Intelligenza Artificiale

Anno Accademico 2022 - 2023

Strutture Dati in Python:
Grafi

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Sommario

- Definizione
- Rappresentazione dei Grafi in Python
- Visite di Grafi
  - Breadth-first
  - Depth-first

Grafi
Introduzione

Da un punto di vista matematico, un *grafo orientato* (o *diretto*) G consiste delle due componenti seguenti:

- Un insieme V di *nodi*.
- Una relazione binaria E su V. Chiamiamo E l'insieme degli *archi orientati* del grafo. Gli archi orientati sono quindi coppie di nodi.

A volte può essere utile collegare dei nodi per mezzo di linee che non abbiano una direzione, dette *archi non orientati* o *spigoli*. Un grafo con archi non orientati, cioè un grafo con una relazione di arco simmetrica, è detto *grafo non orientato*.

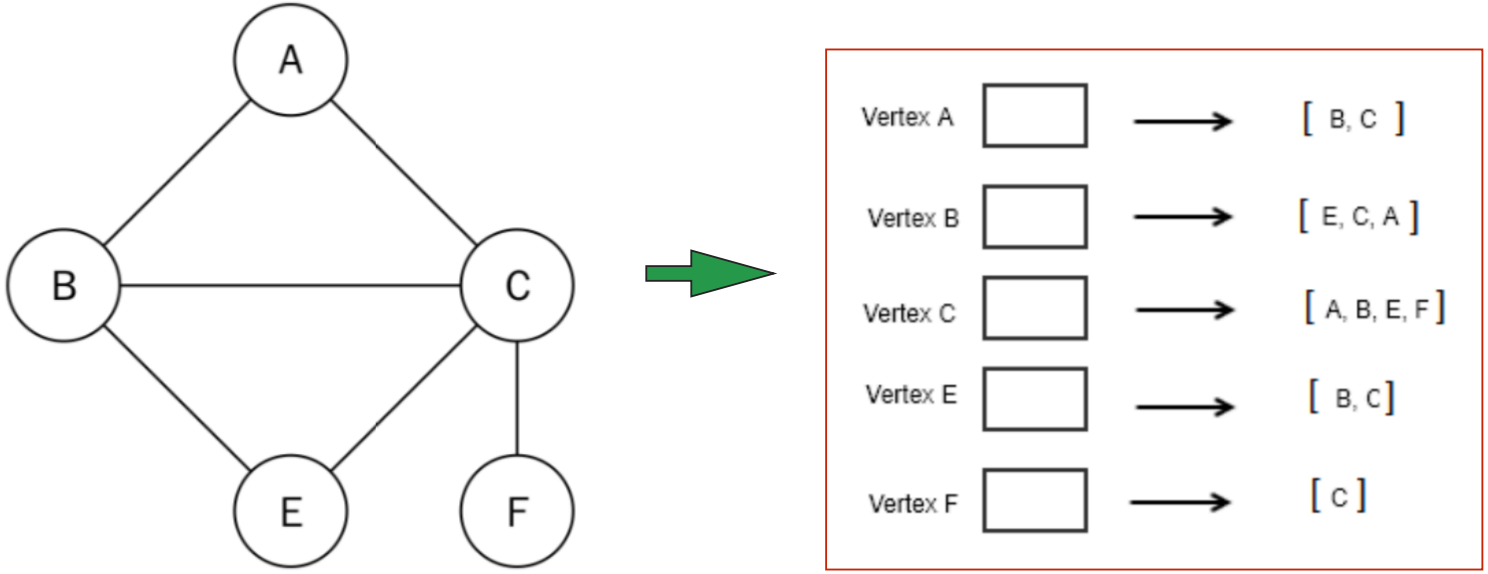
Grafi
Rappresentazione di grafi

I due modi più comuni di rappresentare un grafo sono i seguenti:

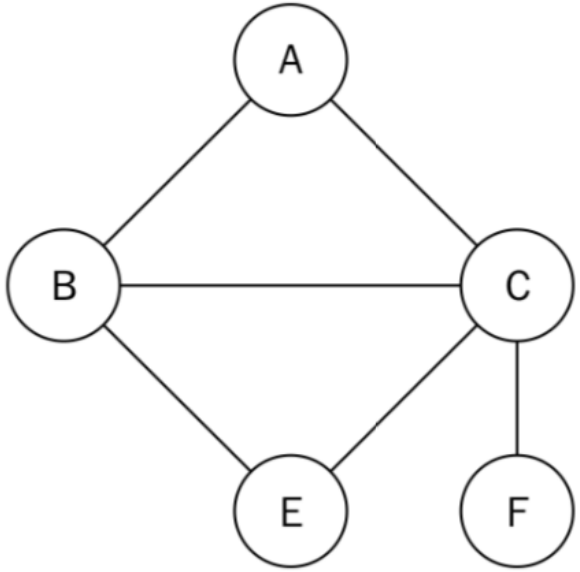
- **Liste di adiacenza**.
- **Matrici di adiacenza**.

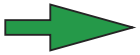
Il secondo modo è adatto in modo particolare a quelle relazioni in cui il numero delle coppie che appartengono alla relazione rappresenta una frazione piuttosto grande del numero di tutte le possibili coppie che si possono formare su un dato dominio.

Rappresentazione di Grafi
Liste di adiacenza



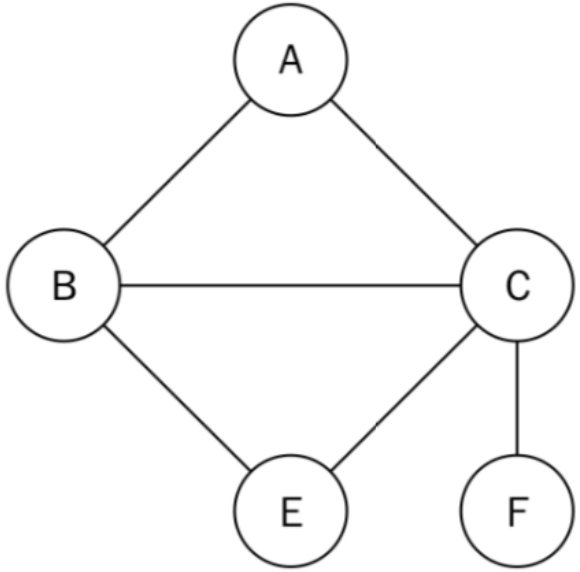
Rappresentazione di Grafi
Liste di adiacenza in Python

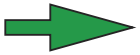




```
graph = dict()
graph['A'] = ['B', 'C']
graph['B'] = ['E','C', 'A']
graph['C'] = ['A', 'B', 'E','F']
graph['E'] = ['B', 'C']
graph['F'] = ['C']
```

Rappresentazione di Grafi
Matrici di adiacenza





	A	B	C	E	F
A	0	1	1	0	0
B	1	0	1	1	0
C	1	1	0	1	1
E	0	1	1	0	0
F	0	0	1	0	0

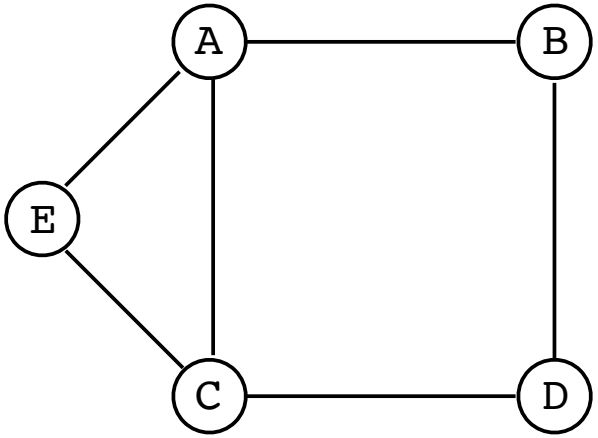
Visita di un grafo
Introduzione

Analizziamo ora le seguenti modalità di visita di un grafo:

- Visita in ampiezza (**breadth first**).
- Visita in profondità (**depth first**).

Visita di un Grafo
Breadth-First

Vediamo un esempio di visita breadth-first applicata al grafo seguente:

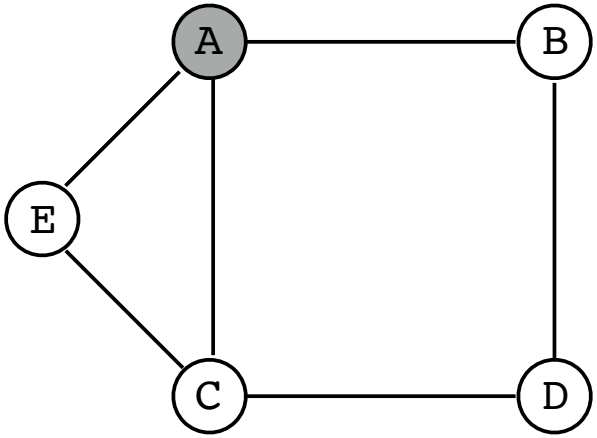


Visited:

Queue:

Visita di un Grafo
Breadth-First

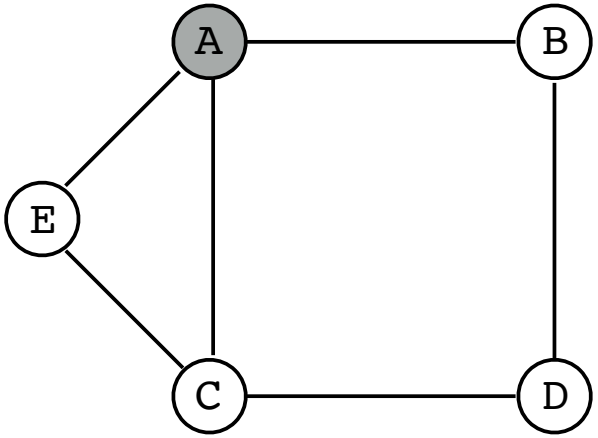
Partiamo dal nodo A:

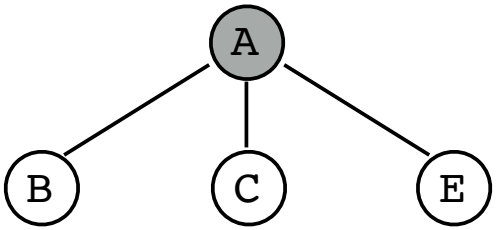


Visited:

Queue:

Visita di un Grafo
Breadth-First





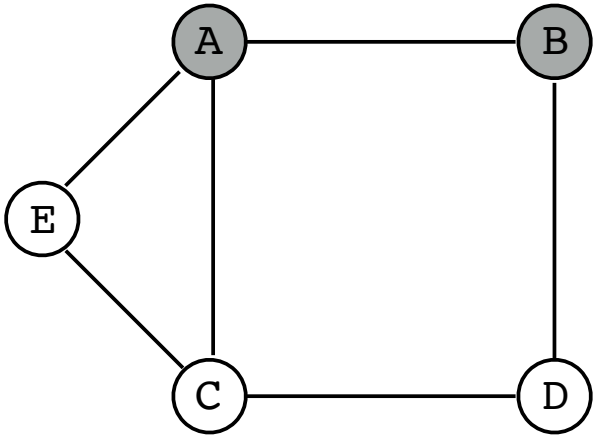
Visited:

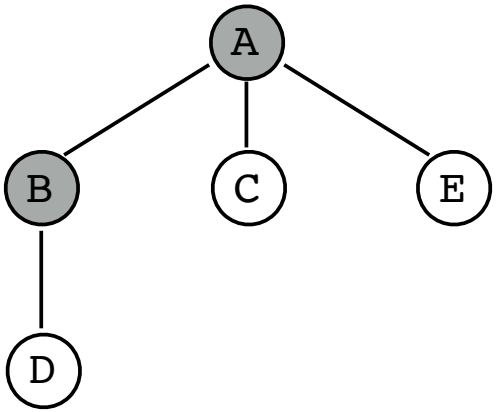
A								

Queue:

B	C	E						

Visita di un Grafo
Breadth-First





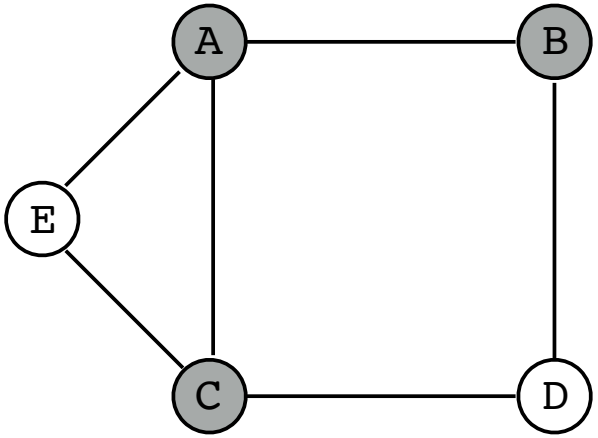
Visited:

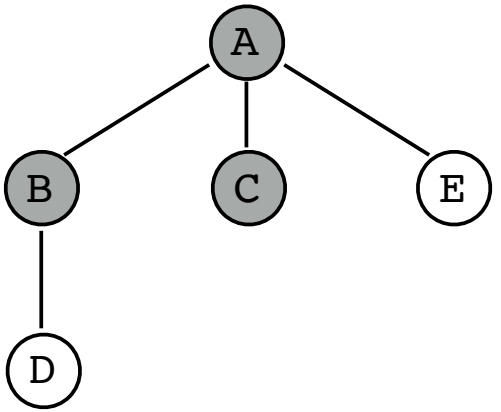
A	B							

Queue:

C	E	D						

Visita di un Grafo
Breadth-First





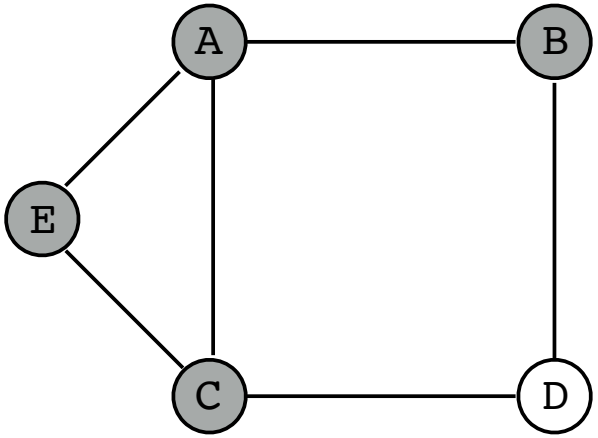
Visited:

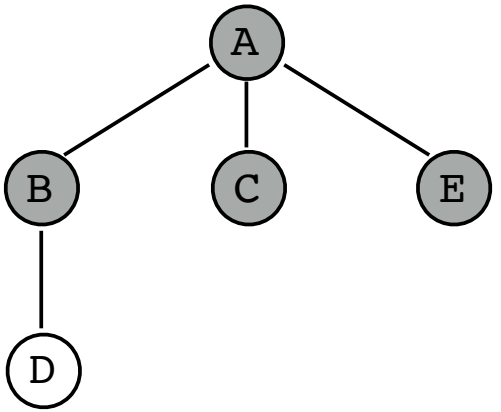
A	B	C						

Queue:

E	D							

Visita di un Grafo
Breadth-First





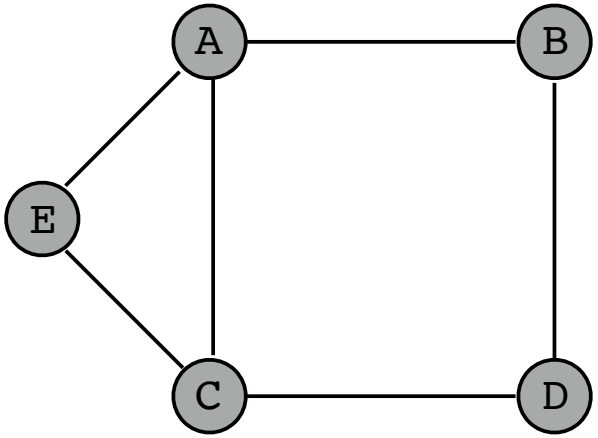
Visited:

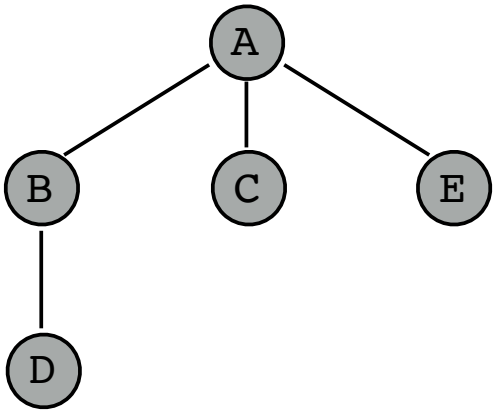
A	B	C	E					

Queue:

D								

Visita di un Grafo
Breadth-First





Visited:

A	B	C	E	D				

Queue:

Visita di un Grafo
Breadth-First

Algoritmo in codice Python:

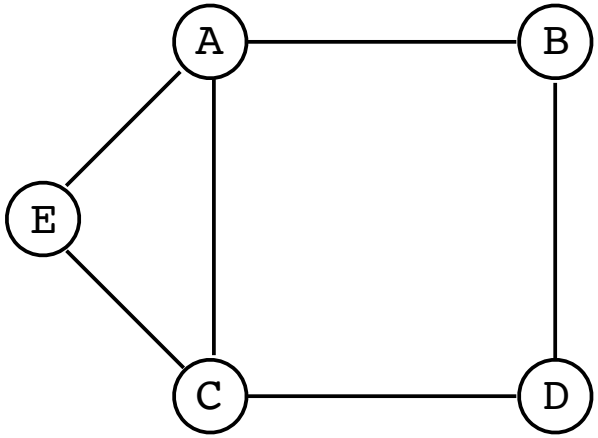
```
def breadth_first_search(visitati, grafo, nodo):
    visitati.append(nodo)
    coda.append(nodo)
    while coda:
        s = coda.pop(0)
        print(s, end = " ")
        for neighbour in grafo[s]:
            if neighbour not in visitati:
                visitati.append(neighbour)
                coda.append(neighbour)
```

```
visitati = []  # Lista che tiene traccia dei nodi visitati.
coda = []      #Inizializzazione della coda.

# Driver Code
breadth_first_search(visitati, graph, 'A')
```

Visita di un Grafo
Breadth-First

Applichiamo l'algoritmo al grafo che abbiamo appena visto:




```
graph1 = {
   'A' : ['B','C', 'E'],
   'B' : ['A', 'D'],
   'C' : ['A', 'D', 'E'],
   'D' : ['B', 'C'],
   'E' : ['A', 'C']
}
```

Visita di un Grafo
Breadth-First

Risultato dell'elaborazione:

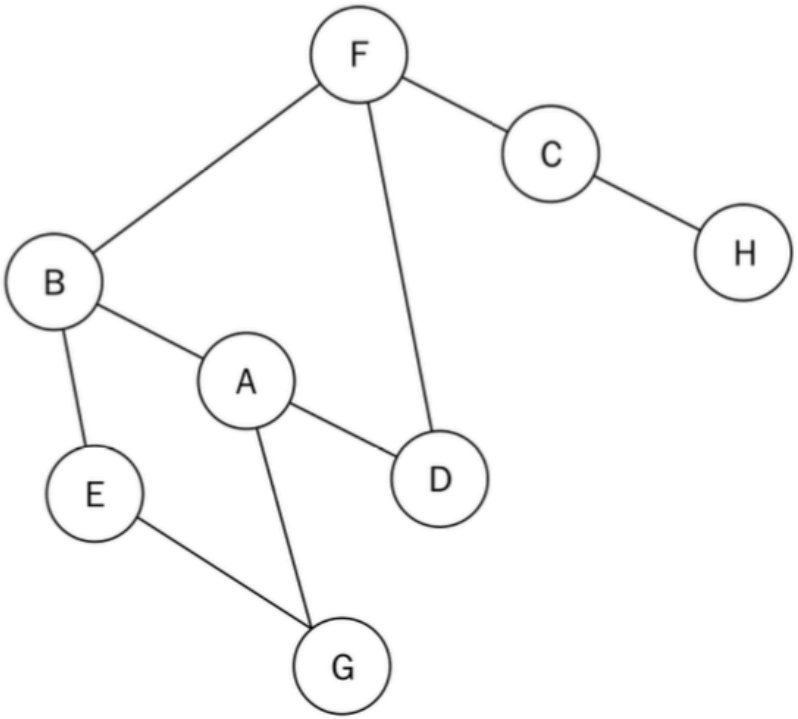
```
visitati = []  # Lista che tiene traccia dei nodi visitati.
coda = []      #Inizializzazione della coda.

# Driver Code
breadth_first_search(visitati, graph1, 'A')
```

```
A B C E D
```

Visita di un Grafo
Breadth-First

Applichiamo l'algoritmo al grafo seguente:



Visita di un Grafo
Breadth-First

Rappresentazione in Python del grafo:

```
graph2 = dict()
graph2['A'] = ['B', 'D', 'G']
graph2['B'] = ['A', 'E', 'F']
graph2['C'] = ['F', 'H']
graph2['D'] = ['A', 'F']
graph2['E'] = ['B', 'G']
graph2['F'] = ['B', 'C', 'D']
graph2['G'] = ['A', 'E']
graph2['H'] = ['C']
```

Visita di un Grafo
Breadth-First

Risultato dell'elaborazione:

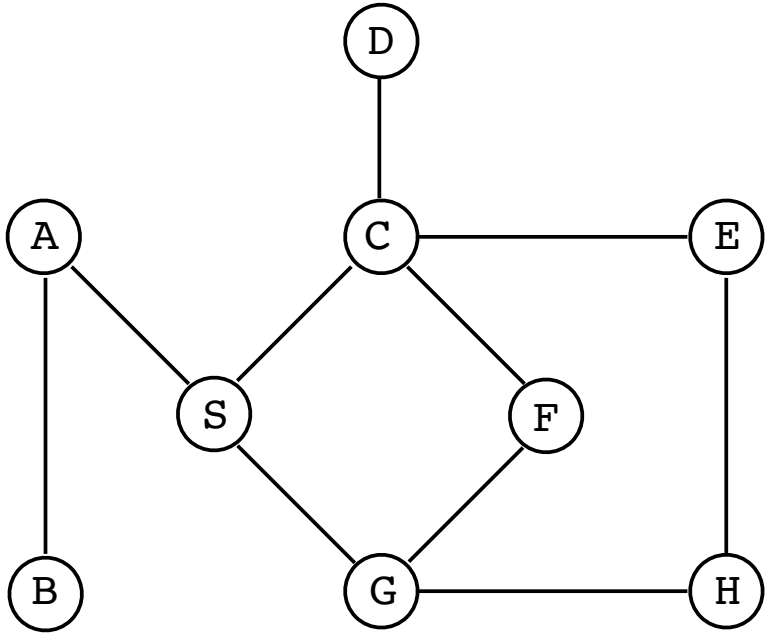
```
visitati = []  # Lista che tiene traccia dei nodi visitati.
coda = []      #Inizializzazione della coda.

# Driver Code
breadth_first_search(visitati, graph2, 'A')
```

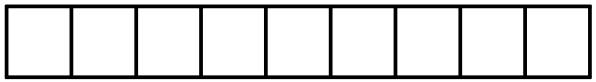
```
A B D G E F C H
```

Visita di un Grafo
Depth-First

Vediamo un esempio di visita applicata al grafo seguente:

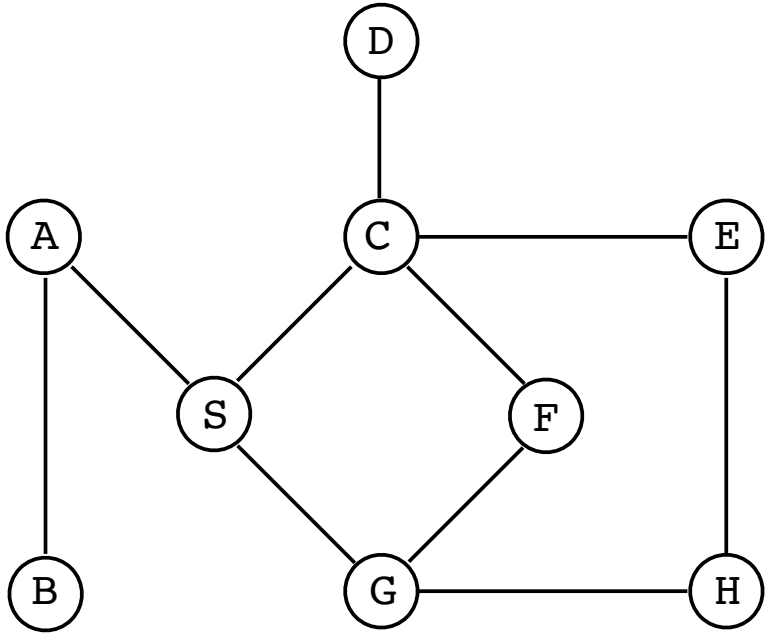


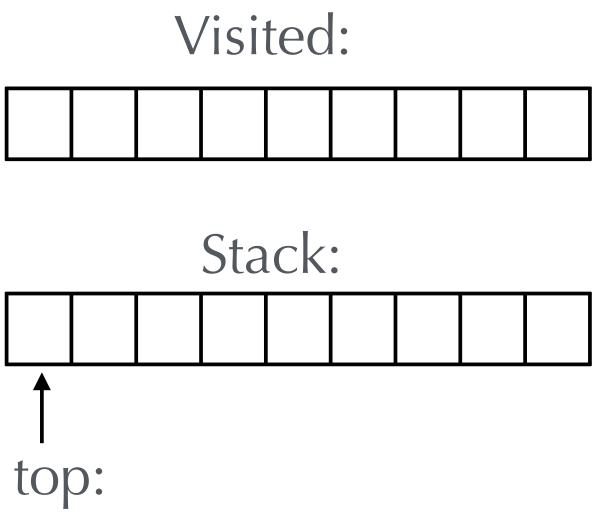
Visited:



Stack:

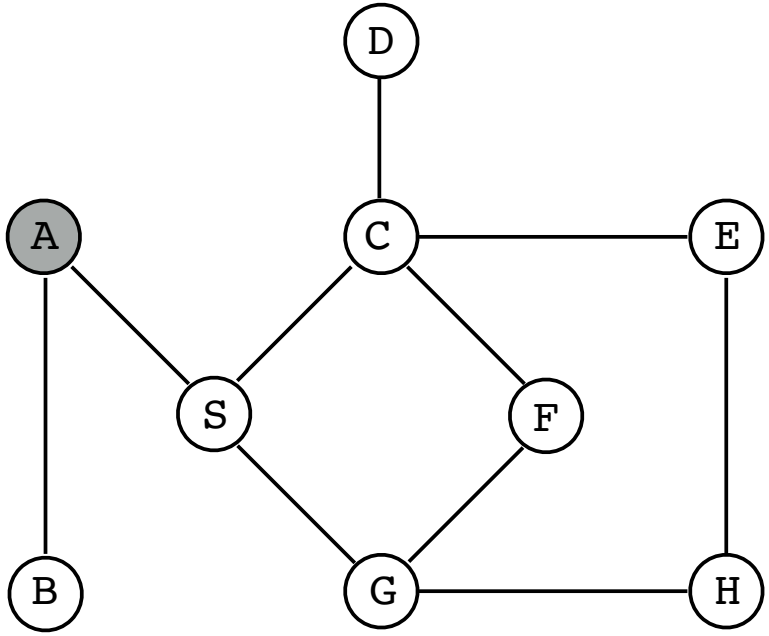
Visita di un Grafo
Depth-First



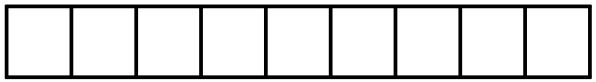


Visita di un Grafo
Depth-First

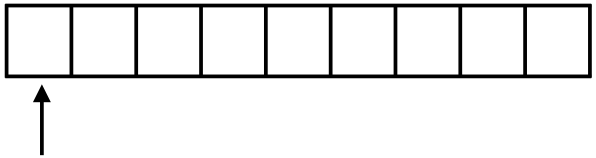
Partiamo dal nodo A:



Visited:

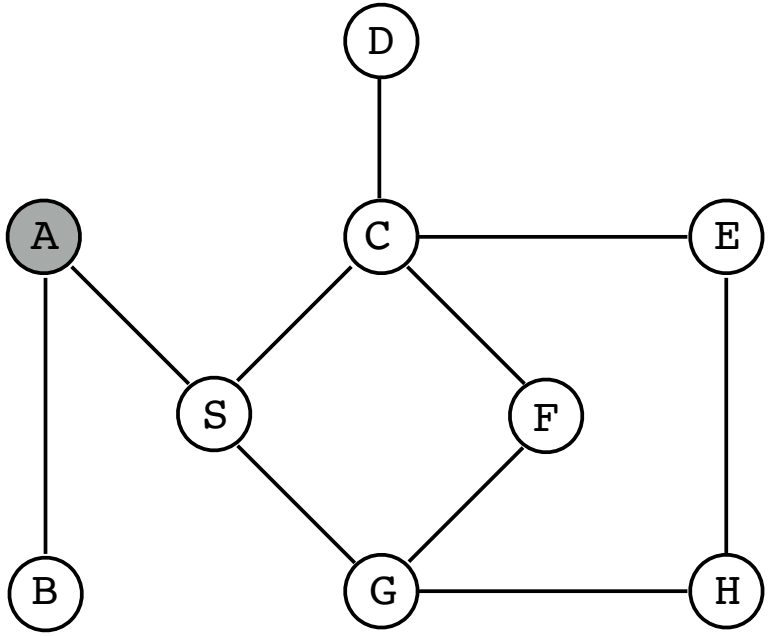


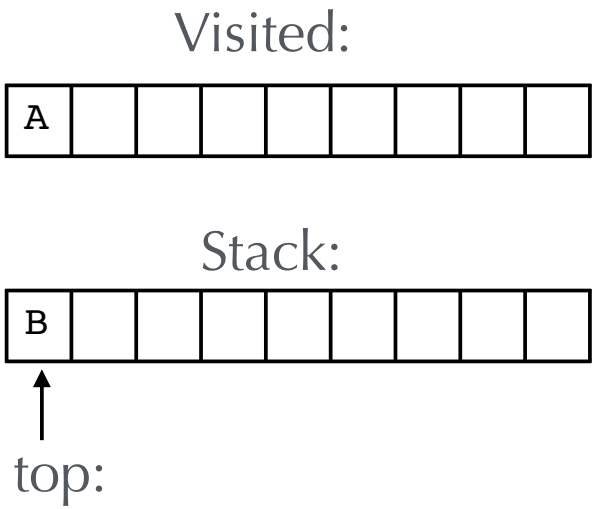
Stack:



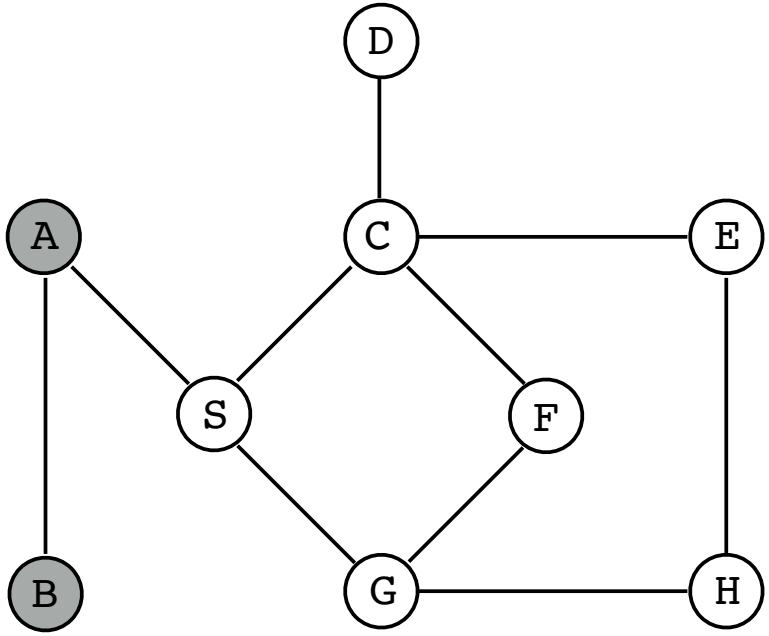
top:

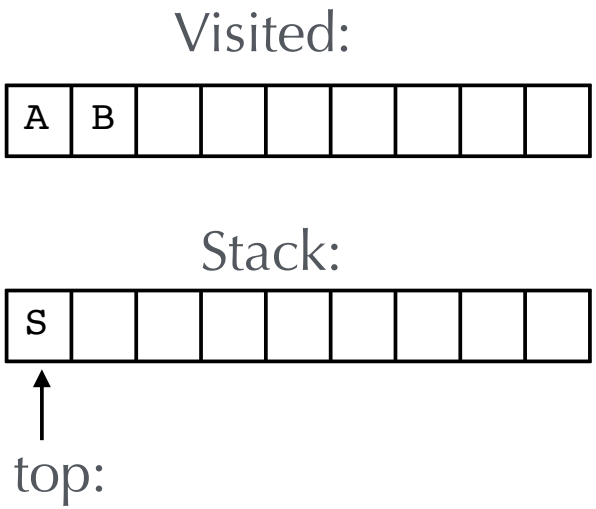
Visita di un Grafo
Depth-First



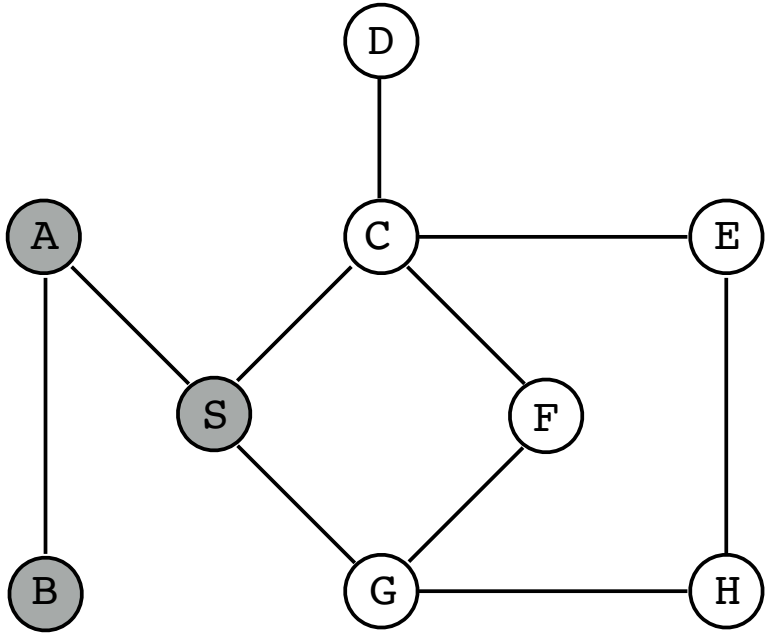


Visita di un Grafo
Depth-First





Visita di un Grafo
Depth-First



Visited:

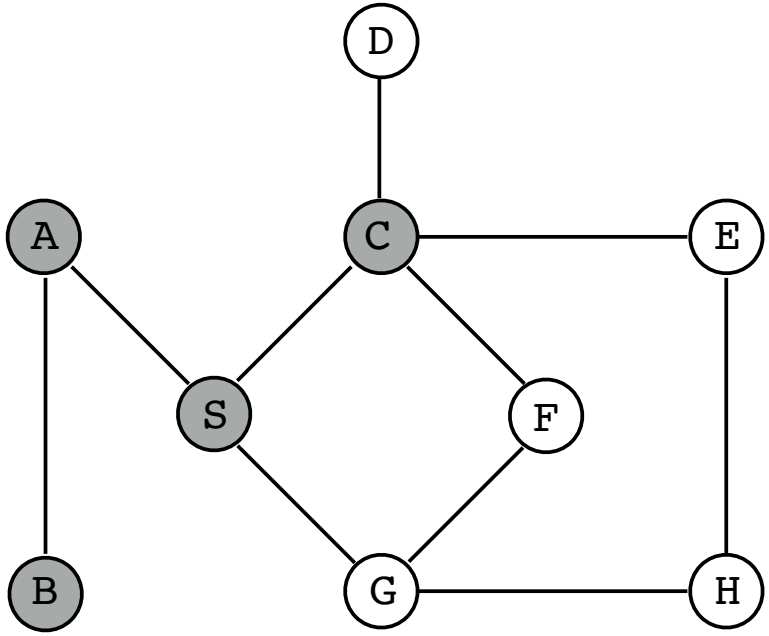
A	B	S						

Stack:

C								

top:

Visita di un Grafo
Depth-First



Visited:

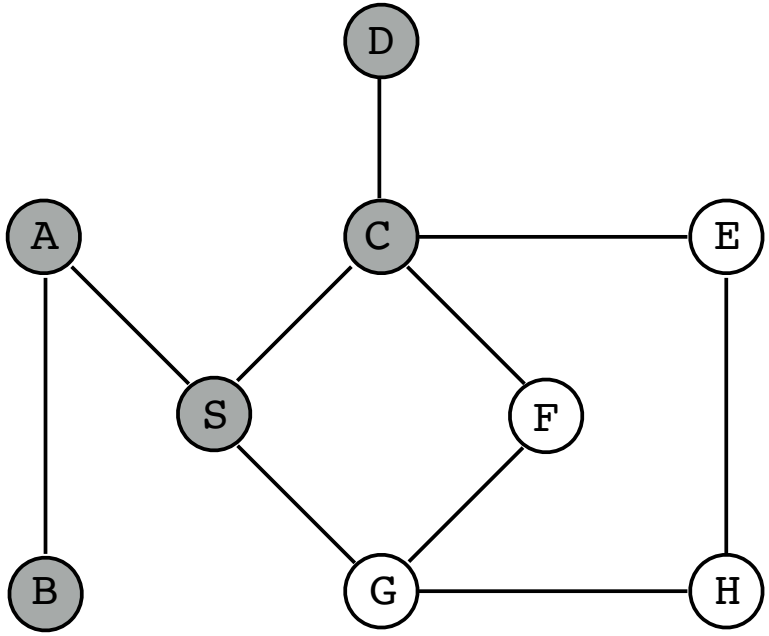
A	B	S	C					

Stack:

D								

top:

Visita di un Grafo
Depth-First



Visited:

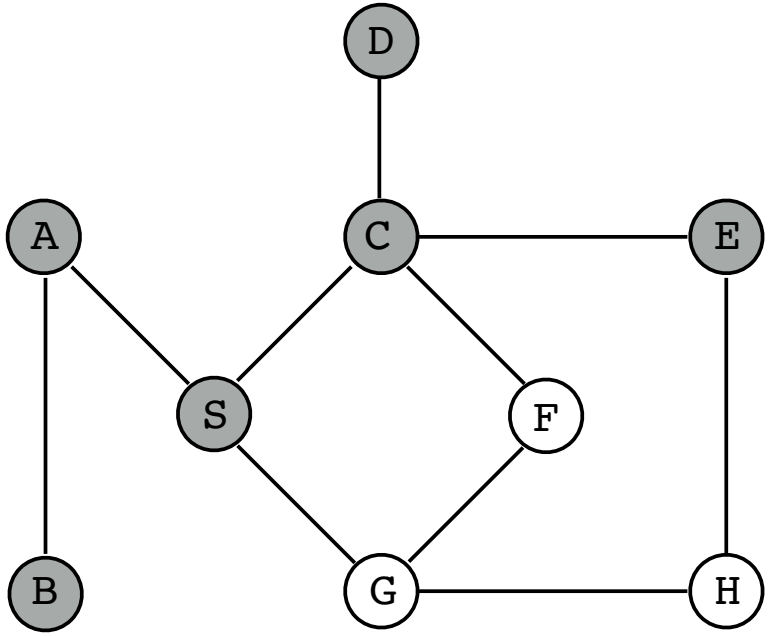
A	B	S	C	D				

Stack:

E								

top:

Visita di un Grafo
Depth-First



Visited:

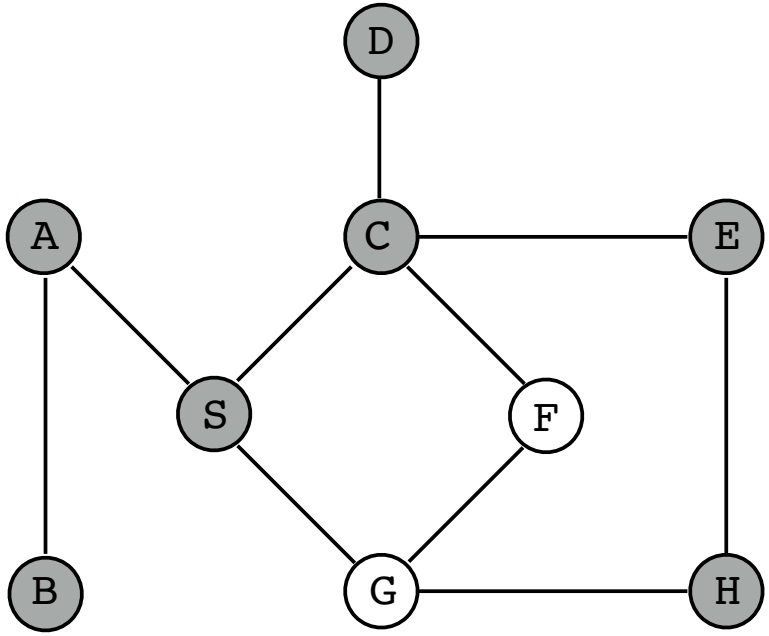
A	B	S	C	D	E			

Stack:

H								

top:

Visita di un Grafo
Depth-First



Visited:

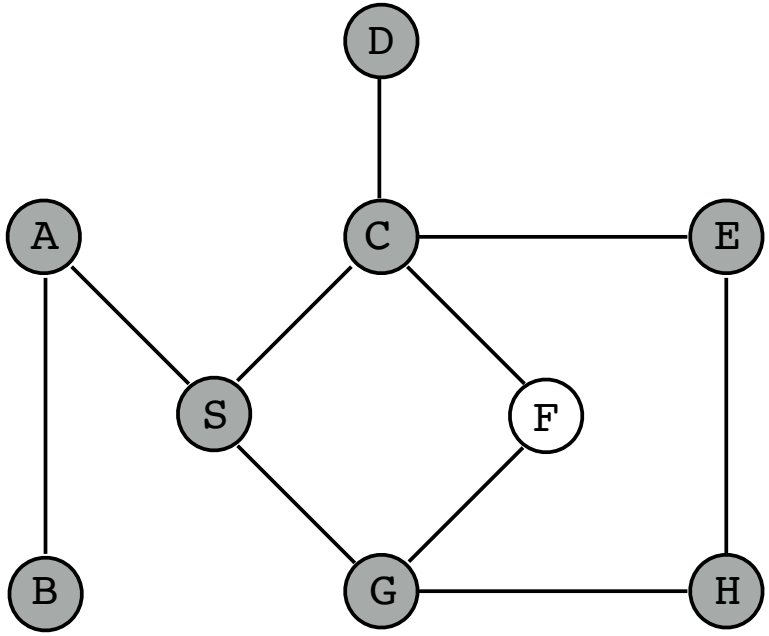
A	B	S	C	D	E	H		

Stack:

G								

top:

Visita di un Grafo
Depth-First



Visited:

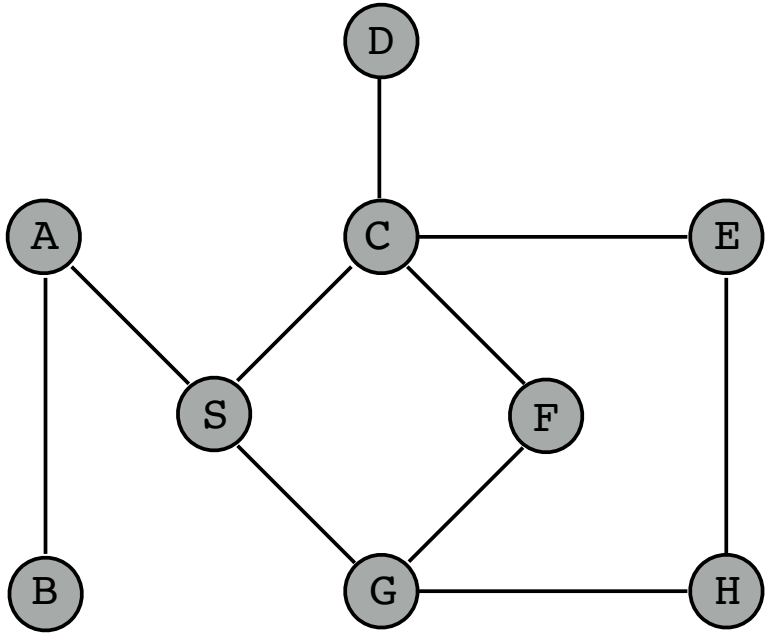
A	B	S	C	D	E	H	G	

Stack:

F								

top:

Visita di un Grafo
Depth-First



Visited:

A	B	S	C	D	E	H	G	F

Stack:

top:

Visita di un Grafo
Depth-First

Algoritmo in codice Python (versione ricorsiva):

```
def depth_first_search_R(graph, root, visited_vertices):
    node = root
    visited_vertices.append(node)
    adj_nodes = graph[node]
    for n in adj_nodes:
        if n not in visited_vertices:
            depth_first_search_R(graph, n, visited_vertices)
    return visited_vertices
```

Visita di un Grafo
Depth-First

Rappresentazione del grafo in Python:

```
graph = dict()
graph['A'] = ['B', 'S']
graph['B'] = ['A']
graph['S'] = ['A','C','G']
graph['D'] = ['C']
graph['G'] = ['F','H', 'S']
graph['H'] = ['E', 'G']
graph['E'] = ['C','H']
graph['F'] = ['C','G']
graph['C'] = ['D','E','F','S']
```

Visita di un Grafo
Depth-First

Risultato della elaborazione (vedi Notebook 6):

```
depth_first_search_R(graph, 'A', [])
```

```
['A', 'B', 'S', 'C', 'D', 'E', 'H', 'G', 'F']
```

Riferimenti

Agarwal, B., Baka, B. *Data Structures and Algorithms with Python*, Packt, 2018.

Lubanovic, B. *Introducing Python*, O'Reilly, 2020.

Horstmann, C., Necaise, R.D. *Python for Everyone*, John Wiley & Sons, 2014.

Hu, Y. *Easy learning Data Structures & Algorithms Python*, second edition, 2021.

Aho, A.V., Ullman, J.D. *Fondamenti di Informatica*, Zanichelli, 1994.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. *Introduzione agli algoritmi e strutture dati*, terza edizione, McGraw-Hill, 2010.