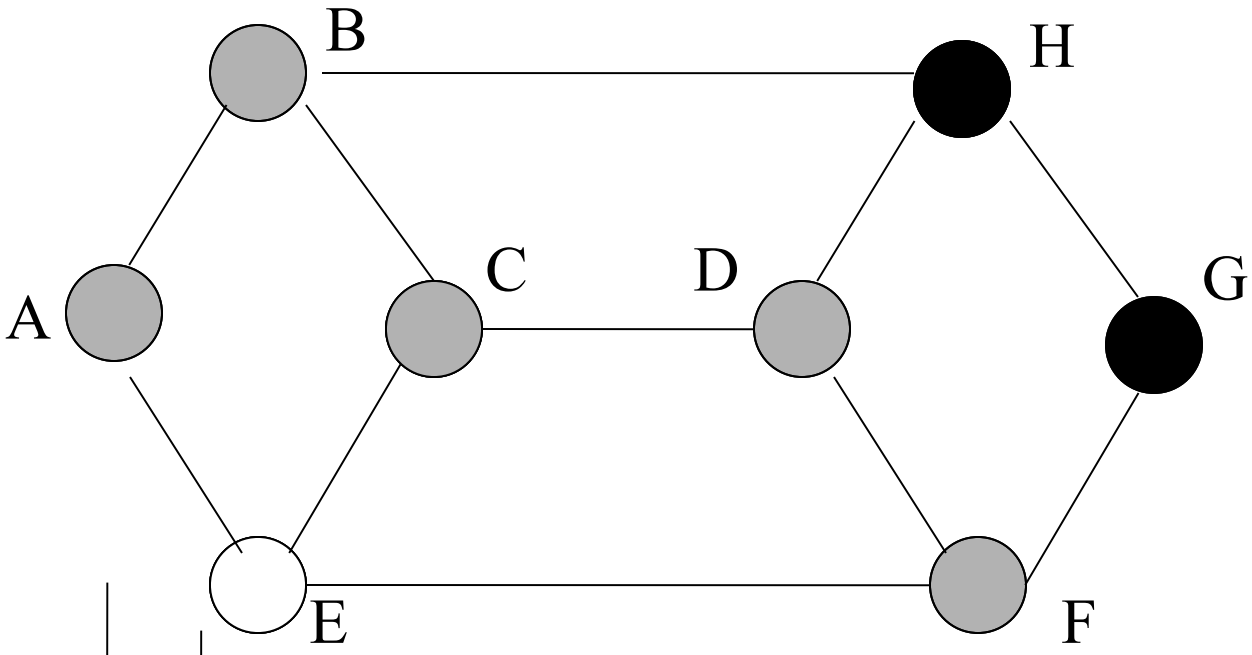


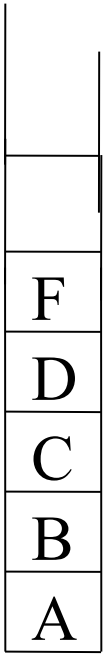


S

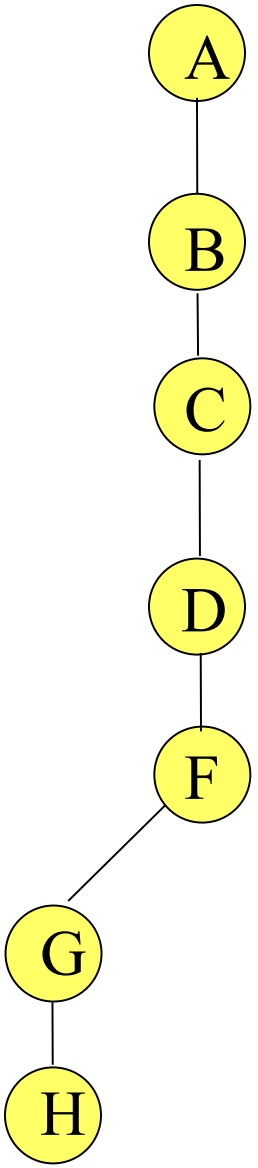


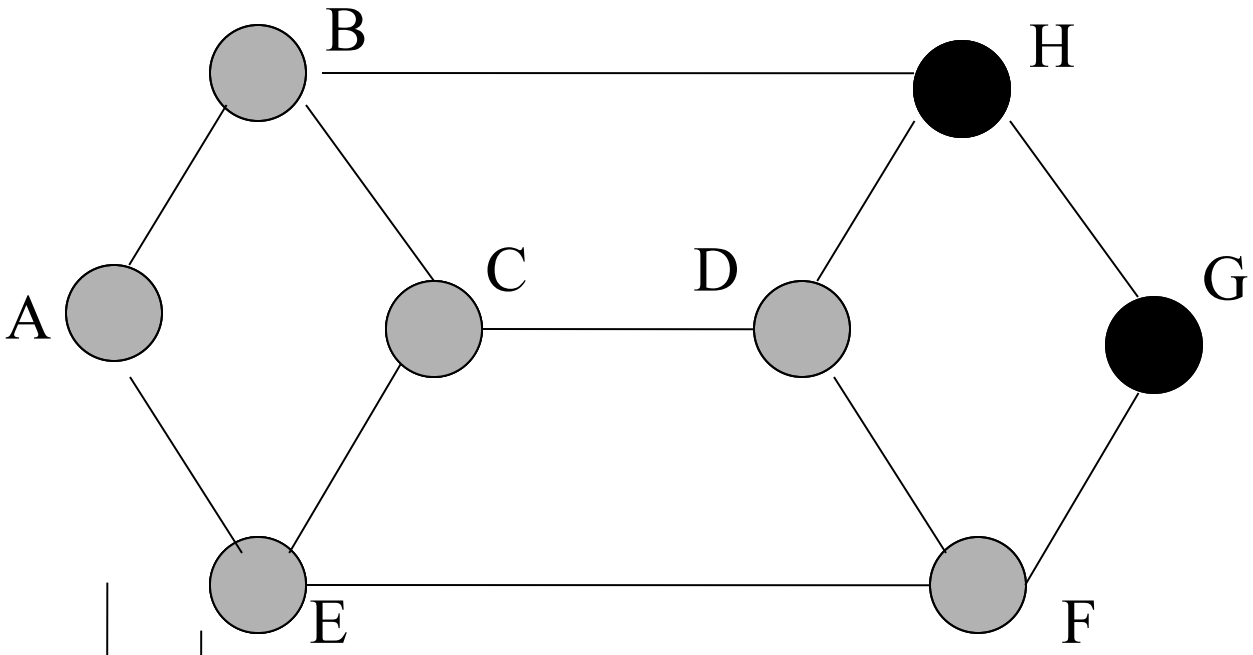


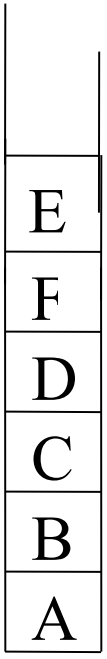




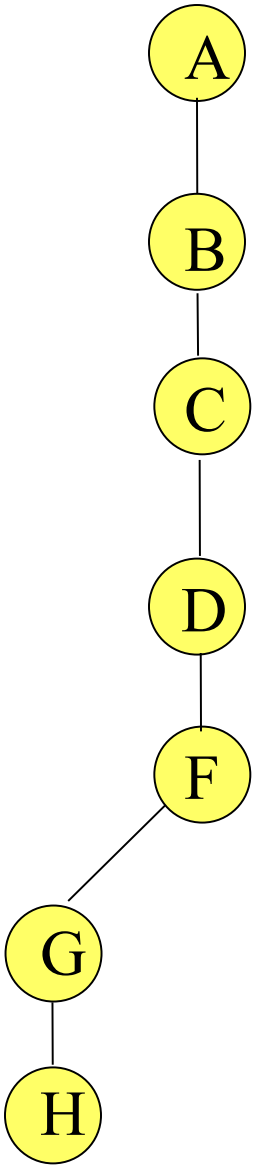
S

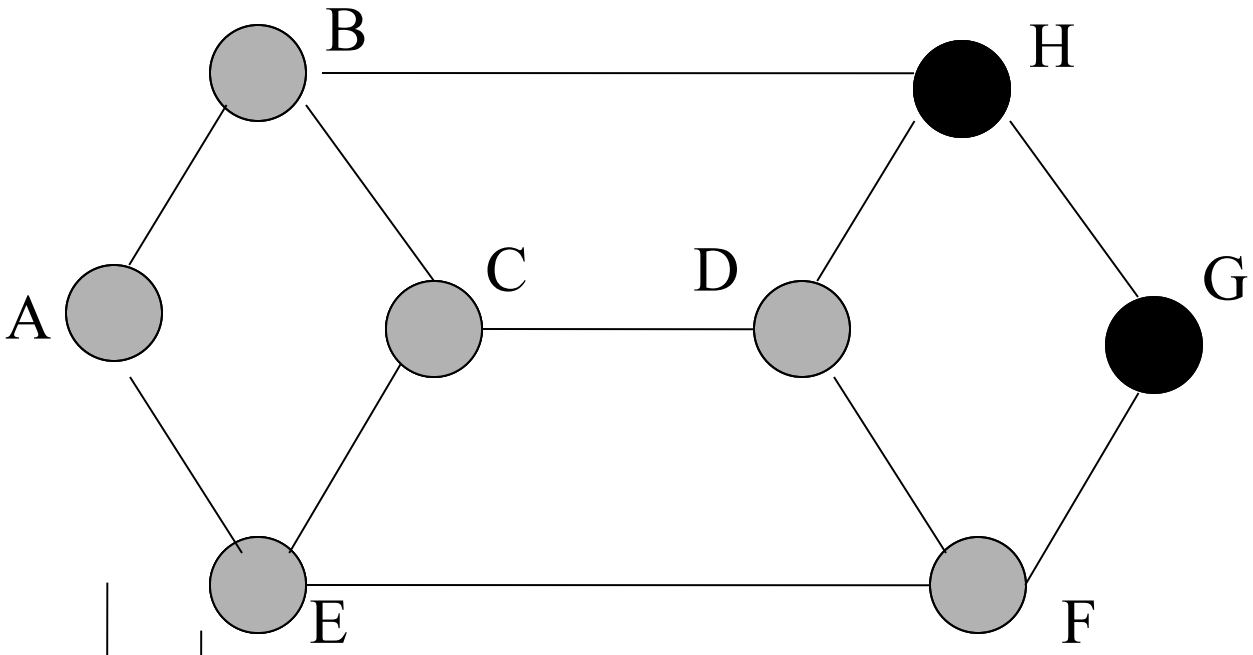


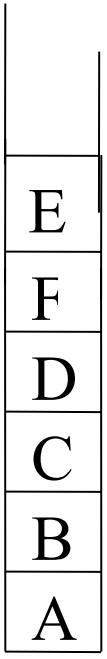




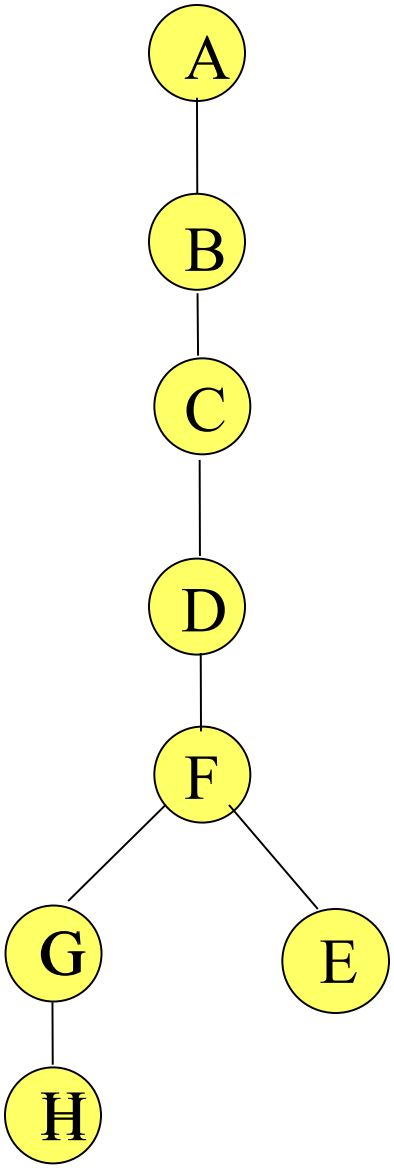
S

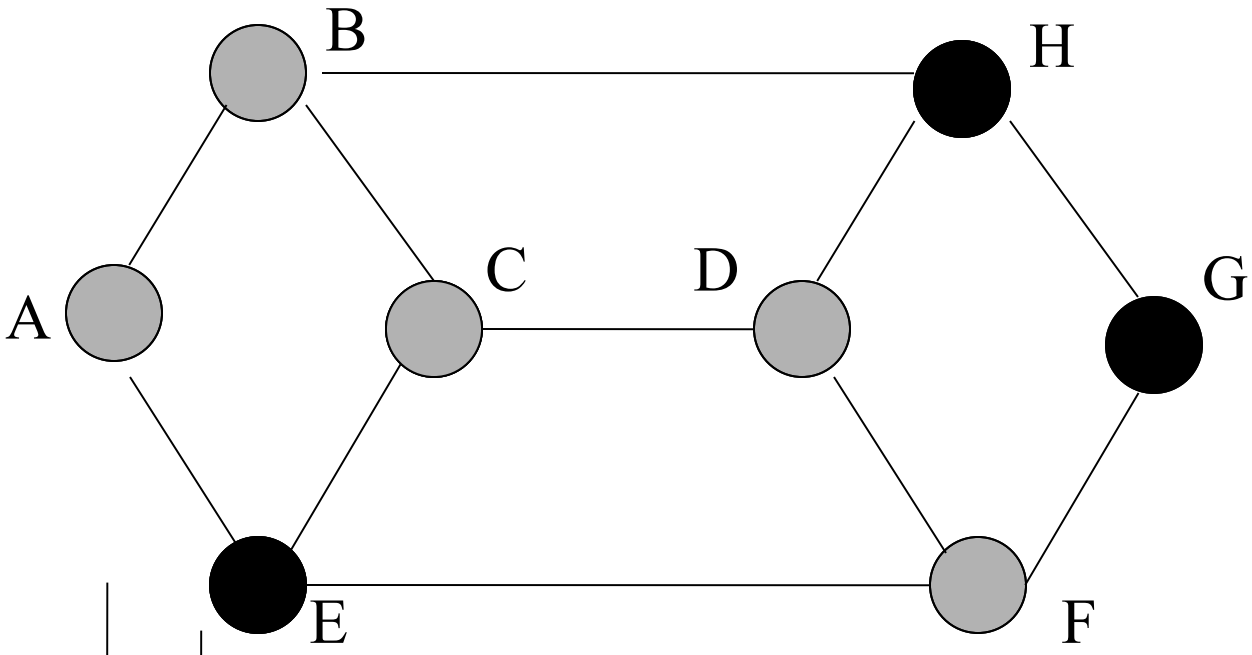


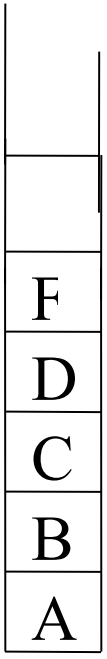




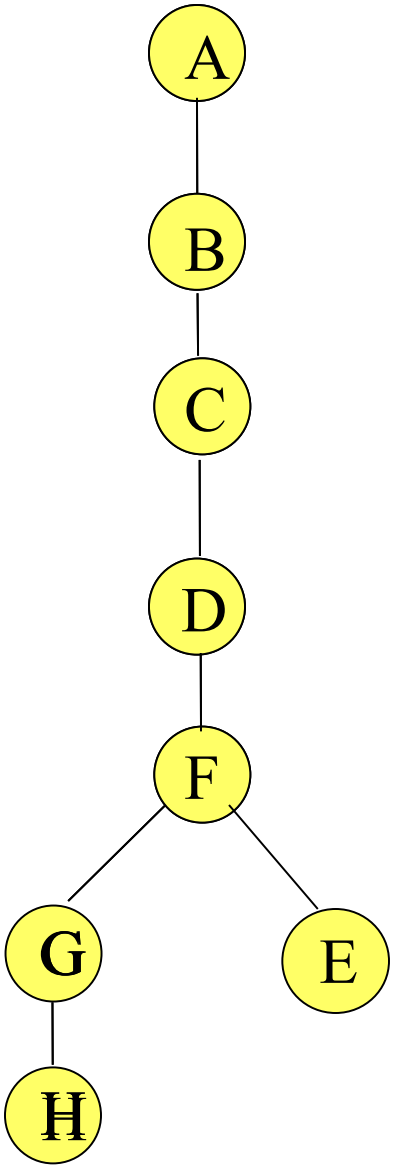
S

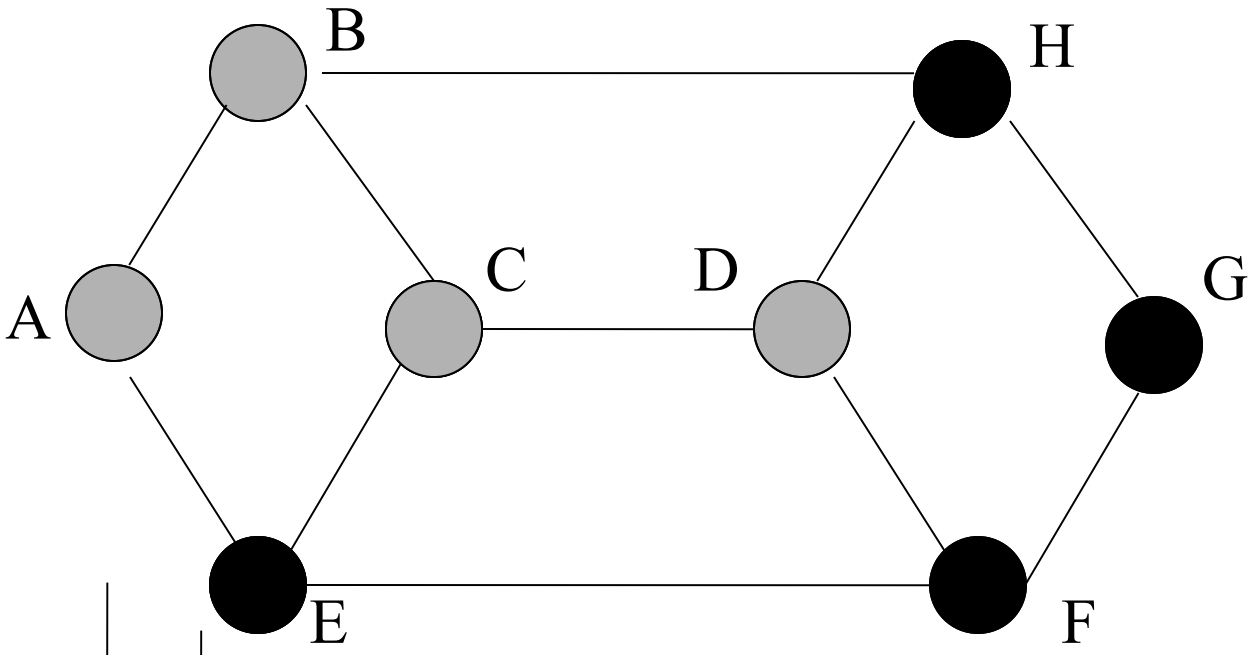


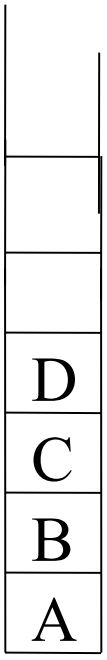




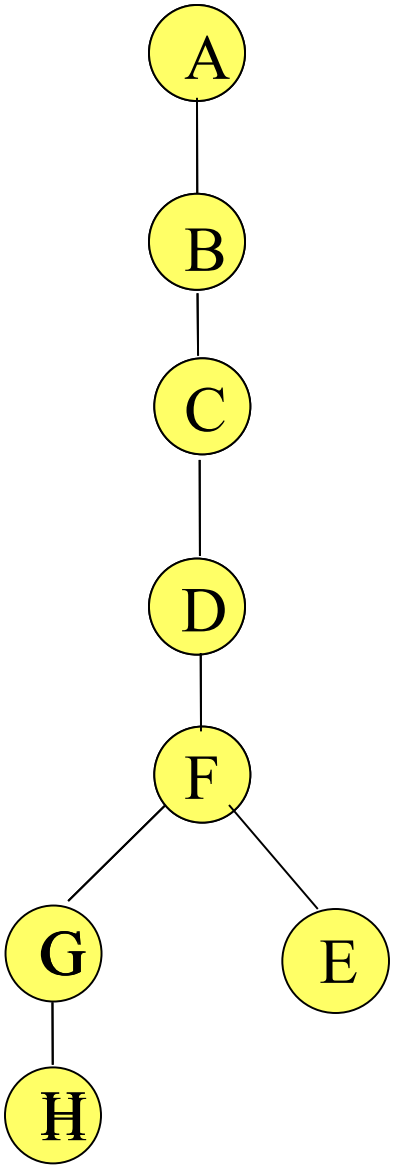
S

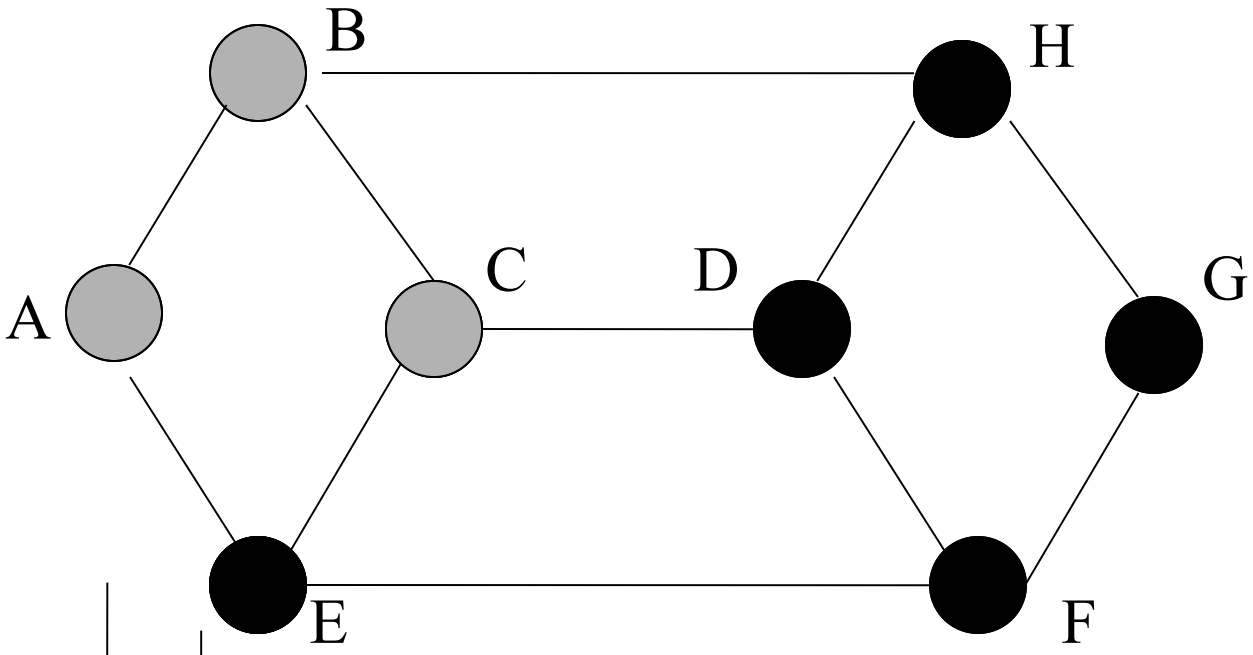


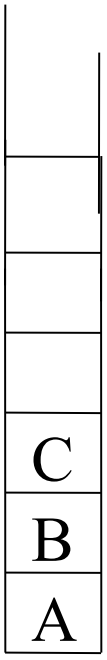




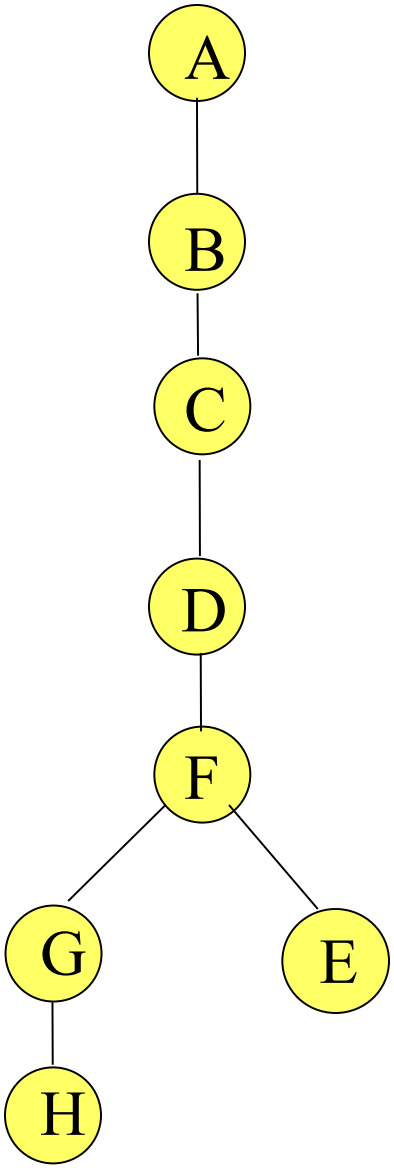
S

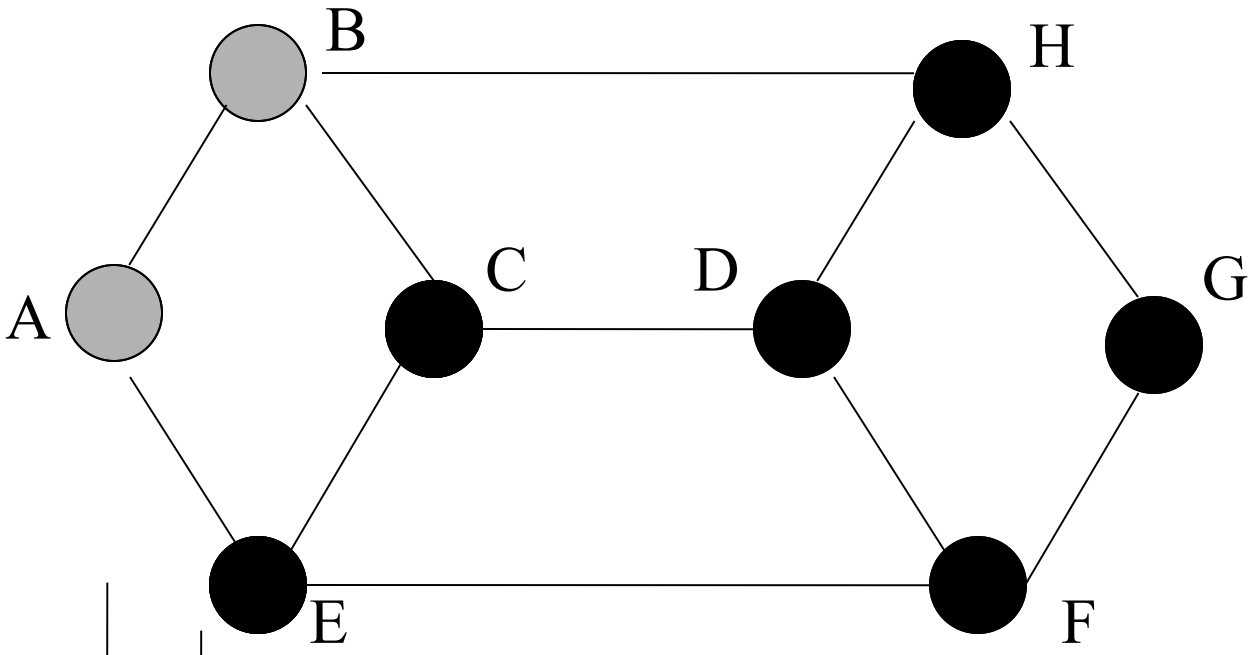


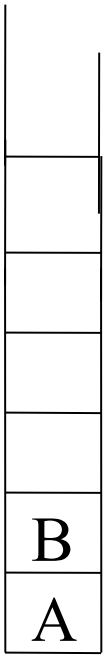




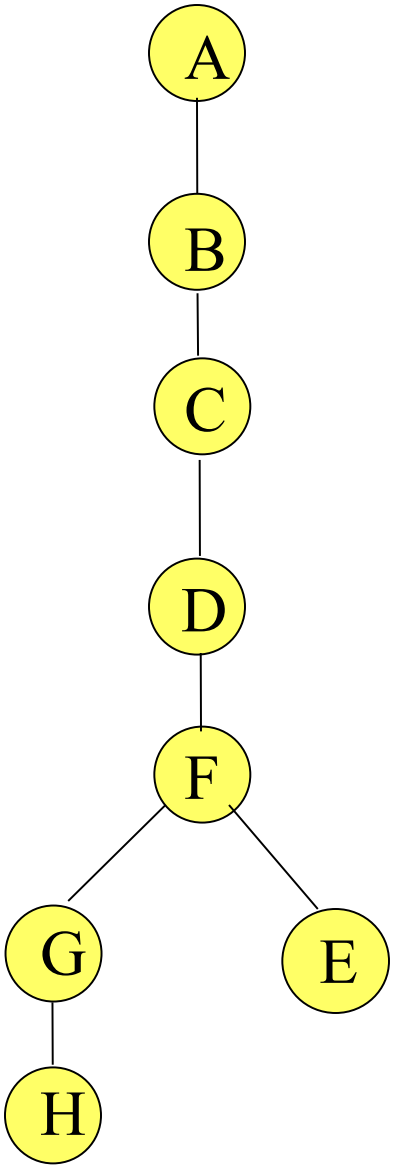
S

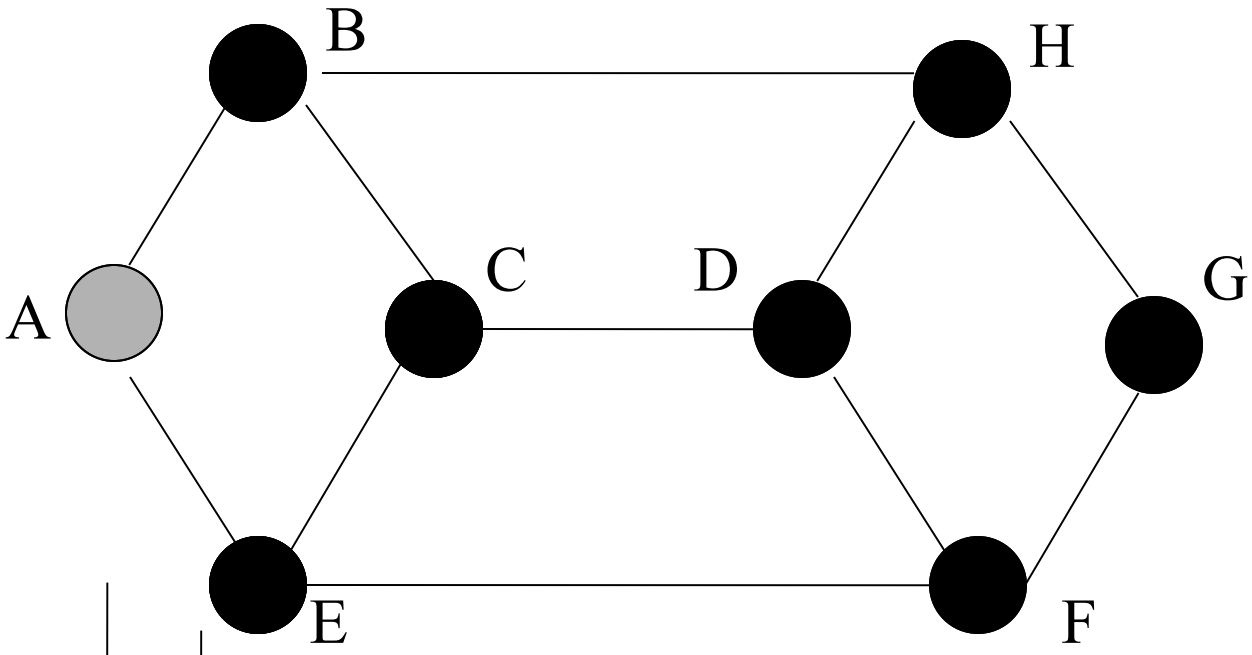


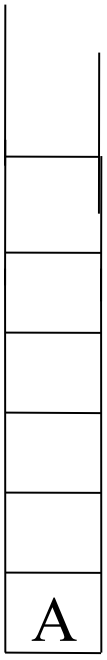




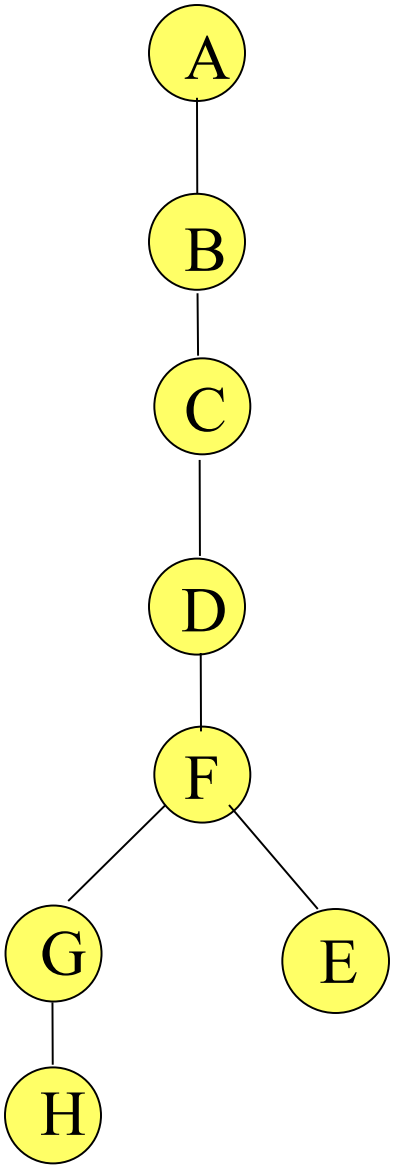
S

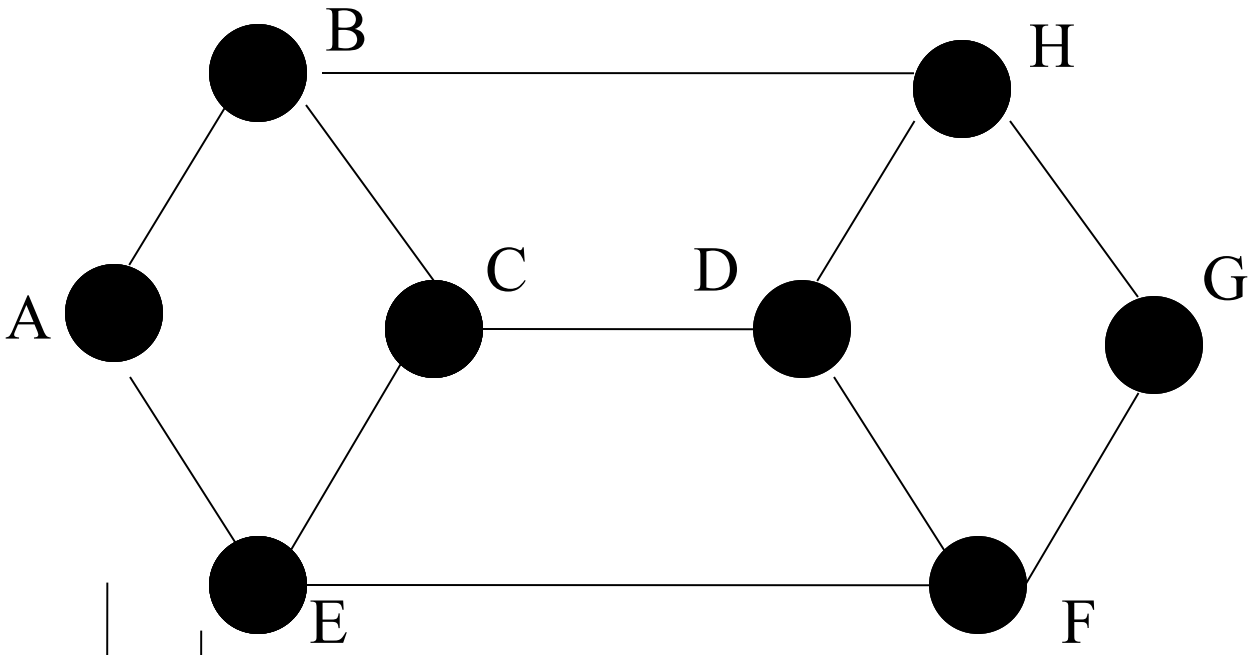


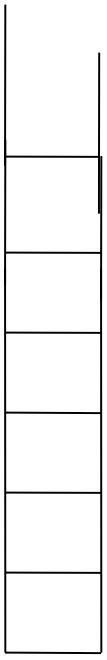




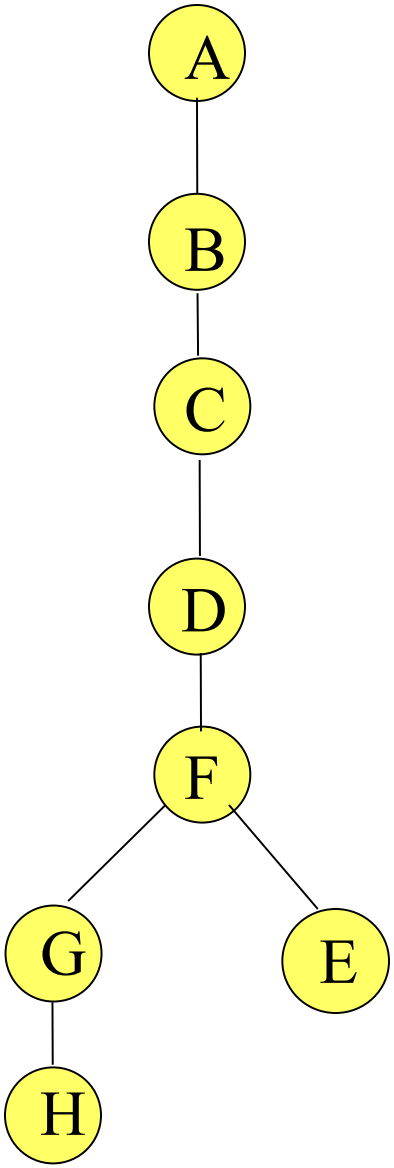
S







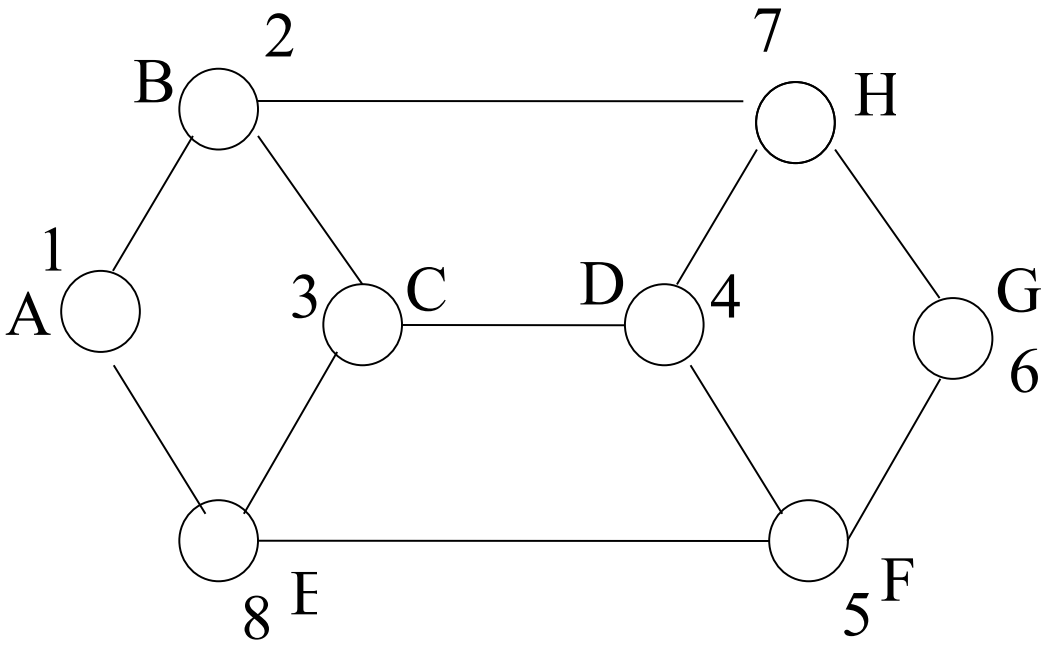
S

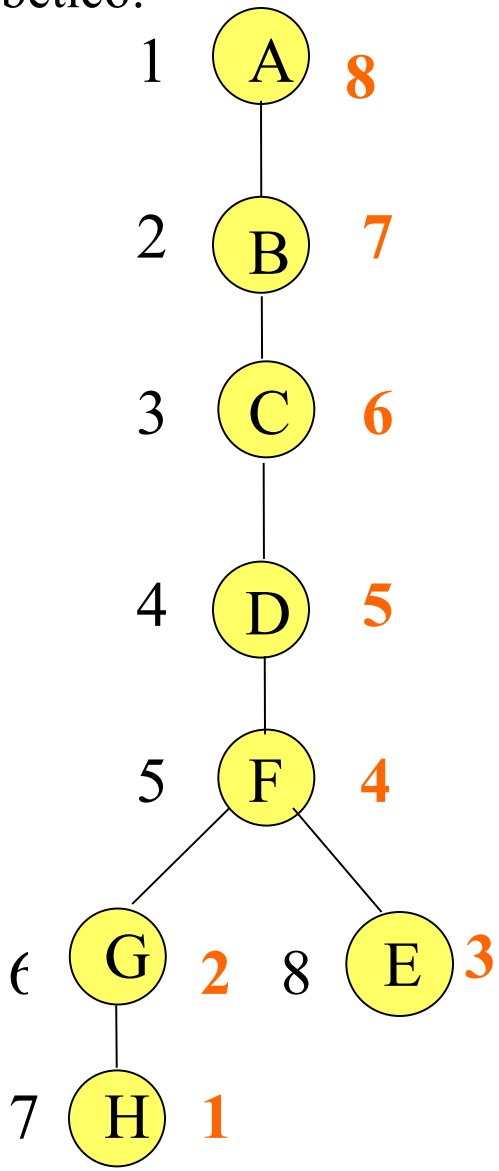


Vertici memorizzati nelle liste di adiacenza in ordine alfabetico:

CONSIDERIAMO L'ORDINE IN CUI I VERTICI DIVENTANO GRIGI: viene creato l'albero

E L'ORDINE IN CUI DIVENTANO NERI





L’albero viene creato dall’alto al basso: “in profondità”

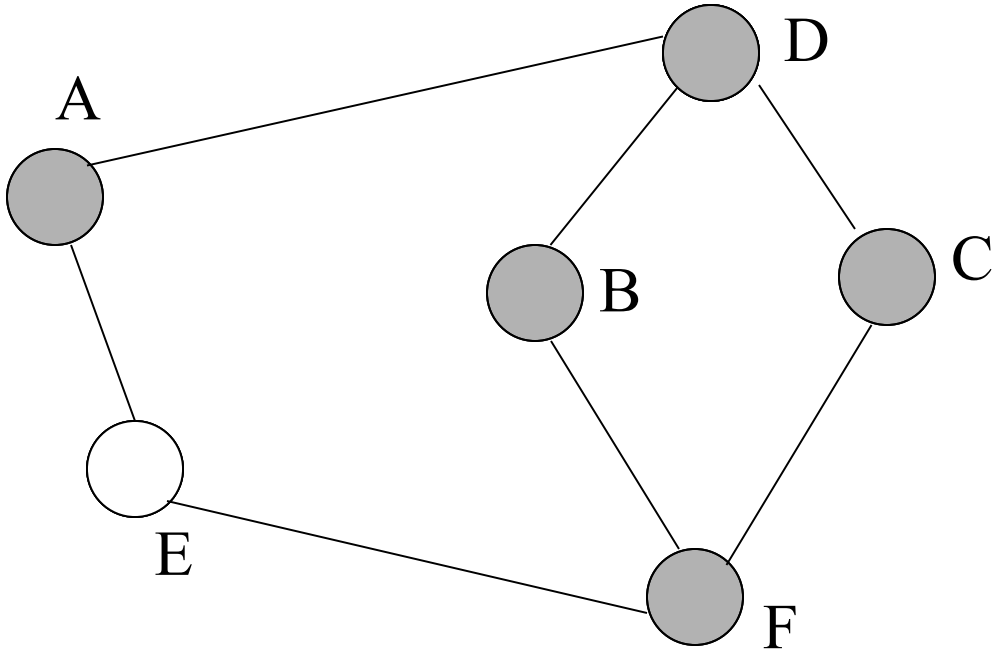
La visita prende il nome di “visita in profondità”:

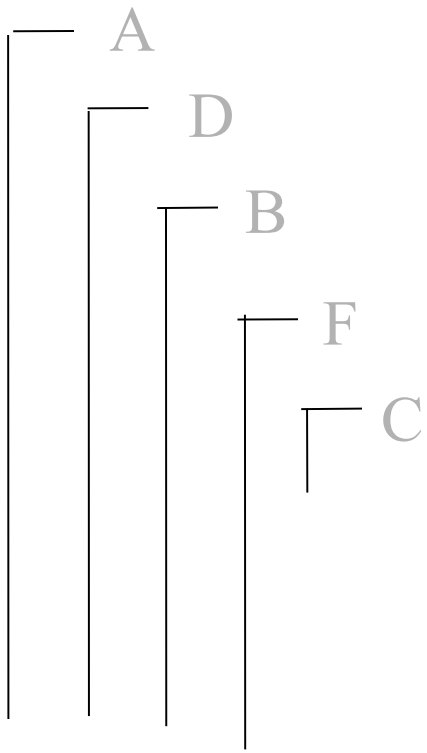
Depth First Search

DFS-VISITA (G, s)

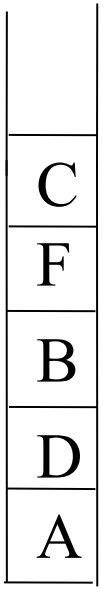
S ← make_empty_stack
color [s] ← gray
push (S, s)
while not_empty (S) **do**
while c'è v ∈ ADJ [top (S)]
non considerato **do**

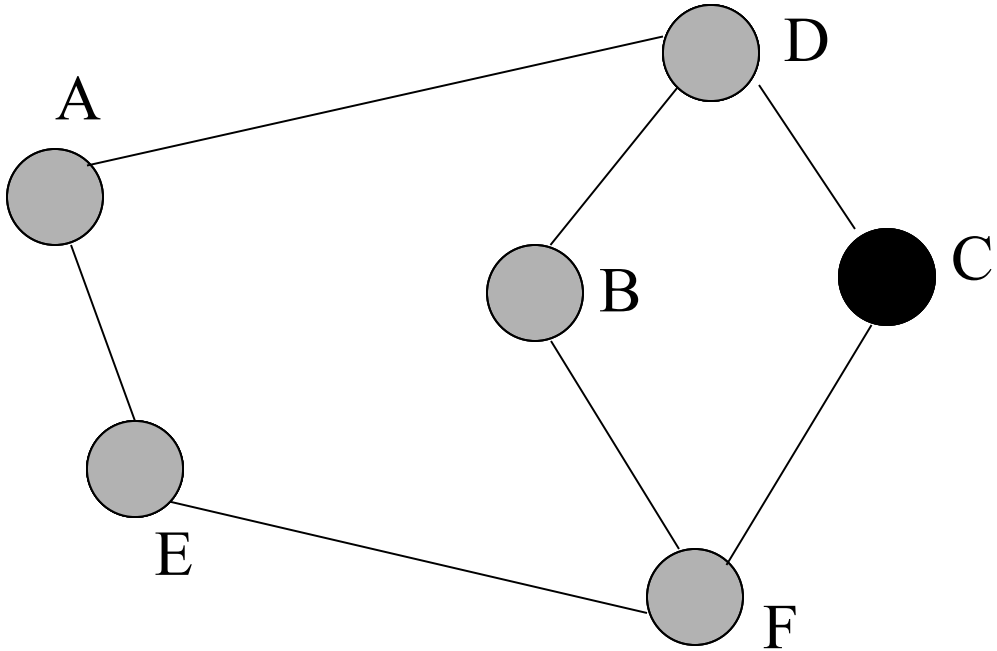
if color [v] = white
then color [v] ← gray
π [v] ← u
push (S, v)
color [u] ← black
pop (S)

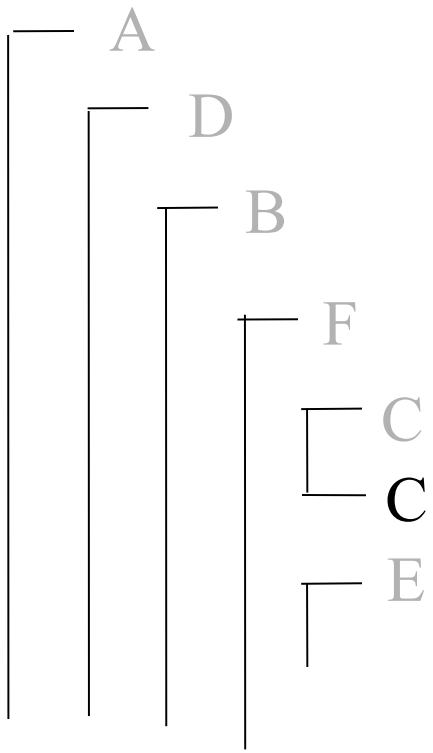




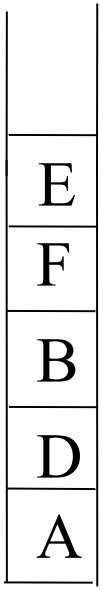
S

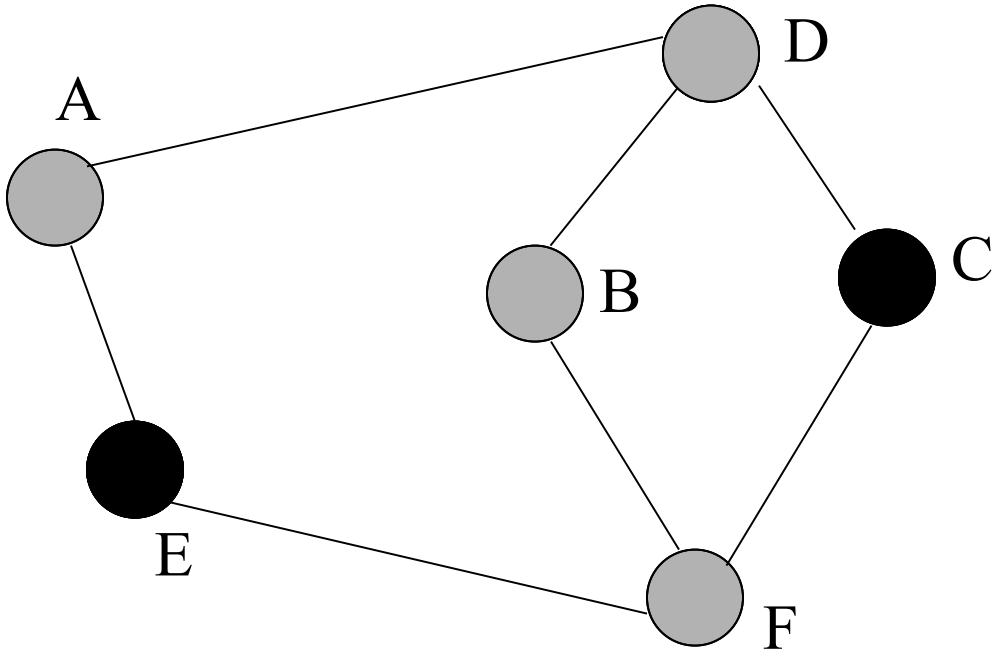


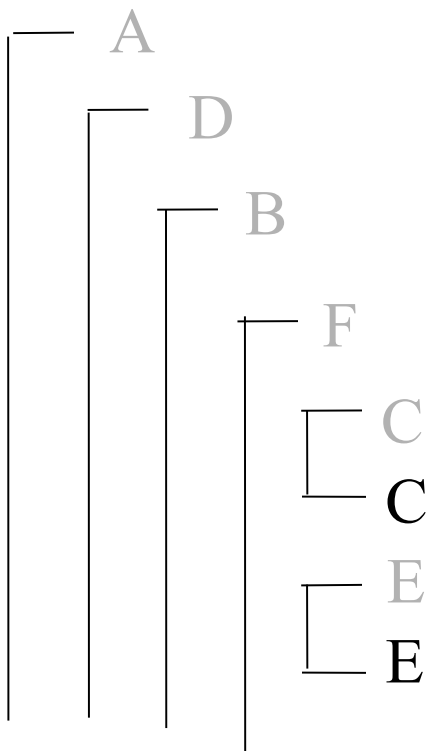


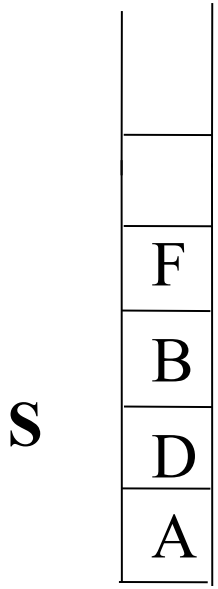


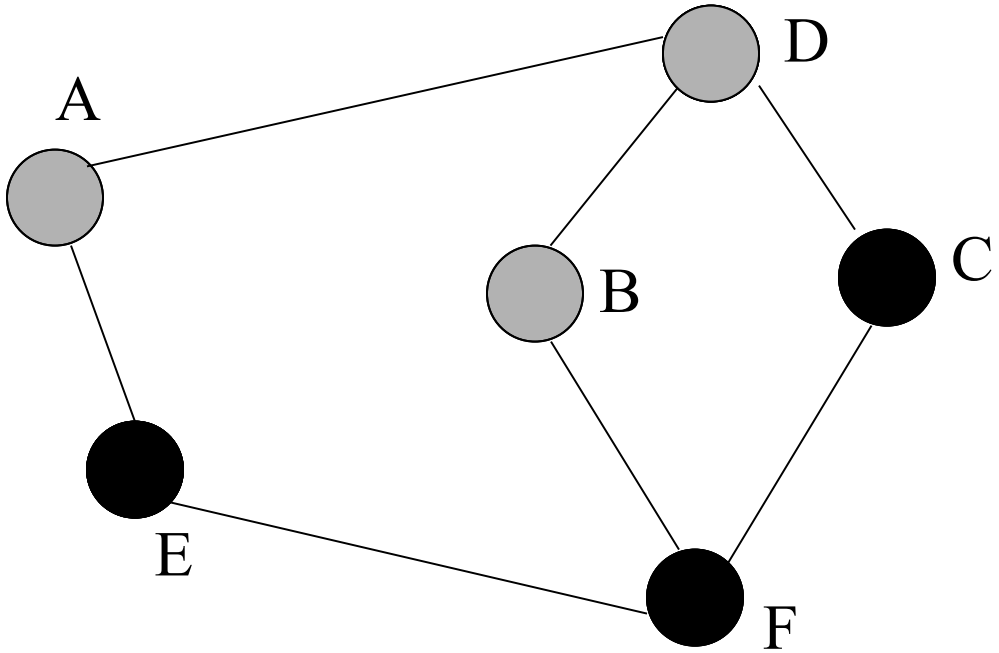
S

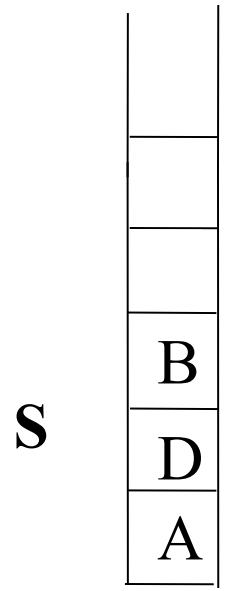


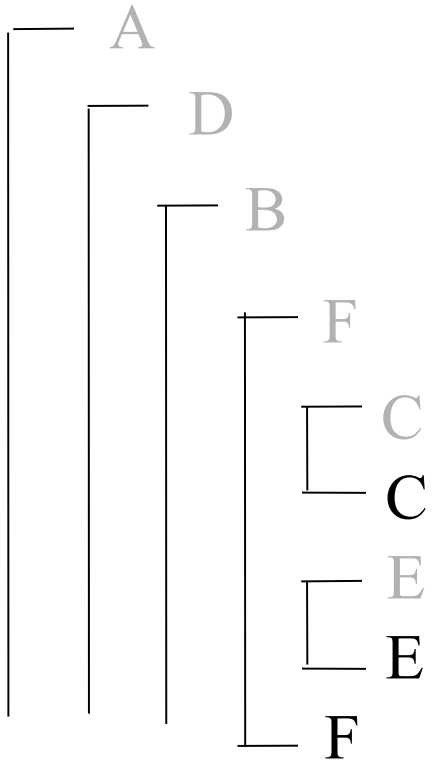

A
D
B
C
E
F

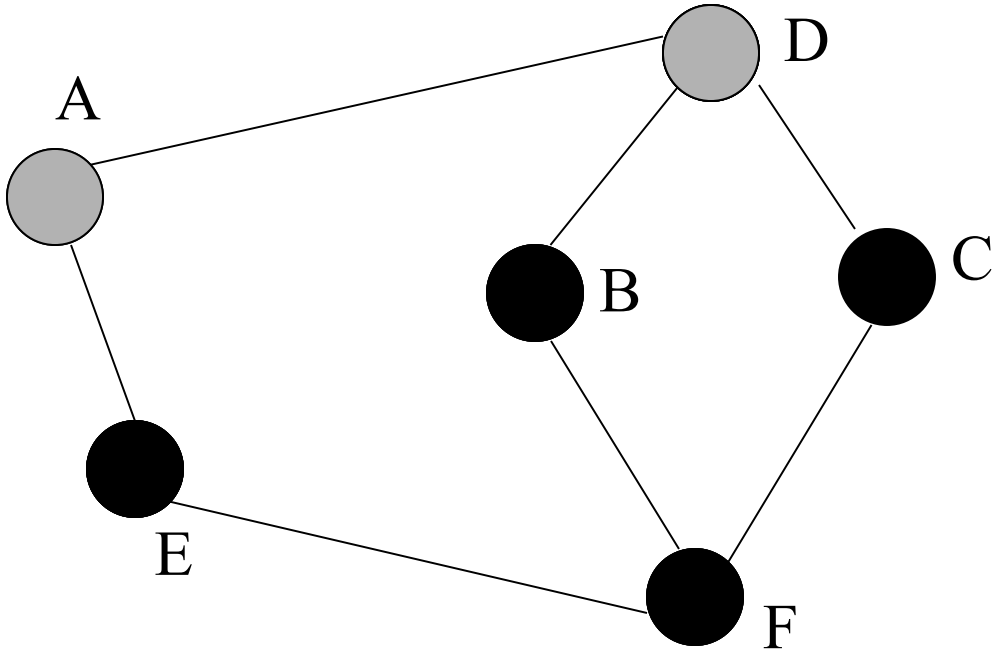

A
D
B
F
C
C
E
E

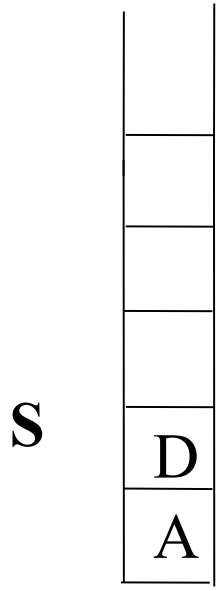

S
F
B
D
A

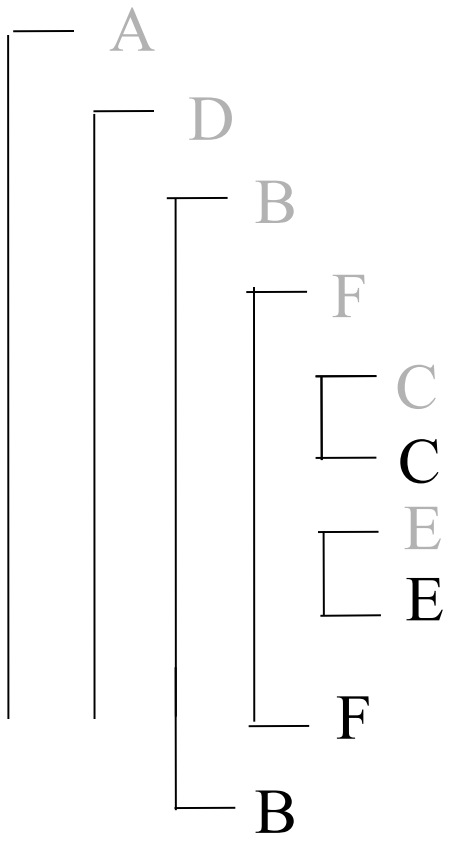

A
D
B
C
E
F

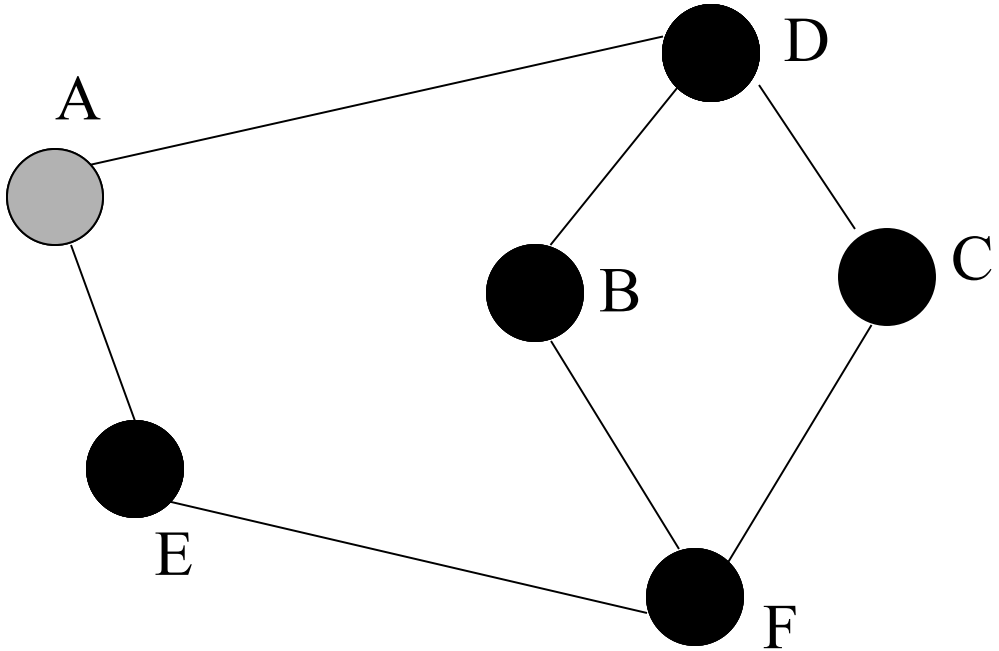

S
B
D
A

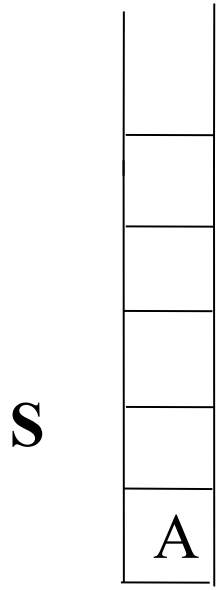

A
D
B
F
C
C
E
E
F

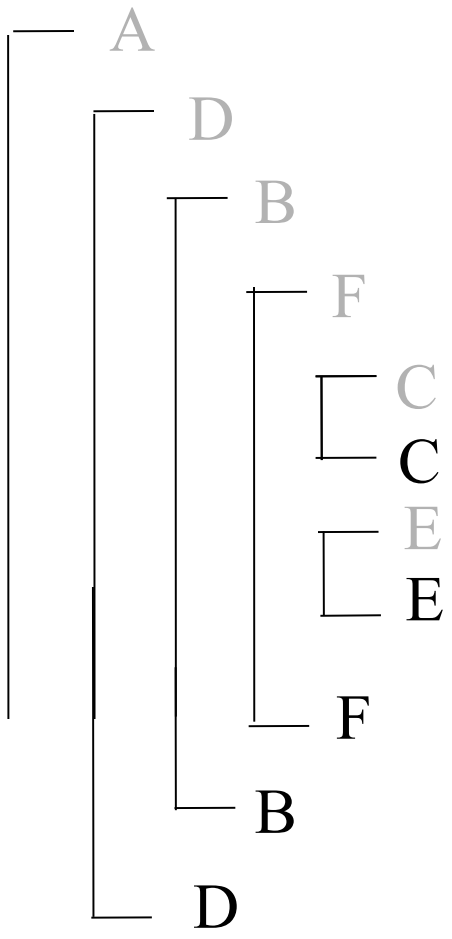

A
D
B
C
E
F

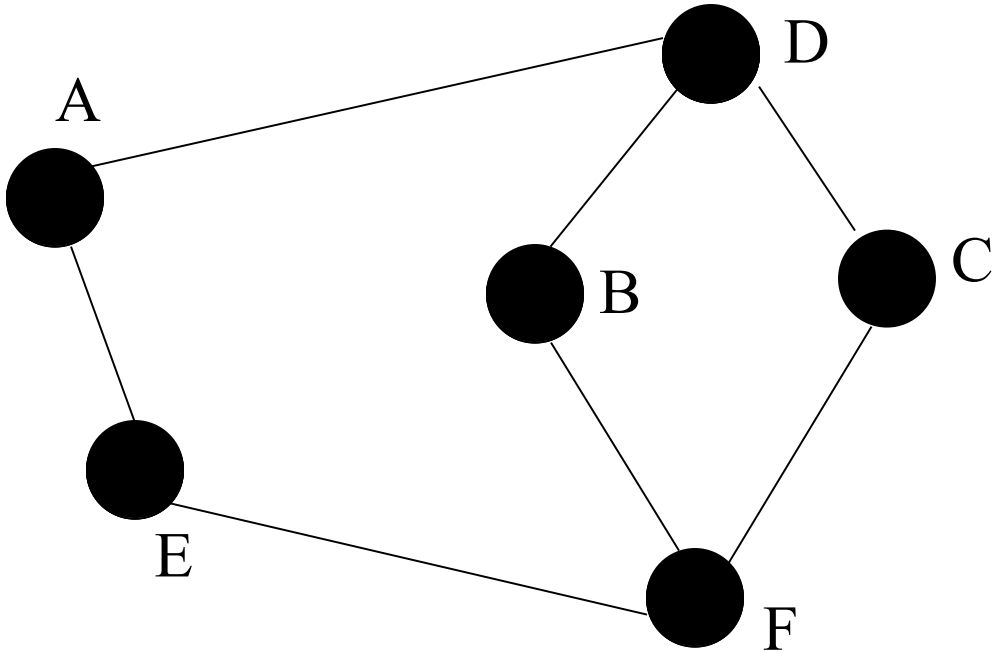

S
D
A

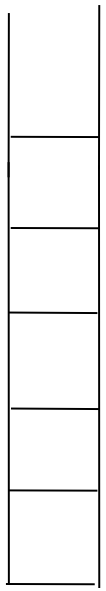

A
D
B
F
C
C
E
E
F
B

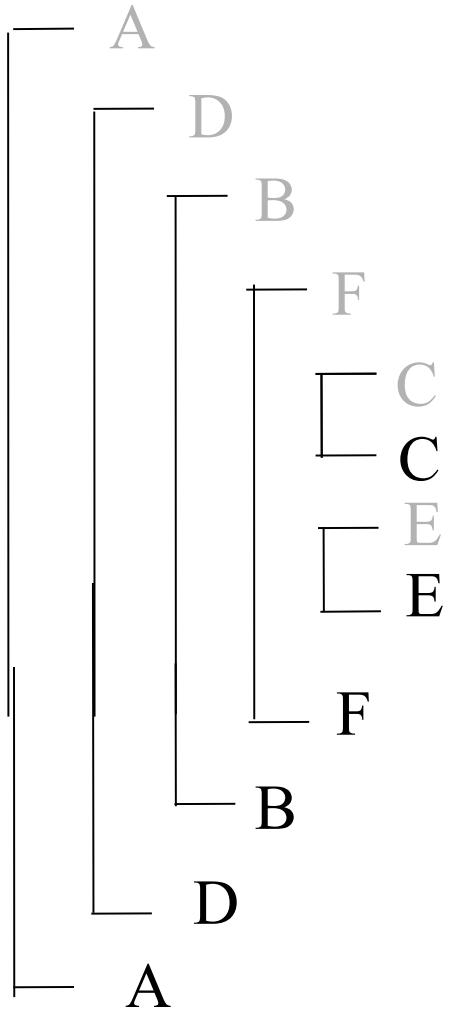

A
D
B
C
E
F


S
A


A
D
B
F
C
C
E
E
F
B
D


A
D
B
C
E
F


S


A
D
B
F
C
C
E
E
F
B
D
A

DFS-VISITA-ricorsiva (G, u)

color [u] ← gray

while c'è v ∈ ADJ [u] non considerato **do**
if color [v] = white
then π [v] ← u
DFS-VISITA-ricorsiva (G, v)

color [u] ← black

Algoritmi Greedy

Idea: "per trovare una soluzione ***globalmente ottima***, scegli ripetutamente soluzioni ***ottime localmente***"

Cammini minimi da un vertice sorgente in un grafo orientato e pesato

Definiamo “distanza” di un vertice u da un vertice v: $\delta(v, u)$, in un grafo orientato e pesato, il peso di un cammino di peso minimo tra tutti i cammini da v a u.

$$\delta(v, u) = \min \{W(p) / p \text{ cammino da } v \text{ a } u\}$$

dove W(p) e’ la somma dei pesi degli archi che formano il cammino.

N.B. $\delta(v, u)$ e’ ben definita solo se nessun cammino tra v e u contiene un ciclo di peso negativo

Input: $G = \langle V, E \rangle$ grafo orientato e pesato,
$w: E \rightarrow R$ funzione peso, $s \in V$, G senza cicli di peso negativo raggiungibili da s

Output: $\forall v \in V[G]$, l'attributo d[v] indica la "distanza" di v dal vertice sorgente: $d[v] = \delta(s, v)$

Per costruire un algoritmo greedy, dobbiamo individuare l' insieme di oggetti e definire per essi un grado di *appetibilita'*.

Si puo' pensare ai vertici come agli oggetti tra i quali scegliere; mantenendo per ogni vertice v, nell'attributo d[v] una stima (maggiore o uguale) della distanza di v da s.
Un vertice risultera' tanto piu' appetibile quanto minore e' la stima della sua distanza dal sorgente stesso.

L’idea dell’algoritmo e’ la seguente:

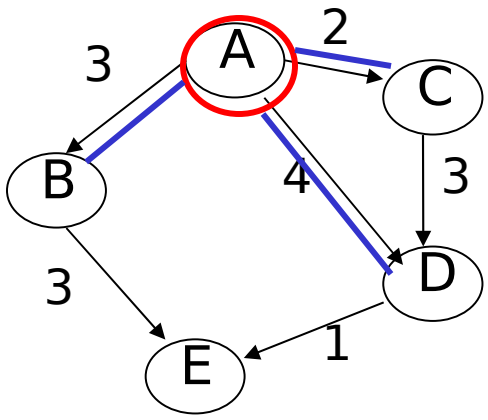
Inizialmente si stima la distanza di ogni vertice uguale a ∞, tranne che per il sorgente stesso, di cui si conosce la distanza vera: $\delta(s, s) = 0$ (si definisce pertanto $d[s] = 0$)

Si costruisce poi un albero, di radice s, in cui viene inserito un vertice per volta: il piu’ appetibile.

L’albero e’ memorizzato implicitamente come l’insieme dei suoi archi $< \pi[v], v>$

Quando un vertice u e’ inserito nell’albero, si aggiornano le stime delle distanze dei vertici v ad esso adiacenti, in quanto potrebbe esitere un cammino da s a v, attraverso il vertice u, meno pesante del cammino da s a v considerato fino a quel momento.

Prima di scrivere l'algoritmo vediamo un esempio:

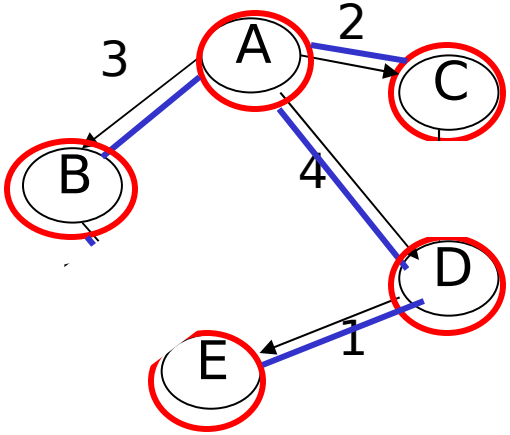


Se il vertice sorgente è A, si ha inizialmente:

$d[A] = 0,\ d[B] = d[C] = d[D] = d[E] = \infty$

La strategia greedy sceglie percio' il vertice A.

Da A sono raggiungibili con un arco i vertici B, C e D.
Si possono modificare pertanto le stime delle loro distanze da A.
$d[B] = 3,\ d[C] = 2,\ d[D] = 4$, mentre resta $d[E] = \infty$



Scegliendo C si scopre il cammino A - C - D, ma d[D] non viene modificata in quanto il suo peso (5) e' maggiore del peso del cammino A - D (di peso 4).

La scelta e' ora tra i vertici B, D ed E, con d[B] = 3, d[D] = 4, d[E] = ∞, e la stategia greedy impone di scegliere B.

Si scopre cosi' il cammino A - B - E e diventa d[E] = 6.

La scelta di D permette infine di ottenere d[E] = 5, ed E sara' l'ultimo vertice scelto.

Dijkstra (G, w, s)

$S \leftarrow \Phi$

for ogni $v \in V$ **do**

$d[v] \leftarrow \infty$

$\pi[v] \leftarrow$ **nil**

$d[s] \leftarrow 0$

while $S \neq V[G]$ **do**

$u \leftarrow$ "vertice con d minore tra quelli non ancora scelti"

$S \leftarrow S \cup \{u\}$

for ogni $v \in ADJ[u]$ **do**

if $v \notin S$ **and** $d[v] > d[u] + W(u, v)$

then $d[v] \leftarrow d[u] + W(u, v)$

$\pi[v] \leftarrow u$

▲ valuta le *appetibilita'* iniziali dei vertici

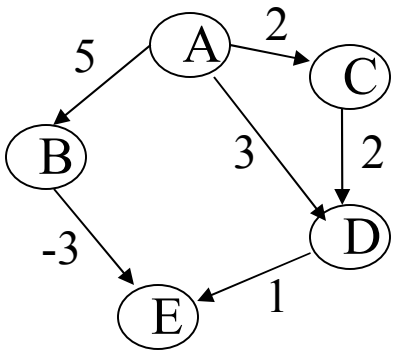
▲ ci sono elementi da scegliere

▲ scegli il vertice piu' *appetibile*

▲ "aggiorna le *appetibilita'* dei vertici"

Non sempre la strategia funziona!

Ad esempio:



Se il vertice sorgente è A, dopo le opportune modifiche alle stime delle distanze, vengono scelti, nell'ordine:

A (d[A] = 0, d[B] = d[C] = d[D] = d[E] = ∞)
C (d[B] = 5, d[C] = 2, d[D] = 3, d[E] = ∞)
D (d[B] = 5, d[D] = 3, d[E] = ∞)
E (d[B] = 5, d[E] = 4)
B (d[B] = 5)

Ma δ (A, E) = 2, mentre d[E] = 4

Nell'esempio considerato e' facile capire che il non corretto comportamento e' dovuto alla presenza dell'arco <B, E>, il cui peso e' negativo.

Allora, se la strategia greedy funziona, essa funziona solo in assenza di archi di peso negativo.