Laurea Triennale in Ingegneria Gestionale

Corso di Fondamenti di Informatica
A.A. 2017/2018

DEPARTMENT OF COMPUTER, CONTROL, AND MANAGEMENT ENGINEERING ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Grafi

Riferimento: Problem solving with Algorithms and Data Structures using Python

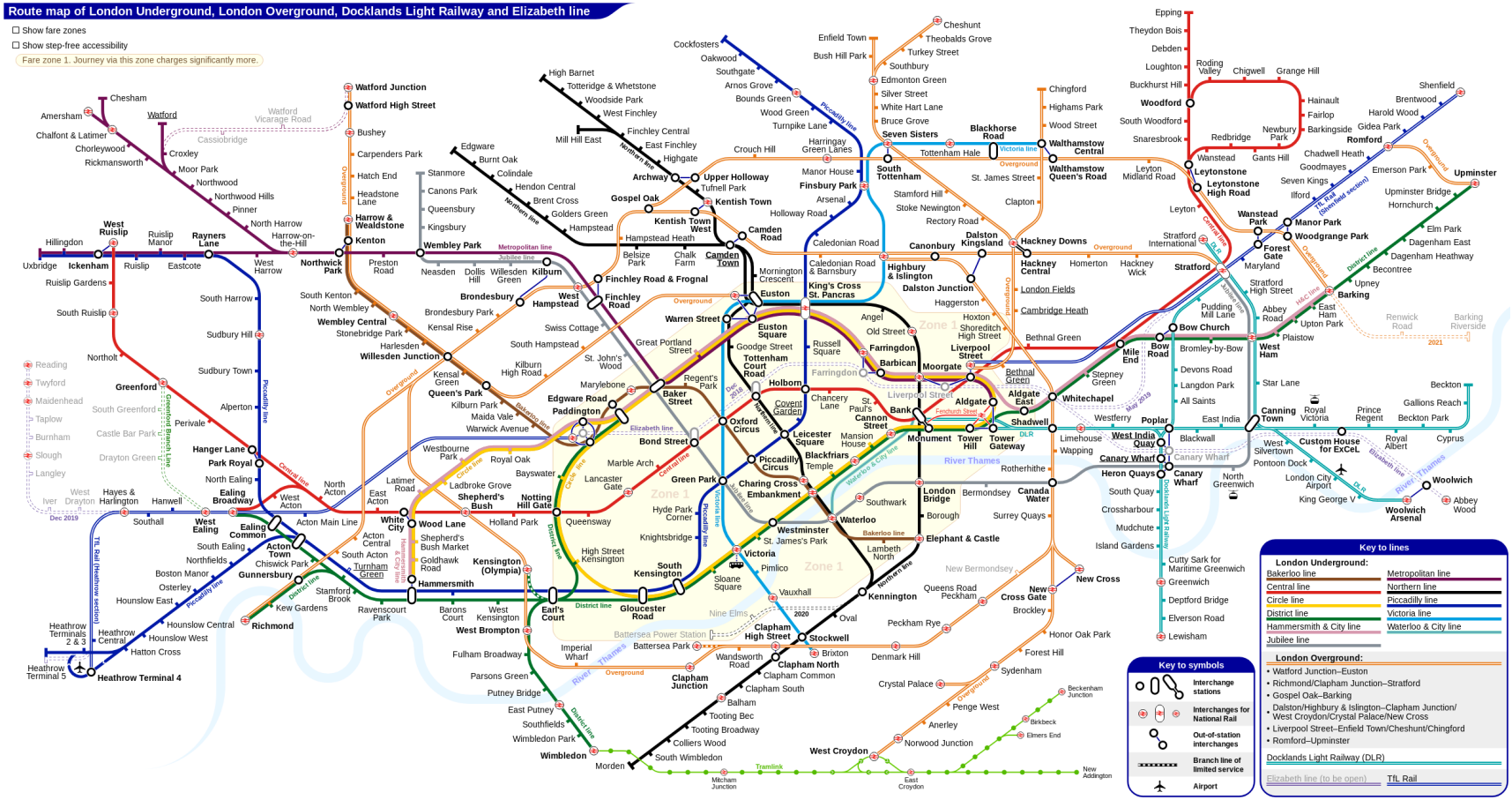
Introduzione

Grafi

- Oggetto di importanza fondamentale in vari campi
- Tanti problemi possono essere rappresentati come problemi su un grafo
  - logistica
  - trasporti
  - reti
- Struttura dati più generale rispetto agli alberi
- gli alberi sono una sottocategoria di grafi

Esempi

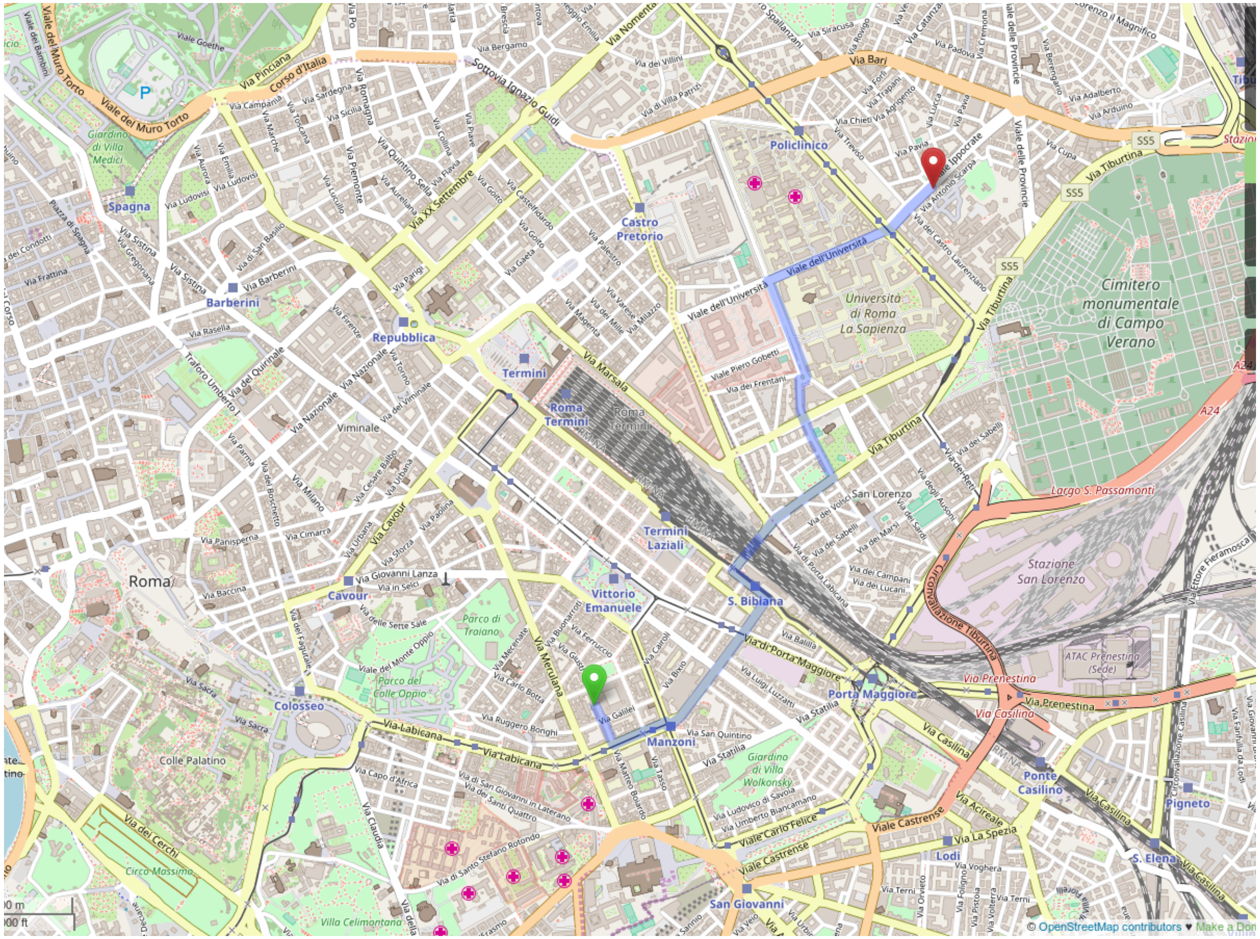
trasporti



Problemi: trovare percorso più veloce, meno costoso, ecc.

Esempi

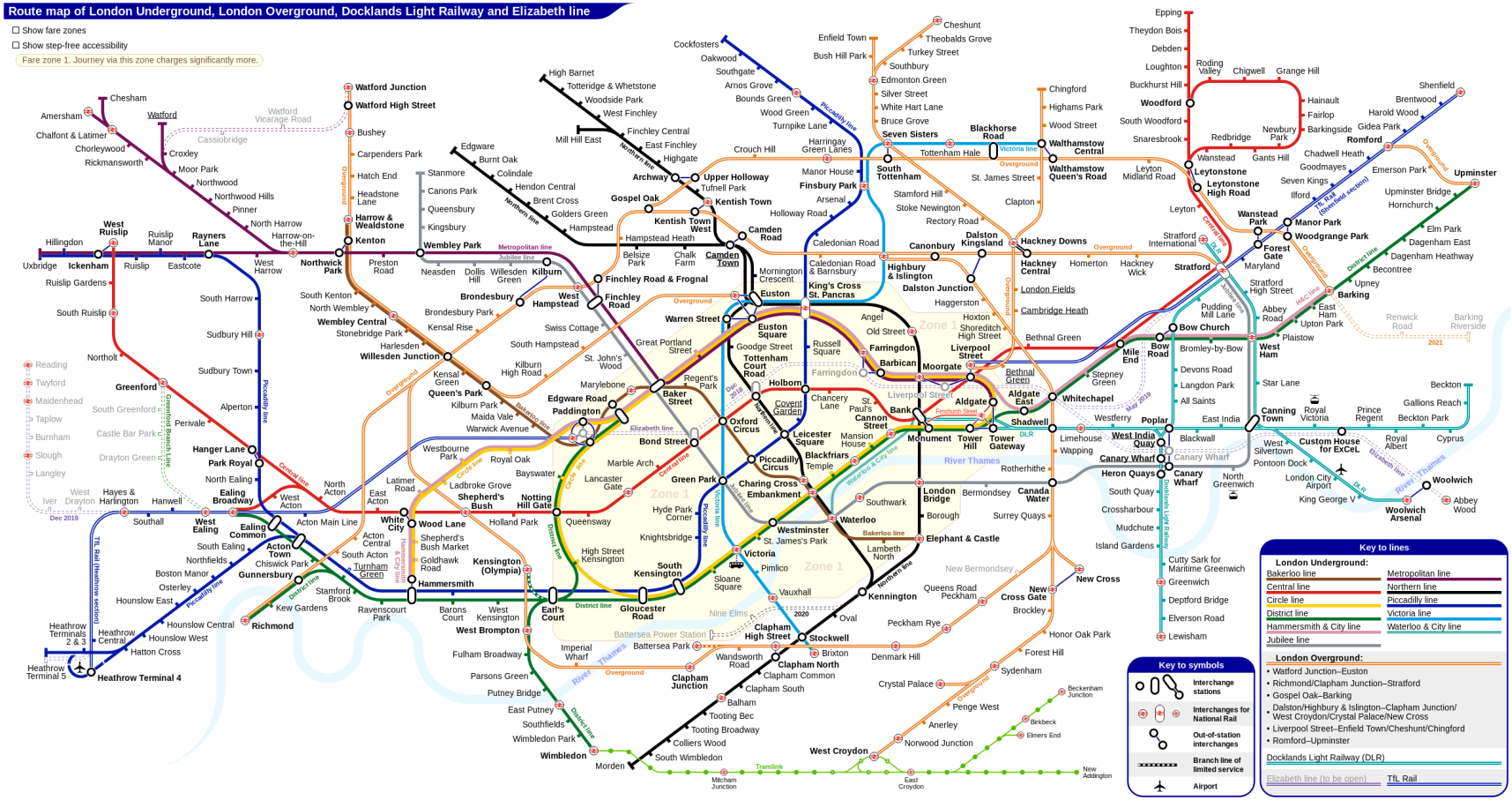
indicazioni stradali



Problemi: trovare percorso più veloce, più breve, ecc.

Esempi

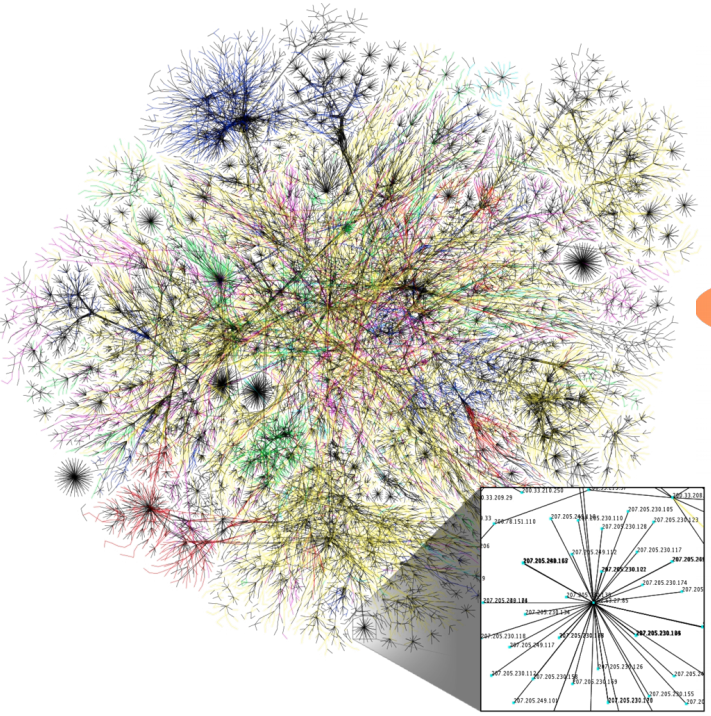
trasporti

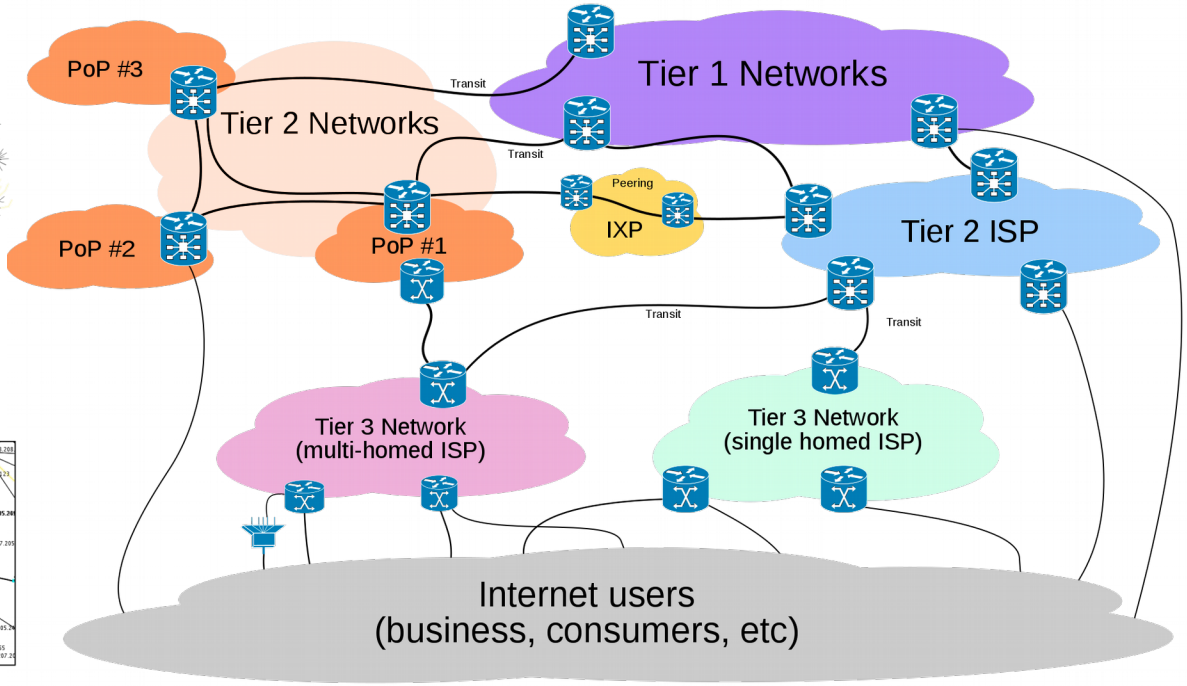


Problemi: trovare percorso più veloce, meno costoso, ecc.

Esempi

Internet



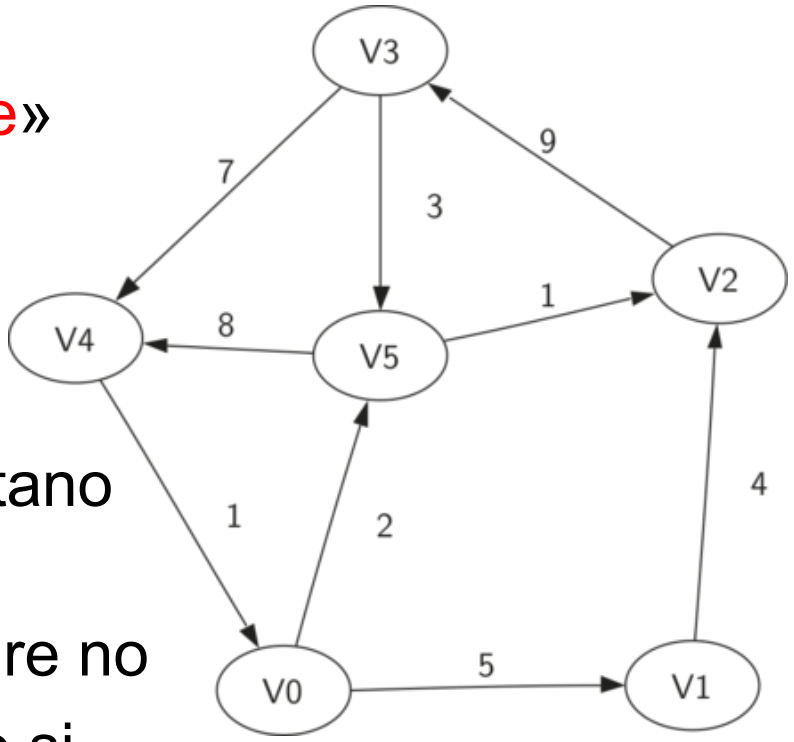


Problemi: trovare percorso più veloce, più breve, ecc.

Definizioni/Terminologia

Vertici/Nodi (vertices/nodes)

- possono avere un nome «chiave»
- possono avere dei dati associati



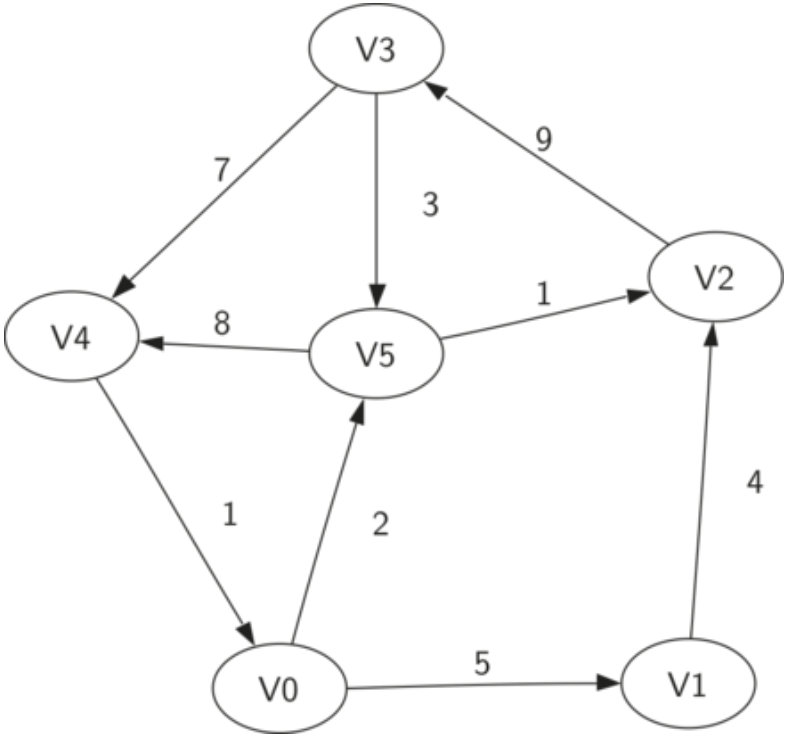
Archi (edges/arcs)

- collegano due nodi e rappresentano una relazione fra di loro
- possono essere a orientati oppure no
- se gli archi sono orientati il grafo si chiama orientato, diretto o digrafo
- altrimenti si chiama non orientato o indiretto

Definizioni/Terminologia

Pesi (Weights)

- gli archi possono aver associato un peso
- Tipicamente rappresenta il costo per attraversare l'arco (tempo, distanza, prezzo, ecc.)



Definizione

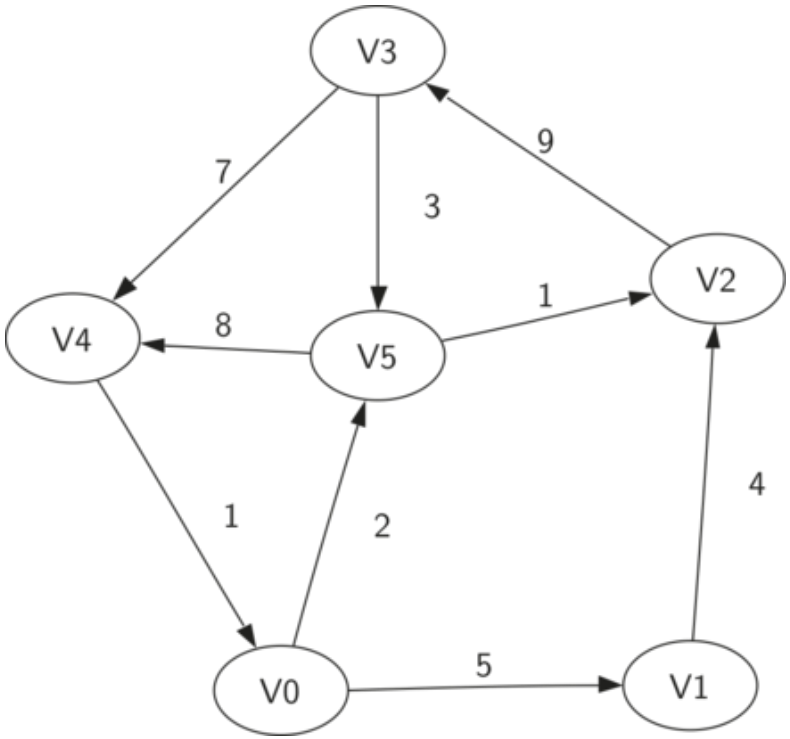
Grafo

Un grafo $G = (E,V)$ consiste di un insieme di vertici V ed un insieme di archi E

Ogni arco è una tupla (u,w) dove $u,w \in V$

Se il grafo è pesato la tupla contiene un terzo elemento che è il peso associato al arco

Un sottografo s è un insieme di vertici v e un insieme di archi e, tali che $v \in V$ e $e \in E$



$$V = \{V0, V1, V2, V3, V4, V5\}$$

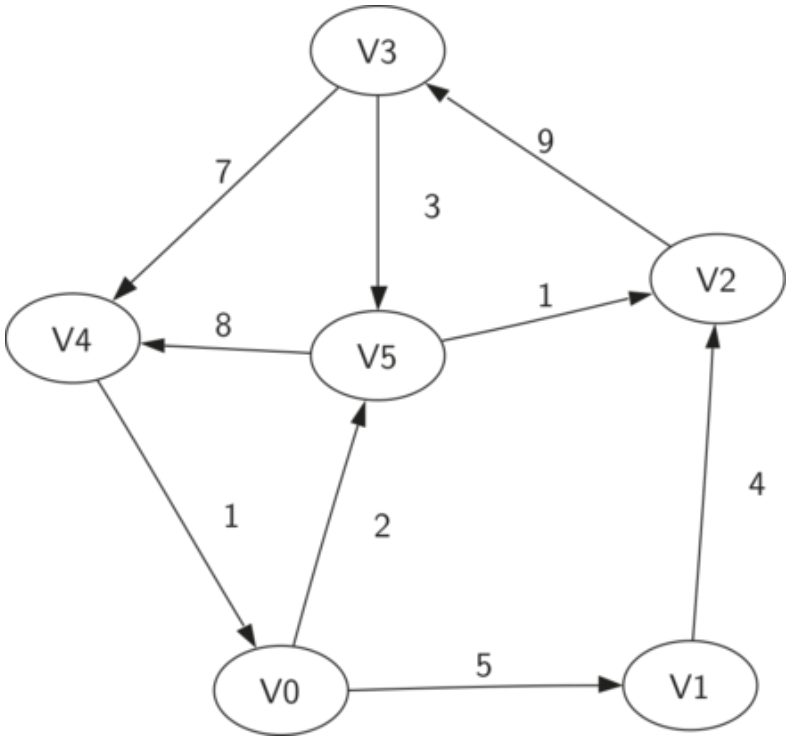
$$E = \left\{ \begin{matrix} (V0, V1, 5), (V1, V2, 4), (V2, V3, 9) \\ (V3, V4, 7), (V4, V0, 1), (V0, V5, 2) \\ (V5, V4, 8), (V3, V5, 3), (V5, V2, 1) \end{matrix} \right\}$$

Definizioni

Percorso (Path)

Una sequenza di vertici collegata da archi, ovvero un insieme di vertici $w_1, w_2, \ldots, w_n$ tale che $(w_i, w_{i+1}) \in V$ per ogni $1 \leq i \leq n-1$

- per un grafo non pesato la lunghezza del percorso è uguale al numero degli archi del percorso
- per un grafo pesato la lunghezza corrisponde alla somma dei pasi degli archi

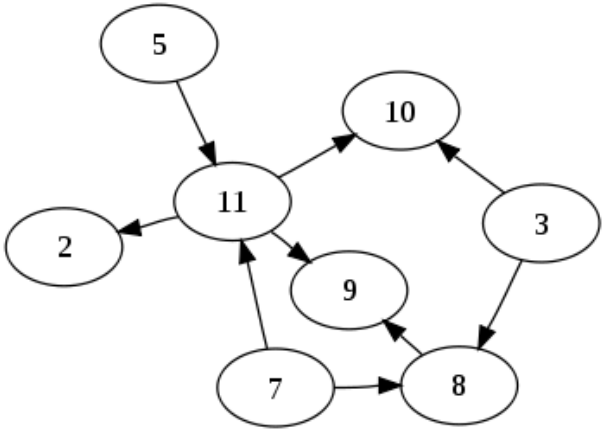


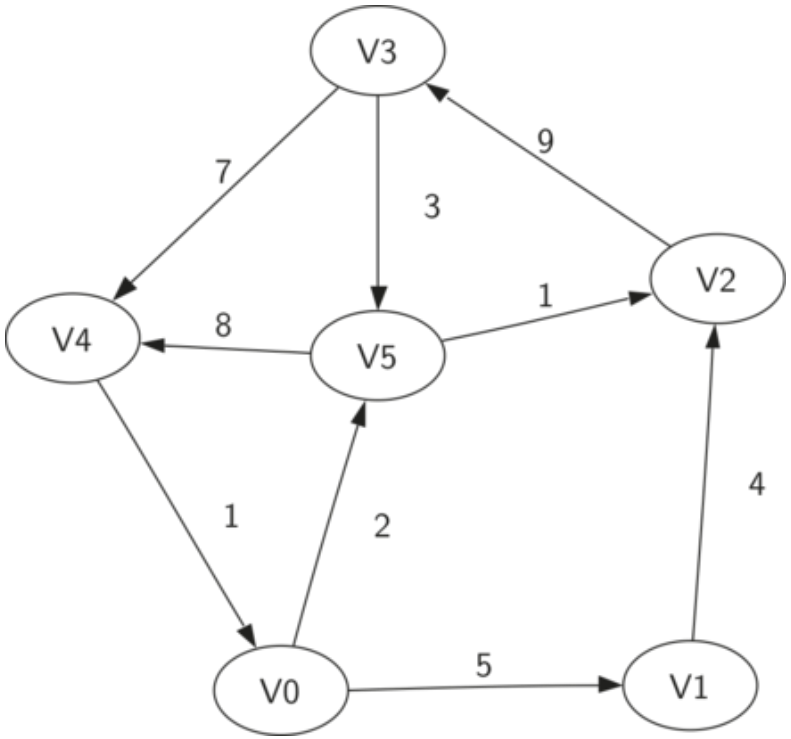
Definizioni

Ciclo (Cycle)

Un percorso che inizia e finisce con lo stesso nodo

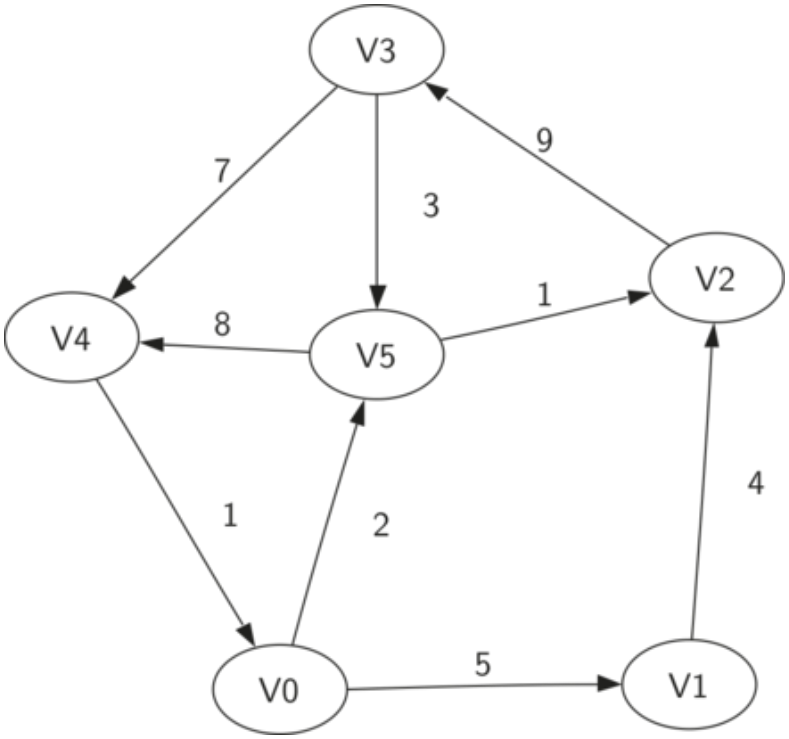
Un grafo orientato che non contiene cicli si chiama digrafo aciclico o DAG (Directed Acyclic Graph)





Matrice delle adiacenze

Matrice (sparsa) che nel posto (i,j) contiene il peso dell'arco corrispondente



	V0	V1	V2	V3	V4	V5
V0		5				2
V1			4			
V2				9		
V3					7	3
V4	1					
V5			1		8	

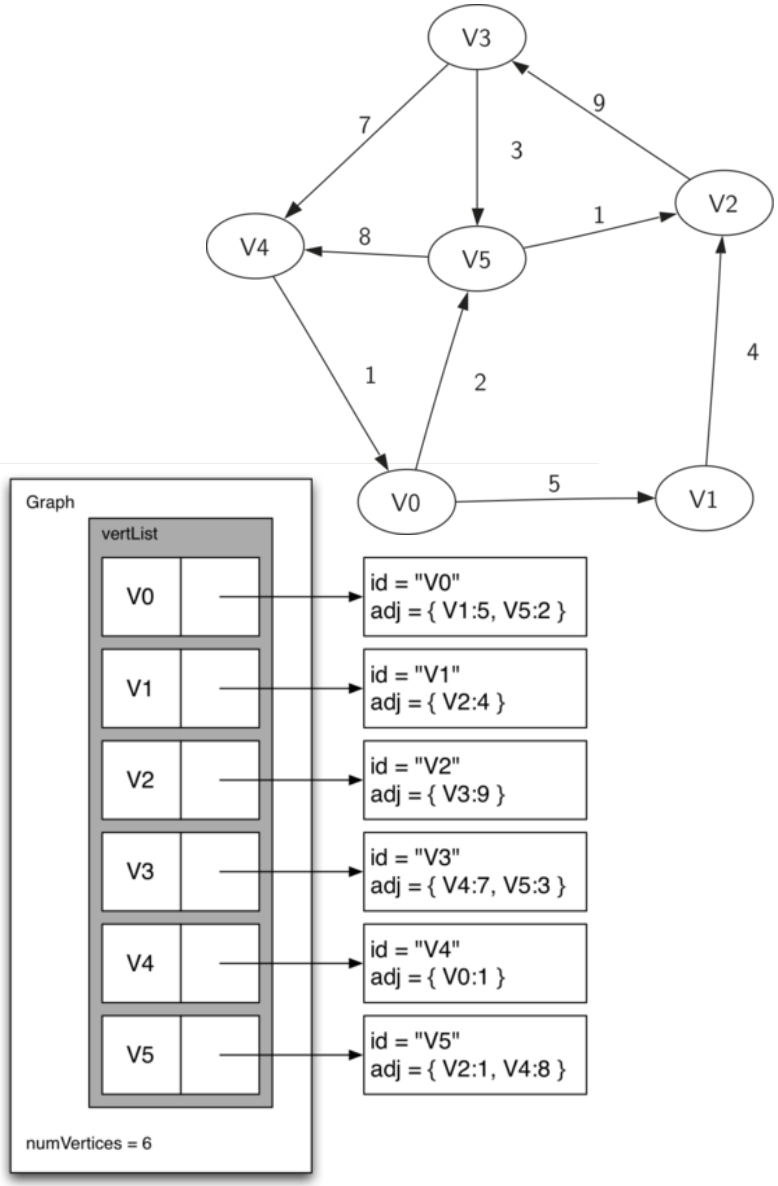
Lista delle adiacenze

Rappresentazione che richiede meno memoria

Ogni oggetto di tipo vertice contiene un dizionario dei vertici adiacenti raggiungibili dal vertice

Le chiavi del dizionario sono le id dei vertici adiacenti

I valori sono i pesi associati agli archi



Implementazione (1/2)

Classe: **`Vertex`**

Usa un dizionario per salvare i vertici adiacenti (raggiungibili dal vertice) e i pesi associati

Metodi

- `__init__(self,key)`: Crea il vertice e associa la sua chiave
- `addNeighbor(self,nbr,weight=1)`: Aggiunge un vertice adiacente e associa un peso al arco corrispondente
- `getConnections(self)`: Restituisce le chiavi dei vertici adiacenti
- `getId(self)`: Restituisce l'id (chiave) del vertice
- `getWeight(self,nbr)`: Restituisce il peso del arco diretto verso un vertice adiacente

Implementazione (2/2)

Classe: **`Graph`**

Usa un dizionario che associa la chiave (id) di ogni vertice del grafo all'oggetto corrispondente di tipo **`Vertex`**

Metodi

- `__init__(self)`: Crea l'oggetto del grafo
- `__iter__(self)`: Metodo per iterare fra i vertici del grafo
- `__contains__(self,n)`: Controlla se un vertice fa parte del grafo
- `addVertex(self,key)`: Aggiunge un vertice al grafo
- `getVertex(self,n)`: Restituisce l'oggetto del vertice `n`
- `addEdge(self,f,t,cost=1)`: Aggiunge un arco dal vertice `f` al `t` con un peso associato `cost` (valore di default: 1)
- `getVertices(self)`: Restituisce le id di tutti i vertici del grafo

Visita dei vertici – Ricerca in ampiezza

Ricerca in ampiezza (Breadth first search – BFS)

Dato un grafo G

- si sceglie un vertice di partenza s
- si esamina prima tutti i vertici raggiungibili da s (distanza 1)
- dopo aver esaminato i vertici di distanza 1 si esamina tutti i vertici raggiungibili da loro (distanza 2)
- si procede cosi finché non ci siano più vertici da visitare
-

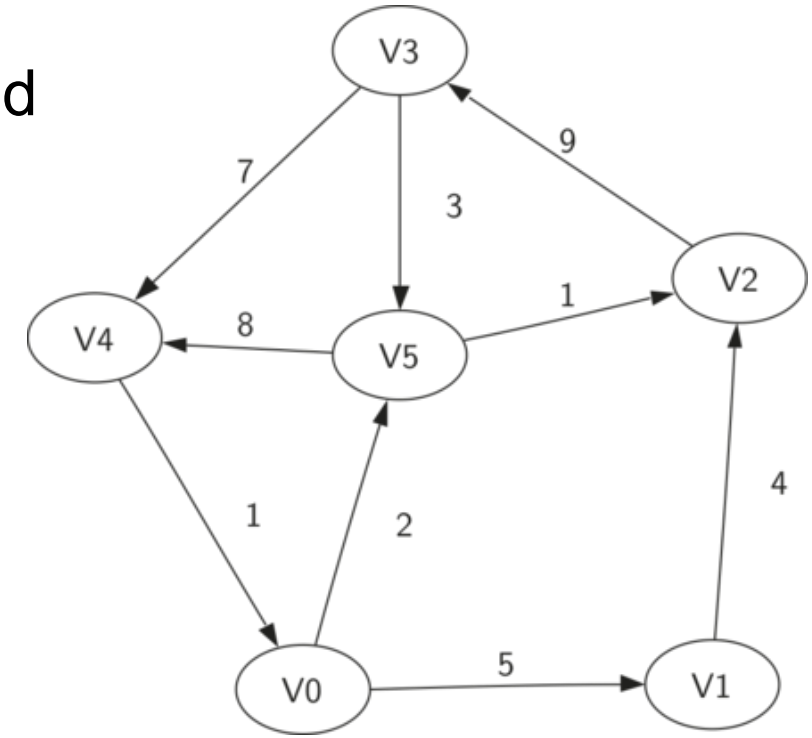
Visita dei vertici – Ricerca in ampiezza

Ricerca in ampiezza

Vertice di partenza $V0$

Lista di nodi visitati/da visitare ad ogni step:

1. $\emptyset$ | {V0}
2. {V0} | {V1,V5}
3. {V0,V1,V5} | {V2,V4}
4. {V0,V1,V5,V2,V4} | {V3}
5. {V0,V1,V5,V2,V4,V3} | $\emptyset$



Ricerca in ampiezza - Implementazione

```
def bfs(g, sorgente):
    queue = [sorgente]
    visitati = set()
    while queue != []:
        v = queue.pop(0)
        visitati.add(v)
        for vicino in g.getVertex(v).getConnections():
            if vicino not in visitati:
                queue.append(vicino)
    return visitati
```