Cammini minimi

Roberto Baldacci

DEI, Università di Bologna

Ricerca Operativa - Modulo II

Outline

Cammini minimi

- Sia $G = (V, A)$ un grafo orientato con $n = |V|$ vertici e $m = |A|$ archi. Sia c_{ij} il costo associato ad ogni arco $(i, j) \in A$.
- Il costo di un cammino da $s \in V$ a $t \in V$ è pari alla somma dei costi degli archi che lo compongono.
- Il cammino di costo minimo (*cammino minimo*) da s a t è quello che, fra tutti i cammini da s a t, ha il costo più piccolo.
- Se $c_{ij} \geq 0$, $\forall (i, j) \in A$, il cammino minimo è elementare.
- Trasformazione: lati in archi. Se G è *non orientato* o *misto* (archi e lati), può essere trasformato in un grafo orientato sostituendo ogni lato $\{i, j\}$ con costo c_{ij} in due archi (i, j) e (j, i) entrambi con costo c_{ij}.

Cammini di costo minimo: complessità

- Se alcuni dei costi c_{ij} sono negativi allora il grafo G può contenere circuiti di costo negativo;
- In questo caso il circuito di costo negativo può essere usato un numero infinito di volte per minimizzare il costo;
- Bisogna imporre la restrizione che il cammino passi attraverso ciascun vertice al massimo una sola volta (*cammino elementare*):
  $\Rightarrow$ problema è NP-difficile.
- Esistono tuttavia casi particolari che possono essere risolti in tempo polinomiale, fra i quali: grafi senza circuiti di costo negativo, grafi aciclici e grafi con costi positivi (algoritmo di Dijkstra).

Formulazione matematica (1)

- Per ogni arco $(i,j) \in A$ si consideri la variabile decisionale:

$$x_{ij} = \begin{cases} 1 & \text{se } (i,j) \text{ viene scelto nel cammino;} \\ 0 & \text{altrimenti.} \end{cases}$$

- Per ogni $S \subseteq V$, sia $A(S)$ l'insieme degli archi con entrambi gli estremi in S, $A(S) = \{(i,j) \in A : i \in S, j \in S\}$.

Formulazione matematica (2)

- Supponendo $s \neq t$, il problema del cammino minimo semplice da s a t può essere formulato come:

$$Min \quad \underbrace{\sum_{(i,j)\in A} c_{ij}x_{ij}}_{\text{costo cammino}}$$

$$\underbrace{\sum_{(h,j)\in\delta^+(h)} x_{hj}}_{\text{n. archi uscenti}} - \underbrace{\sum_{(i,h)\in\delta^-(h)} x_{ih}}_{\text{n. archi entranti}} = \begin{cases} 1 & \text{se } h = s \\ -1 & \text{se } h = t \\ 0 & \forall h \in V \setminus \{s,t\} \end{cases}$$

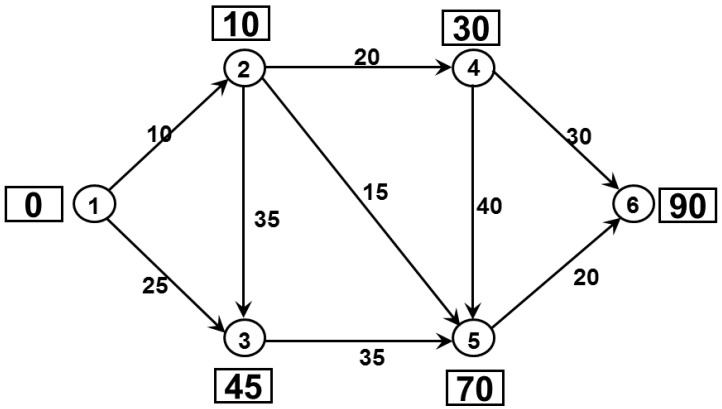
$$\underbrace{\sum_{(i,j)\in A(S)} x_{ij}}_{\text{n. archi in } S} \leq |S| - 1, \quad \forall S \subseteq V, S \neq \emptyset \qquad (*)$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A$$

- I $2^n - 1$ vincoli $(*)$ impediscono il formarsi di *subtour* (vincoli di *subtour elimination*).

Distance Label

- La maggior parte degli algoritmi per calcolare i cammini minimi impiegano il vettore delle distance label. Per ogni vertice è definita una label $L(i)$ (indicate anche con $d(i)$).
- La distance label $L(i)$ rappresenta il costo di un qualche cammino diretto dal vertice sorgente s al nodo i.

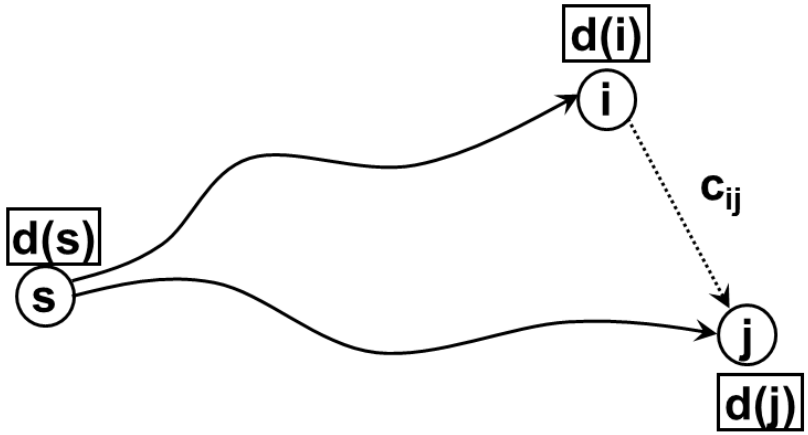


- Ad una generica iterazione dell'algoritmo, le distance label sono un upper bound al costo del cammino minimo dal vertice sorgente s al nodo i. Al termine, sono il costo del cammino minimo.

Condizioni di Ottimalità

Teorema. Sia $L(j)$, $\forall j \in V$, la lunghezza di un cammino dal nodo s a j. Allora, le distanze $L(j)$ rappresentano le distanze minime se e solo se soddisfano le seguenti condizioni:

$$L(j) \leq L(i) + c_{ij} \quad \text{per ogni arco } (i,j) \in A.$$

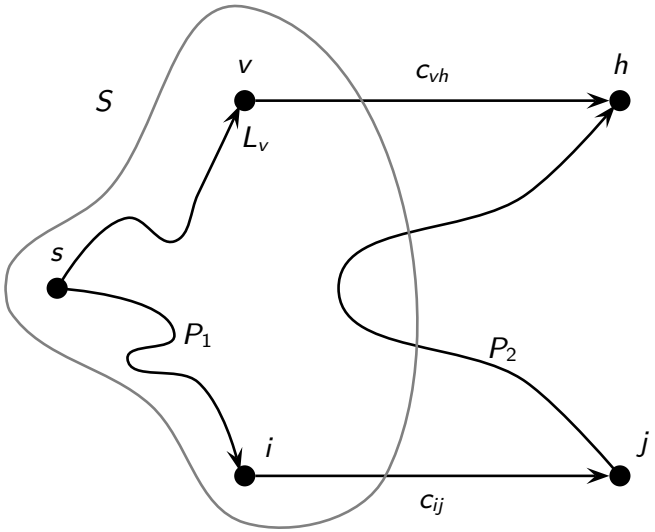


- Due possibili approcci: label setting and label correcting.
- Label setting: calcola una valore $L(j)$, $j \in V$, *ottimo* ad ogni iterazione.
- Label correcting: i valori ottimi $L(j)$, $\forall j \in V$, sono definiti sono al termine dell'algoritmo.

Algoritmo di Dijkstra (1)

- Algoritmo di tipo *label setting*.
- L'algoritmo di Dijkstra permette di calcolare i cammini minimi da $s \in V$ a ogni $t \in V$ nel caso in cui i costi degli archi sono non negativi ($c_{ij} \geq 0$, $\forall (i,j) \in A$).
- **Teorema**.
  Sia L_i il costo del cammino minimo da s ad un vertice i, per ogni vertice $i \in S \subset V$ con $s \in S$.
  Sia inoltre $(v,h) = argmin\{L_i + c_{ij} : (i,j) \in \delta^+(S)\}$.
  Allora $L_v + c_{vh}$ rappresenta il costo del cammino minimo da s ad h.

Algoritmo di Dijkstra (2)



- **Dimostrazione**. $L_v + c_{vh}$ rappresenta il costo di un cammino da s ad h. Si consideri un altro cammino P che termina in h.
  Sia $(i,j) \in P \cap \delta^+(S)$ e si partizioni P in $P_1 \cup \{(i,j)\} \cup P_2$, dove P_1 e P_2 sono due cammini da s ad i e da j ad h, rispettivamente. Si ha

$$C(P) = \underbrace{c(P_1)}_{\geq L_i} + c_{ij} + \underbrace{C(P_2)}_{\geq 0} \geq L_i + c_{ij} \geq L_v + c_{vh}.$$

  Per cui $L_v + c_{vh}$ è il costo del cammino di costo minimo da s ad h.

Algoritmo di Dijkstra (3)

- Il teorema precedente suggerisce il seguente algoritmo iterativo per la determinazione dei cammini minimi da $s \in V$ ad ogni $t \in V$.

$S = \{s\}$;
$L[s] = 0$;
$pred[s] = s$;
while $(|S| \neq n)$ **do**
  if $(\delta^+(S) \neq \emptyset)$ **then**
    $(v, h) = argmin\{L[i] + c_{ij} : (i, j) \in \delta^+(S)\}$;
    $L[h] = L[v] + c_{vh}$;
    $pred[h] = v$;
    $S = S \cup \{h\}$;
  else
    Grafo G disconnesso;
  end if
end while

- La complessità dell'algoritmo è pari a $O(nm)$.

Miglioramento dell'algoritmo di Dijkstra

- E' possibile ottenere una complessità $O(n^2)$ se ad ogni iterazione si sfrutta opportunamente l'informazione già acquisita nelle iterazioni precedenti, in modo simile a quello visto per l'algoritmo di Prim-Dijkstra per il problema dello SST. L'algoritmo utilizza le seguenti strutture dati:
  - $flag[j] = \begin{cases} 1 & \text{se } j \in S \\ 0 & \text{altrimenti} \end{cases}$, per ogni $j \in V$;
  - $L[j] = \begin{cases} \text{costo del cammino minimo da } s \text{ a } j, & \text{se } j \in S; \\ min\{L[i] + c_{ij} : i \in S\}, & \text{se } j \notin S; \end{cases}$
  - $pred[j] = \begin{cases} \textit{predecessore} \text{ di } j \text{ nel cammino minimo da } s \text{ a } j, & \text{se } j \in S \\ argmin\{L[i] + c_{ij} : i \in S\}, & \text{se } j \notin S \end{cases}$

Miglioramento dell'algoritmo di Dijkstra

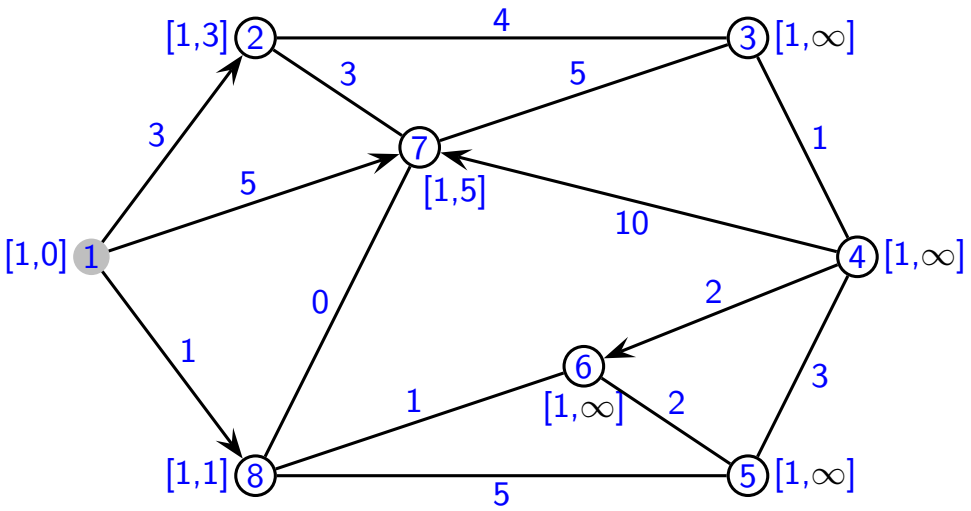
Require: Grafo orientato connesso con costi $\{c_{ij}\}$ non-negativi;
Ensure: Cammini minimi da s a $V \setminus \{s\}$ in $\{pred[j], j\}$, $j \in V \setminus \{s\}$;

```
{Inizializzazione}
for j = 1 to n do
    flag[j] = 0;
    pred[j] = s;
    L[j] = c_sj;
end for
flag[s] = 1; {Vertice sorgente s}
L[s] = 0;
for k = 1 to n − 1 do
    min = +∞;
    {Individua h = argmin{L[j] : j ∉ S}}
    for j = 1 to n do
        if (flag[j] = 0) and (L[j] < min) then
            min = L[j];
            h = j;
```

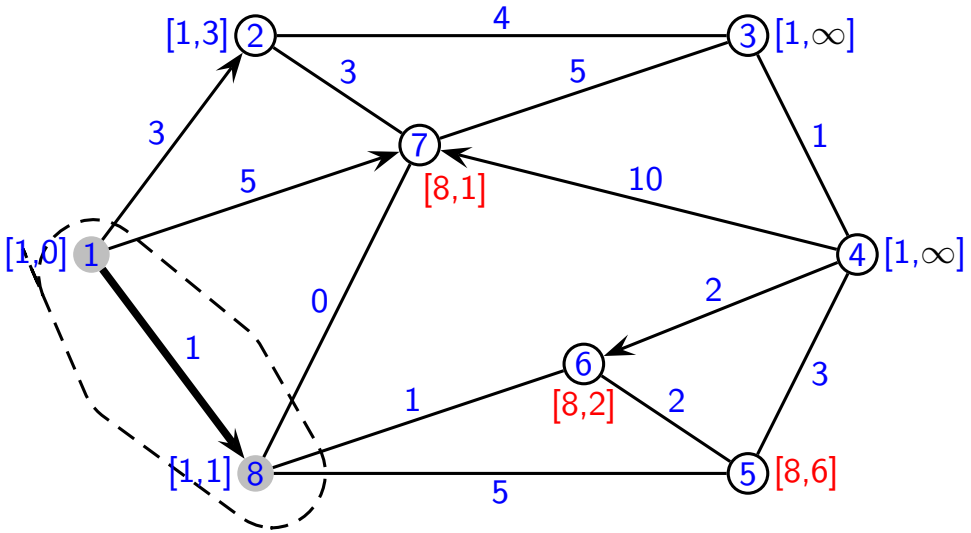
Miglioramento dell'algoritmo di Dijkstra (2)

```
    end if
  end for
  {Aggiorna S = S ∪ {h}}
  flag[h] = 1;
  {Aggiorna L[j] e pred[j] per ogni j ∉ S}
  for j = 1 to n do
    if (flag[j] = 0) and (L[h] + c_hj < L[j]) then
      L[j] = L[h] + c_hj;
      pred[j] = h;
    end if
  end for
end for
```

Esempio dell'algoritmo di Dijkstra

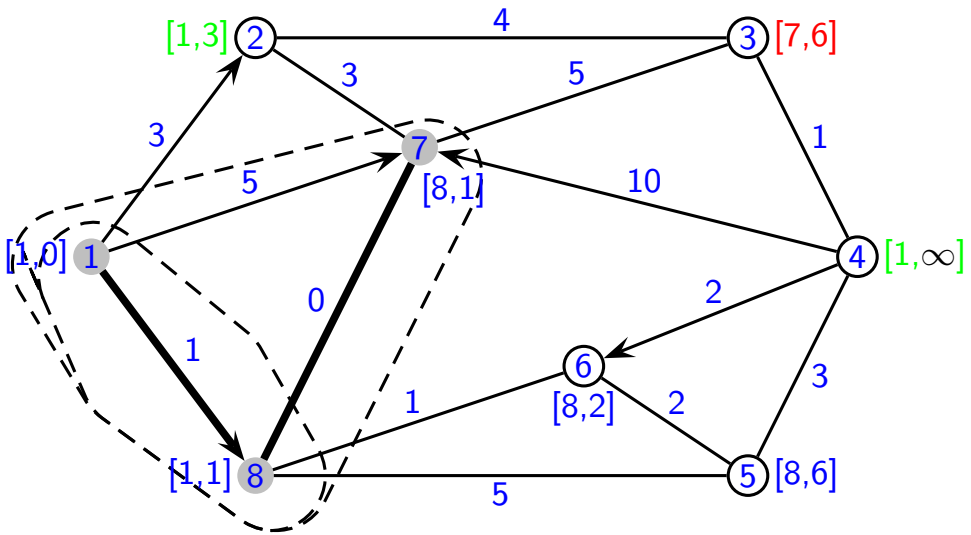


Inizializzazione, $s = 1$, etichette $[pred[j], L[j]]$ sui vertici.

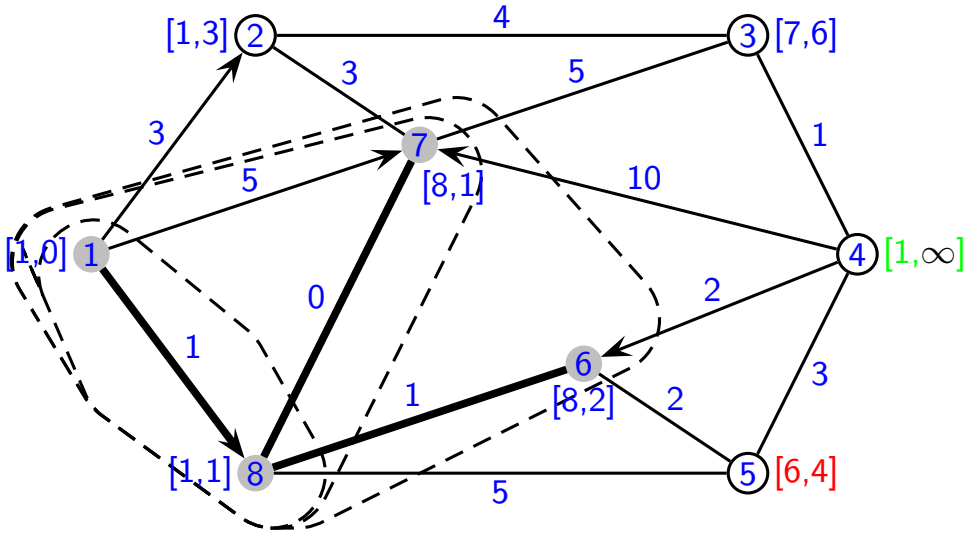


Vertice scelto 8.

Esempio dell'algoritmo di Dijkstra (2)

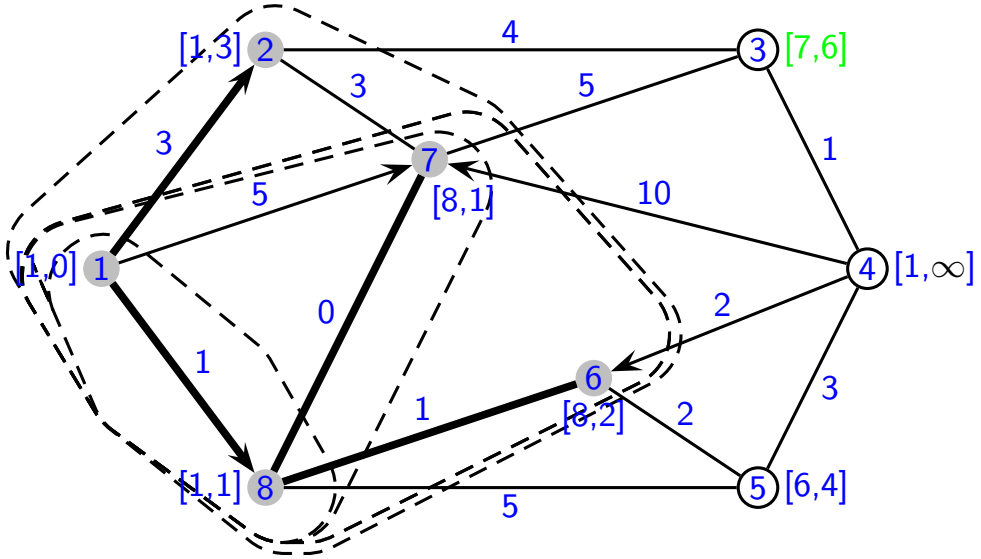


Vertice scelto 7.

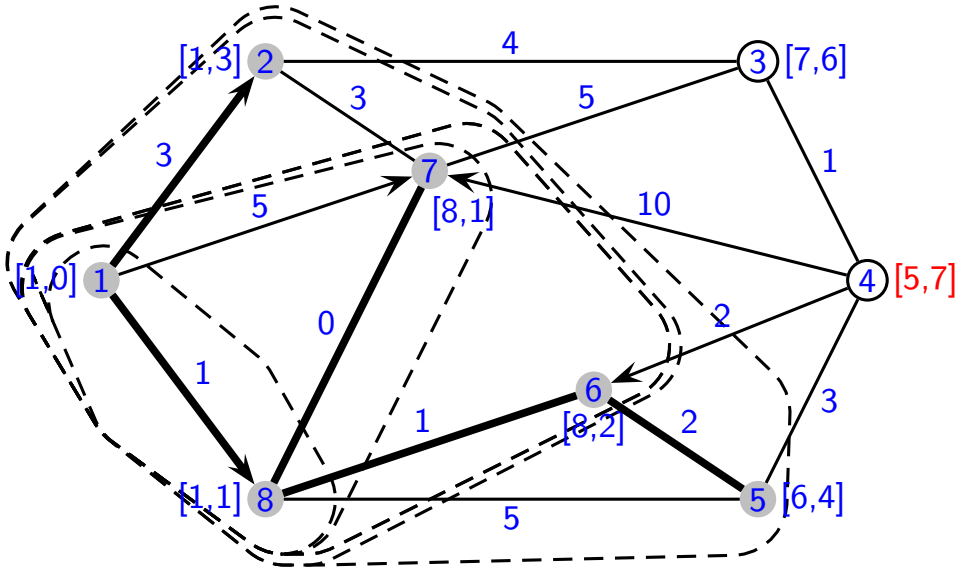


Vertice scelto 6.

Esempio dell'algoritmo di Dijkstra (3)

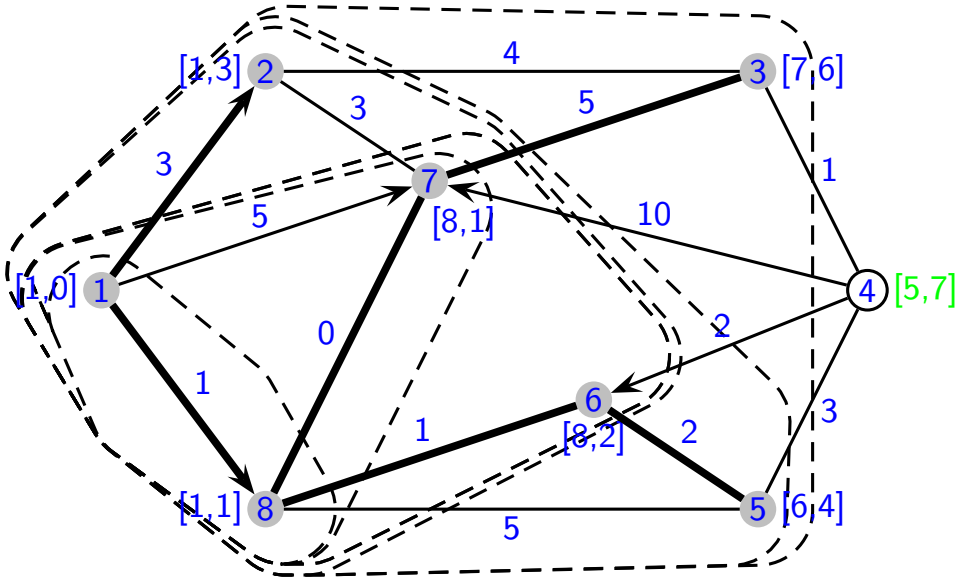


Vertice scelto 2.

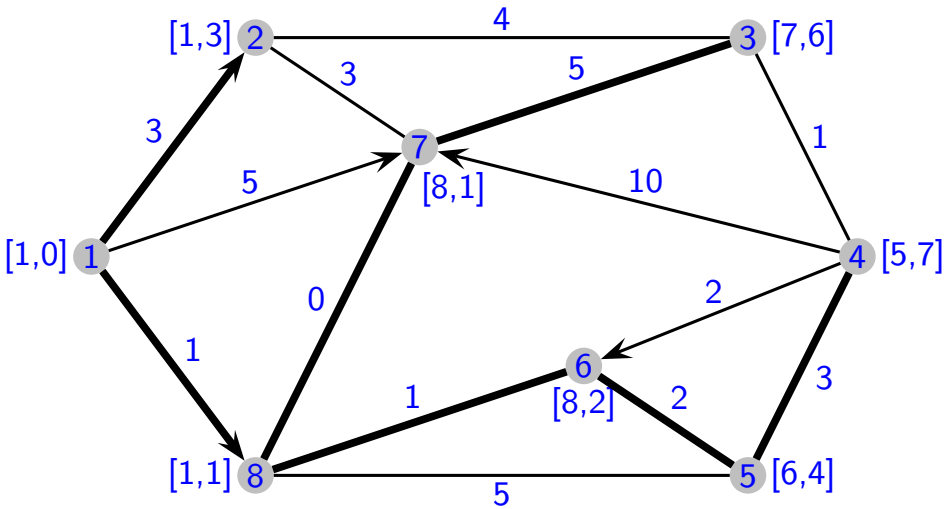


Vertice scelto 5.

Esempio dell'algoritmo di Dijkstra (4)

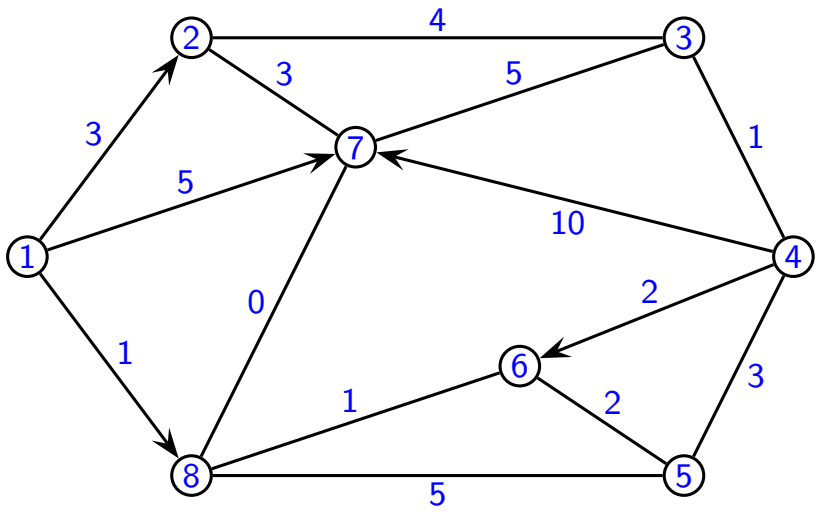


Vertice scelto 3.



Vertice scelto 4.

Formato tabellare dell'algoritmo di Dijkstra



$$[c_{ij}] = \begin{bmatrix} & 3 & & & & & 5 & 1 \\ & & 4 & & & & 3 & \\ & 4 & & 1 & & & 5 & \\ & & 1 & & 3 & 2 & 10 & \\ & & & 3 & & 2 & & 5 \\ & & & & 2 & & & 1 \\ & 3 & 5 & & & & & 0 \\ & & & & 5 & 1 & 0 & \end{bmatrix}$$

Formato tabellare dell'algoritmo di Dijkstra (2)

S	$L[j]$						
	2	3	4	5	6	7	8
$\{1\}$	3	∞	∞	∞	∞	5	**1**
$\{1,8\}$	3	∞	∞	6	2	**1**	1
$\{1,8,7\}$	3	6	∞	6	**2**	1	1
$\{1,8,7,6\}$	**3**	6	∞	4	2	1	1
$\{1,8,7,6,2\}$	3	6	∞	**4**	2	1	1
$\{1,8,7,6,2,5\}$	3	**6**	7	4	2	1	1
$\{1,8,7,6,2,5,3\}$	3	6	**7**	4	2	1	1
$\{1,8,7,6,2,5,3,4\}$	3	6	7	4	2	1	1

$pred[j]$						
2	3	4	5	6	7	8
1	-	-	-	-	1	1
1	-	-	8	8	8	1
1	7	-	8	8	8	1
1	7	-	6	8	8	1
1	7	-	6	8	8	1
1	7	5	6	8	8	1
1	7	5	6	8	8	1
1	7	5	6	8	8	1

$$[c_{ij}] = \begin{bmatrix} & 3 & & & & & 5 & 1 \\ & & 4 & & & & 3 & \\ & 4 & & 1 & & & 5 & \\ & & 1 & & 3 & 2 & 10 & \\ & & & 3 & & 2 & & 5 \\ & & & & 2 & & & 1 \\ & 3 & 5 & & & & & 0 \\ & & & & 5 & 1 & 0 & \end{bmatrix}$$

Cammini minimi fra tutte le coppie di vertici

- Possono essere calcolati eseguendo n volte l'algoritmo di Dijkstra (grafo con costi positivi) utilizzando come vertice iniziale s ogni vertice del grafo. La complessità dell'algoritmo risultante è $O(n^3)$.
- Un diverso metodo è quello dell'algoritmo di Floyd-Warshall:
  i) ha complessità $O(n^3)$;
  ii) si applica a grafi con costi qualunque ed è in grado di riconoscere circuiti di costo negativo.
- L'algoritmo si applica ad un grafo orientato definito dalla matrice $n \times n$ dei costi $[c_{ij}]$.

Cammini minimi fra tutte le coppie di vertici (2)

- Sia u_{ij} la lunghezza del cammino di costo minimo fra i e j.
- Sia u_{ij}^h la lunghezza del cammino di costo minimo fra i e j con la condizione aggiuntiva che il cammino non passi attraverso i nodi $h, h+1, \ldots, n$ (eccetto i e j).
- I valori $\{u_{ij}^h\}$ possono essere calcolati come segue:

$$\left.\begin{array}{l} u_{ij}^1 = c_{ij} \\ u_{ij}^{h+1} = \min\{u_{ij}^h, u_{ih}^h + u_{hj}^h\}, h = 1, \ldots, n \end{array}\right\}$$

- Si ha $u_{ij} = u_{ij}^{n+1}$.

Cammini minimi fra tutte le coppie di vertici (3)

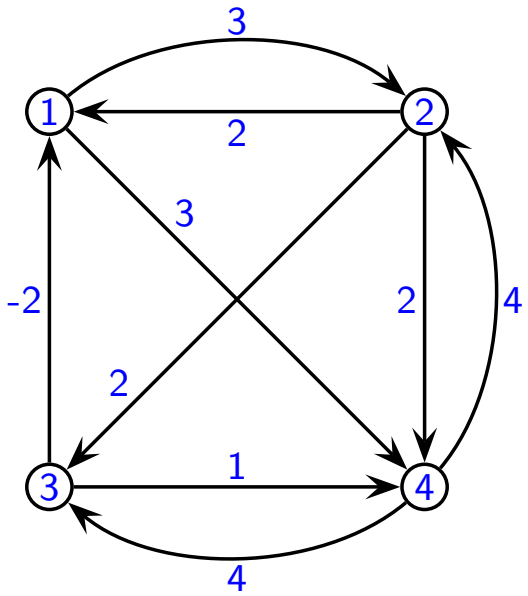
- L'implementazione dell'algoritmo di Floyd-Warshall richiede:
  - una matrice U di ordine $n \times n$ per memorizzare i costi dei cammini minimi;
  - una matrice $pred$ di ordine $n \times n$ per ricostruire i cammini minimi.
- Al termine dell'algoritmo, per ogni $i, j \in V$, u_{ij} rappresenta il costo del cammino minimo da i a j mentre $pred[i, j]$ rappresenta il predecessore di j nel cammino minimo da i a j.
- Se $u_{ii} < 0$ allora esiste un circuito negativo (ricostruibile a partire da $pred[i, i]$).

Algoritmo di Floyd-Warshall

Require: Grafo orientato definito dalla matrice dei costi $[c_{ij}]$;
Ensure: Matrici $[u_{ij}]$ e $[pred[i,j]]$;

```
{Inizializzazione}
for i = 1 to n do
    for j = 1 to n do
        u_ij = c_ij;
        pred[i,j] = i;
    end for
end for
{Operazione triangolare su h}
for h = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            if (u_ih + u_hj < u_ij) then
                u_ij = u_ih + u_hj;
                pred[i,j] = pred[h,j];
            end if
        end for
    end for
    for i = 1 to n do
        if (u_ii < 0) then
            STOP, circuiti negativi;
        end if
    end for
end for
```

Esempio



$$[c_{ij}] = \left[\begin{array}{c|c|c|c} 0 & 3 & \infty & 3 \\ \hline 2 & 0 & 2 & 2 \\ \hline -2 & \infty & 0 & 1 \\ \hline \infty & 4 & 4 & 0 \end{array}\right]$$

Inizializzazione

u_{ij}

0	3	∞	3
2	0	2	2
-2	∞	0	1
∞	4	4	0

$pred[i,j]$

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

Esempio (2)

Alla prima iterazione si ha $h = 1$ quindi

$$u_{ij} = \text{Min}\,\{u_{ij}, (u_{i1} + u_{1j})\}.$$

Ad esempio

$$u_{32} = \text{Min}\,\{u_{32}, (u_{31} + u_{12})\} = \text{Min}\,\{\infty, (-2 + 3)\} = 1.$$

u_{ij}

0	3	∞	3
2	0	2	2
-2	∞	0	1
∞	4	4	0

Matrice u_{ij} aggiornata

0	3	∞	3
2	0	2	2
-2	1	0	1
∞	4	4	0

$pred[i, j]$

1	1	1	1
2	2	2	2
3	1	3	3
4	4	4	4

Esempio (3)

Alla seconda iterazione si ha $h = 2$ quindi

$$u_{ij} = \text{Min}\,\{u_{ij}, (u_{i2} + u_{2j})\}.$$

Ad esempio

$$u_{13} = \text{Min}\,\{u_{13}, (u_{12} + u_{23})\} = \text{Min}\,\{\infty, (3 + 2)\} = 5.$$

u_{ij}

0	3	∞	3
2	0	2	2
-2	1	0	1
∞	4	4	0

Matrice u_{ij} aggiornata

0	3	5	3
2	0	2	2
-2	1	0	1
6	4	4	0

$pred[i, j]$

1	1	2	1
2	2	2	2
3	1	3	3
2	4	4	4

Esempio (4)

Alla terza iterazione si ha $h = 3$ quindi

u_{ij}

0	3	5	3
2	0	2	2
-2	1	0	1
6	4	4	0

Matrice u_{ij} aggiornata

0	3	5	3
0	0	2	2
-2	1	0	1
2	4	4	0

$pred[i, j]$

1	1	2	1
3	2	2	2
3	1	3	3
3	4	4	4

Esempio (5)

Alla quarta ed ultima iterazione si ha $h = 4$ quindi

u_{ij}

0	3	5	3
0	0	2	2
-2	1	0	1
2	4	4	0

$pred[i, j]$

1	1	2	1
3	2	2	2
3	1	3	3
3	4	4	4

Esempio: il costo del cammino minimo dal vertice 1 al vertice 3 è pari a $u_{13} = 5$.
Il cammino minimo $P = (1, 2, 3)$ può essere ricostruito come segue:
$3 \Leftarrow pred[1, 3] = 2 \Leftarrow pred[1, 2] = 1 \Leftarrow pred[1, 1] = 1$.

- Ponendo $c_{ii} = \infty$, $\forall i \in V$, al termine dell'algoritmo u_{ii} rappresenta il costo del circuito di costo minimo che passa per i.