Elementi di Teoria dei Grafi

Argomenti lezione:

- Il problema del percorso orientato ottimo

(“shortest path problem”)

- Algoritmo di Dijkstra
- Implementazione efficiente
- Esercizi

Il problema del percorso orientato ottimo

Dato un grafo orientato : digrafo $G = (N, A)$ con N nodi e A archi

Percorso : sequenza di nodi e archi adiacenti nel digrafo

Percorso orientato ottimo: percorso in un **digrafo pesato**
tale che sia minimizzato il suo costo

Attenzione:
in generale un digrafo G può ammettere dei **cicli orientati**

Problema **ciclo a peso negativo**: un algoritmo risolutivo potrebbe percorrere quel ciclo un infinito numero di volte

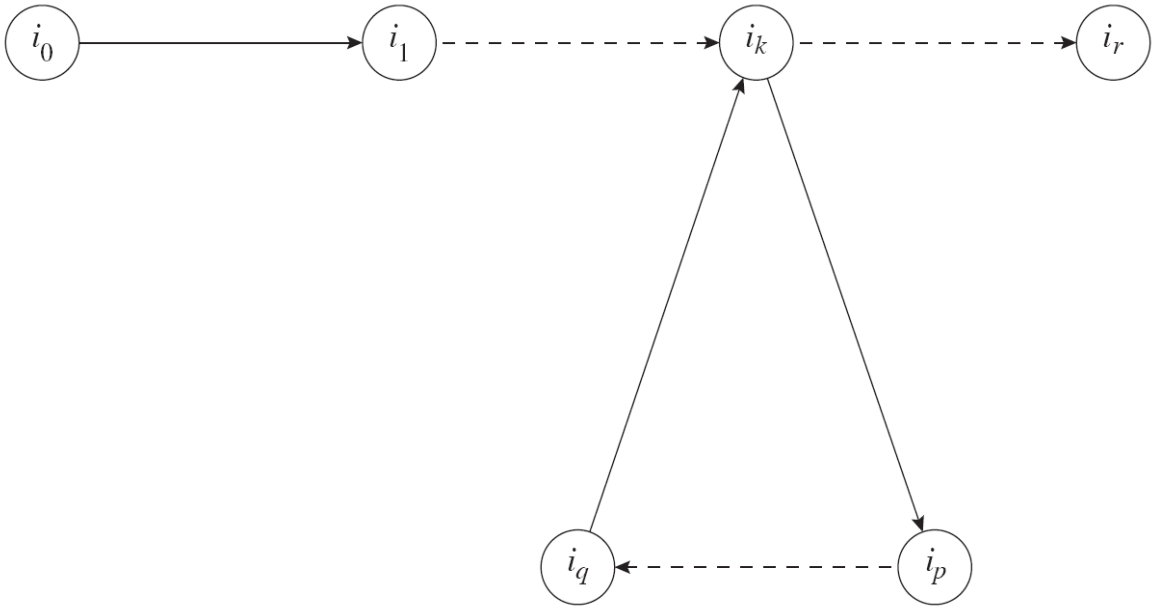
Casi particolari: Esistono algoritmi risolutivi in grado di trovare una soluzione ottima in *tempo polinomiale*

dariano,pacciarelli@dia.uniroma3

Teorema percorso – cammino in un digrafo

Cammino: una sequenza senza ripetizioni di nodi

Teorema: Se esiste un percorso orientato ottimo P^* da i_0 a i_r, esso è un cammino orientato, oppure da P^* si può ricavare un cammino orientato minimo di costo pari alla somma dei costi degli archi che compongono il percorso orientato.



dariano,pacciarelli@dia.uniroma3

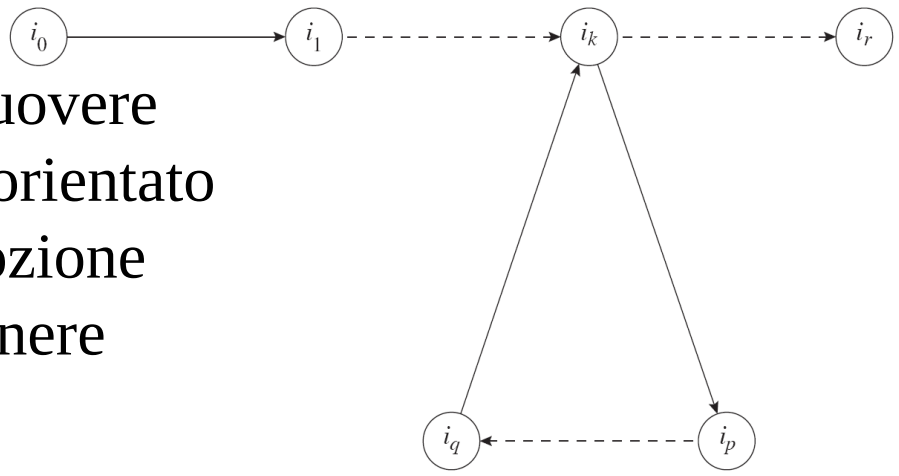
Teorema percorso – cammino in un digrafo

Dimostrazione:

Se il percorso orientato ottimo non contiene cicli allora è anche un cammino orientato ottimo. Altrimenti abbiamo tre casi:

1. Il ciclo è di peso positivo: non può essere altrimenti il percorso non sarebbe ottimo;
2. Il ciclo è di peso negativo: non può essere altrimenti gli archi del ciclo orientato sarebbero percorsi un numero infinito di volte e non si avrebbe una soluzione ottima finita;
3. Il ciclo è di peso nullo:

in questo caso si possono rimuovere gli archi che formano il ciclo orientato e ripetere la procedura di rimozione cicli di peso nullo fino ad ottenere un cammino orientato ottimo.



dariano,pacciarelli@dia.uniroma3

Tipologie di problemi studiati

Percorso orientato di costo minimo tra due nodi (problema singola origine – singola destinazione).
Si chiede di restituire il percorso orientato ottimo in un digrafo da un nodo sorgente ad un nodo destinazione

Percorsi orientati ottimi da un nodo del digrafo (problema singola origine – destinazione multipla).
Si chiede di calcolare tutti i percorsi orientati di costo minimo a partire da un nodo verso tutti gli altri nodi

Percorso orientato ottimo tra tutte le coppie di nodi (problema origine multipla– destinazione multipla).
Si chiede di fornire tutti i percorsi orientati minimi per ogni coppia di nodi nel digrafo

Esempi di applicazione pratica

Possibili applicazioni del problema del percorso orientato minimo sorgono in tutti i problemi in cui bisogna *trovare la strada migliore* (più breve, meno costosa)

Esempi: gestione dei pacchetti in una rete di computer o gestione dei veicoli (treni, aerei, navi) in una rete di trasporti

Il problema del percorso orientato minimo sorge anche in contesti molto diversi, spesso come sottoproblema nell'ambito di un problema complesso, quando bisogna *allocare efficientemente delle risorse per compiere una serie di attività*

Esempi: sequenziamento del DNA o project scheduling

Classificazioni di algoritmi iterativi

Algoritmo iterativo: assegna iterativamente delle stime della lunghezza del percorso (etichette o *label*) ai nodi.

Algoritmo label *setting*: ad ogni iterazione viene fissata in maniera permanente l'etichetta di un nodo. Il valore che assume l'etichetta è ottimo.

Algoritmo label *correcting*: le etichette sono temporanee fino al termine dell'esecuzione dell'algoritmo. Solo a quel punto tutte le lunghezze dei percorsi saranno permanenti e conterranno i valori ottimi.

Considerazioni su algoritmi iterativi

- Gli algoritmi label correcting sono generali ovvero si applicano a tutti i problemi di percorso orientato minimo
- Gli algoritmi label setting sono più efficienti, ma si applicano solamente a digrafi con **pesi non negativi** sugli archi
- Gli algoritmi label setting raggiungono dunque un'efficienza maggiore di label correcting ma su una classe ristretta di problemi
- In questa lezione e nelle prossime lezioni del corso vedremo l'algoritmo di Dijkstra (label setting) e l'algoritmo di Floyd–Warshall (label correcting)

Introduzione all'algoritmo di Dijkstra

Consideriamo il **digrafo pesato** $G = (N, A)$

Condizione restrittiva: supponiamo che G abbia *tutti i pesi non negativi* sugli archi, $c_{ij} >= 0$, per ogni i, j in N

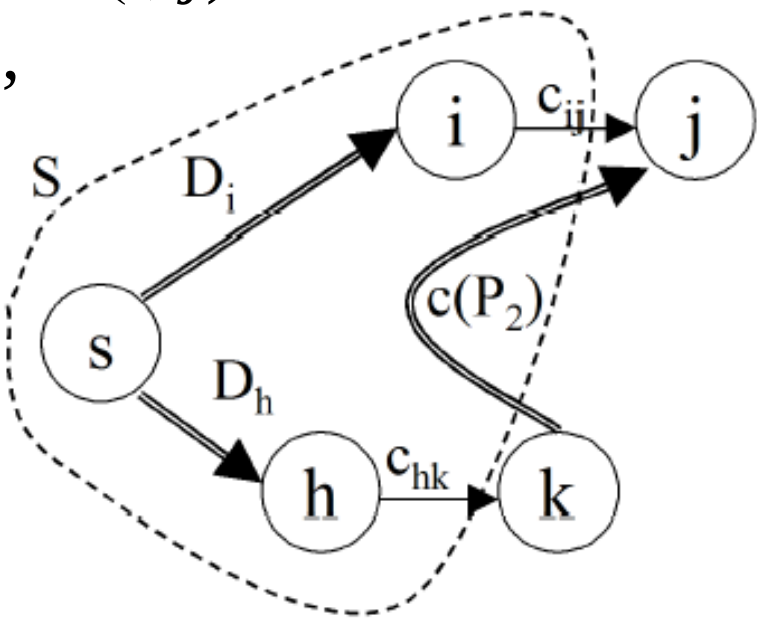
[Attenzione: questa condizione è più forte di quella di imporre che il digrafo non presenti cicli a peso negativo, ovvero è possibile avere un digrafo con alcuni archi negativi ma senza avere cicli a peso negativo!]

Problema da risolvere: trovare a partire da un vertice s i percorsi orientati minimi verso tutti gli altri nodi di G

Condizioni di ottimalità (1)

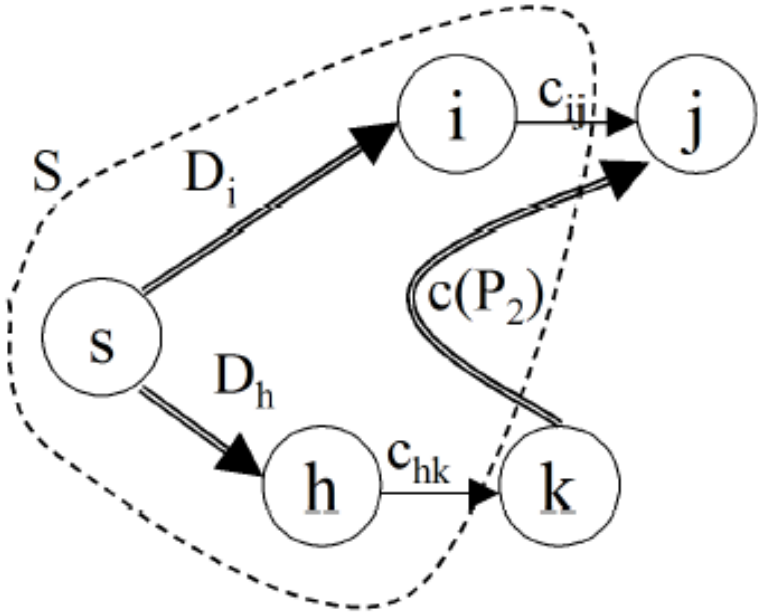
Teorema: Dato un digrafo pesato $G = (N, A)$, con tutti i pesi positivi ($c_{ij} >= 0$, per ogni i, j in N), sia s il nodo sorgente e Di il valore permanentemente assunto dalle etichette dei nodi.

Dato un arco (i, j) che minimizza il costo degli archi del taglio $[S, \underline{S}]+$ con s in S, ovvero l'arco $(i, j) =$ $\arg \min\{Dh + c_{hk} : (h,k) \text{ in } [S, \underline{S}]+\}$, allora $Dj = Di + c_{ij}$ è il costo del percorso orientato minimo da s a j



Condizioni di ottimalità (2)

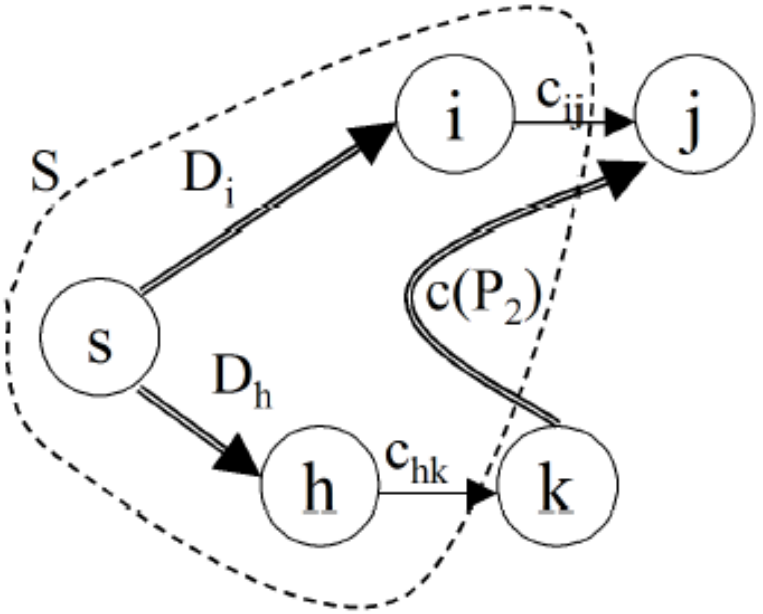
Dimostrazione: L'obiettivo è mostrare che non può esistere un percorso P da s a j di costo minore di $D_i + c_{ij}$.
P può essere partizionato nel percorso $P1$ da s al nodo h, più l'arco (h, k) che attraversa il taglio, più il percorso $P2$ da k al nodo j.
Il percorso $P1$ per definizione conterrà solamente nodi già visitati permanentemente ed appartenenti all'insieme S.
Il percorso $P2$ non avrà limitazioni sui nodi che può attraversare.
Sia $c(P)$ il costo del percorso P.
Segue $c(P) = c(P1) + c_{hk} + c(P2)$.
Considerando che la lunghezza del percorso $P2$ sarà (per ipotesi) positiva, $c(P2) >= 0$, e tenendo presente che $c(P1) + c_{hk} >= D_i + c_{ij}$ (dato che $D_i + c_{ij}$ è minimo tra tutti i percorsi da s ai nodi dell'insieme S), si dimostra la tesi.



Condizioni di ottimalità (3)

Dimostrazione: L'obiettivo è mostrare che non può esistere un percorso P da s a j di costo minore di $D_i + c_{ij}$.
P può essere partizionato nel percorso $P1$ da s al nodo h, più l'arco (h, k) che attraversa il taglio, più il percorso $P2$ da k al nodo j.
Il percorso $P1$ per definizione conterrà solamente nodi già visitati permanentemente ed appartenenti all'insieme S.
Il percorso $P2$ non avrà limitazioni sui nodi che può attraversare.

Sia $c(P)$ il costo del percorso P.
Segue $c(P) = c(P1) + c_{hk} + c(P2)$.
Considerando che la lunghezza del percorso $P2$ sarà (per ipotesi) positiva, $c(P2) >= 0$, e tenendo presente che $c(P1) + c_{hk} >= D_i + c_{ij}$ (dato che $D_i + c_{ij}$ è <u>minimo</u> tra tutti i percorsi da s ai nodi dell'insieme <u>S</u>), si dimostra la tesi.



dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (1)

Algoritmo per calcolare il percorso orientato minimo da un nodo sorgente *s* verso tutti gli altri nodi del digrafo:

- *s* è il nodo sorgente;
- Vettore *pred*[*i*]: per memorizzare permanentemente il nodo predecessore di ogni nodo;
- Vettore *D*[*i*]: per memorizzare permanentemente il valore assunto dalle etichette di ogni nodo.

Algoritmo di Dijkstra (Ver. $O(n^3)$)

// Inizializzazione
$S = \{s\};$
$pred[i] = s, \forall i \in V;$
// Ciclo principale
for ($i = 0; i < n; i++$)
{
  $(h, k) = \arg\min\{D_i + c_{ij} : (i, j) \in [S, \bar{S}]^+\};$
  // Aggiornamento
  $pred[k] = h;$
  $D[k] = D[h] + c_{hk};$
  $S = S \cup k;$
}

dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (2)

- L'algoritmo di Dijkstra per i problemi di percorso orientato minimo in un digrafo è *concettualmente simile* all'algoritmo di Prim–Dijkstra per trovare l'albero ricoprente di costo minimo in un grafo.

- La *differenza sostanziale* è nella scelta dell'arco che minimizza il costo del taglio *S*.

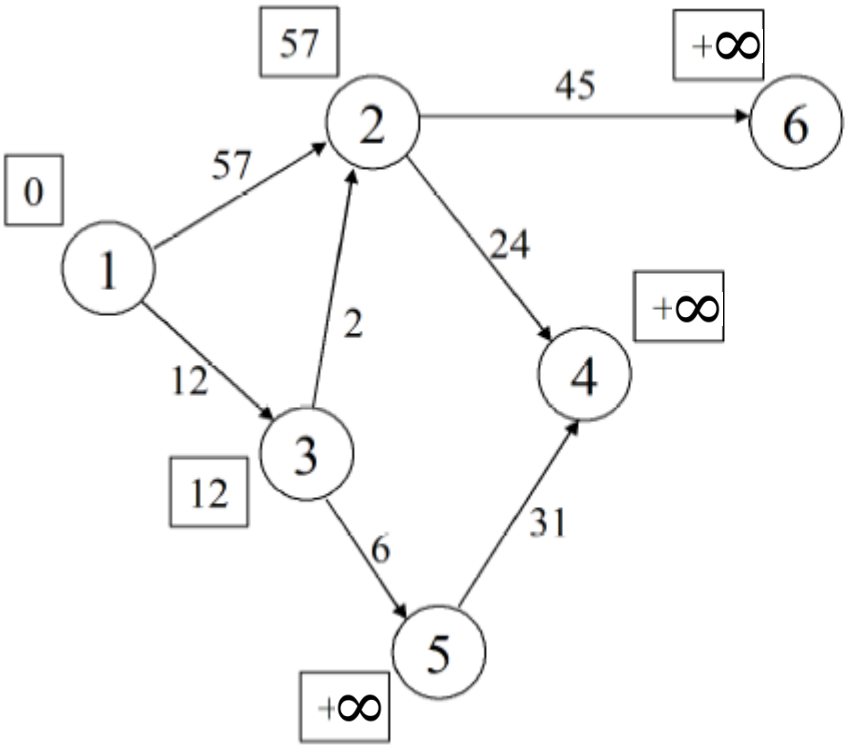
- Una *ulteriore differenza* è che la scelta del nodo iniziale non influisce sulla soluzione fornita.

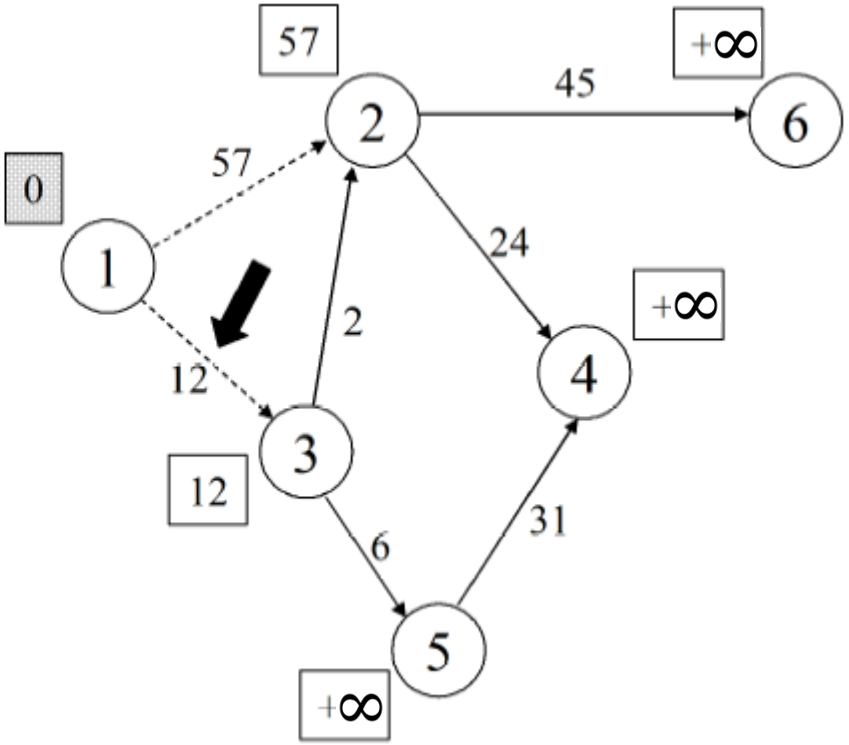
- Il nodo iniziale per l'algoritmo di Dijkstra è invece un requisito del problema che si vuole risolvere.

Algoritmo di Dijkstra (3)

Esempio

- Viene dato come nodo sorgente $s = 1$ da cui aggiorno etichette
- I iterazione: si considera il taglio $[S, \underline{S}]+ = \{(1,2), (1,3)\}$
- L'arco che minimizza il taglio $[S, \underline{S}]+$ è l'arco (1,3)





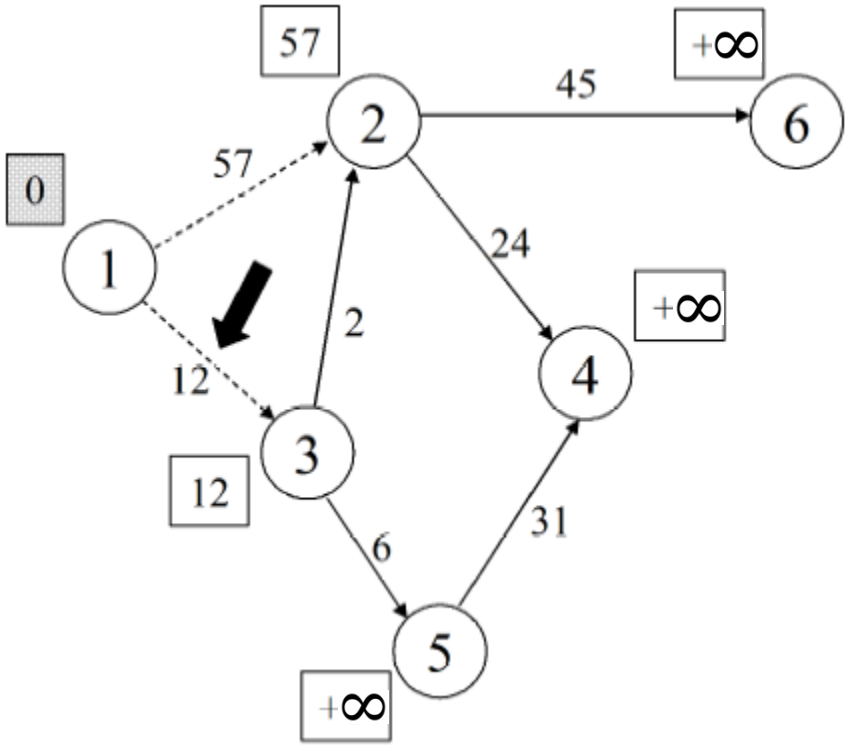
+ infinito

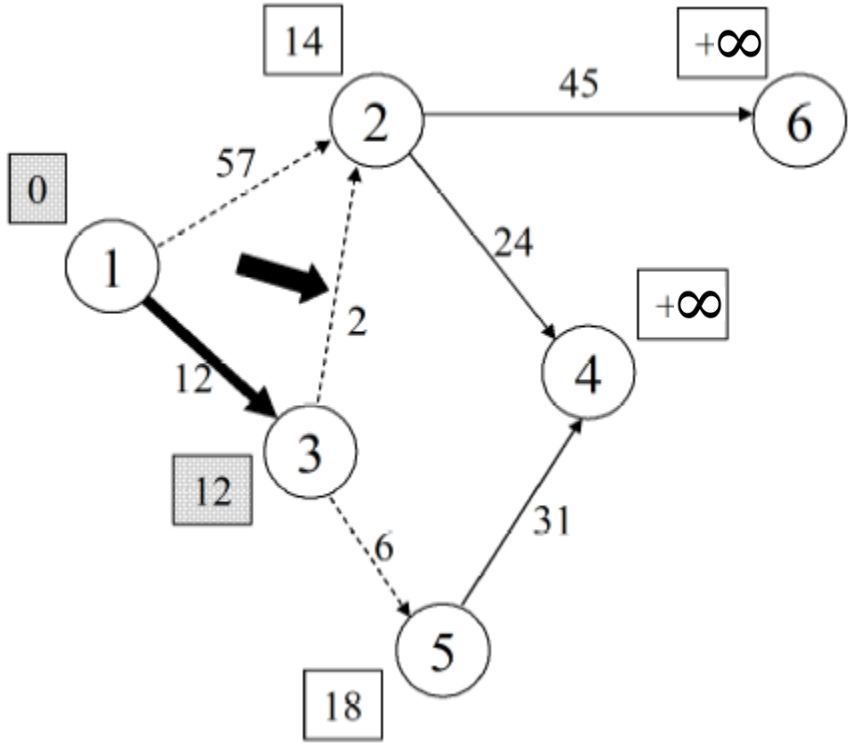
dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (4)

Esempio

- Dalla I iterazione: il nodo 3 viene aggiunto al taglio S e l'etichetta (12) del nodo 3 viene fissata permanentemente.
- II iterazione: si considera il taglio $[S, \underline{S}]+ = \{(1,2), (3,2), (3,5)\}$
- L'arco che minimizza il nuovo taglio $[S, \underline{S}]+$ è l'arco (3,2)



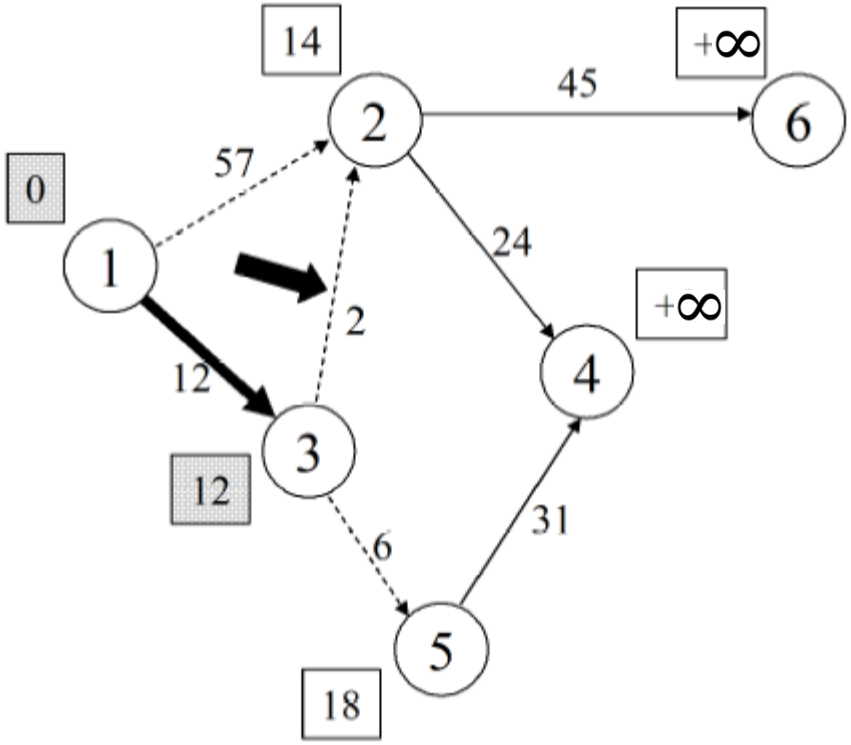


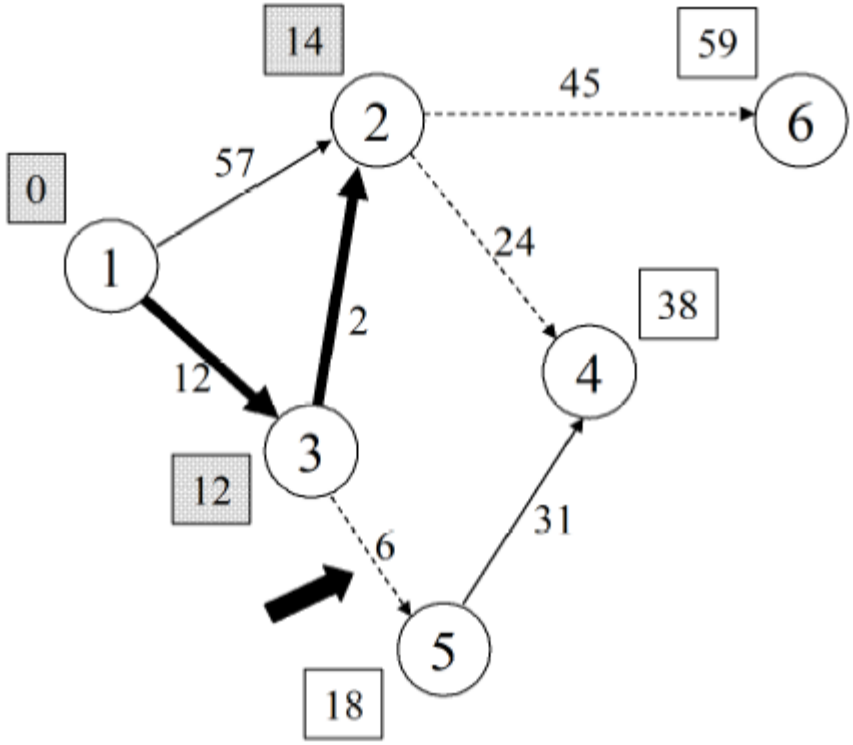
dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (5)

Esempio

- Dalla II iterazione: il nodo 2 viene aggiunto al taglio S e l'etichetta (14) del nodo 2 viene fissata permanentemente.
- III iterazione: si considera $[S, \underline{S}]+ = \{(2,6), (2,4), (3,5)\}$
- L'arco che minimizza il nuovo taglio $[S, \underline{S}]+$ è l'arco (3,5)

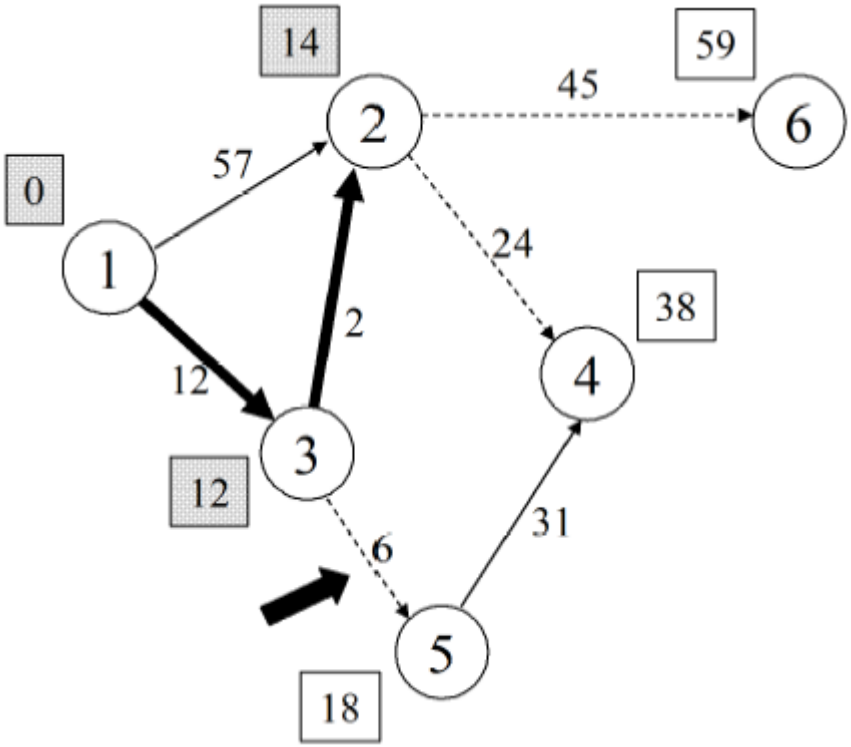


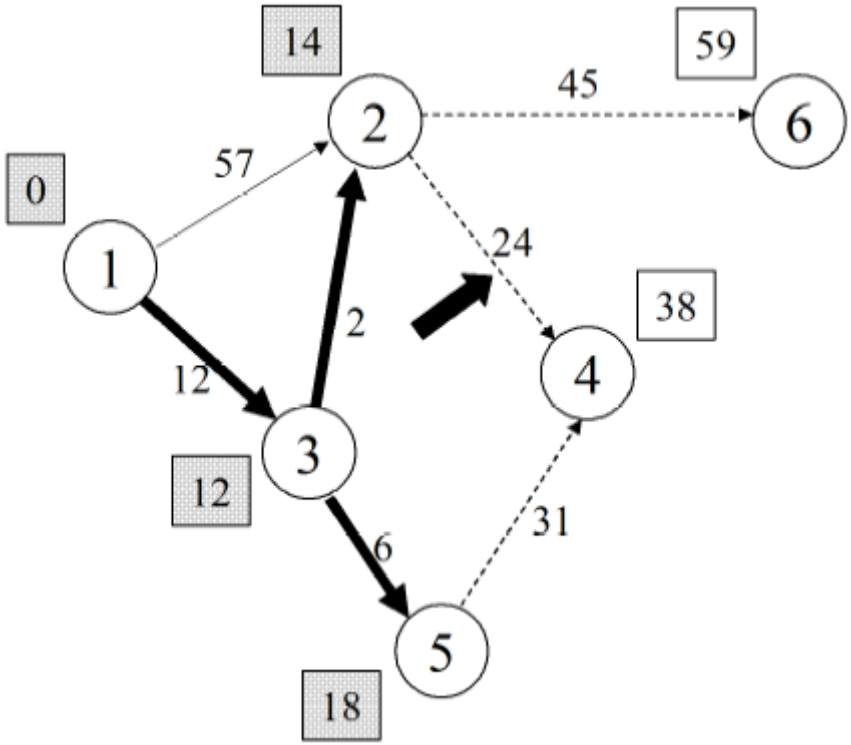


Algoritmo di Dijkstra (6)

Esempio

- Dalla III iterazione: il nodo 5 viene aggiunto al taglio *S* e l'etichetta (18) del nodo 5 viene fissata permanentemente.
- IV iterazione: si considera $[S, \underline{S}]^+ = \{(2,6), (2,4), (5,4)\}$
- L'arco che minimizza il nuovo taglio $[S, \underline{S}]^+$ è l'arco (2,4)



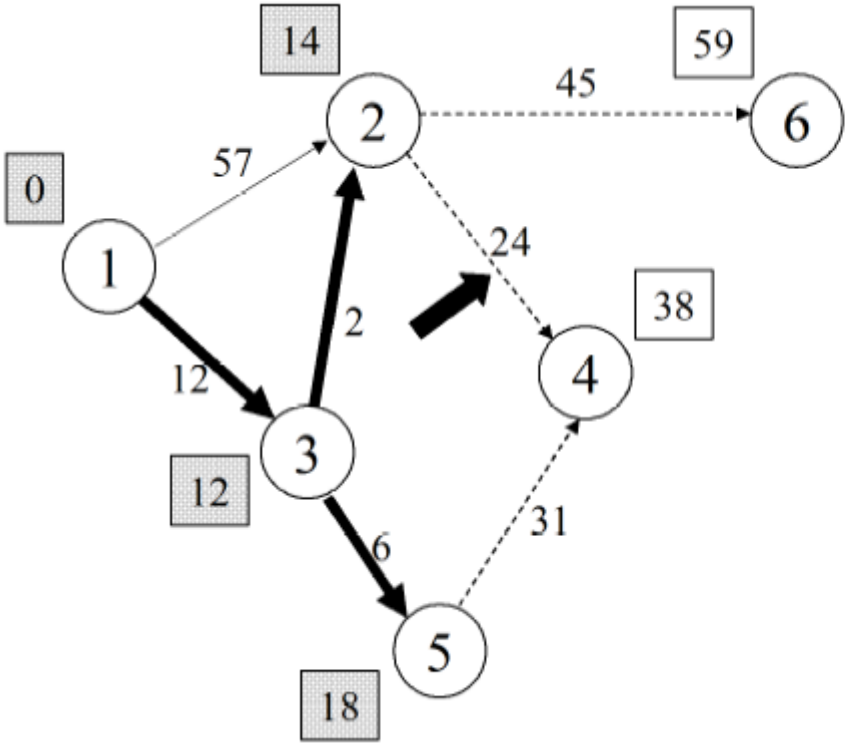


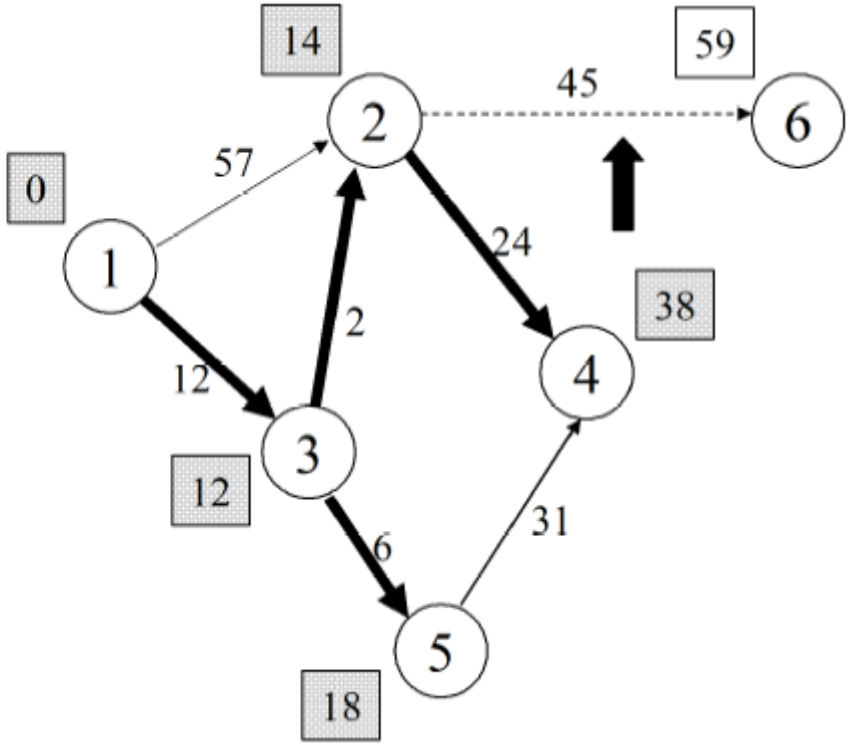
dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (7)

Esempio

- Dalla IV iterazione: il nodo 4 viene aggiunto al taglio S e l'etichetta (38) del nodo 4 viene fissata permanentemente.
- V iterazione: si considera $[S, \underline{S}]^+ = \{(2,6)\}$
- L'arco che minimizza il nuovo taglio $[S, \underline{S}]^+$ è l'arco (2,6)



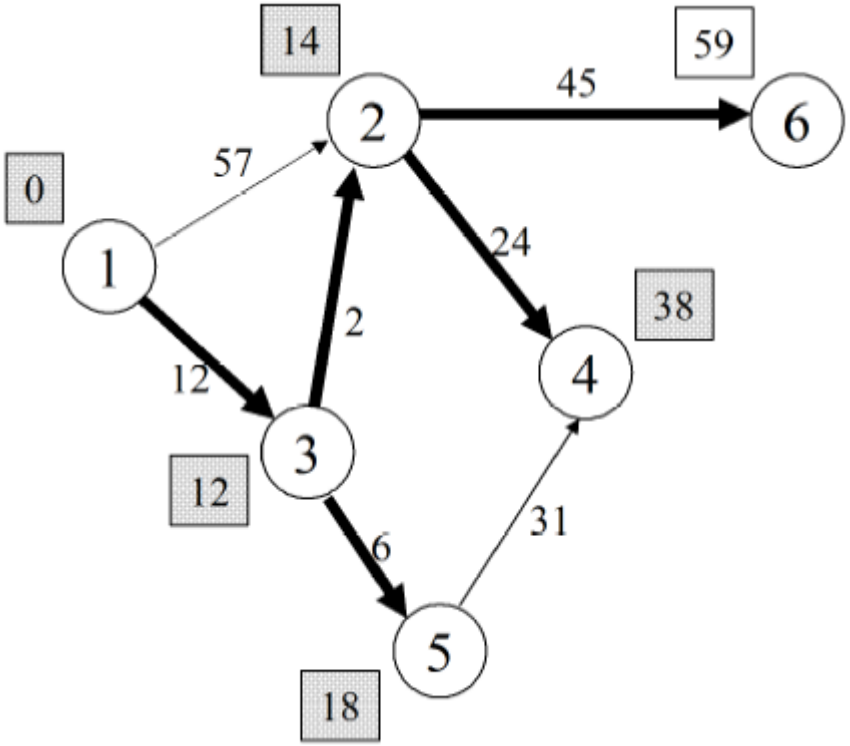


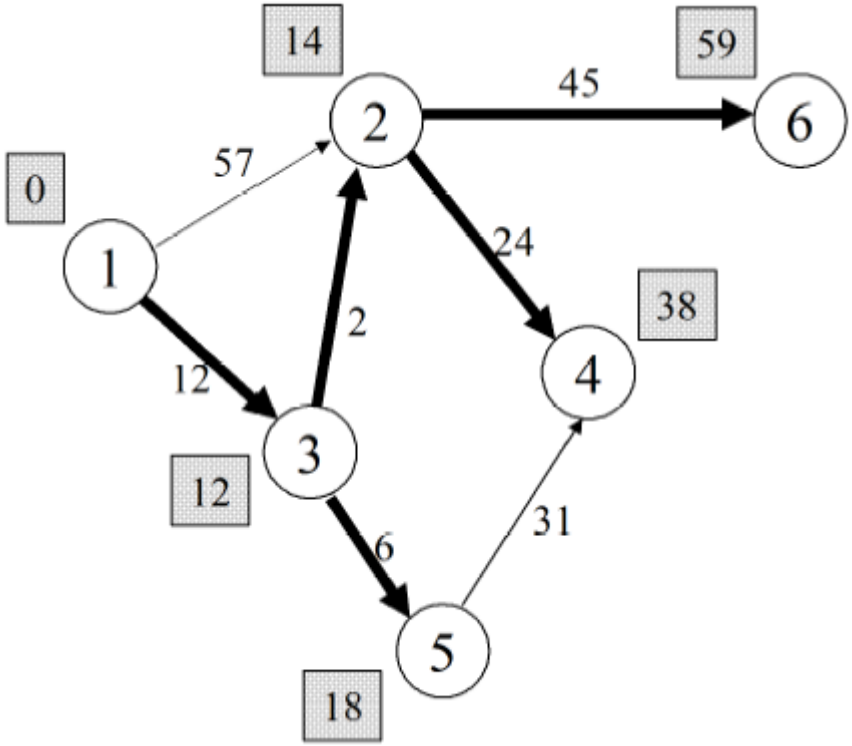
dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (8)

Esempio

- Dalla V iterazione: il nodo 6 viene aggiunto al taglio *S* e l'etichetta (59) del nodo 6 viene fissata permanentemente.
- $S = \{1, 3, 2, 5, 4, 6\}$





dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (9)

- L'algoritmo di Dijkstra eseguirà al più ***n* iterazioni**, infatti ad ogni iterazione l'algoritmo aggiunge un nodo all'insieme *S*

- Il numero di operazioni svolte dall'algoritmo dipende da come viene implementata la fase di selezione dell'arco di peso minimo

- L'implementazione vista di questa operazione richiede di scorrere gli ***m* archi del digrafo** per stabilire se l'arco appartiene o meno all'insieme [*S*, $\underline{S}$], e stabilire se è quello di costo minimo

- La *complessità*, ovvero il maggior numero di operazioni richieste da una esecuzione della corrente versione dell'algoritmo, sarà non superiore a $\mathbf{O(n^3)}$ operazioni (dato che $m <= n^2$)

Algoritmo di Dijkstra (10)

Analisi della complessità:

Ciclo principale viene eseguito *n* volte

Fase di esecuzione richiede di scorrere tutti gli archi del digrafo (*m*)

Dato che $m << n^2$
(un nodo ha al più *n*-1 archi adiacenti, non ci sono anelli o archi multipli)
si ha $\boldsymbol{O(n^3)}$

Algoritmo di Dijkstra (Ver. $O(n^3)$)

```
// Inizializzazione
S = {s};
pred[i] = s, ∀i ∈ V;
// Ciclo principale
for ( i = 0; i < n; i++ )
{
    (h, k) = arg min{D_i + c_ij : (i, j) ∈ [S, S̄]^+};
    // Aggiornamento
    pred[k] = h;
    D[k] = D[h] + c_hk;
    S = S ∪ k;
}
```


dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (11)

Implementazione efficiente dell'algoritmo:

La complessità dell'algoritmo di Dijkstra può essere ridotta tramite una **struttura dati alternativa** che permette di non dover ricalcolare ad ogni passo l'arco di costo minimo all'interno del taglio [S, $\underline{S}$]

Utilizziamo la *stessa struttura dati* utilizzata per la versione efficiente dell'algoritmo di Prim – Dijkstra (per trovare l'albero ricoprente di peso minimo)

dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (12)

Struttura dati per l'implementazione efficiente:

- un vettore booleano di flag (V) che conserva l'informazione se il nodo i appartiene o meno all'insieme S
- un vettore (D) che mantiene aggiornata, per ogni nodo in S, la distanza minima necessaria per raggiungere il nodo a partire dal **nodo sorgente *s***

  { Il vettore delle distanze D sarà inizializzato con i valori degli archi della stella del **nodo sorgente *s*** }
- Inoltre sia H la matrice di adiacenza ($n \times n$) che rappresenta i costi $h[i, j]$ degli archi tra i nodi i e j
  {Nel caso in cui l'arco da i a j non sia presente si assume $h[i, j]$ = +infinito, altrimenti si avrà $h[i, j] = c_{ij}$ }

Algoritmo di Dijkstra (13)

Considerazioni sul vettore delle distanze:

Il vettore *D* rappresenta una stima della distanza minima necessaria per congiungere ogni nodo al **nodo sorgente *s***

Questa distanza viene inizializzata con + infinito
se non esiste un arco diretto dal nodo sorgente al generico nodo, altrimenti viene inizializzata con il peso dell'arco

Durante l'esecuzione dell'algoritmo queste distanze vengono diminuite non appena il nodo diviene raggiungibile, ovvero l'arco appartiene al taglio corrente

Se la distanza rimane + infinito il nodo non è ancora raggiungibile

Algoritmo di Dijkstra (14)

Implement. efficiente:
s è il nodo sorgente

Vettore *pred*[*i*] memorizza il nodo predecessore di ogni nodo

V vettore booleano di flag

D vettore delle distanze

H matrice di adiacenza

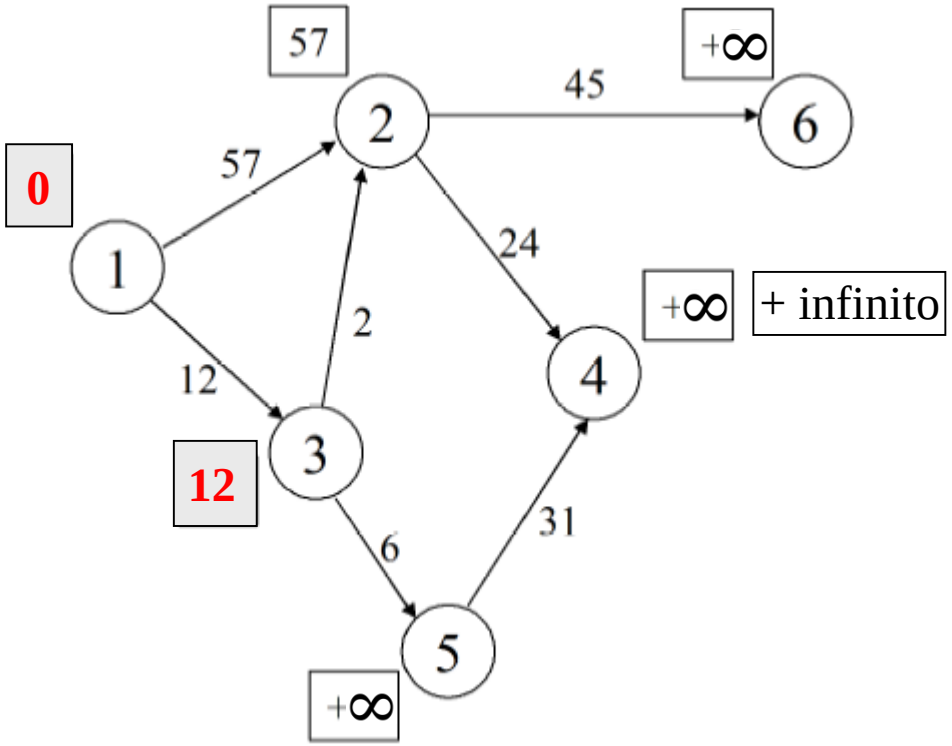
Algoritmo di Dijkstra (Ver. $O(n^2)$)

```
// Inizializzazione
V[i] = 0, ∀i ∈ V;
V[s] = 1;
D[i] = H[s, i], ∀i ∈ V;
pred[i] = s, ∀i ∈ V;
// Ciclo principale
for ( i = 1; i ≤ n − 1; i++ )
{
    Trova il nodo h che minimizza D[h] in D∩!V;
    V[h] = 1;
    for ( k = 1; k ≤ n; k++ )
    {
        if ( V[k] == 0 )&&( D[h] + H[h, k] < D[k] )
        {
            // Aggiornamento
            pred[k] = h;
            D[k] = D[h] + H[h, k];
        }
    }
}
```

Algoritmo di Dijkstra (15)

Riprendiamo l'esempio visto prima:

- Inizializzazione: V = [1, 0, 0, 0, 0, 0] essendo dato s = 1
- I iterazione: nodo 3 viene aggiunto al taglio S e l'etichetta del nodo 3 viene fissata permanentemente (V = [1, 0, 1, 0, 0, 0])

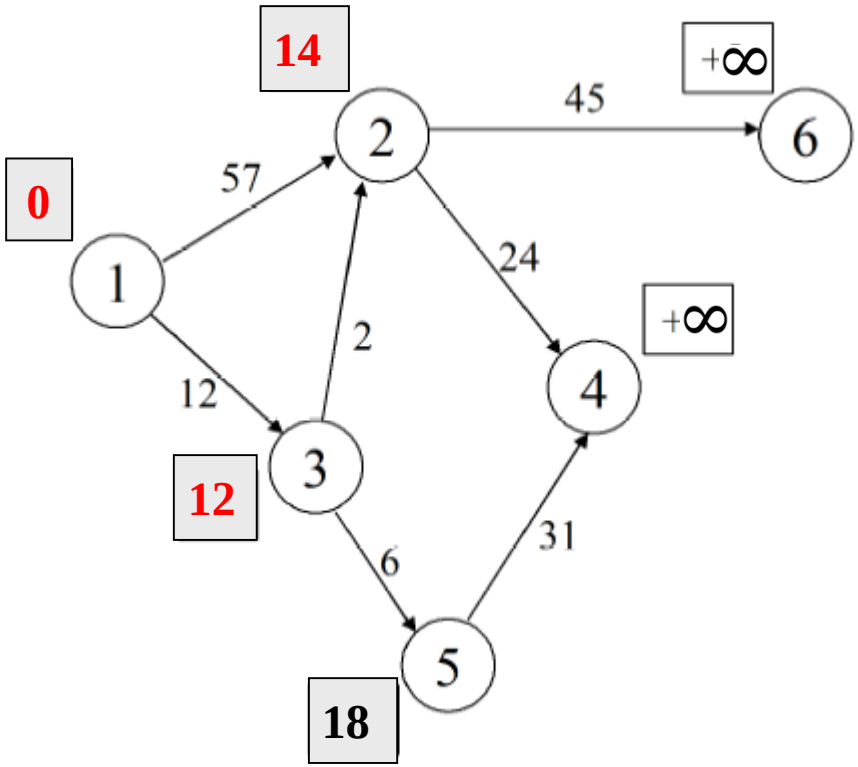


dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (16)

Esempio

- Dalla I iterazione: le etichette dei nodi 2 e 5 vengono aggiornate da 57 a 14 e da +inf a 18
- II iterazione: il nodo 2 viene aggiunto al taglio *S* e la sua etichetta viene resa permanente (*V* = [1, 1, 1, 0, 0, 0])

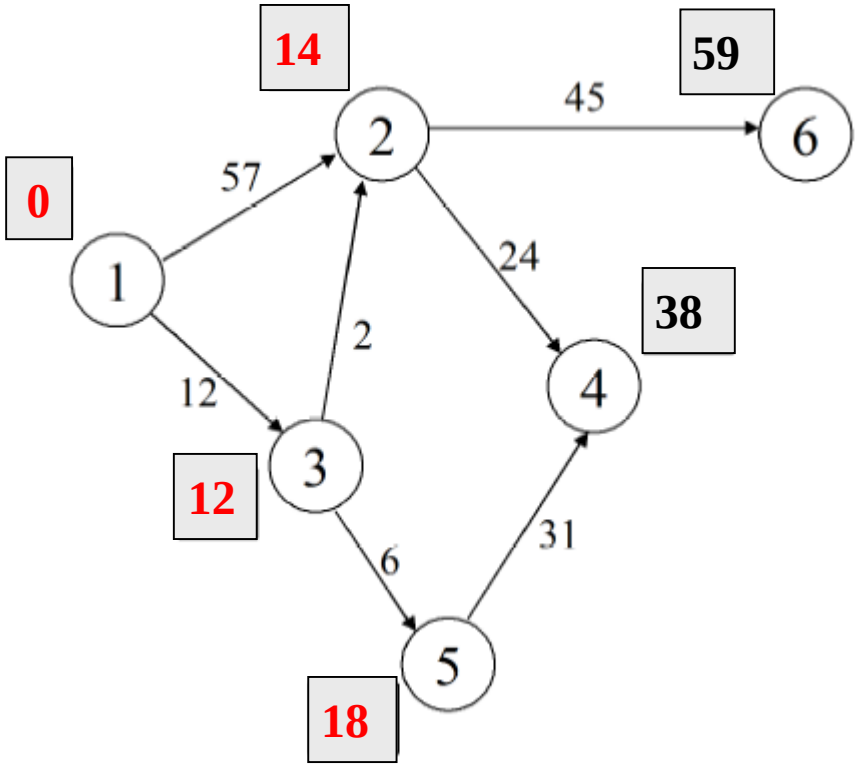


dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (17)

Esempio

- Dalla II iterazione: le etichette dei nodi 4 e 6 vengono aggiornate da +inf a 38 e da +inf a 59
- III iterazione: il nodo 5 viene aggiunto al taglio e la sua etichetta resa permanente (V = [1, 1, 1, 0, 1, 0])

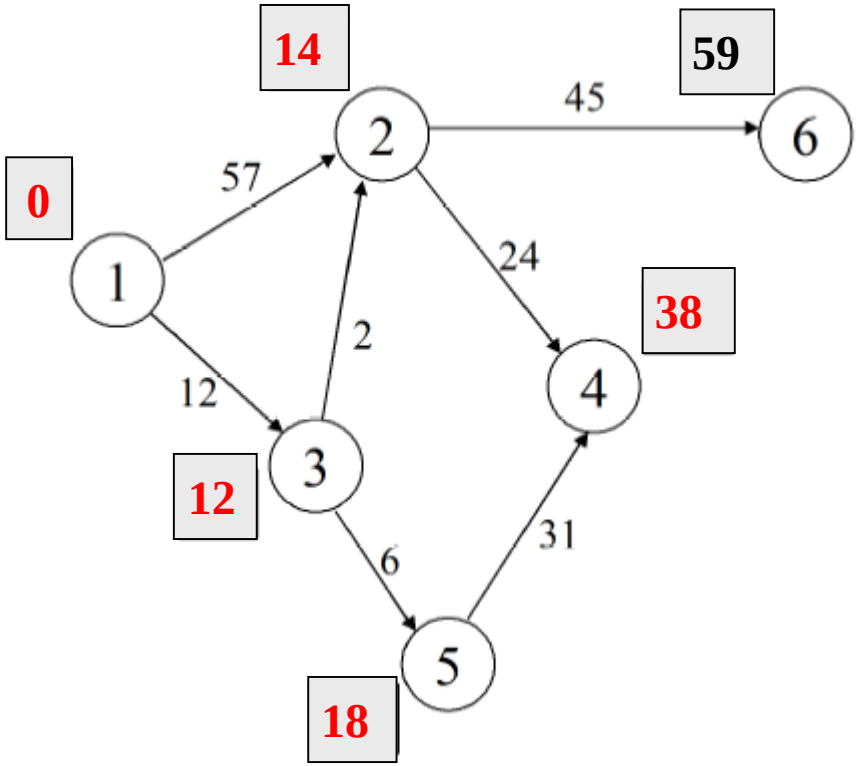


dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (18)

Esempio

- Dalla III iterazione: l'etichetta del nodo 4 non viene aggiornata
- in quanto il valore che la dovrebbe aggiornare (49) risulta essere superiore al suo valore corrente (38)
- IV iterazione:il nodo 4 viene aggiunto e V = [1, 1, 1, 1, 1, 0]

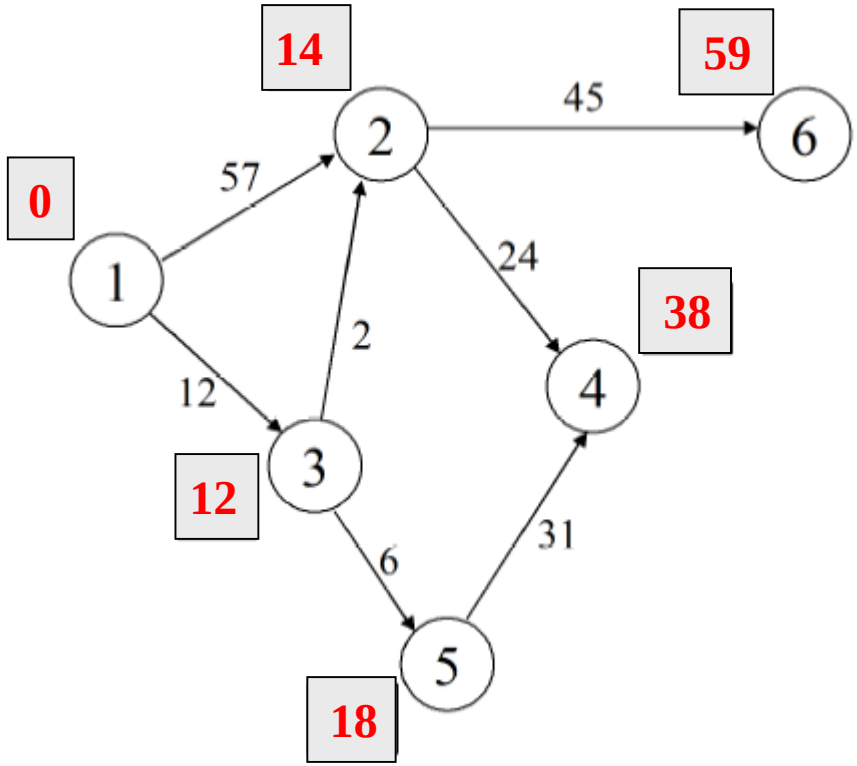


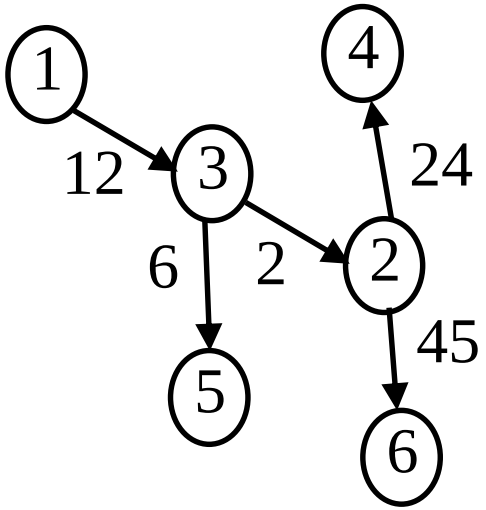
dariano,pacciarelli@dia.uniroma3

Algoritmo di Dijkstra (19)

Esempio

- V iterazione: viene resa permanente l'etichetta del nodo 6 e l'algoritmo termina (V = [1, 1, 1, 1, 1, 1])
- S = {1, 3, 2, 5, 4, 6}





Il peso totale dell'albero è 89

Algoritmo di Dijkstra (20)

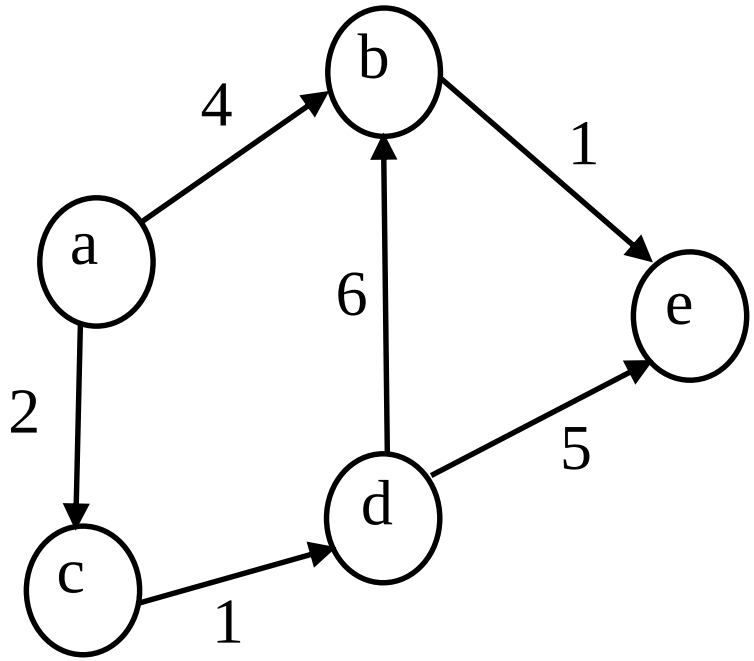
La sequenza della scelta dei nodi è uguale per i due algoritmi visti

Ma nella versione efficiente dell'algoritmo:

- E' possibile individuare il nodo che minimizza il taglio $[S, \underline{S}]^+$ scorrendo i nodi e controllando i valori assunti da V e da D
- In n operazioni è possibile aggiornare tutte le distanze e i predecessori a seguito dell'inserimento di un nodo in S
- Per cui l'algoritmo efficiente richiede non più di $O(n^2)$ operazioni complessive.
- Percorso minimo su digrafi completi = servono esattamente n^2 operazioni per risolvere il problema
- Percorso minimo su digrafi sparsi = esistono ulteriori strutture dati che permettono di ridurre la complessità computazionale

Esercizi su Algo Dijkstra (1)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.



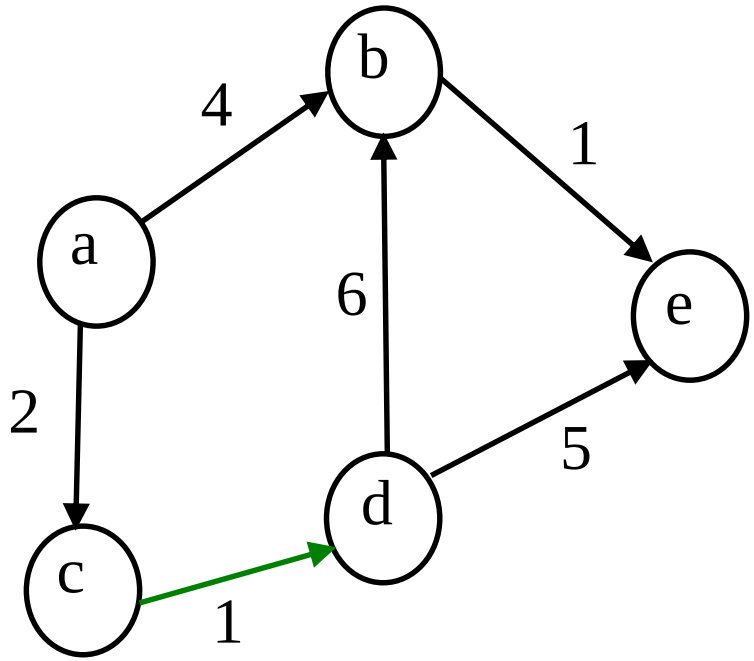
nodo	flag (V)	distanza (D)	predecessore
a	**1**	**0**	**nessuno**
b	**0**	**4**	**a**
c	**0**	**2**	**a**
d	**0**	**inf**	**a**
e	**0**	**inf**	**a**

$S = \{a\}$

dariano,pacciarelli@dia.uniroma3

Esercizi su Algo Dijkstra (2)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.

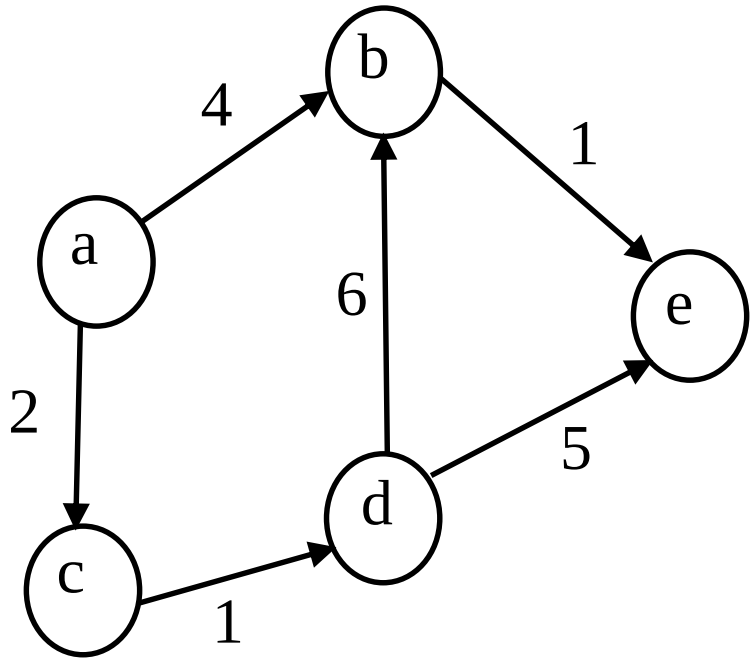


nodo	flag (V)	distanza (D)	predecessore
a	1	0	nessuno
b	0	4	a
c	0	2	a
d	0	inf	a
e	0	inf	a

$S = \{a\}$

Esercizi su Algo Dijkstra (3)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.

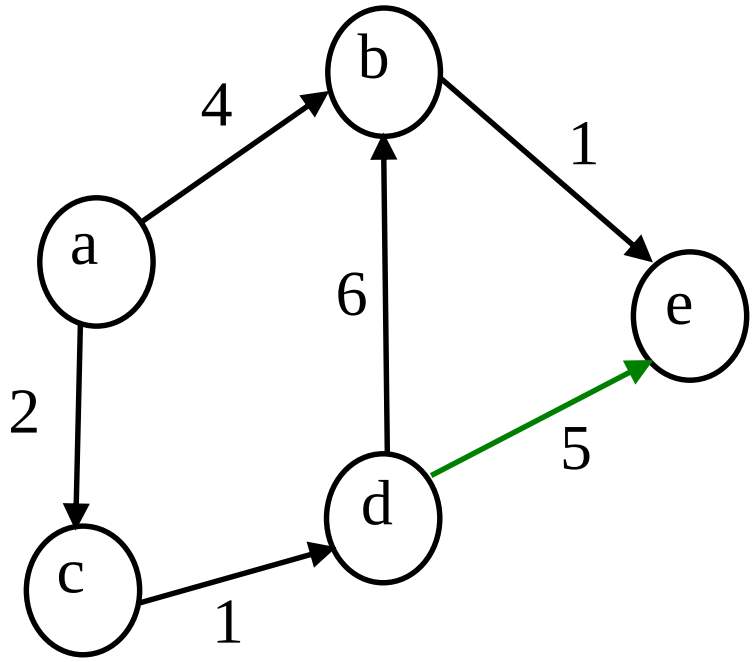


nodo	flag (V)	distanza (D)	predecessore
a	1	0	nessuno
b	0	4	a
c	1	2	a
d	0	3	c
e	0	inf	a

$S = \{a, c\}$

Esercizi su Algo Dijkstra (4)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.

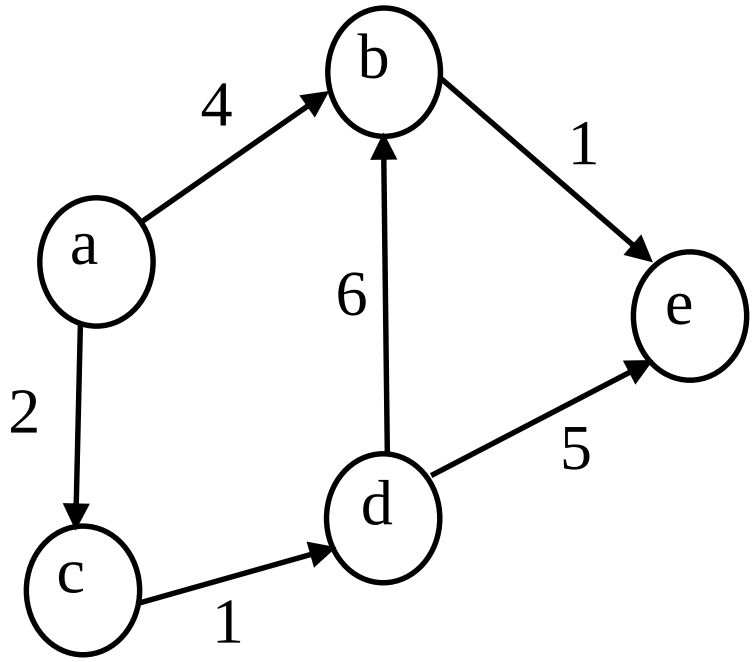


nodo	flag (V)	distanza (D)	predecessore
a	1	0	nessuno
b	0	4	a
c	1	2	a
d	0	3	c
e	0	inf	a

$S = \{a, c\}$

Esercizi su Algo Dijkstra (5)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.

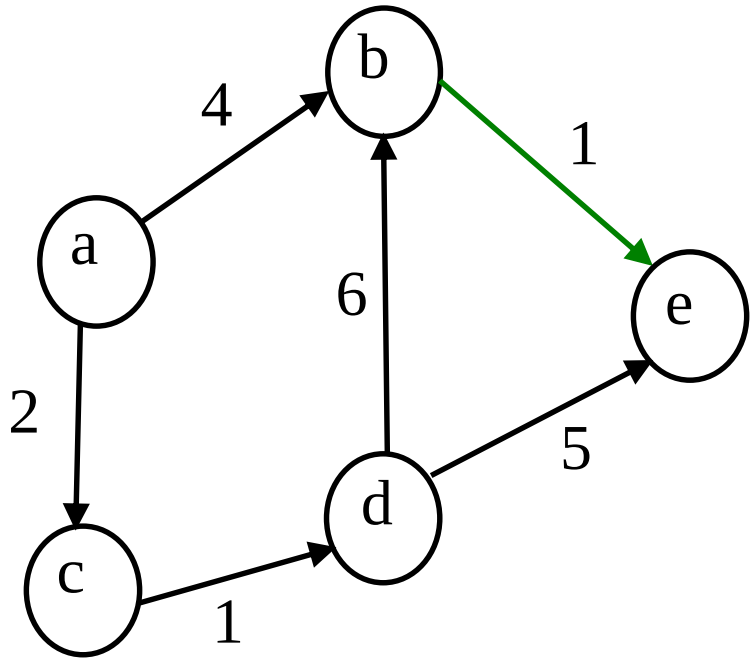


nodo	flag (V)	distanza (D)	predecessore
a	1	0	nessuno
b	0	4	a
c	1	2	a
d	1	3	c
e	0	8	d

$S = \{a, c, d\}$

Esercizi su Algo Dijkstra (6)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.

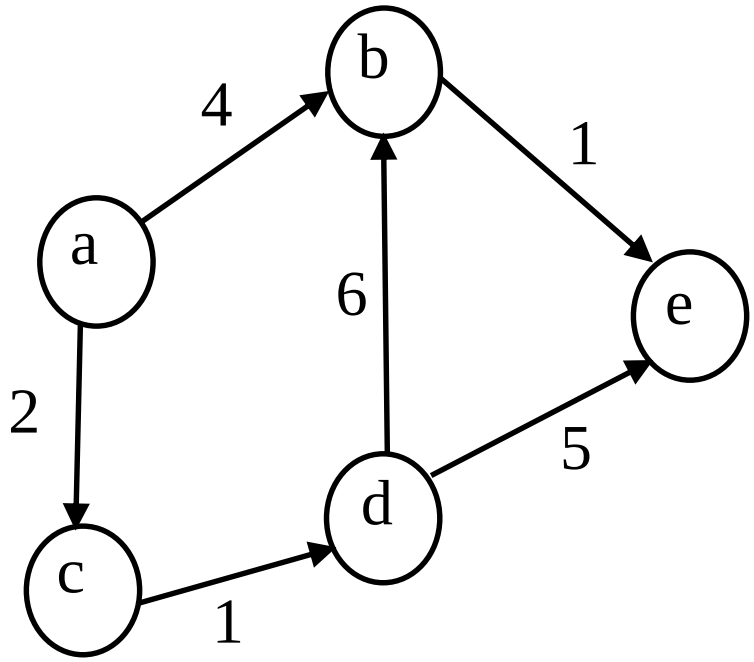


nodo	flag (V)	distanza (D)	predecessore
a	1	0	nessuno
b	0	4	a
c	1	2	a
d	1	3	c
e	0	8	d

$S = \{a, c, d\}$

Esercizi su Algo Dijkstra (7)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.

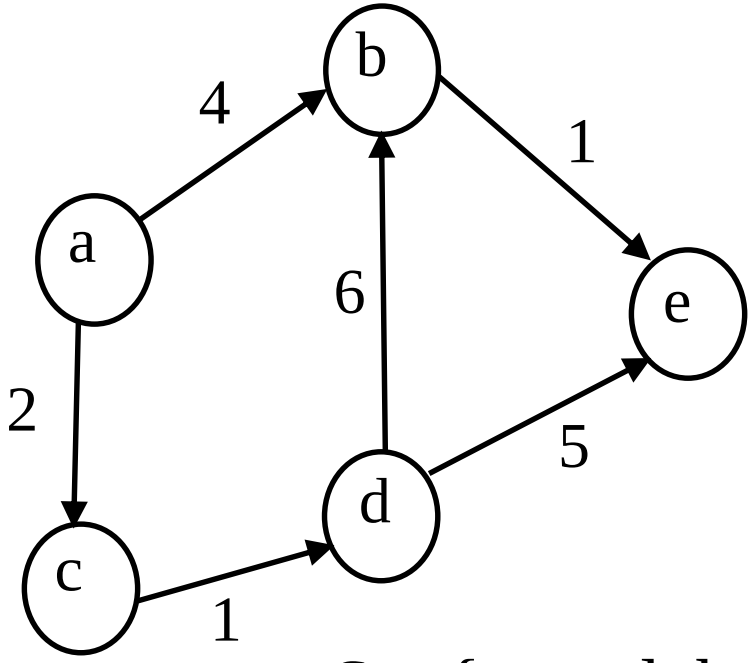


nodo	flag (V)	distanza (D)	predecessore
a	1	0	nessuno
b	1	4	a
c	1	2	a
d	1	3	c
e	0	5	b

S = {a, c, d, b}

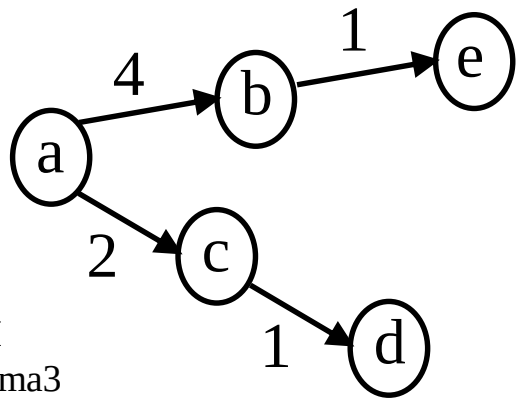
Esercizi su Algo Dijkstra (8)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.



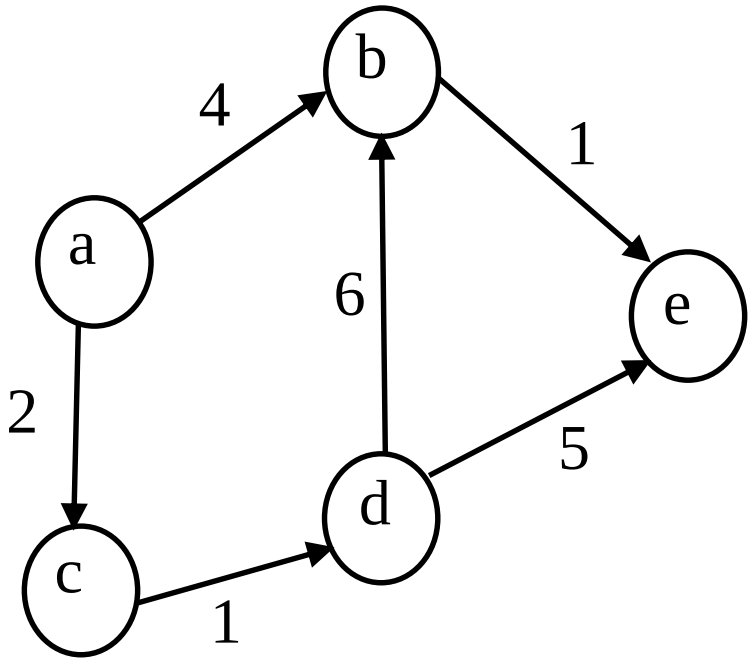
nodo	flag (V)	distanza (D)	predecessore
a	**1**	**0**	**nessuno**
b	**1**	**4**	**a**
c	**1**	**2**	**a**
d	**1**	**3**	**c**
e	**1**	**5**	**b**

S = {a, c, d, b, e}

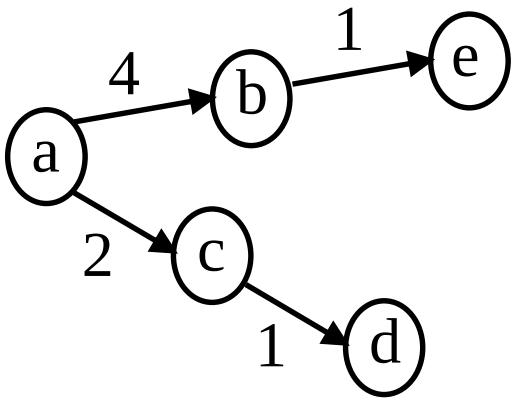


Esercizi su Algo Dijkstra (9)

Abbiamo un digrafo con 5 nodi *a*…*e*. Trovare l'albero dei cammini orientati minimi dal nodo *a* a tutti gli altri nodi utilizzando la versione efficiente dell'algoritmo di Dijkstra. Evidenziare il percorso (cammino) orientato minimo dal nodo *a* al nodo *e*.



S = {a, c, d, b, e}



Il percorso minimo da **a** a **e** è il cammino (**a**, **b**, **e**) di costo 5.

Esercizi su Algo Dijkstra (10)

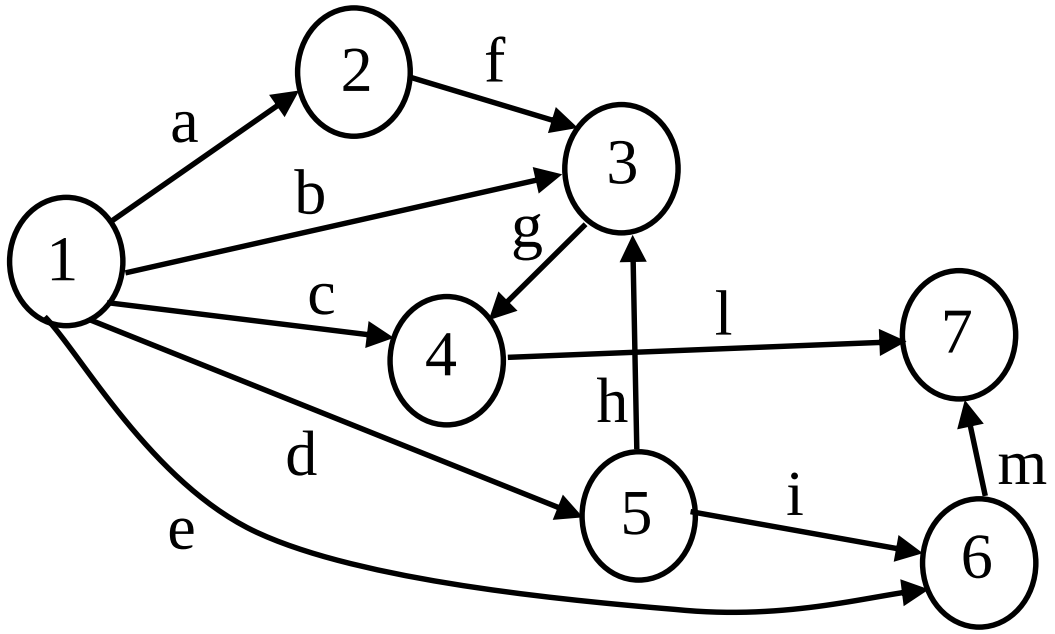
In tabella è riportata la matrice di incidenza nodi/archi del digrafo.
Per convenzione +1 indica un arco uscente dal nodo.

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*l*	*m*
1	+1	+1	+1	+1	+1	0	0	0	0	0	0
2	-1	0	0	0	0	+1	0	0	0	0	0
3	0	-1	0	0	0	-1	+1	-1	0	0	0
4	0	0	-1	0	0	0	-1	0	0	+1	0
5	0	0	0	-1	0	0	0	+1	+1	0	0
6	0	0	0	0	-1	0	0	0	-1	0	+1
7	0	0	0	0	0	0	0	0	0	-1	-1
Pesi	3	6	10	4	8	1	2	4	3	19	10

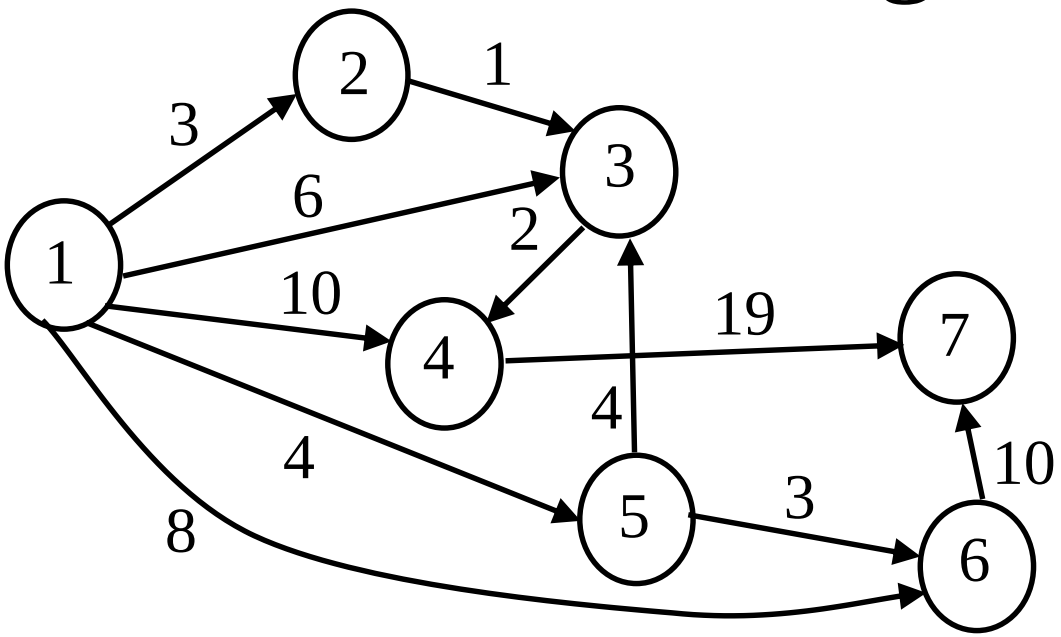
Trovare l'albero dei cammini orientati minimi, a partire dal nodo **1**, utilizzando l'algoritmo di Dijkstra in versione efficiente.

Esercizi su Algo Dijkstra (11)

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*l*	*m*
1	+1	+1	+1	+1	+1	0	0	0	0	0	0
2	-1	0	0	0	0	+1	0	0	0	0	0
3	0	-1	0	0	0	-1	+1	-1	0	0	0
4	0	0	-1	0	0	0	-1	0	0	+1	0
5	0	0	0	-1	0	0	0	+1	+1	0	0
6	0	0	0	0	-1	0	0	0	-1	0	+1
7	0	0	0	0	0	0	0	0	0	-1	-1
Pesi	3	6	10	4	8	1	2	4	3	19	10



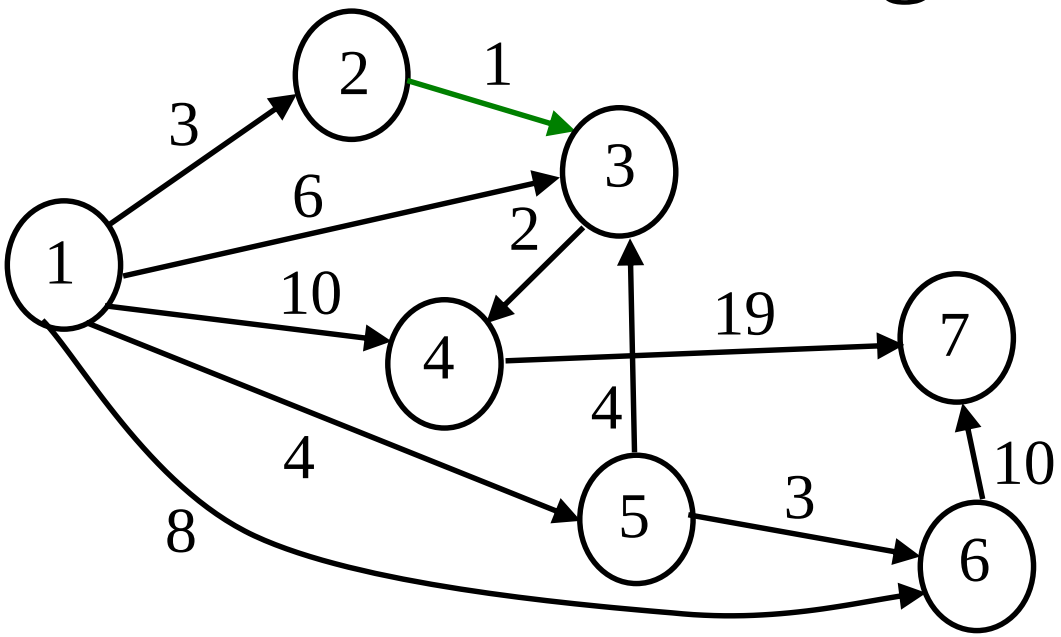
Esercizi su Algo Dijkstra (12)



$S = \{1\}$

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	0	3	1
3	0	6	1
4	0	10	1
5	0	4	1
6	0	8	1
7	0	inf	1

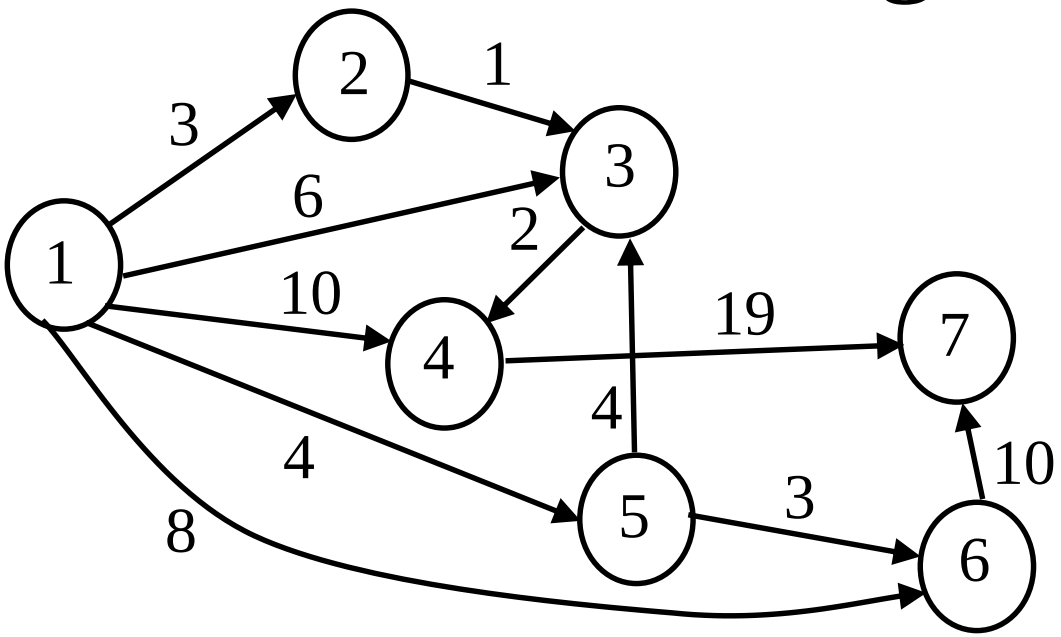
Esercizi su Algo Dijkstra (13)



$S = \{1\}$

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	0	3	1
3	0	6	1
4	0	10	1
5	0	4	1
6	0	8	1
7	0	inf	1

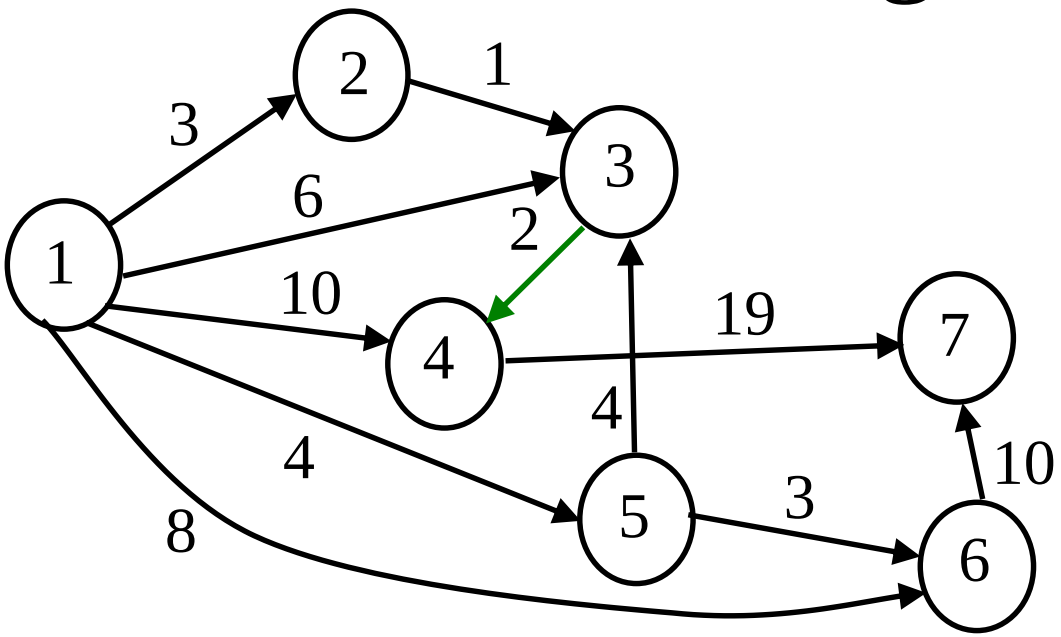
Esercizi su Algo Dijkstra (14)



$S = \{1, 2\}$

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	0	4	2
4	0	10	1
5	0	4	1
6	0	8	1
7	0	inf	1

Esercizi su Algo Dijkstra (15)



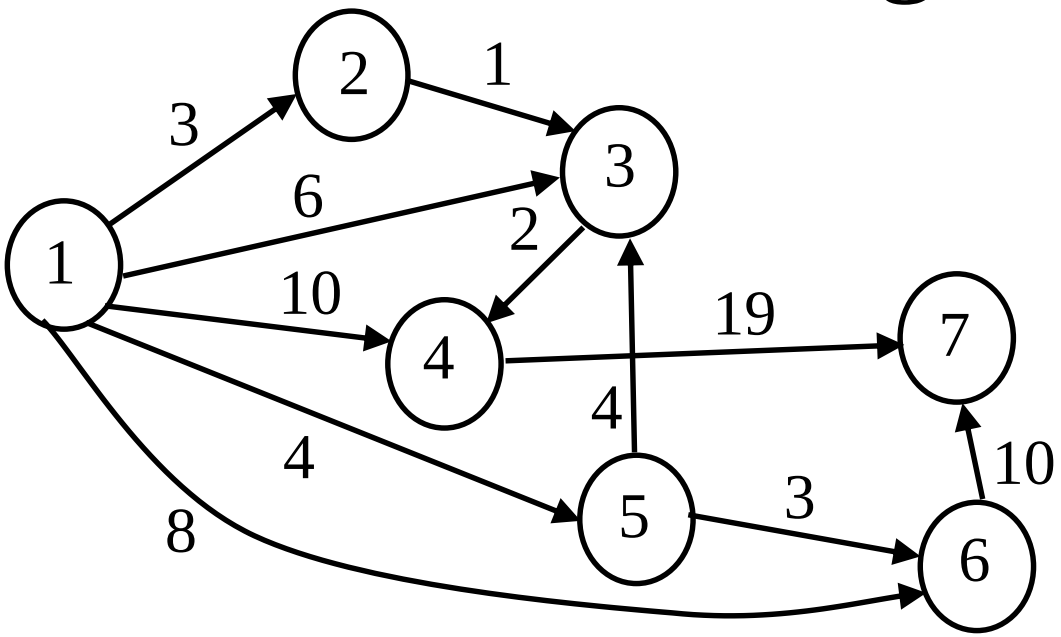
$S = \{1, 2, 3\}$

oppure

$S = \{1, 2, 5\}$

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	0	4	2
4	0	10	1
5	0	4	1
6	0	8	1
7	0	inf	1

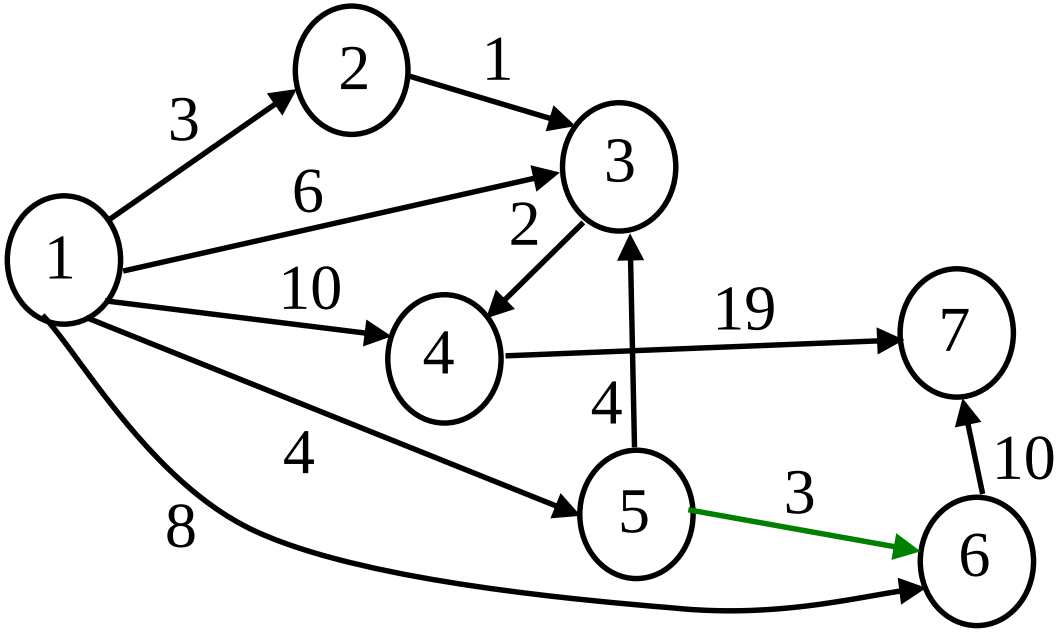
Esercizi su Algo Dijkstra (16)



$S = \{1, 2, 3\}$

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	0	6	3
5	0	4	1
6	0	8	1
7	0	inf	1

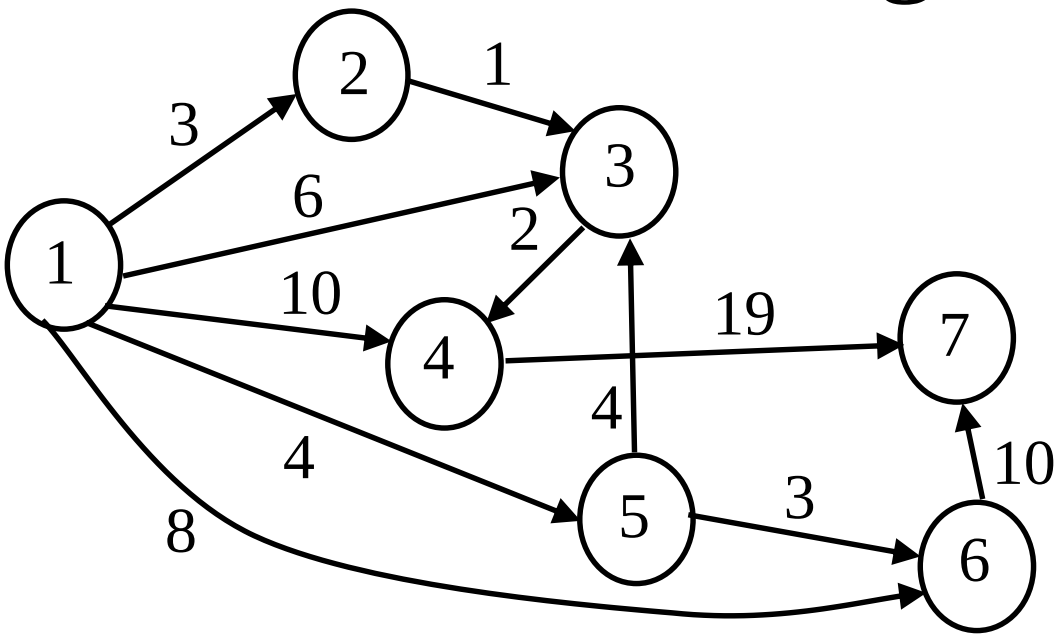
Esercizi su Algo Dijkstra (17)



$S = \{1, 2, 3\}$

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	0	6	3
5	0	4	1
6	0	8	1
7	0	inf	1

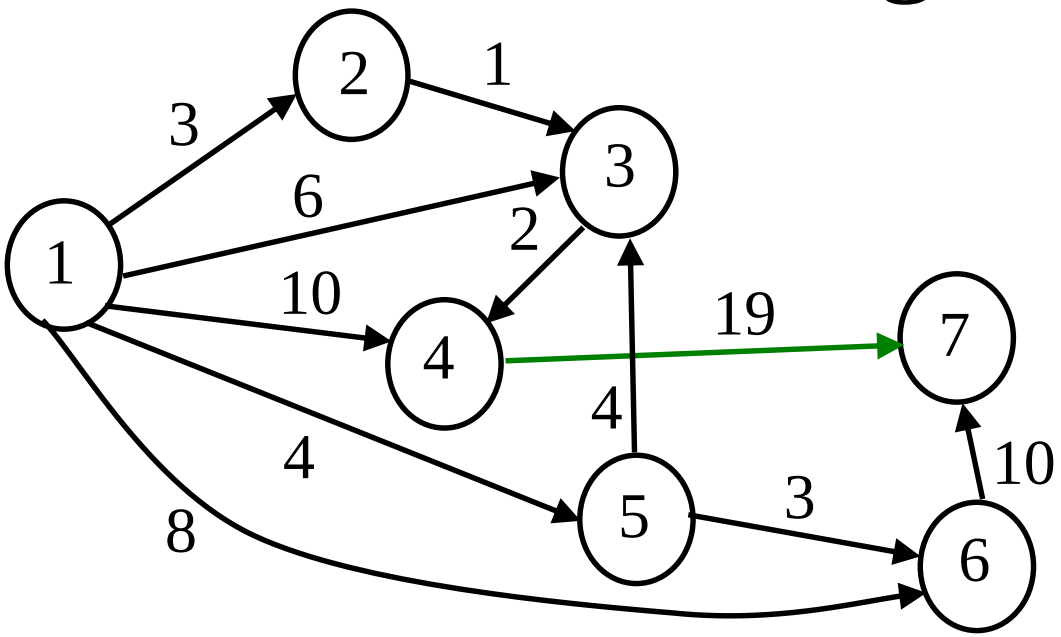
Esercizi su Algo Dijkstra (18)



$S = \{1, 2, 3, 5\}$

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	0	6	3
5	1	4	1
6	0	7	5
7	0	inf	1

Esercizi su Algo Dijkstra (19)

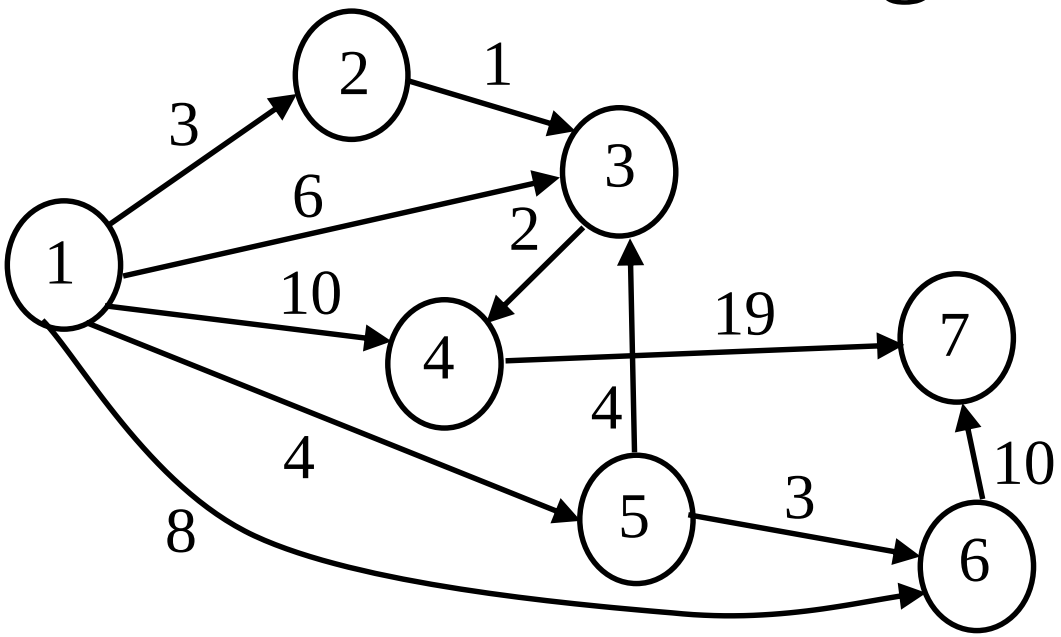


$S = \{1, 2, 3, 5\}$

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	0	6	3
5	1	4	1
6	0	7	5
7	0	inf	1

dariano,pacciarelli@dia.uniroma3

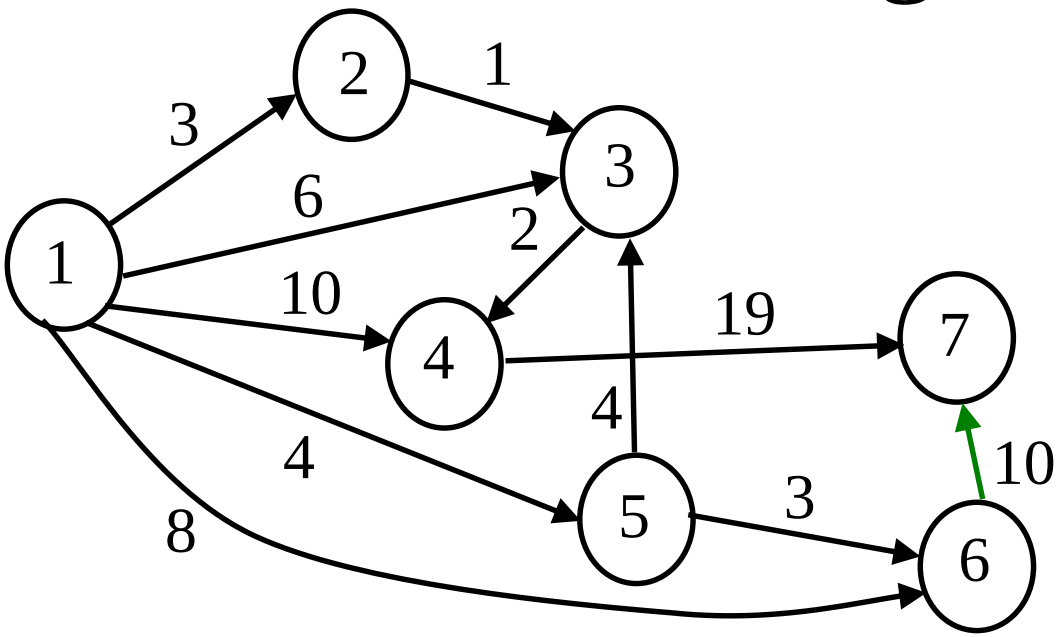
Esercizi su Algo Dijkstra (20)



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

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	1	6	3
5	1	4	1
6	0	7	5
7	0	25	4

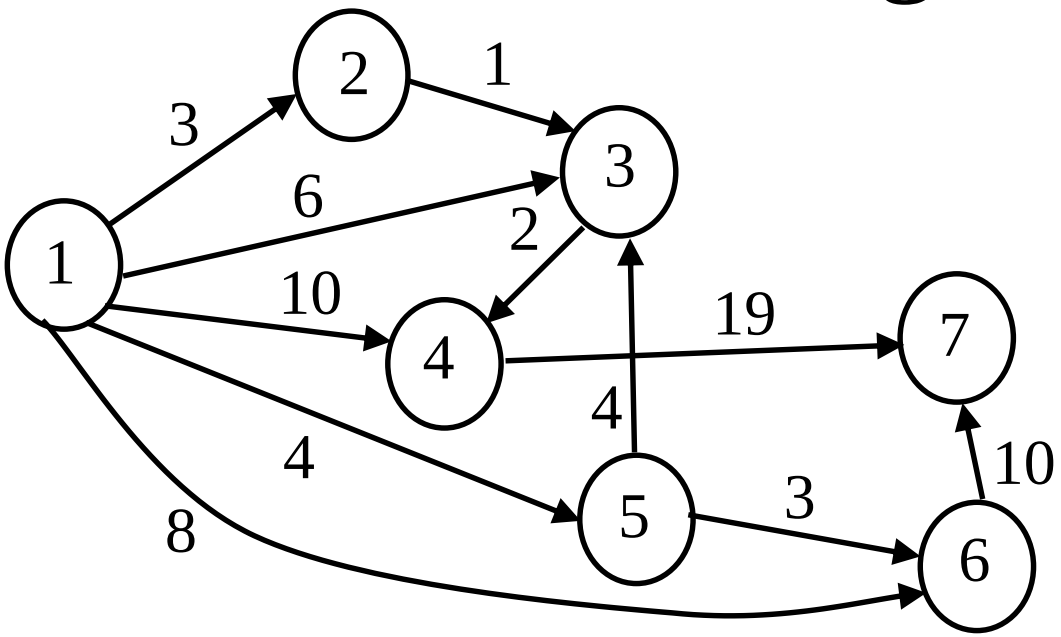
Esercizi su Algo Dijkstra (21)



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

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	1	6	3
5	1	4	1
6	0	7	5
7	0	25	7

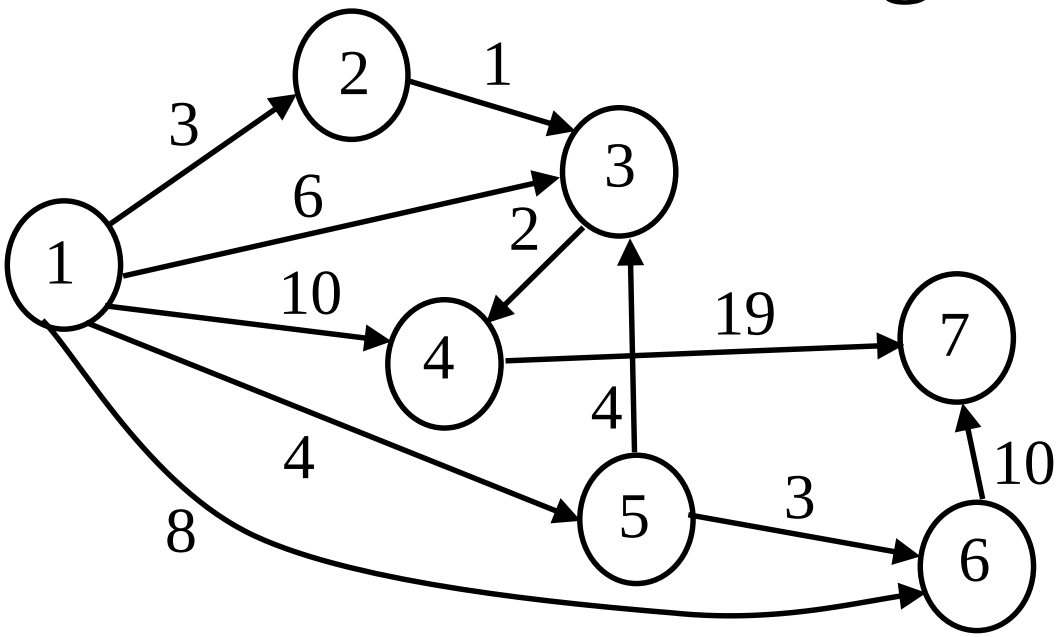
Esercizi su Algo Dijkstra (22)



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

nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	1	6	3
5	1	4	1
6	1	7	5
7	0	17	6

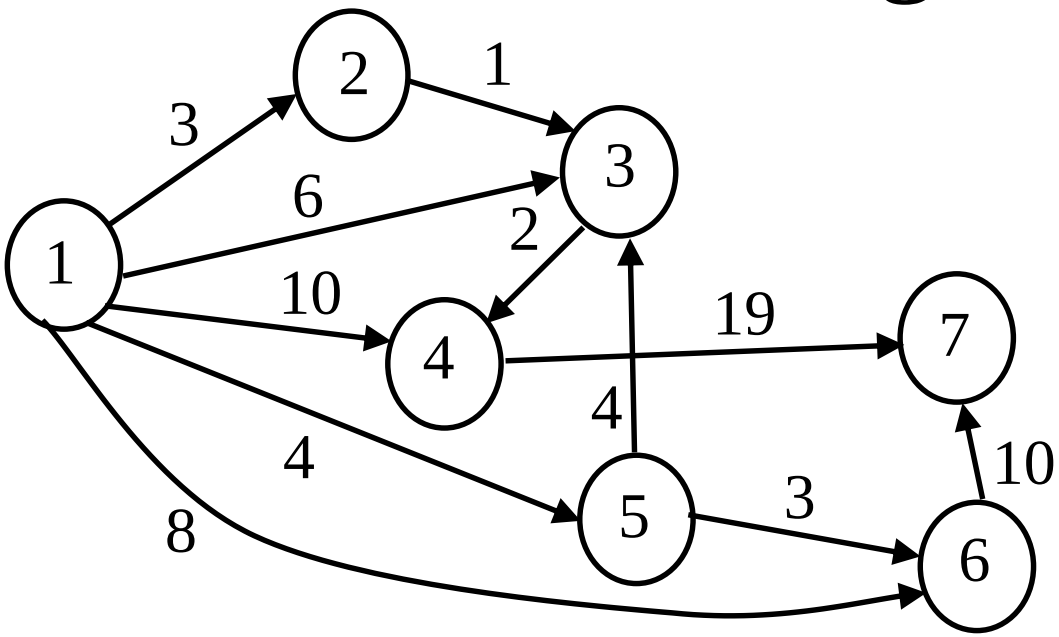
Esercizi su Algo Dijkstra (23)



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

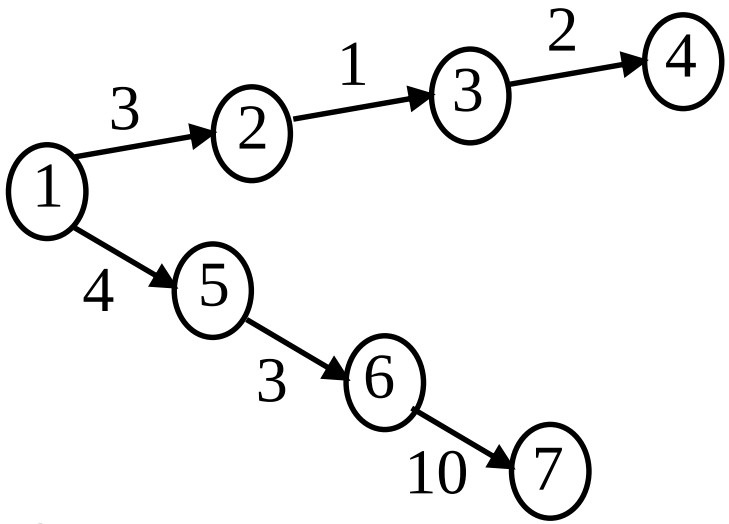
nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	1	6	3
5	1	4	1
6	1	7	5
7	1	17	6

Esercizi su Algo Dijkstra (24)



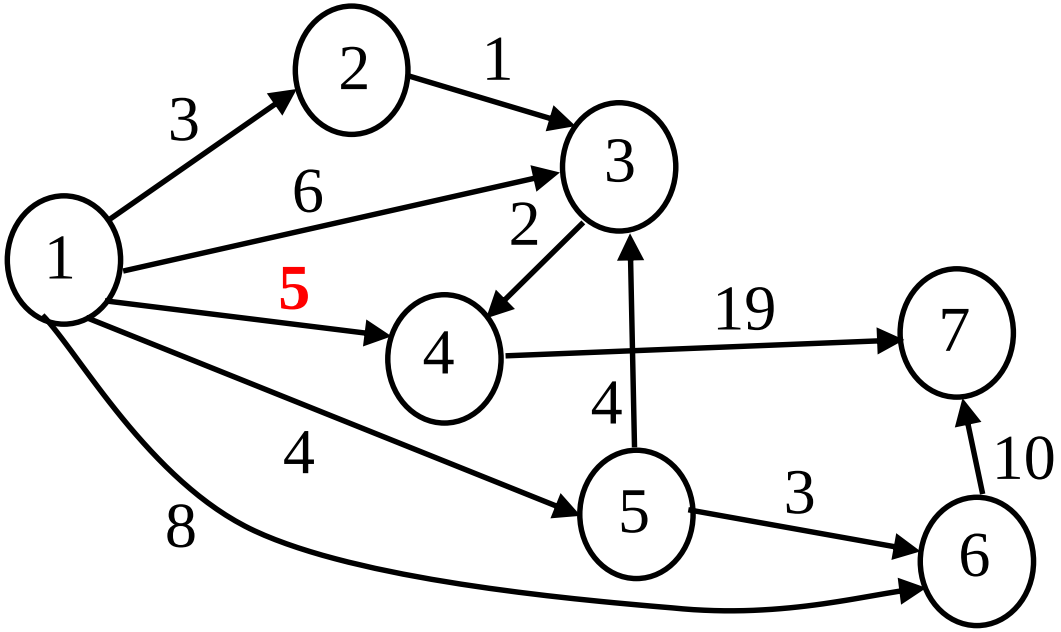
nodo	flag (V)	distanza (D)	predecessore
1	1	0	nessuno
2	1	3	1
3	1	4	2
4	1	6	3
5	1	4	1
6	1	7	5
7	1	17	6

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

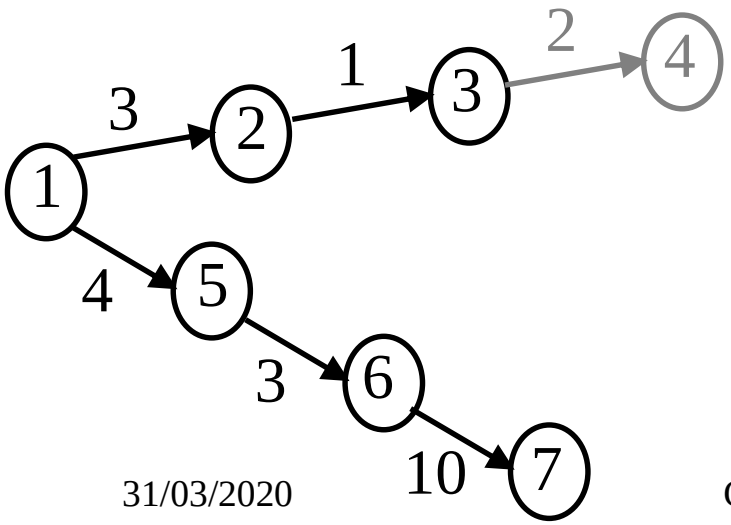


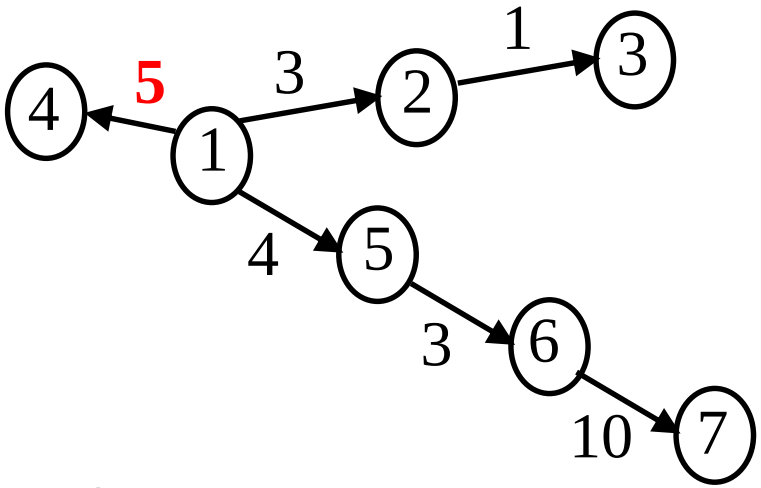
dariano,pacciarelli@dia.uniroma3

Esercizi su Algo Dijkstra (25)



Supponendo che il peso dell'arco (1,4) diventi 5: dire se e come cambia nell'albero dei cammini orientati minimi





dariano,pacciarelli@dia.uniroma3

Esercizi su Algo Dijkstra (26)

La produzione di ceramica può effettuarsi con una bicottura che prevede:
(1) preparazione grezzo crudo, (2) I cottura, (3) pittura terracotta, (4) II cottura.
La monocottura evita la fase di I cottura e prevede la pittura del grezzo crudo.
Un'azienda ceramica che produce vasi dispone di:

· 5 operai per la fase (1) pagati a ora che producono 1 vaso in 30 minuti al costo di 2 € (1A) oppure in 20 minuti al costo di 3 € (1B);

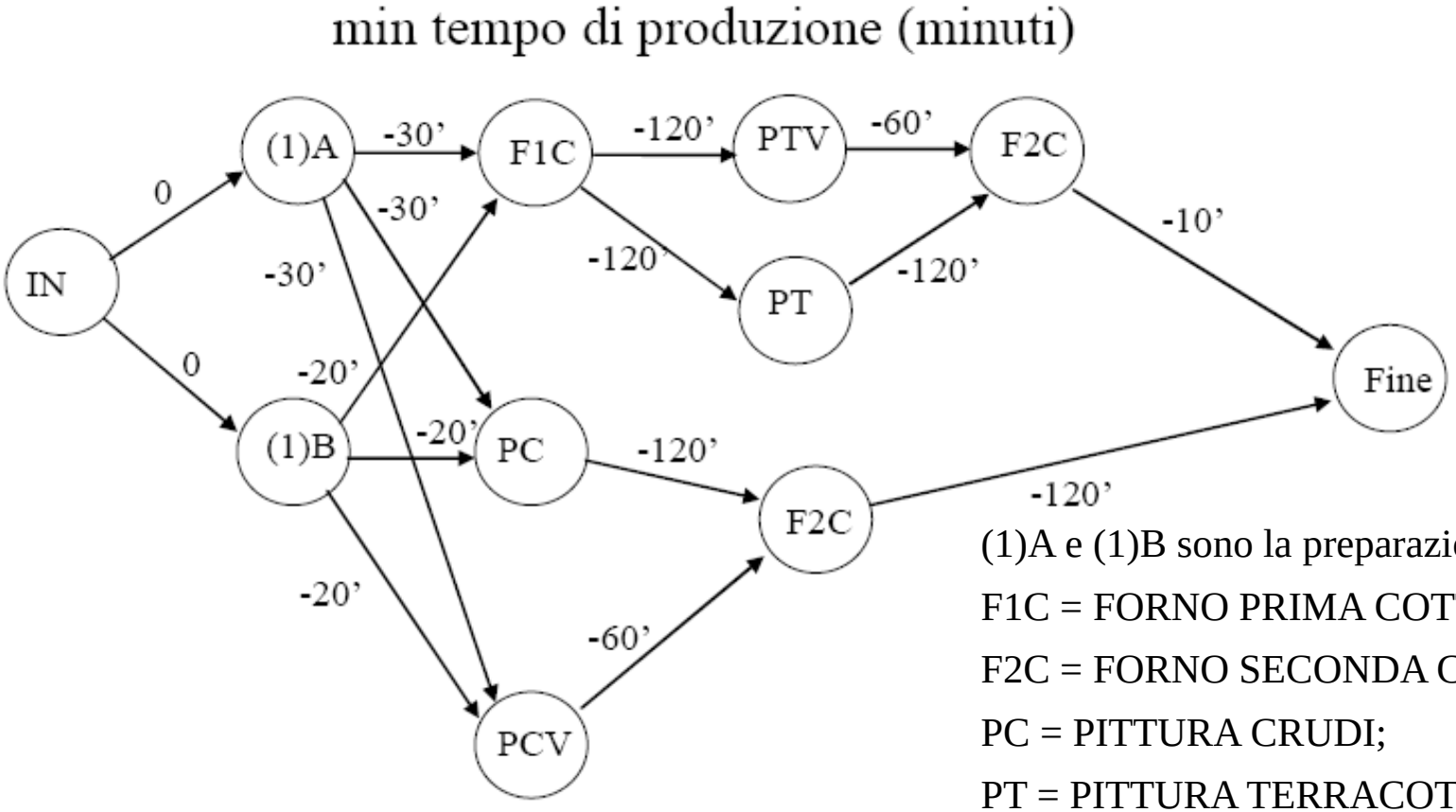
· un forno per I cottura (F1C) che cuoce un vaso crudo in 2 ore al costo di 0,9 €/vaso;

· un forno per II cottura (F2C) che cuoce vasi crudi in 2 ore al costo di 1,3 €/vaso e terrecotte in 10 minuti al costo di 0,3 €/vaso.

Per la pittura l'azienda si rivolge ad artisti esterni pagati a cottimo: nella modalità *standard* un pittore prende 6 €/vaso per la pittura dei vasi crudi (PC) e 5 €/vaso per la pittura di una terracotta (PT) e restituisce il vaso dipinto dopo 2 ore.
Nella modalità *veloce* (PCV e PTV) viene restituito il vaso dipinto dopo 1 ora con un aumento di costo del 20%. Si vuole determinare la modalità di produzione di **un vaso** a costo minimo e la modalità di produzione di un vaso nel tempo minimo.

Esercizi su Algo Dijkstra (27)

Il problema di determinare la modalità di produzione di un vaso nel tempo minimo si può formulare come problema di percorso minimo sul digrafo che segue.



(1)A e (1)B sono la preparazione grezzo crudo

F1C = FORNO PRIMA COTTURA;

F2C = FORNO SECONDA COTTURA

PC = PITTURA CRUDI;

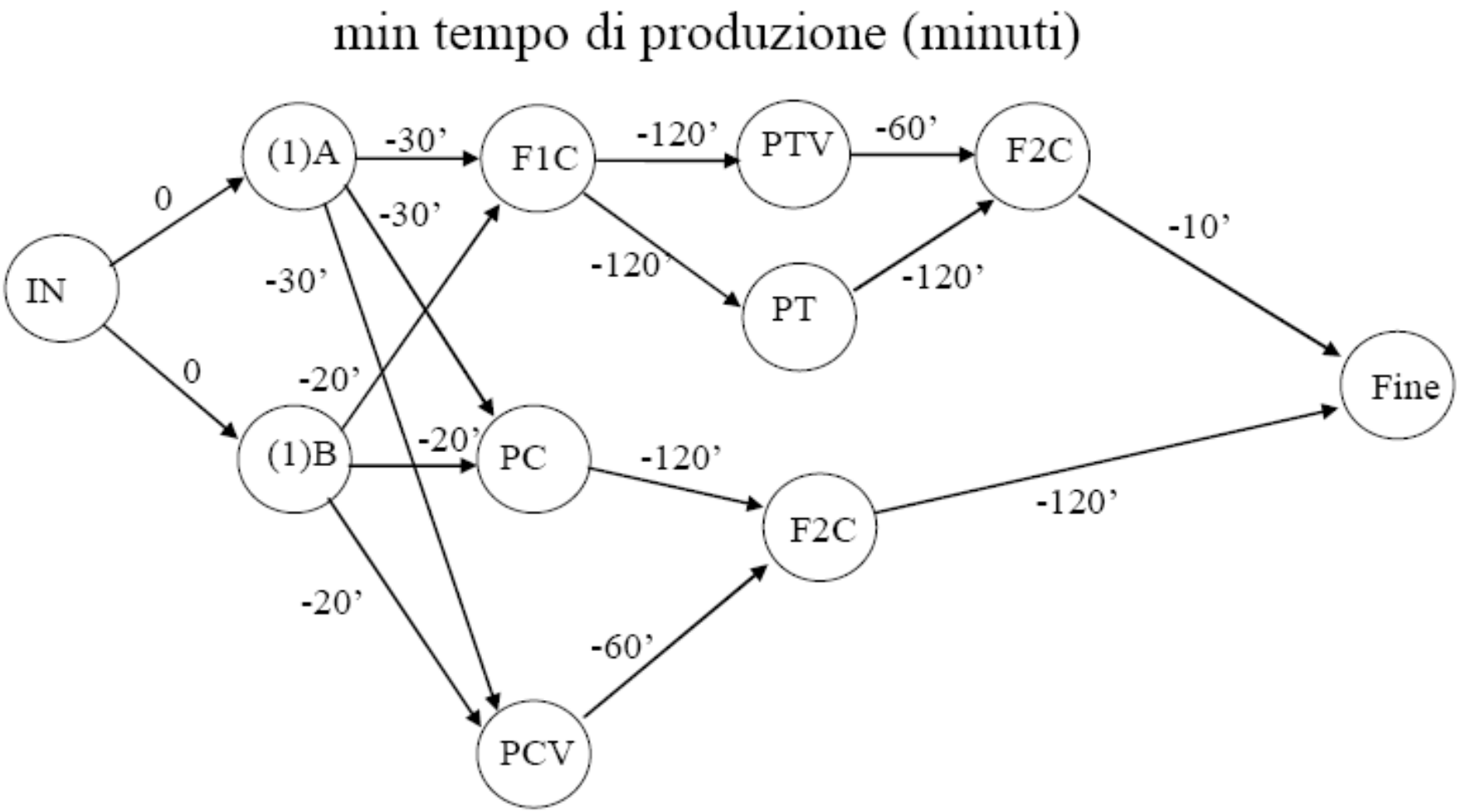
PT = PITTURA TERRACOTTA

PCV = PITTURA CRUDI VELOCE

PTV = PITTURA TERRACOTTA VELOCE

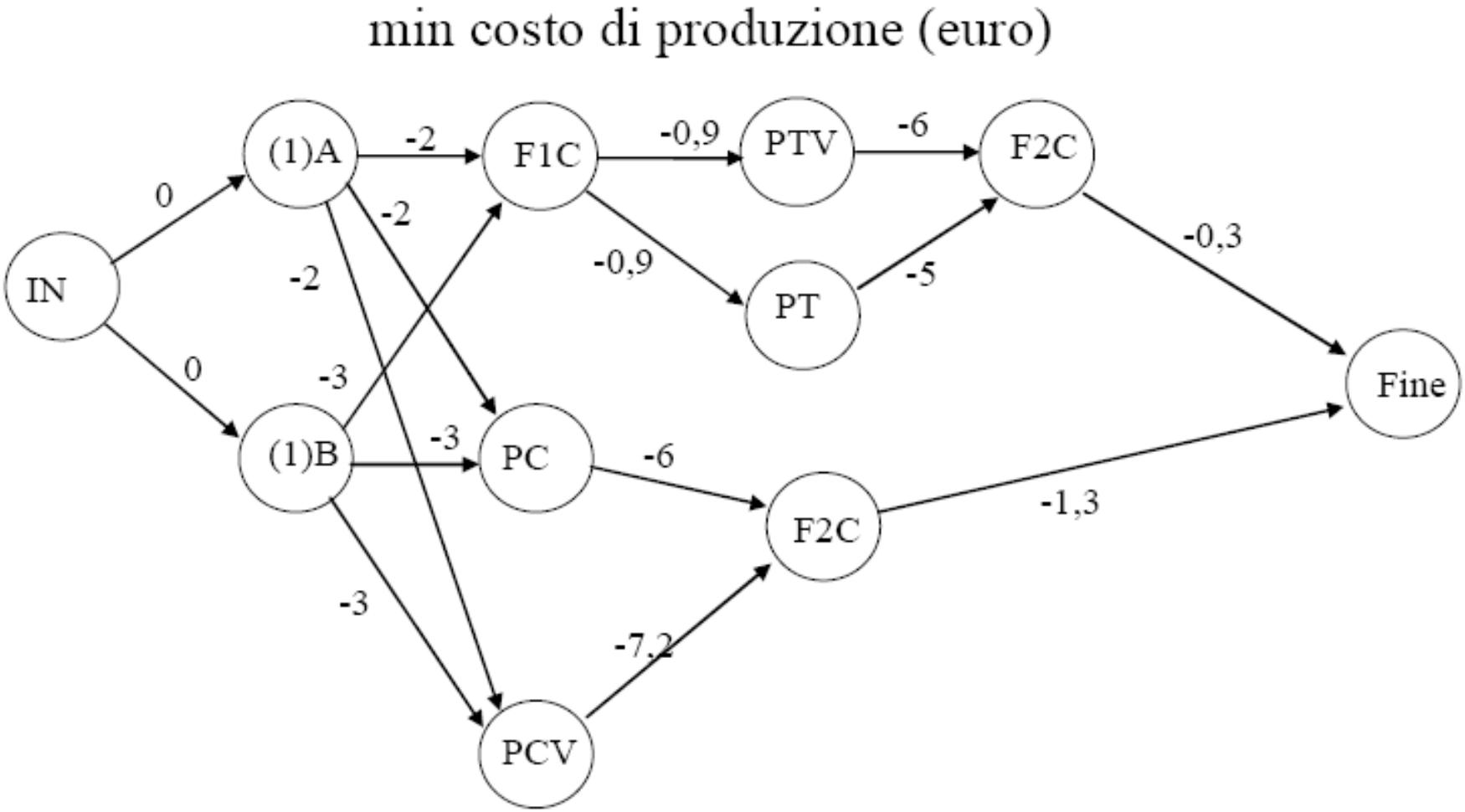
Esercizi su Algo Dijkstra (28)

Determinare la modalità di produzione di un vaso nel tempo minimo:



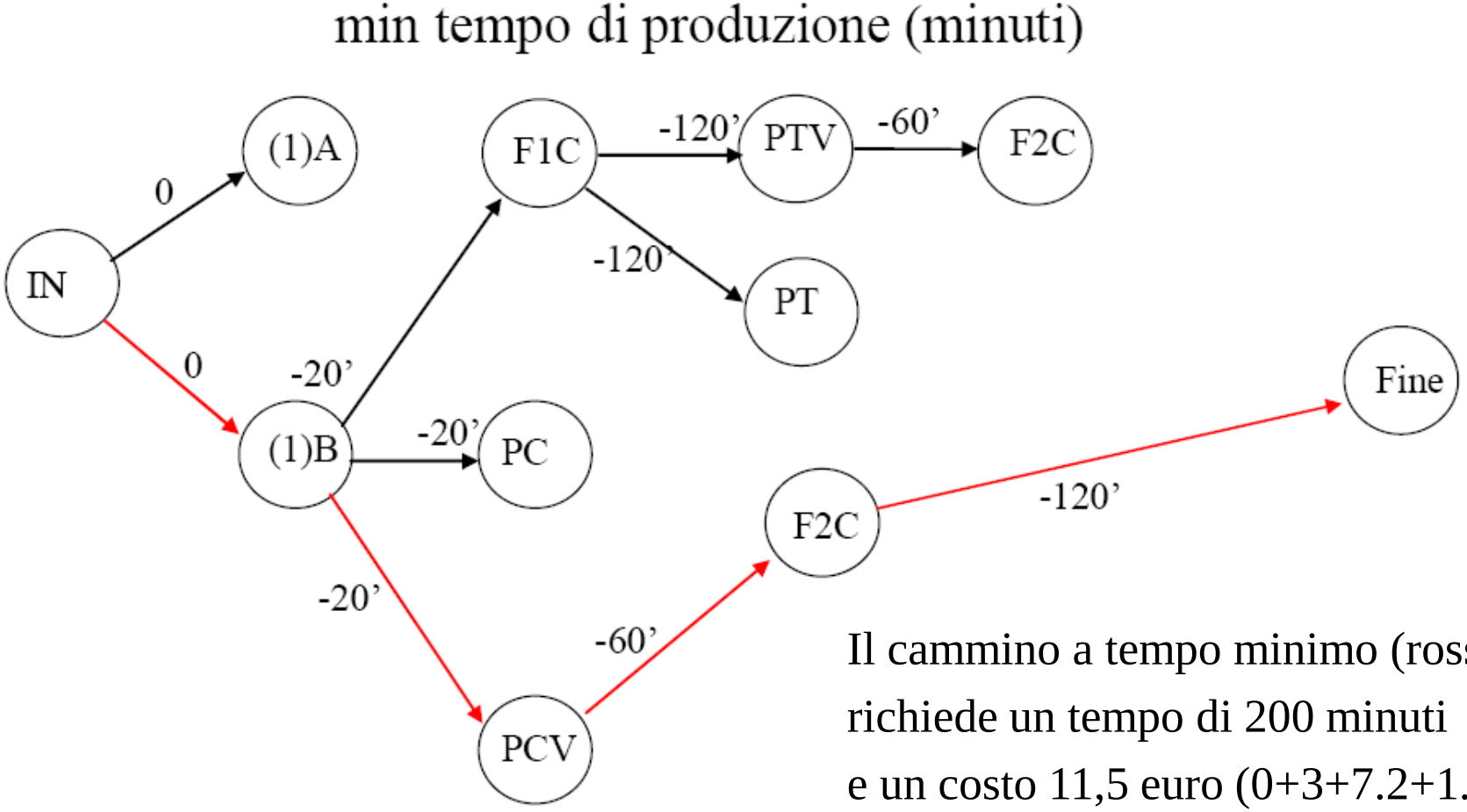
Esercizi su Algo Dijkstra (29)

Determinare la modalità di produzione di un vaso a costo minimo:



Esercizi su Algo Dijkstra (30)

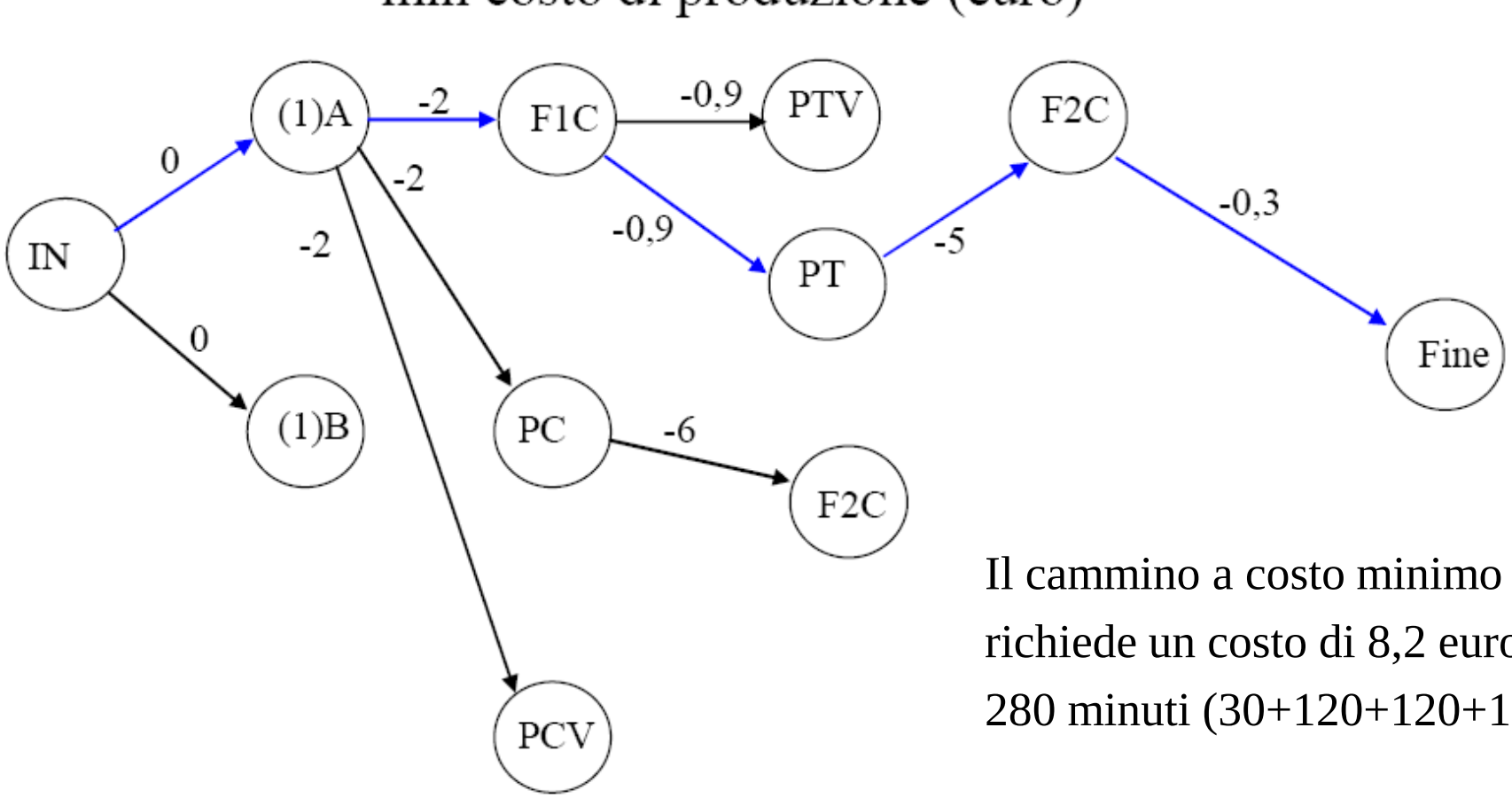
Cambiando di segno i pesi su gli archi del digrafo e risolvendo con l'algoritmo di Dijkstra si ottiene questo albero dei cammini minimi:



Il cammino a tempo minimo (rosso) richiede un tempo di 200 minuti e un costo 11,5 euro (0+3+7.2+1.3)

Esercizi su Algo Dijkstra (31)

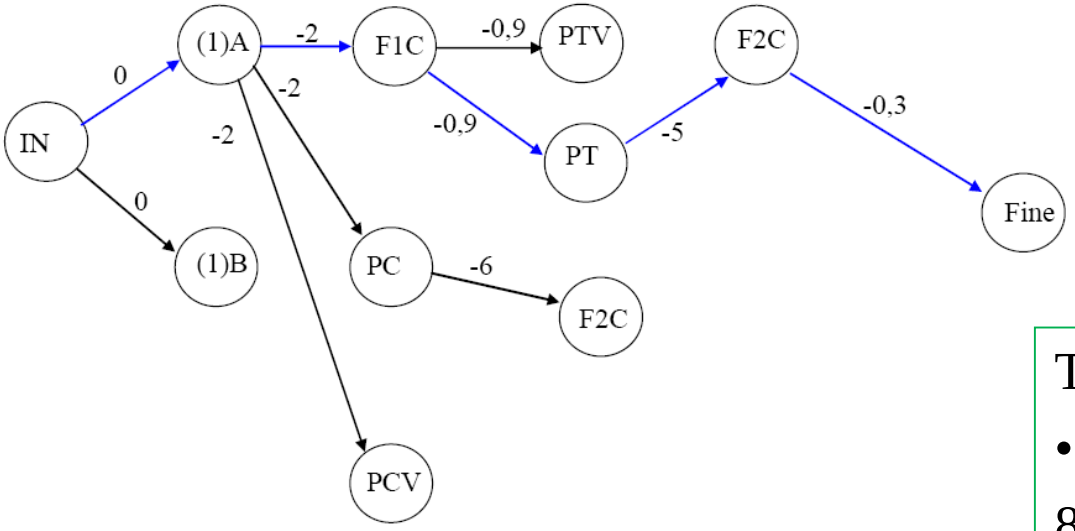
Cambiando di segno i pesi su gli archi del digrafo e risolvendo con l'algoritmo di Dijkstra si ottiene questo albero dei cammini minimi:



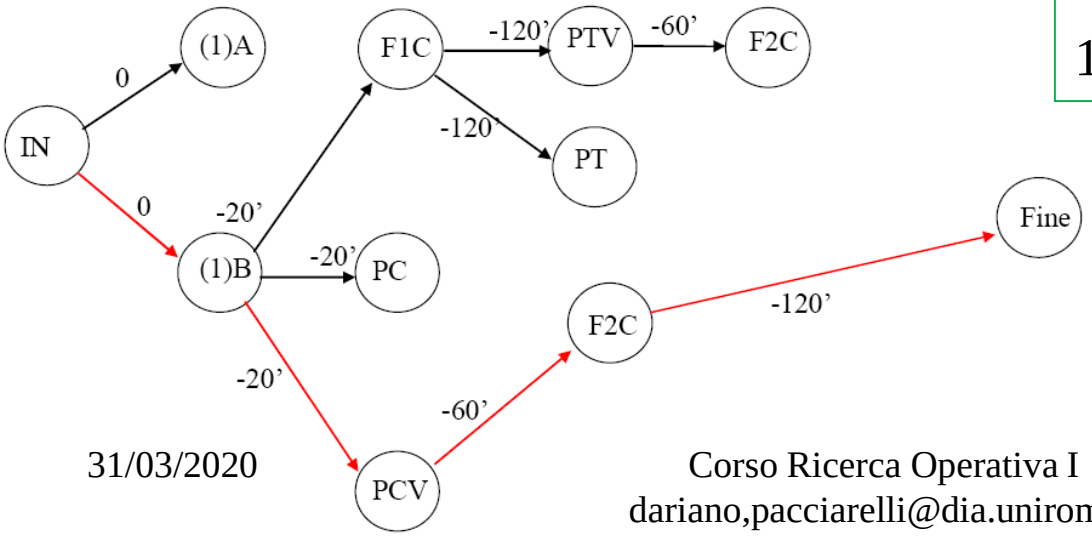
Il cammino a costo minimo (blu) richiede un costo di 8,2 euro e 280 minuti (30+120+120+10).

Esercizi su Algo Dijkstra (32)

min costo di produzione (euro)



min tempo di produzione (minuti)



Trade-off tra costi e durate:

- Min costi di produzione:

8,2 euro e 280 minuti

- Min tempi di produzione:

11,5 euro e 200 minuti

Esercizi su Algo Dijkstra (33)

La tabella mostra i dati di un digrafo con 14 archi e 9 nodi:

Archi	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(2,3)	(2,6)	(3,8)	(4,5)	(5,7)	(6,8)	(7,9)	(8,9)
Peso	2	10	2	1	3	6	9	3	11	4	1	5	10	12

Quesiti: Considerando gli archi del digrafo:

- Applicare l'algoritmo di Dijkstra per calcolare l'albero dei cammini minimi dal nodo 1 verso tutti gli altri nodi del digrafo.
- Sull'albero dei cammini minimi, calcolare il peso complessivo di tutti gli archi (appartenenti all'albero dei cammini minimi).
- Calcolare la lunghezza del percorso tra il nodo 1 ed il nodo 9.

Esercizi su Algo Dijkstra (34)

La tabella mostra i dati di un digrafo con 14 archi e 9 nodi:

Archi	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(2,3)	(2,6)	(3,8)	(4,5)	(5,7)	(6,8)	(7,9)	(8,9)
Peso	2	10	2	1	3	6	9	3	11	4	1	5	10	12

Risposte:

- L'ordine in cui vengono fissati i flag dei nodi è 1, 5, {2, 4, 7}, 6, 8, 3, 9. Avendo indicato tra graffe i nodi che possono essere raggiunti alla stessa distanza (ovvero con distanza 2 da 1). L'albero dei cammini minimi è composto dai seguenti archi (1,2), (1,3), (1,4), (1,5), (1,6), (5,7), (6,8), (7,9).
- Il peso complessivo dell'albero è pari a 34.
- Il percorso minimo tra il nodo 1 e il nodo 9 è pari a 12.

Esercizi su Algo Dijkstra (35)

La tabella mostra i dati di un grafo con 14 lati e 9 vertici:

Lati	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(2,3)	(2,6)	(3,8)	(4,5)	(5,7)	(6,8)	(7,9)	(8,9)
Peso	2	10	2	1	3	6	9	3	11	4	1	5	10	12

Quesiti: Considerando il grafo ottenuto dal digrafo precedente:

- Applicare l'algoritmo di Prim-Dijkstra per calcolare l'albero ricoprente di costo minimo (partendo dal vertice 1).
- Sull'albero ricoprente, calcolare il peso complessivo di tutti i suoi lati (appartenenti all'albero ricoprente).
- Calcolare la lunghezza del percorso tra il vertice 1 e il vertice 9.

Esercizi su Algo Dijkstra (36)

La tabella mostra i dati di un grafo con 14 lati e 9 vertici:

Lati	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(2,3)	(2,6)	(3,8)	(4,5)	(5,7)	(6,8)	(7,9)	(8,9)
Peso	2	10	2	1	3	6	9	3	11	4	1	5	10	12

Risposte:

- L'ordine in cui vengono fissati i flag dei vertici è 1, 5, 7, {2, 4}, 6, 8, 3, 9. L'albero ricoprente di costo minimo è composto dai seguenti lati (1,2),(1,4),(1,5),(1,6),(2,3),(5,7),(6,8),(7,9). Si osservi come al posto del lato (1,6) si possa selezionare il lato (2,6) ottenendo comunque una soluzione ottima che segue esattamente lo stesso ordinamento dei flag.
- Il peso complessivo dell'albero (di costo minimo) è pari a 33.
- Il percorso tra il vertice 1 e il vertice 9 è pari a 12.

Esercizi su Algo Dijkstra (37)

P1: La tabella mostra i dati di un digrafo con 14 archi e 9 nodi

Archi	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(2,3)	(2,6)	(3,8)	(4,5)	(5,7)	(6,8)	(7,9)	(8,9)
Peso	2	10	2	1	3	6	9	3	11	4	1	5	10	12

P2: La tabella mostra i dati di un grafo con 14 lati e 9 vertici

Lati	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(2,3)	(2,6)	(3,8)	(4,5)	(5,7)	(6,8)	(7,9)	(8,9)
Peso	2	10	2	1	3	6	9	3	11	4	1	5	10	12

Ulteriore Quesito:

In cosa differiscono i due alberi e i due percorsi tra il nodo/vertice 1 e il nodo/vertice 9 calcolati per i problemi P1 e P2? Perché?

dariano,pacciarelli@dia.uniroma3

Esercizi su Algo Dijkstra (38)

P1: La tabella mostra i dati di un digrafo con 14 archi e 9 nodi

Archi	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(2,3)	(2,6)	(3,8)	(4,5)	(5,7)	(6,8)	(7,9)	(8,9)
Peso	2	10	2	1	3	6	9	3	11	4	1	5	10	12

P2: La tabella mostra i dati di un grafo con 14 lati e 9 vertici

Lati	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(2,3)	(2,6)	(3,8)	(4,5)	(5,7)	(6,8)	(7,9)	(8,9)
Peso	2	10	2	1	3	6	9	3	11	4	1	5	10	12

Risposta:

L'albero calcolato dall'algoritmo di Prim-Dijkstra (soluzione ottima del problema dell'albero ricoprente di costo minimo) ha un peso minore rispetto all'albero (dei percorsi minimi) calcolato dall'algoritmo di Dijkstra.

- I due algoritmi, pur risolvendo due problemi diversi, ottengono lo stesso percorso per congiungere i nodi/vertici 1 e 9 nel digrafo/grafo.