

Parte 5 - Grafi

Implementazione dell'algoritmo di Dijkstra e dell'algoritmo di Prim

Algoritmo di Dijkstra

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

DEFINIZIONE

Dato un grafo pesato $G=(V,E)$ e un vertice $s \in V$ l'algoritmo di Dijkstra restituisce *l'albero dei cammini minimi radicato in s*


Per comodità, l'albero dei cammini minimi è rappresentato come **vettore dei padri**

Tecnica del rilassamento

- Dijkstra e diversi altri algoritmi per la ricerca dei cammini minimi usano la tecnica del rilassamento
- Per ogni vertice $v \in V$ viene mantenuta la **stima $\text{dist}(v)$ del cammino minimo** ovvero un valore che rappresenta il limite superiore per il peso di un cammino minimo dalla sorgente s al vertice v
- Il **processo di rilassamento** di un arco $(u, v) \in E$ consiste nel verificare se, passando da u , è possibile migliorare il cammino per v precedentemente trovato e nel caso aggiornare *dist*

L'algoritmo di Dijkstra

DIJKSTRA(grafo $G=(V,E)$, pesi w , sorgente s)

1. INIZIALIZE (G,s)
2. $S = \emptyset$ 
3. $Q = V$
4. while $Q \neq \emptyset$
5. $u = \text{extract-min}(Q)$
6. $S = S \cup \{u\}$
7. for each vertex v of $\text{Adj}[u]$
8. RELAX(u,v)

Insieme S di vertici i cui pesi finali dei cammini minimi dalla sorgente s sono già stati determinati

Passo 5: extract-min seleziona ripetutamente il vertice $u \in V-S$ con la stima $d(u)$ minima

Passo 6: aggiunge u a S

Passo 7: rilassa tutti gli archi che escono da u

L'algoritmo di Dijkstra

INIZIALIZE(G,s)

1. for each $v \in V$
2. $\text{dest}(v) = \infty$
3. $\text{parent}[v] = \text{NULL}$
4. $\text{dest}[s] = 0$

RELAX(u,v,w)

1. If $\text{dest}[v] > \text{dest}[u] + w(u,v)$
2. $\text{dest}[v] = \text{dest}[u] + w(u,v)$
3. $\text{Decrease_Priority}(Q, v, \text{dest}[v])$
4. $\text{parent}[v] = u$

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

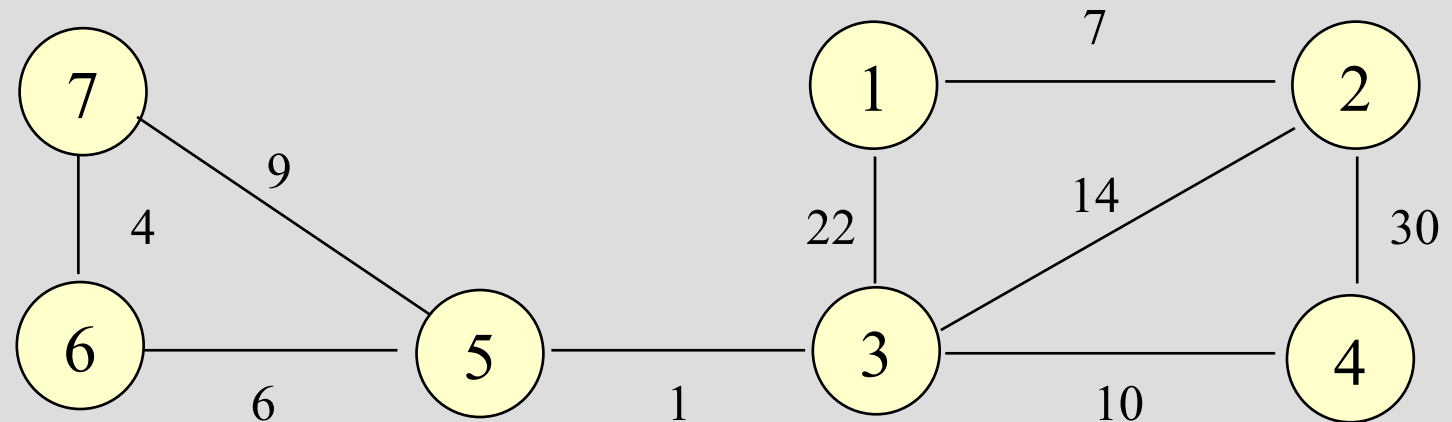
4 3 10

3 5 1

5 6 6

5 7 9

7 6 4



Nota: interpretato come
non orientato

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

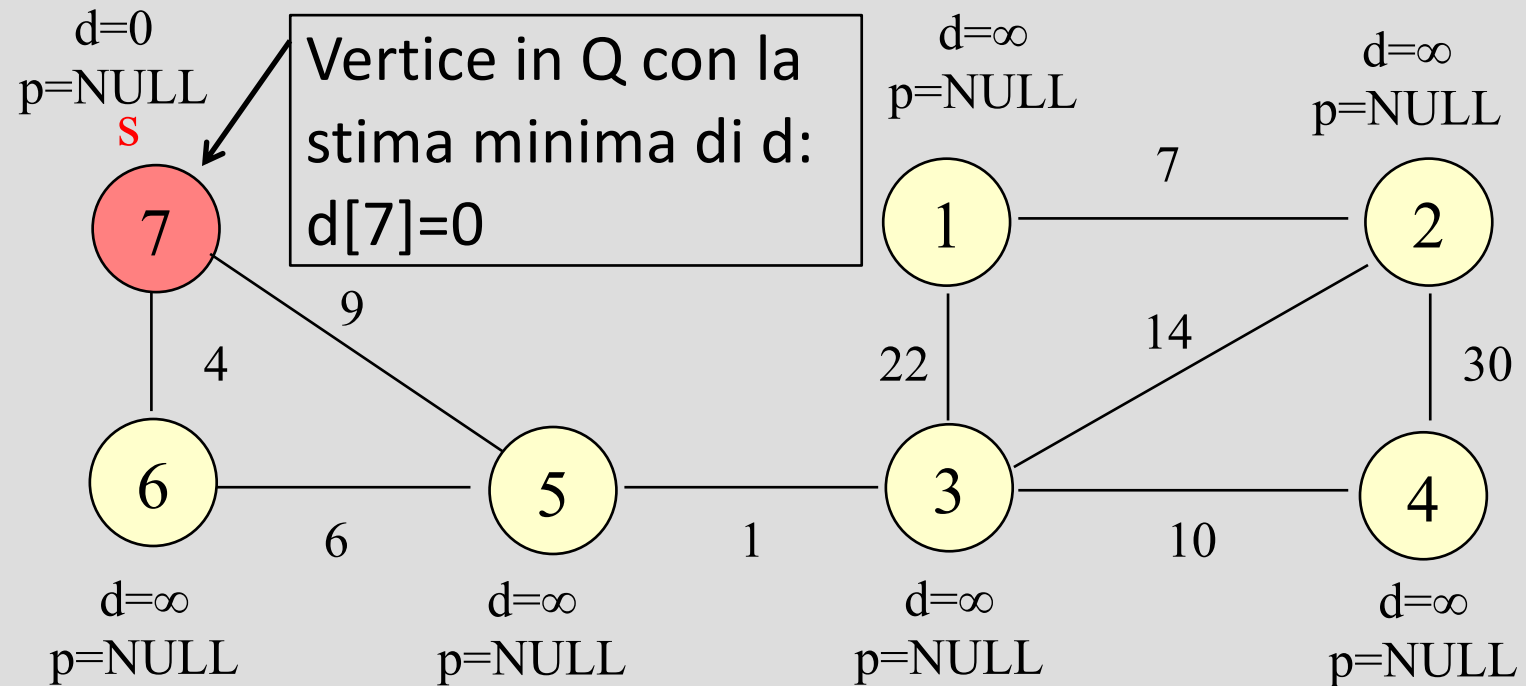
5 7 9

7 6 4

Sorgente $s=7$

$S=\{\}$

$Q=\{1,2,3,4,5,6,7\}$



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

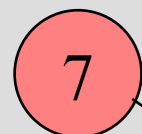
7 6 4

$S=\{7\}$

$Q=\{1,2,3,4,5,6\}$

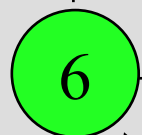
CM=0
p=NULL

S



9

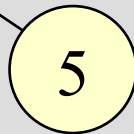
4



d=4
p=7

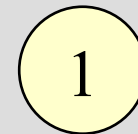
6

d=9
p=7

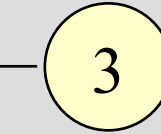


1

d=∞
p=NULL

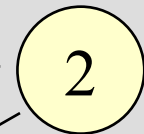


22



d=∞
p=NULL

d=∞
p=NULL



14

30



d=∞
p=NULL

Vertice in Q con la
stima minima di d: d[6]=4

Passi dell'algoritmo Dijkstra

$S=\{7,6\}$
 $Q=\{1,2,3,4,5\}$

Es.

7

1 2 7

1 3 22

2 3 14

2 4 30

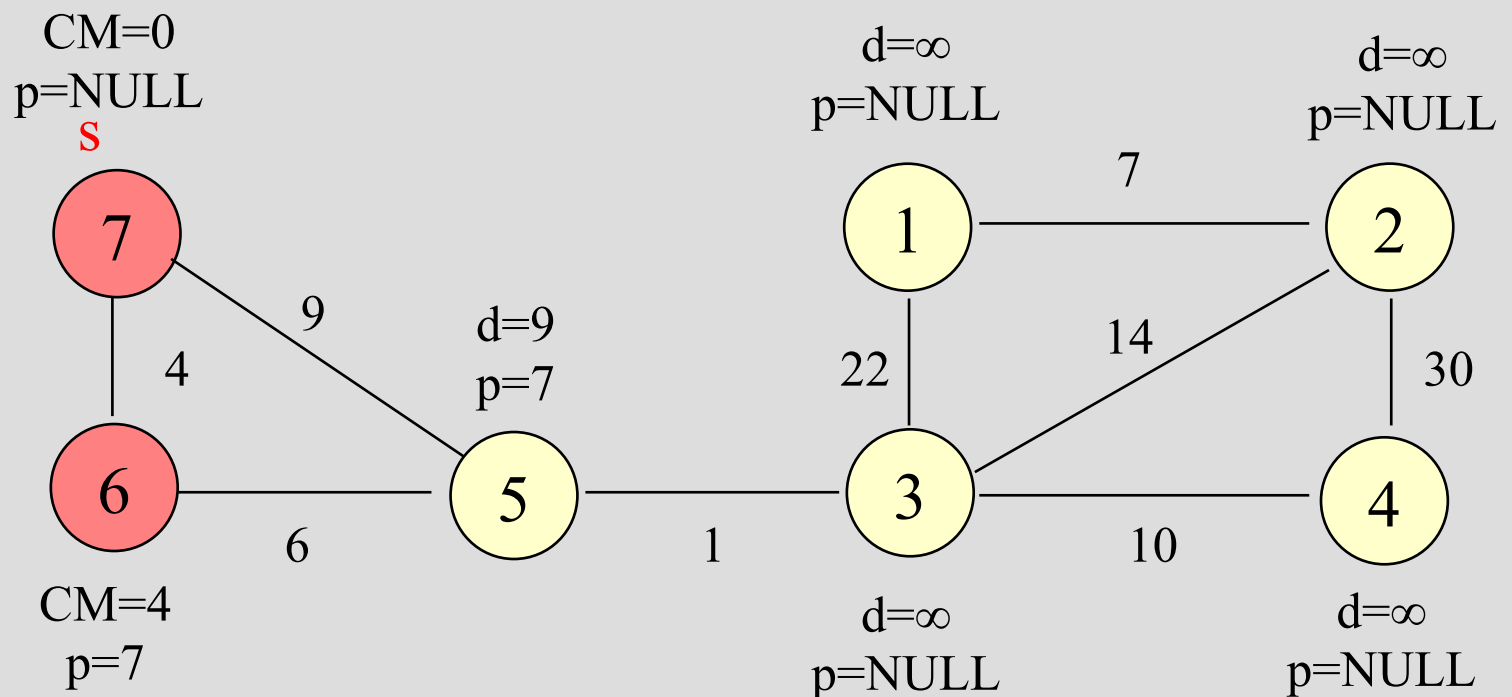
4 3 10

3 5 1

5 6 6

5 7 9

7 6 4



Passi dell'algoritmo Dijkstra

$S=\{7,6\}$
 $Q=\{1,2,3,4,5\}$

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

CM=0
p=NULL

7

6

CM=4
p=7

9

4

6

d=9
p=7

5

1

d=∞
p=NULL

1

22

d=∞
p=NULL

3

d=∞
p=NULL

2

30

d=∞
p=NULL

4

7

14

10

Passi dell'algoritmo Dijkstra

$S=\{7,6,5\}$
 $Q=\{1,2,3,4\}$

Es.

7

1 2 7

1 3 22

2 3 14

2 4 30

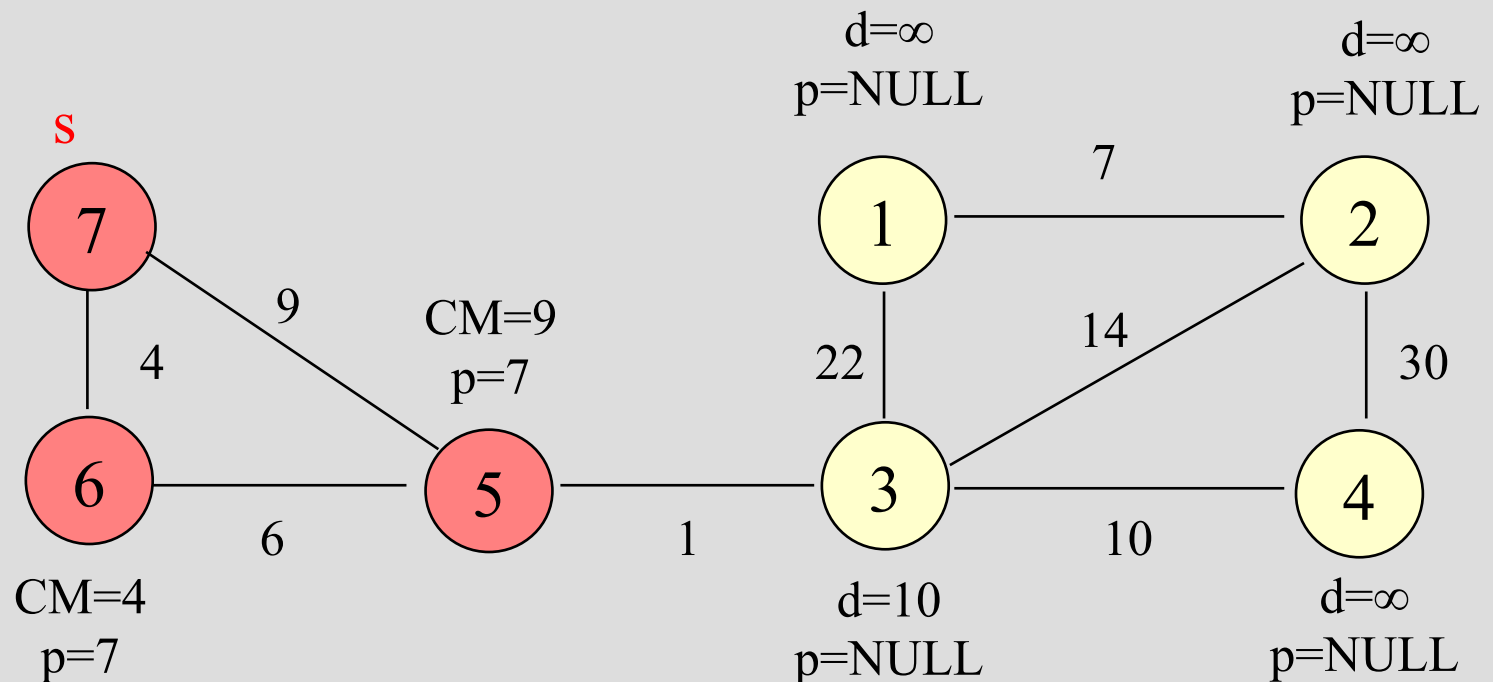
4 3 10

3 5 1

5 6 6

5 7 9

7 6 4



Passi dell'algoritmo Dijkstra

$S=\{7,6,5,3,4\}$
 $Q=\{1,2\}$

Es.

7

1 2 7

1 3 22

2 3 14

2 4 30

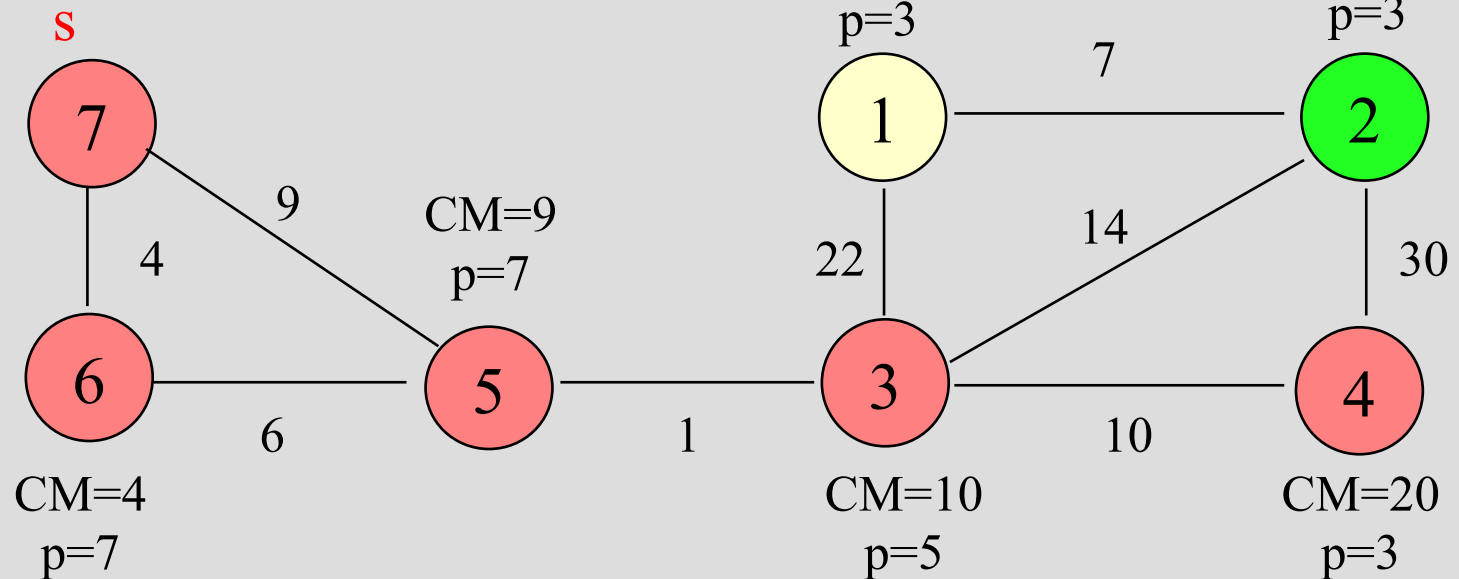
4 3 10

3 5 1

5 6 6

5 7 9

7 6 4



Passi dell'algoritmo Dijkstra

$S=\{7,6,5,3,4\}$
 $Q=\{1,2\}$

Es.

7

1 2 7

1 3 22

2 3 14

2 4 30

4 3 10

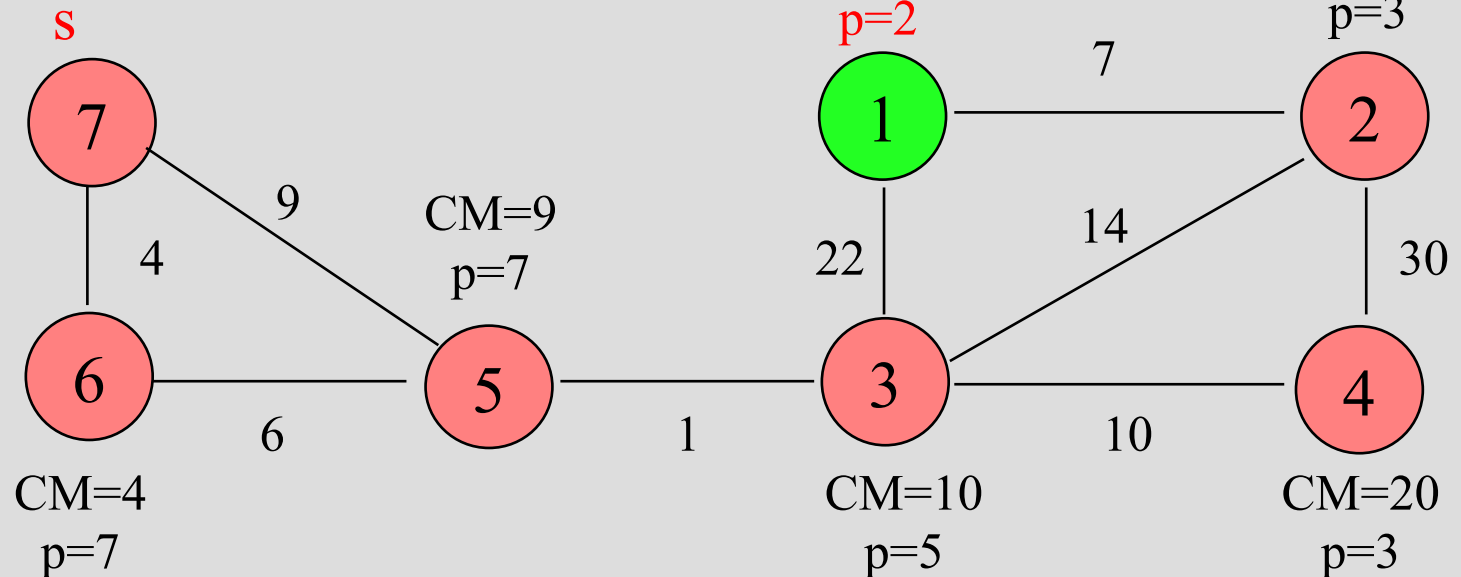
3 5 1

5 6 6

5 7 9

7 6 4

Aggiornate distanza e
predecessore



Visione finale

Da file graph1.w

Es.

7

1 2 7

1 3 22

2 3 14

2 4 30

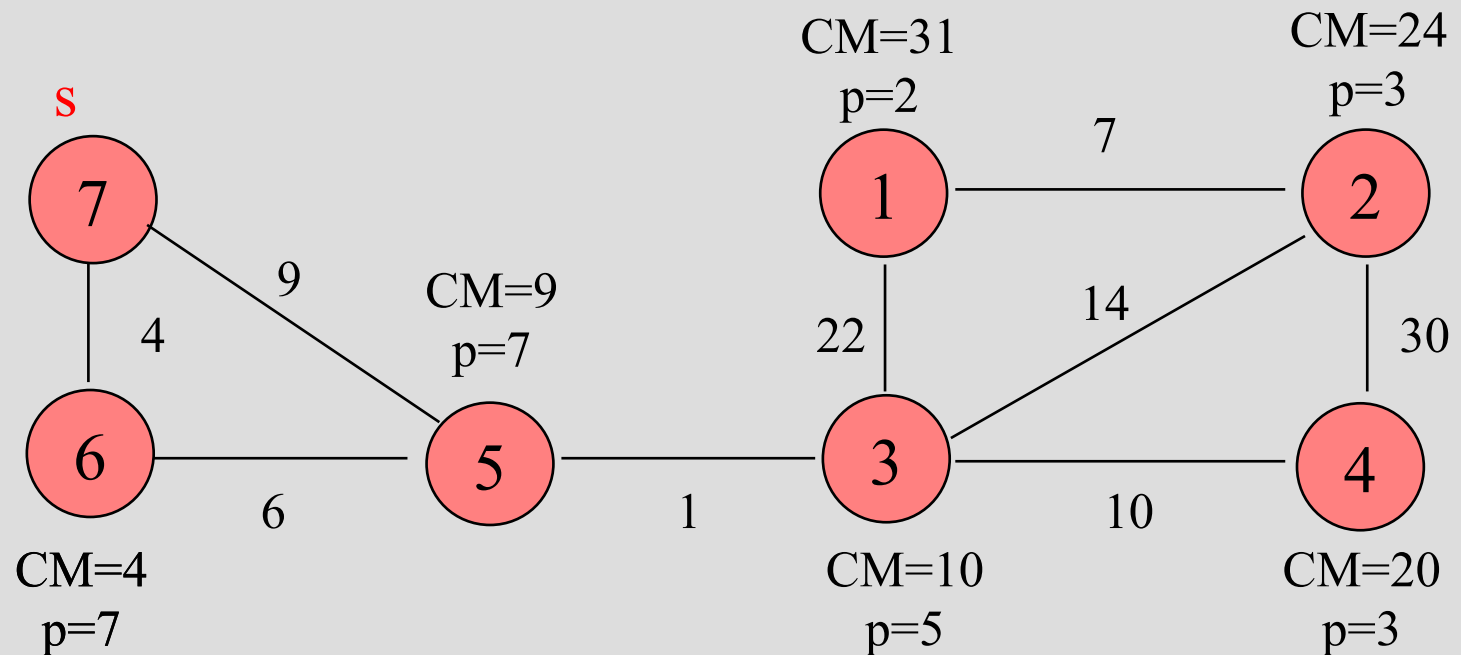
4 3 10

3 5 1

5 6 6

5 7 9

7 6 4



Strutture dati

Come rappresentare le strutture dati dell'algoritmo?

- Il *grafo* può rappresentato con liste di adiacenze:
 - $\text{Adj}[v]$ è la lista di adiacenza del vertice v
 - $w(u,v)$ è il peso associato a $v \in \text{Adj}[u]$

USIAMO il modulo GRAFO

- `dest` è un vettore dinamico della dimensione di V che contiene la stima del cammino minimo
- `parent` è un vettore dinamico che rappresenta il vettore dei padri
- Q è una **coda con priorità** dove ogni vertice $v \in V$ ha associato la *stima del cammino minimo* $\text{dest}(v)$

INTRODUCIAMO il modulo CODAP

- L'insieme S è il complementare di Q rispetto a V e non lo useremo nell'implementazione

Il modulo CODAP

Vedi file *codap.h* e *codap.cc*

Implementa la coda con priorità come *lista ordinata* dove gli elementi vengono mantenuti in ordine crescente per peso w

```
struct elem {  
    int inf;  
    float w;  
    elem* pun ; };  
  
typedef elem* codap;
```


Primitive del modulo CODAP

`codap enqueue(codap c, int i, int w)`: Inserisce l'elemento nella lista ordinata *c* sulla base del peso *w*. L'inserimento mantiene la proprietà di ordinamento.

`int dequeue(codap& c)`: Restituisce ed elimina l'elemento dal peso minimo *ovvero* l'elemento che si trova in testa alla lista

`int minQueue(codap c)`: Restituisce l'elemento dal peso minimo *ovvero* l'elemento che si trova in testa alla lista.

`codap Decrease_Priority(codap c, int i, float d)`: Cerca nella lista ordinata l'elemento *i* (se esiste) aggiorna il peso a *d* (con $d < \text{peso attuale di } i$). Questa operazione può richiedere il *riposizionamento* dell'elemento nella lista ordinate perché il suo peso è cambiato.

`bool isEmpty(codap c)`: Verifica se la coda è vuota

Implementazione dell'algoritmo

- Per implementare l'algoritmo di Dijkstra il progetto deve includere:
 - il modulo GRAFO
 - il modulo CODAP
- Per inizializzare le stime possiamo usare il valore costante **FLT_MAX** (libreria `float.h`)

ESERCIZIO Creare un nuovo progetto per l'implementazione dell'algoritmo di Dijkstra a partire dal file *graph_sol.cc*

SOLUZIONE Vedi sorgente *graph_dijkstra.cc*

Algoritmo di Prim

L'algoritmo di Prim risolve il problema della ricerca di un **albero di copertura minimo** (**minimum spanning tree**) in un grafo non orientato

DEFINIZIONE

Dato un grafo pesato $G=(V,E)$ e un vertice $v \in V$ di partenza, l'albero di copertura minimo è l'albero di copertura con la somma dei pesi minima:

$$w(MST) = \min_{MST \in S} \sum_{(u,v) \in MST} w(u,v).$$

Dove S è l'insieme di tutti gli alberi di copertura di G

Prim e Dijkstra

L'algoritmo di Prim *opera in modo molto simile* all'algoritmo di Dijkstra:

- L'algoritmo è definito su grafi non orientati
- L'algoritmo parte da un vertice
- L'algoritmo mantiene per ogni vertice v il peso minimo, $key(v)$, di un arco qualsiasi che collega v ad un vertice nell'albero
- L'output dell'algoritmo è un albero, l'albero di connessione minimo, rappresentato come vettore dei padri

Pseudo-codice dell'algoritmo

Prim(grafo $G=(V,E)$, pesi w , radice r)

1. INIZIALIZE (G,s)
2. $Q = V$
3. while $Q \neq \emptyset$
4. $u = \text{extract-min}(Q)$
5. for each vertex v of $\text{Adj}[u]$
6. RELAX(u,v,w)

INIZIALIZE(G,r)

1. for each $v \in V$
2. $\text{key}(v) = \infty$
3. $\text{parent}[v] = \text{NULL}$
4. $\text{key}[r] = 0$

RELAX(u,v,w)

1. If $\text{key}[v] > w(u,v)$
2. $\text{key}[v] = w(u,v)$
3. $\text{parent}[v] = u$
4. Decrease_Priority($Q,v,\text{key}[v]$)

Esercizio non risolto

Implementare l'algoritmo di Prim come variante dell'algoritmo di Dijkstra