Algoritmi

UniVR - Dipartimento di Informatica

Fabio Irimie

2° Semestre 2024/2025

Indice

1 Grafi

I grafi permettono di risolvere problemi particolarmente complessi, ma la parte difficile è la conversione di un problema in un grafo. I grafi sono costituiti da nodi e archi:

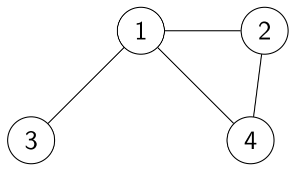


Figura 1: Esempio di grafo

- **Nodi**: rappresentano gli elementi del problema.
- **Archi**: rappresentano le relazioni tra i nodi.

I grafi in cui gli archi hanno un valore (o peso) vengono chiamati **grafi pesati**. Si possono anche aggiungere delle direzioni agli archi, ottenendo così un **grafo orientato**, in cui un arco si può attraversare in un solo verso.

Definizione 1.1 (Cammino)**.** Un **cammino** è una sequenza di nodi per cui esiste un arco tra ogni coppia di nodi adiacenti.

In un cammino, la ripetizione di un nodo rappresenta un **loop** e questo cammino viene detto **cammino ciclico**. (un cammino senza cicli si dice **cammino semplice**)

Il **grado** di un nodo è il numero di archi che incidono sul nodo. Ha senso parlare di grado di un nodo solo quando il grafo non è orientato perchè così ogni arco viene contato una sola volta.

- **Grado entrante**: numero di archi entranti in un nodo.
- **Grado uscente**: numero di archi uscenti da un nodo.

La definizione formale di un grafo è la seguente:

Definizione 1.2. Un grafo è definito come una coppia $G = (V, E)$ dove:

- V è un insieme di nodi.
- E è un insieme di archi:
$$E \subseteq V \times V$$

Dallla figura 1 si ha che:

- $V = \{1, 2, 3, 4\}$.
- $E = \{(1, 3), (3, 1), (1, 1), (1, 4), (4, 1), (1, 2), (2, 4), (4, 2)\}$.

La definizione formale dei concetti precedenti è:

Definizione 1.3. Il **grado uscente** di un nodo v in un grafo orientato $G = (V, E)$ è il numero di archi uscenti da v ($|\dots|$ è la cardinalità di un insieme):

$$\text{grado\_uscente}(v) = |\{u \mid (v, u) \in E\}|$$

Definizione 1.4. Il **grado entrante** di un nodo v in un grafo orientato $G = (V, E)$ è il numero di archi entranti in v:

$$\text{grado\_entrante}(v) = |\{u \mid (u, v) \in E\}|$$

Definizione 1.5. Un cammino è una sequenza di nodi in cui per ogni coppia di nodi consecutivi esiste un arco:

$$\forall i \in \{0 \dots n-1\} \quad (v_i, v_{i+1}) \in E$$

1.1 Rappresentazione di un grafo

Per rappresentare un grafo ci sono due modi:

- **Rappresentazione per liste di adiacenza**: Si crea una lista in cui si rappresentano i nodi e ad ogni nodo si associa la lista di tutti i nodi raggiungibili tramite un arco. Prendiamo in considerazione la figura 1:

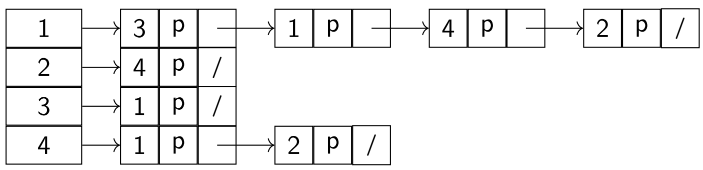


Figura 2: Rappresentazione per liste di adiacenza

  Lo spazio in memoria occupato è $\Theta(|V| + |E|)$.

- **Rappresentazione per matrice di adiacenza**: Si crea una matrice A di dimensione $|V| \times |V|$ in cui $A_{ij} = 1$ se esiste un arco tra i nodi i e j, altrimenti $A_{ij} = 0$. Prendiamo in considerazione la figura 1, dove p è il peso dell'arco:

/	1	2	3	4
1	1	1	1	1
2	0	0	0	1
3	1	0	0	0
4	1	1	0	0

Tabella 1: Rappresentazione per matrice di adiacenza

  Lo spazio in memoria occupato è $\Theta(|V|^2)$.

- Un **grafo trasposto** è un grafo in cui tutti gli archi sono invertiti.
- La **chiusura transitiva di un grafo** è un grafo in cui se esiste un cammino tra due nodi allora esiste un arco diretto tra i due nodi:

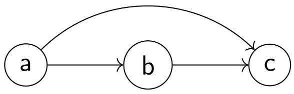


Figura 3: Grafo con chiusura transitiva

- Il **diametro** è il percorso più lungo fra i percorsi minimi

1.2 Esplorazione di un grafo

1.2.1 Visita in ampiezza (BFS: Breath First Search)

La visita in ampiezza (o a ventaglio) è un algoritmo che permette di visitare tutti i nodi di un grafo partendo da un nodo iniziale. L'algoritmo è il seguente:

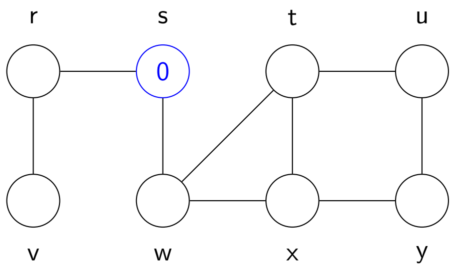
```
// G e' un grafo composto da un insieme di nodi V e un insieme di archi E
// s e' un nodo dell'arco
bfs(G, s)
  for u in G.V
    u.color <- white // non esplorato
    u.distance <- +inf // distanza dal nodo s
    u.parent <- NIL // nodo da cui si arriva a u

  s.color <- gray // scoperto, ma non esplorato
  s.distance <- 0
  s.parent <- NIL
  Q <- {s} // coda FIFO che contiene i nodi scoperti non esplorati

  while Q != empty
    u <- q.head

    for v in G.adj(u) // lista di nodi adiacenti a u
      if v.color == white
        v.color <- gray
        v.distance <- u.distance + 1
        v.parent <- u
        Q.enqueue(v)

  Q.dequeue()
  u.color <- black // esplorato
```

Esempio 1.1. L'algoritmo passo per passo è il seguente, dove i colori rappresentano:

- Nero: non esplorato,
- Blu: scoperto, ma non esplorato,
- Rosso: esplorato,

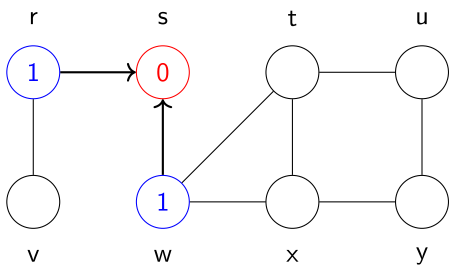
1. Primo passo:

Distanza	0
Coda	s



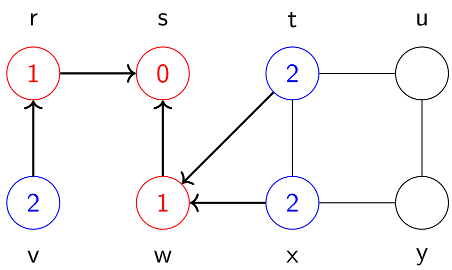
2. Secondo passo:

Distanza	0	1	1
Coda	s	w	r



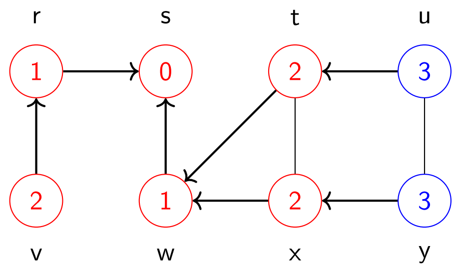
3. Terzo passo:

Distanza	0	1	1	2	2	2
Coda	~~s~~	~~w~~	~~r~~	t	x	v



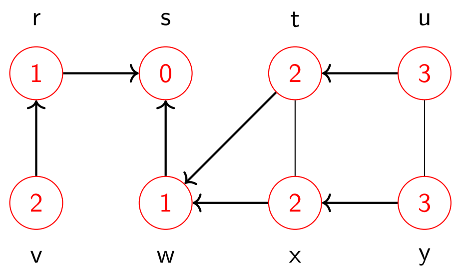
4. Quarto passo:

Distanza	0	1	1	2	2	2	3	3
Coda	~~s~~	~~w~~	~~r~~	~~t~~	~~x~~	~~v~~	u	y



5. Quinto passo:

Distanza	0	1	1	2	2	2	3	3
Coda	~~s~~	~~w~~	~~r~~	~~t~~	~~x~~	~~v~~	~~u~~	~~y~~



Se si vuole trovare il cammino minimo tra due nodi, si parte dal nodo di destinazione e si risale al nodo di partenza seguendo il campo `parent` di ogni nodo.