Esercitazione 8

Algoritmo di Dijkstra

Algoritmo di Dijkstra

- Risolve il ***problema dei cammini minimi*** da sorgente unica in un grafo pesato (con pesi non negativi)
- ***Grafo G = (V,E)***
- Consideriamo un grafo non orientato per semplicità

Rappresentazione tramite liste di adiacenza

- Supponiamo di rappresentare il grafo **G = (V,E)** con ***liste di adiacenza***
- Array **Adj** di **|V|** liste, una per ogni vertice in **V**
- Per ogni **u** in **V**, **Adj[u]** contiene tutti i ***vertici adiacenti*** a **u** in **G**, memorizzati in ***ordine arbitrario***

Strutture dati e idea base

- L'algoritmo di Dijkstra mantiene un insieme **S** di vertici i cui cammini minimi verso la **sorgente s** sono già stati determinati
- L'algoritmo utilizza l'insieme **V - S** di vertici del grafo per cui si ha solo una ***stima per eccesso della distanza dalla sorgente s***
  - Ad es., i valori dell'albero BFT
  - Oppure, più semplicemente, + infinito

Inizializzazione

- Supponiamo di usare i valori ottenuti durante la visita in ampiezza BFS, memorizzando le seguenti informazioni (creazione del BFT):
  - Il ***padre (o predecessore)*** **parent[u]**: il vertice da cui si arriva ad **u** durante la visita del grafo
  - la ***distanza dalla sorgente s*** **d[u]**: la somma dei pesi degli archi che formano il cammino da **s** ad **u**
    NOTA: **non è detto che sia la distanza minima se il grafo è pesato**

Fasi dell'algoritmo (1)

- Si parte da un insieme **S, che inizialmente contiene solo la sorgente s**
- L'algoritmo seleziona dall'insieme **V-S** il vertice **u** con ***la stima minima della distanza d[u] da s***
- ***Fissa il cammino minimo di u pari a d[u]***
- Aggiunge **u** ad **S**
- Esamina tutti i vertici **v** adiacenti di **u** che sono ancora in **V-S**
  - Vertici per cui non è ancora stabilito il valore definitivo del cammino minimo

Fasi dell'algoritmo (2)

Per ogni vertice **v** adiacente di **u** e ancora in **V-S**

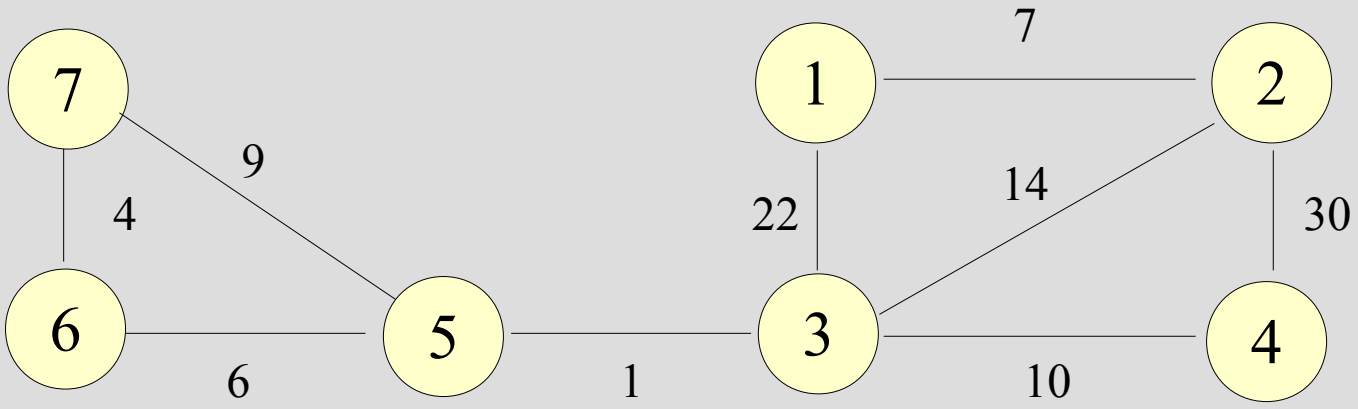
- Verifica se il cammino minimo di **v** può essere migliorato passando per **u**, cioè se
  - **d[u] + peso dell'arco tra u e v < d[v]**
- Aggiorna se necessario la stima **d[v]**
- Aggiorna se necessario il predecessore **parent[v]**

Lettura grafo da file

graph1.w e **graph2.w**: file di input che contengono una lista di archi pesati

Esempio da graph1.w

```
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4
```

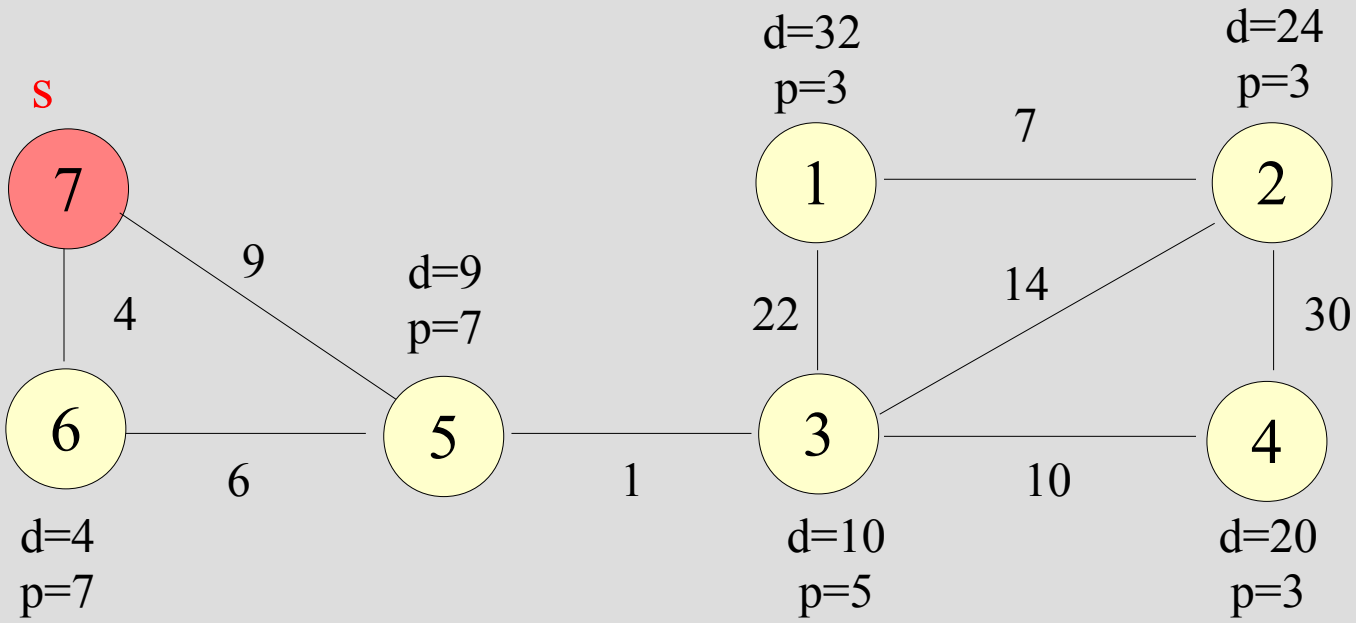


Nota: interpretato come non orientato

Grafo dopo BFS

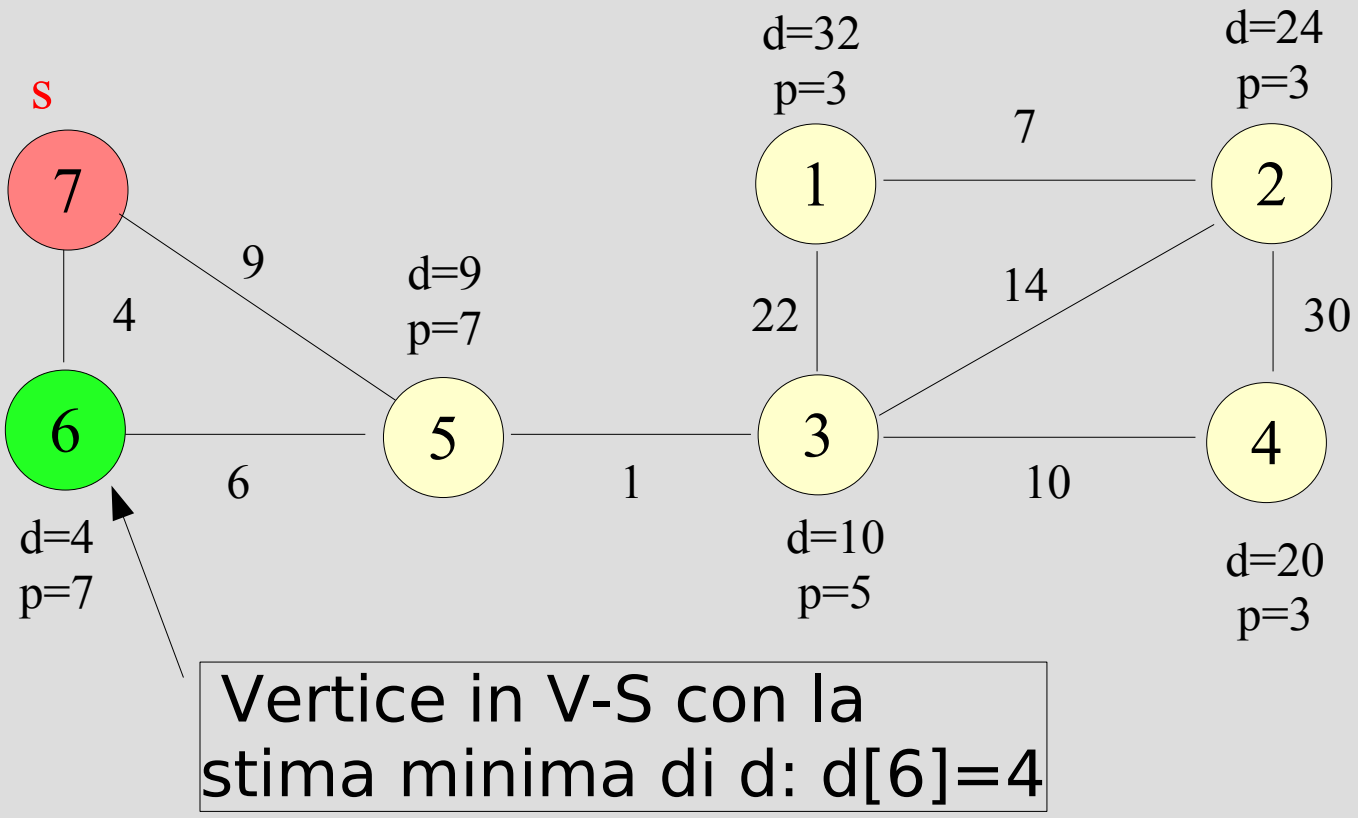
Es.
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4

Sorgente s=7
S={7}
V-S={1,2,3,4,5,6}



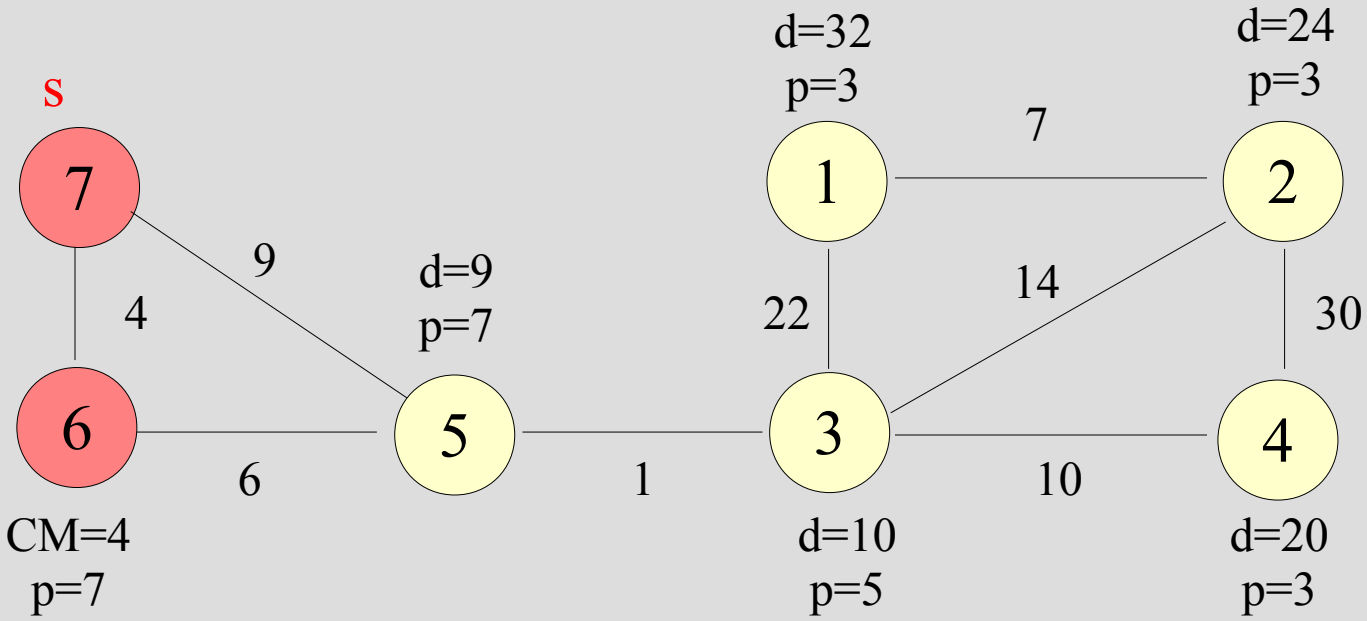
Passi dell'algoritmo Dijkstra

Es.
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4



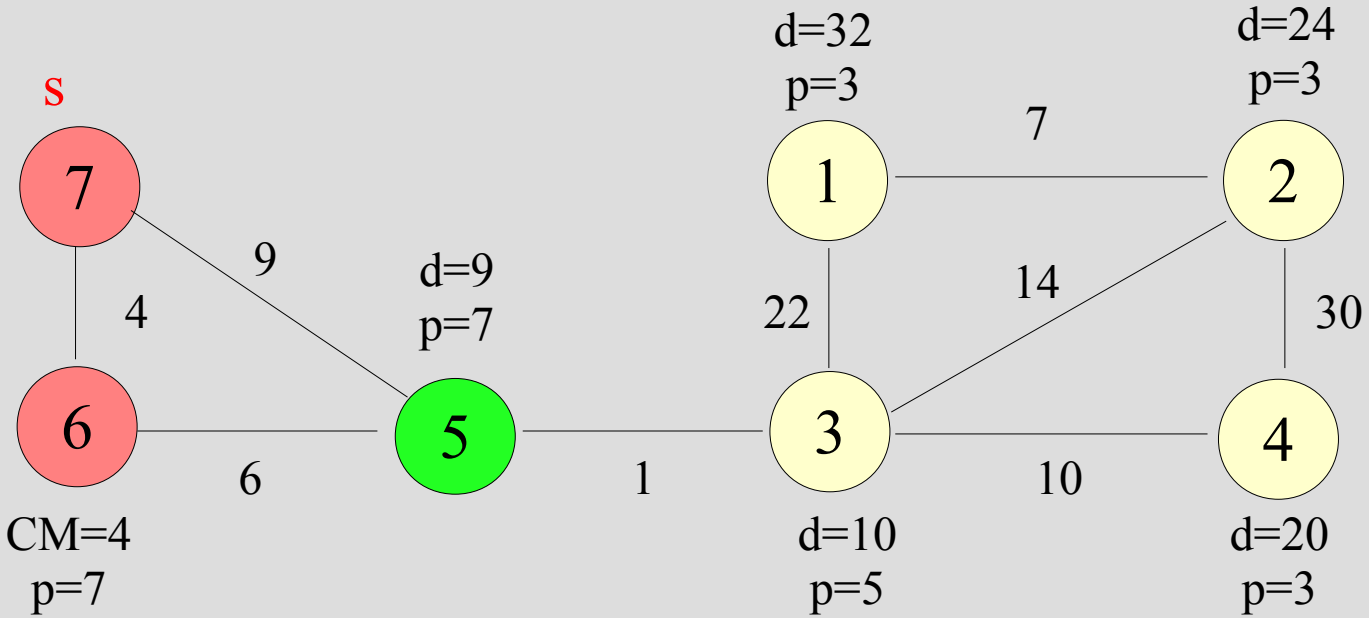
Passi dell'algoritmo Dijkstra

Es.
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4



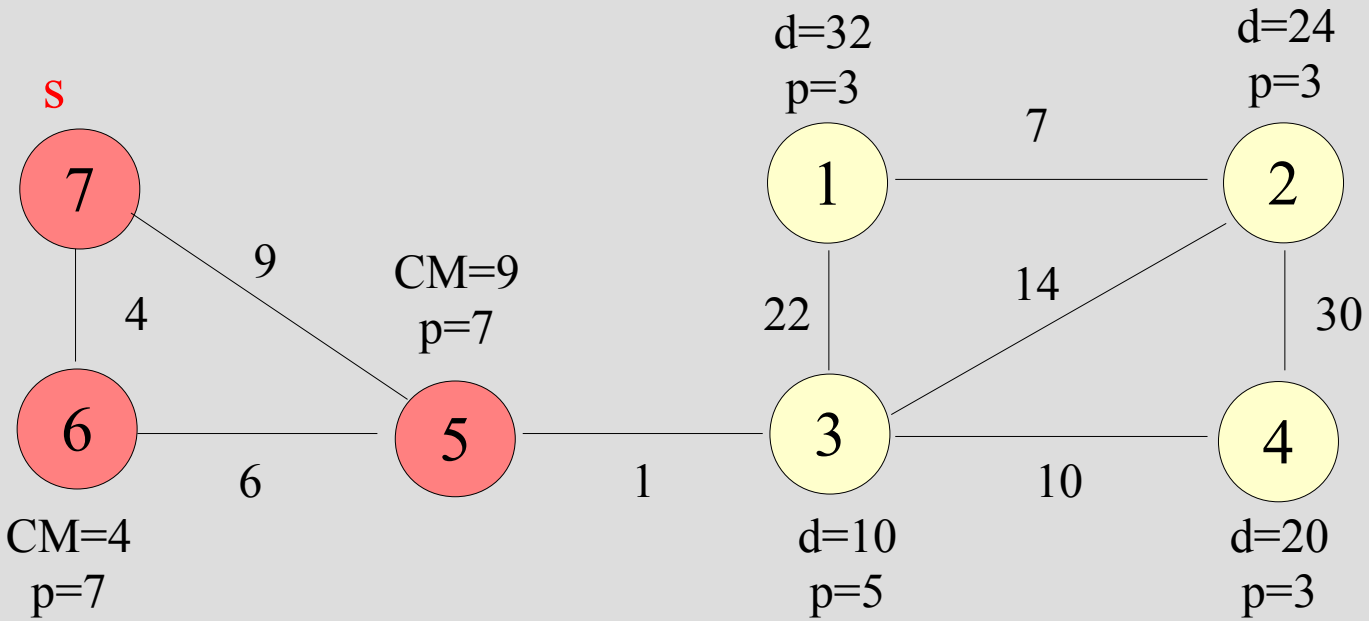
Passi dell'algoritmo Dijkstra

Es.
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4



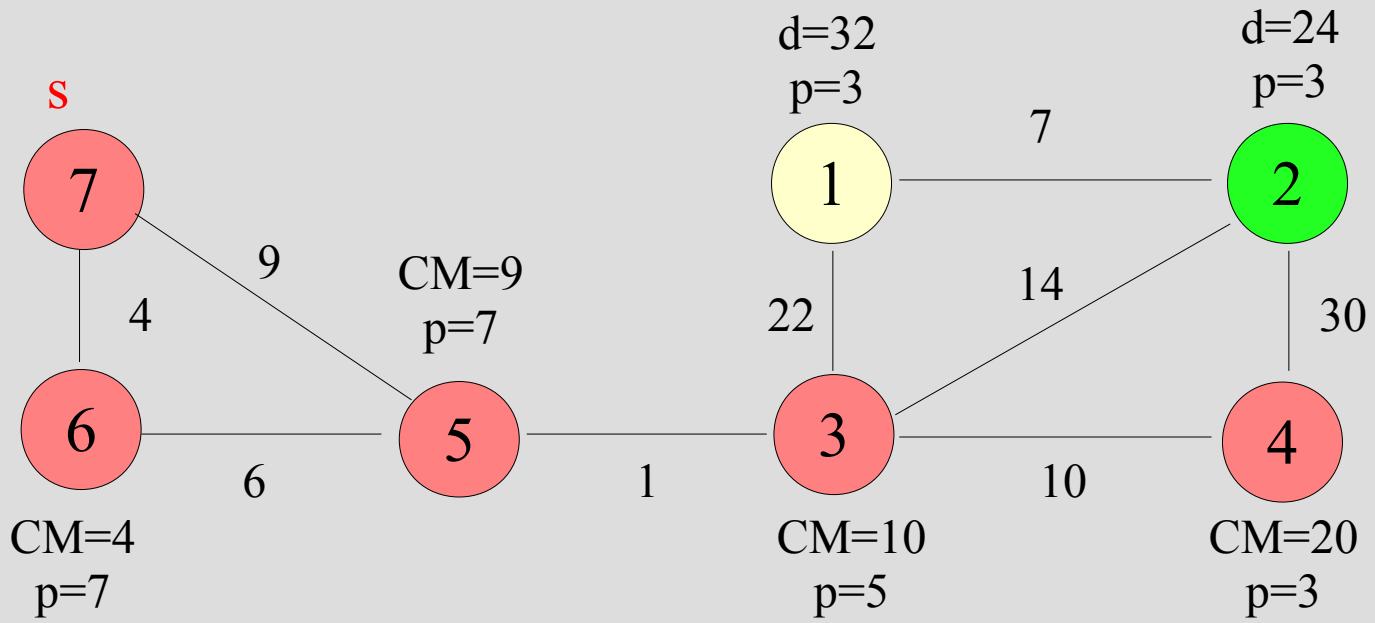
Passi dell'algoritmo Dijkstra

Es.
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4



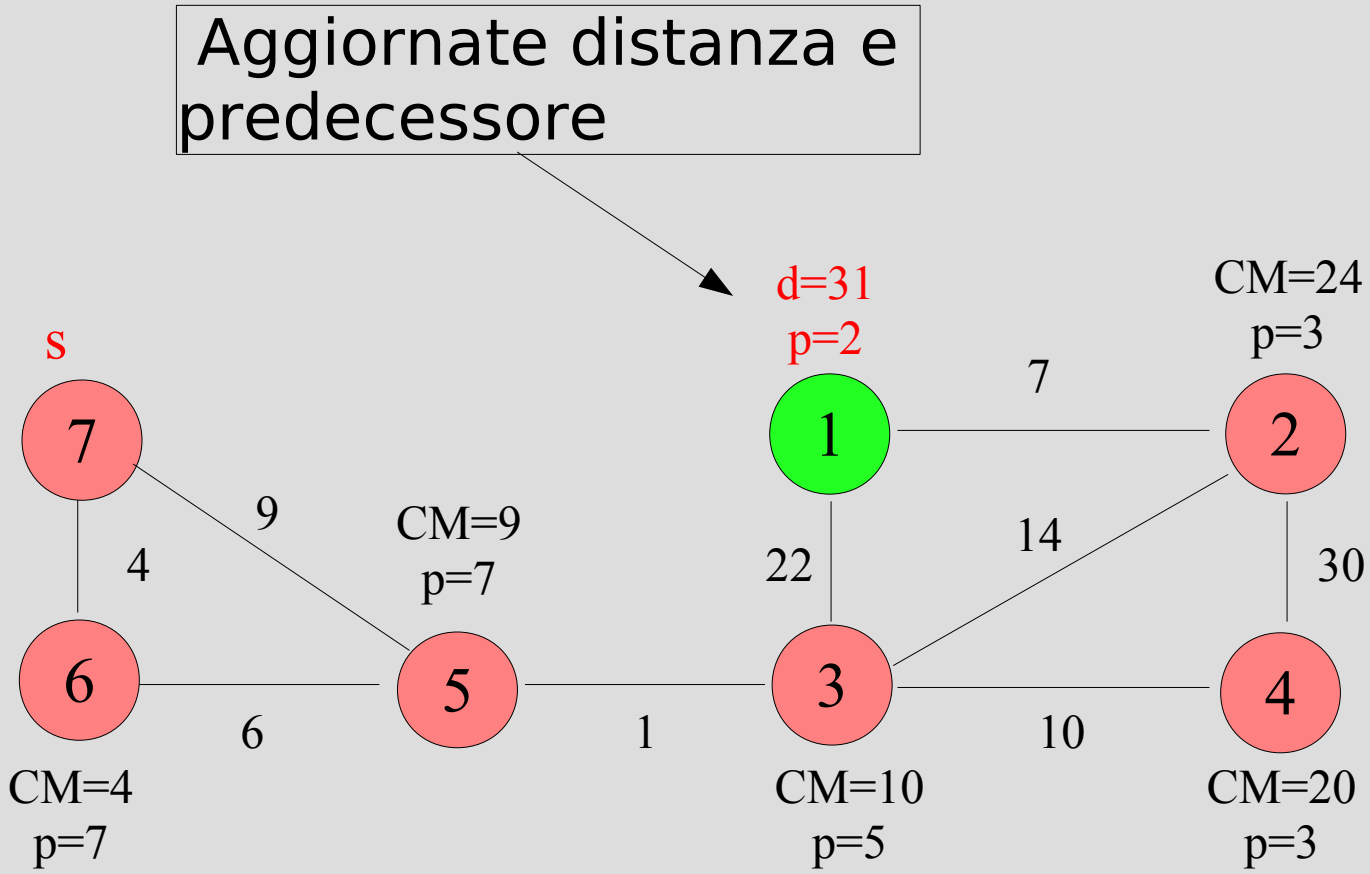
Passi dell'algoritmo Dijkstra

Es.
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4



Passi dell'algoritmo Dijkstra

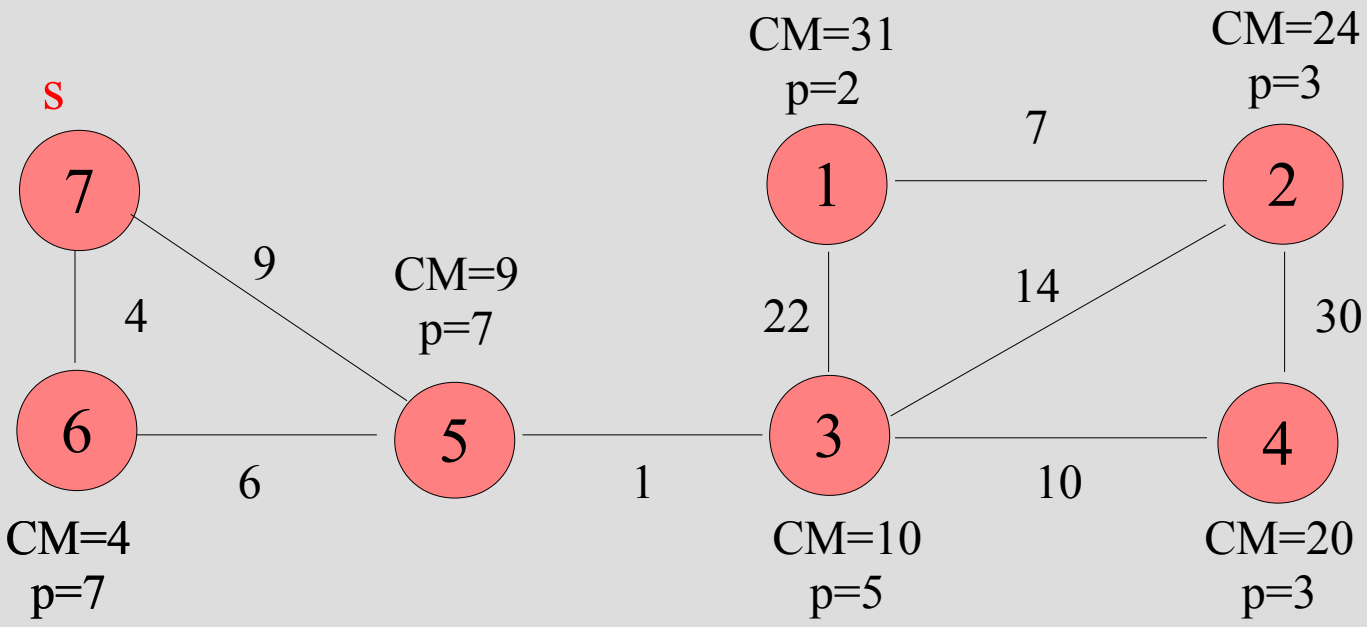
Es.
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4



Visione finale

Da file graph1.w

Es.
7
1 2 7
1 3 22
2 3 14
2 4 30
4 3 10
3 5 1
5 6 6
5 7 9
7 6 4



Suggerimenti implementativi

- L'insieme **S** e **V-S** possono essere realizzati attraverso **liste concatenate**
  - Come la coda che mantiene la '**frontiera**' nell'algoritmo di visita del grafo
- Serve una funzione per l'estrazione da **V-S** del vertice **u** con stima minore della distanza **d[u]** dalla sorgente **s**
  - **Ricerca** + **estrazione** da una lista
  - Necessità di **puntatori ausiliari** che tengano traccia del vertice col valore minimo!
    - Pensate alla ricerca del **min** in un array

Suggerimenti implementativi

- Per mantenere i cammini minimi per ogni vertice si può utilizzare un array **CM[u]**
- **Aggiornamento** dei cammini minimi dei vertici adiacenti ad **u** (*quelli ancora in **V-S**!*)
  - Serve una funzione che verifichi la presenza di un dato vertice nella lista → Ricerca
- Scansione della lista di adiacenza di **u**
  - Utilizzo delle funzioni **first_neighbor** e **next_neighbor** come durante la visita in ampiezza del grafo
  - Eventuale aggiornamento della distanza

Programma

- *dijkstra.cc*
- Programma che, partendo dalla rappresentazione attraverso liste di adiacenza di un grafo letto da file, calcola i cammini minimi di ogni vertice da una sorgente **s** (inserita a run-time) usando l'algoritmo di **Dijkstra**
- Per il calcolo dei cammini minimi si sfruttino le informazioni ottenute durante la visita in ampiezza del grafo

Soluzione parziale

Suggerimento

- Iniziare a implementare una soluzione parziale che includa soltanto le funzioni:
  - int extract_min(adj_node** q, float* d)

    //Estrae dalla coda q il vertice u col valore
    //minore di d[u] e torna 0 se la coda è vuota
  - float extract_weight(adj_node* gu, int v)

    //Estrae e ritorna il peso dell'arco tra il nodo u e
    // il nodo v

Esempio

Da file graph2.w