Grafi e algoritmo di Dijkstra

Matteo Ferrara

Dipartimento di Informatica - Scienza e Ingegneria

matteo.ferrara@unibo.it

Grafi (1)

Definizione

Un **grafo** $G = (V, E)$ è costituito da:

- un insieme di elementi detti *vertici*, denominato V;
- un insieme di elementi detti *archi*, denominato E, tale che $E \subset V \times V$.

L'insieme degli archi rappresenta quindi un sottoinsieme del prodotto cartesiano dei vertici. E è quindi un insieme di coppie di vertici (ordinate), che indicano in quale modo i vertici del grafo siano connessi fra loro.

In particolare:

- se gli archi hanno una direzione, il grafo è detto *diretto* o orientato
- se gli archi non hanno una direzione, il grafo è detto *indiretto* o non orientato

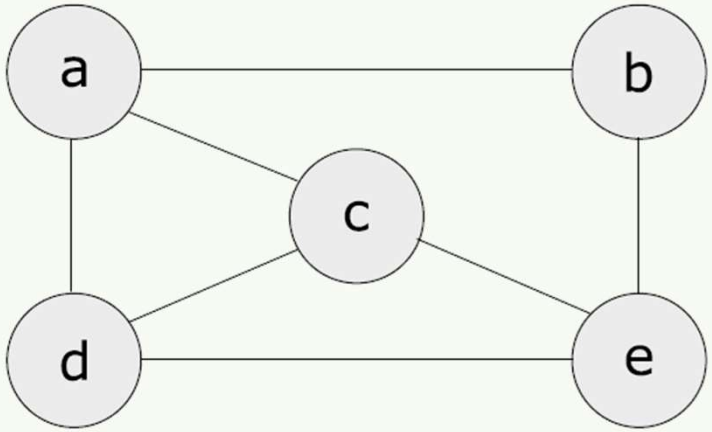
Grafi (2)

Esempio

Consideriamo i seguenti insiemi V ed E:

$V = \{a, b, c, d, e\}$
$E = \{(a, b), (a, c), (a, d), (b, e), (c, d), (c, e), (d, e)\}$

Supponendo che gli archi non abbiano un'orientazione, il grafo $G=(V, E)$ può essere rappresentato come segue:



Grafi (3)

- **Adiacenza**: due vertici v_i e v_j si dicono *adiacenti* se sono connessi da un arco, ovvero se $(v_i, v_j) \in E$.
- **Cammino**: un cammino rappresenta una sequenza ordinata di vertici $v_1, v_2, \ldots, v_k$ tale che, per ogni coppia di vertici consecutivi (v_i, v_{i+1}), tali vertici sono adiacenti.
- **Predecessore**: nella rappresentazione di un cammino P tra due vertici, si dice che v_i è il *predecessore* di v_j se v_i precede v_j in P.
- **Grafo pesato**: un grafo $G=(V,E)$ si dice *pesato* se ad ogni suo arco è associato un peso $w \in \mathbb{N}^+$. Il peso associato all'arco (v_i,v_j) si indica con $w_{i,j}$ e rappresenta il costo di attraversamento dell'arco (eventualmente orientato), ovvero il costo da pagare per spostarsi dal vertice v_i al vertice v_j. Un grafo pesato è quindi costituito dagli insiemi dei vertici V, degli archi E e da una funzione peso $\omega : E \rightarrow \mathbb{N}^+$.
- **Costo di un cammino**: in un grafo pesato, il *costo di un cammino* $P=(v_1,v_2, \ldots, v_k)$ consiste nella somma dei pesi coinvolti per muoversi dal primo nodo (*sorgente*) all'ultimo nodo (*destinazione*), seguendo gli archi inclusi nel cammino.

Grafi (4)

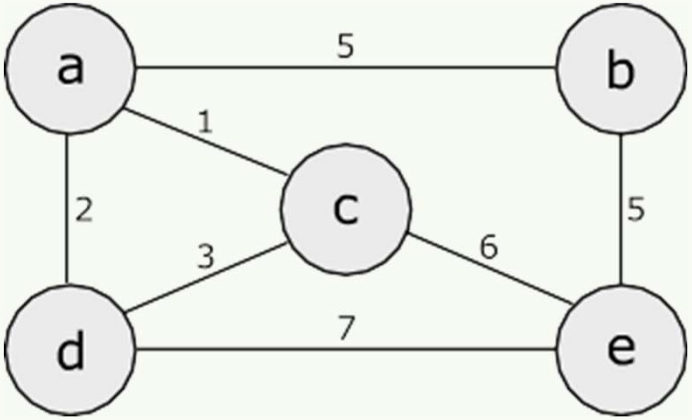
Esempio

Consideriamo il grafo pesato sottostante:

Alcuni possibili cammini:

$(a, b, e), (a, c, e), (a, d, c, e) \ldots$

Il **costo del cammino** (a, b, e) è 10.
Il **costo del cammino** (a, c, e) è 7.
Il **costo del cammino** (a, d, c, e) è 11.

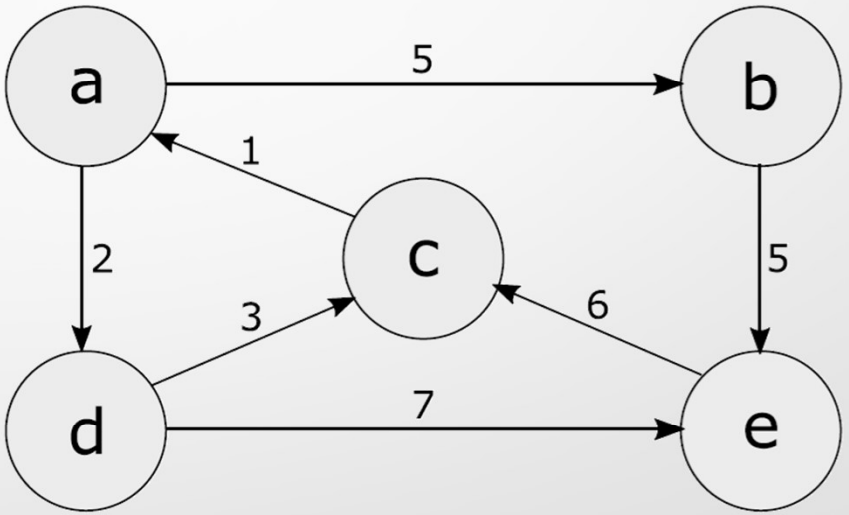


Matrice di adiacenza (1)

- Matrice di dimensione $n \times n$.
- Ogni cella rappresenta l'adiacenza tra il vertice corrispondente all'indice di riga e quello corrispondente all'indice di colonna.
- La cella contiene 0 (o altro simbolo apposito) se i due vertici non sono connessi.
- Se la cella contiene un valore diverso, tale valore rappresenta il peso dell'arco che lega i due vertici.
- Per grafi non orientati, la matrice di adiacenza è simmetrica.
- Per verificare l'adiacenza di due nodi dati, basta leggere la corrispondente cella della matrice.
- Questa rappresentazione è conveniente per grafi fortemente connessi.

Matrice di adiacenza (2)

	a	b	c	d	e
a	0	5	0	2	0
b	0	0	0	0	5
c	1	0	0	0	0
d	0	0	3	0	7
e	0	0	6	0	0



Lettura dei dataset

Nel materiale dell'esercitazione sono disponibili alcuni grafi d'esempio. I dati all'interno di ciascun file sono rappresentati con il seguente formato:

Sorgente	Destinazione	Peso
1,	5,	11
1,	4,	3
…	…	…

```
Lettura del file I:\CorsiUniversitari\AlgoritmiStruttureDati\19_20\es5\grafo1.txt

Matrice di adiacenza:
        [1]     [2]     [3]     [4]     [5]
[1]      0       5       0       2       0
[2]      0       0       0       0       5
[3]      1       0       0       0       0
[4]      0       0       3       0       7
[5]      0       0       6       0       0
```

Algoritmo di Dijkstra (1)

Cammini minimi con sorgente singola

Dato un grafo pesato $G=(V,E)$, si definisce problema dei *cammini minimi con sorgente singola* la ricerca dei cammini di costo minimo che conducono da un nodo $s \in V$ detto *sorgente* a ciascun altro nodo del grafo.

L'**algoritmo di Dijkstra** è uno dei metodi maggiormente noti per la risoluzione di questo problema. Si assume in generale che:

- esista un cammino dalla sorgente ad ogni altro nodo del grafo;
- il grafo non contenga cicli di costo negativo.

Alcune implementazioni, inoltre, assumono per praticità che il peso associato ad ogni arco sia **maggiore di zero**.

Algoritmo di Dijkstra (2)

Algoritmo di Dijkstra $O(n^2)$

INPUT: $G=(V,E)$, $s \in V$ OUTPUT: $G_\pi=(V_\pi,E_\pi)$

1. **Initialize**(G, s)
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V_d$
4. **while** $Q \neq \emptyset$ **do**
5. $u \leftarrow$ **ExtractMin**(Q)
6. $S \leftarrow S \cup \{u\}$
7. **for each** v **in** ADJ(u) **do**
8. **Relax**(u, v)

V_d rappresenta l'insieme dei vertici etichettati per distanza minima dalla radice s ($d[v]$), ed è inizializzato dalla procedura **Initialize**.

Si consideri che ADJ(u) rappresenta l'insieme dei vertici adiacenti ad u.

S contiene i vertici il cui peso del cammino minimo da s è già stato determinato.

Algoritmo di Dijkstra (3)

Initialize $O(n)$

INPUT: $G=(V,E)$, $s \in V$

OUTPUT:

1. **for each** v **in** V **do**
2. $\quad d[v] \leftarrow \infty$
3. $\quad p[v] \leftarrow 0$
4. $d[s] \leftarrow 0$

Questa procedura inizializza l'array dei predecessori (p) e delle distanze minime dalla radice s (d). Se $|V| = n$, la procedura ha costo $O(n)$.

L'algoritmo di Dijkstra è basato su un **rilassamento**: in $d[v]$ si memorizza un **upper bound** al costo del cammino minimo da s a v (stima di cammino minimo).

Con $p[v]$ indichiamo il predecessore del vertice v.

Algoritmo di Dijkstra (4)

ExtractMin $O(n)$

INPUT: $Q=\{(v_1,d_1),\ldots,(v_n,d_n)\}$ OUTPUT: v_i, $Q'=Q\setminus(v_i,d_i)$

1. $vmin \leftarrow$ **null**
2. $dmin \leftarrow \infty$
3. **for each** (v,d) **in** Q **do**
4. **if** $d<dmin$ **then**
5. $dmin \leftarrow d$
6. $vmin \leftarrow v$
7. $Q \leftarrow Q\setminus(vmin,dmin)$
8. **return** $vmin$

Questa procedura restituisce il vertice v_i (tra quelli il cui cammino minimo da s non è stato ancora calcolato) con la distanza d_i minima da s.

Algoritmo di Dijkstra (5)

Relax $O(1)$

INPUT: $u,v \in V$

OUTPUT:

1. **if** $d[v] > d[u] + w_{u,v}$ **then**
2. $d[v] \leftarrow d[u] + w_{u,v}$
3. $p[v] \leftarrow u$

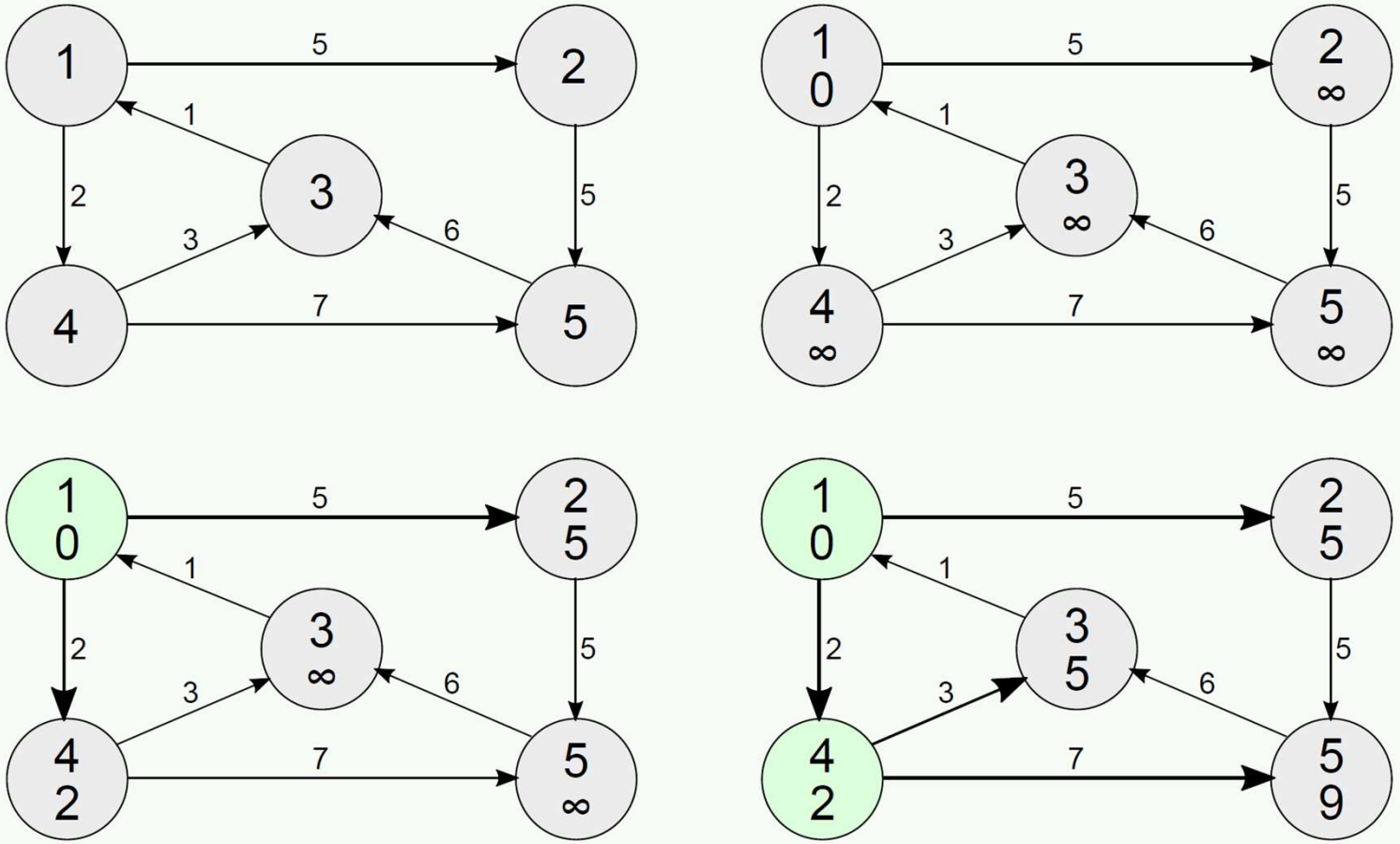
Questa procedura effettua il rilassamento dell'arco $(u,v) \in E$.

Considerato il costo attualmente noto per raggiungere il nodo v dalla sorgente s ($d[v]$), la procedura verifica se, attraversando l'arco (u,v), tale costo possa essere migliorato. In tal caso aggiorna l'array dei costi minimi e dei predecessori.

Algoritmo di Dijkstra (6)

Esempio - prima parte

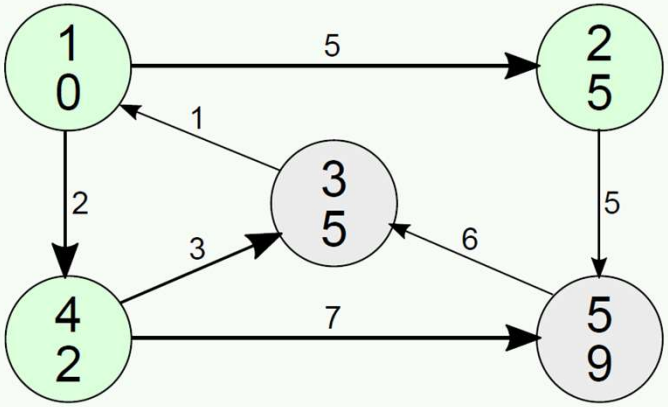
Consideriamo il seguente grafo orientato e pesato e supponiamo di eseguire l'algoritmo di Dijkstra con sorgente 1:

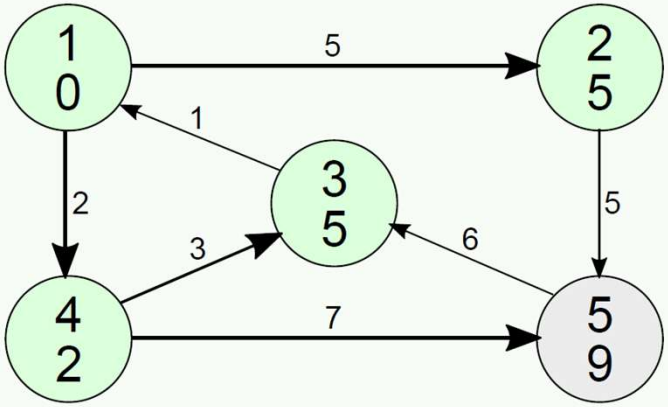


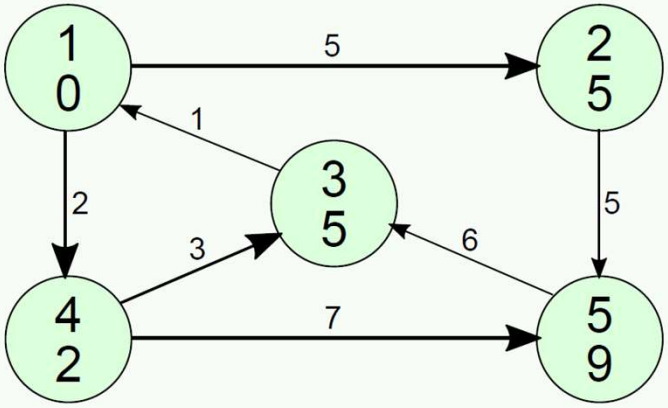
Algoritmo di Dijkstra (7)

Esempio - seconda parte

Al termine della procedura, l'insieme dei nodi visitati coincide con quello del grafo iniziale, mentre, tra gli archi del grafo originale, solo quattro sono stati inseriti nell'albero dei cammini minimi.







Algoritmo di Dijkstra (8)

- Per tenere traccia dei nodi che devono essere ancora visitati (e inclusi nell'insieme risultato) è sufficiente mantenere un unico array di valori booleani, che rappresenterà al contempo l'insieme risultato *S* e il suo complementare *Q*.
- Utilizzando una matrice di adiacenza per memorizzare la struttura del grafo, la lista di adiacenza (ADJ) del nodo *i*-esimo va ricercata sulla riga *i*-esima della matrice (per valori diversi da 0).

Obiettivo (1)

1. Implementare la **Initialize**, l'**ExtractMin** e la **Relax** dell'algoritmo di *Dijkstra* seguendo lo pseudo-codice visto nelle slide precedenti.
2. Testare l'algoritmo implementato con i dati contenuti nel materiale dell'esercitazione (*grafo1.txt*, *grafo2.txt* e *airports.txt*).

N.B.: per velocizzare lo sviluppo si consiglia di utilizzare il codice sorgente presente nel materiale dell'esercitazione.

```
Lettura del file I:\CorsiUniversitari\AlgoritmiStruttureDati\19_20\es5\grafo1.txt

Matrice di adiacenza:
        [1]     [2]     [3]     [4]     [5]
[1]      0       5       0       2       0
[2]      0       0       0       0       5
[3]      1       0       0       0       0
[4]      0       0       3       0       7
[5]      0       0       6       0       0


Tempo di esecuzione: 0.000000 sec

Src     Nodo    Costo    Percorso ottimo

1       2       5       1->2->
1       3       5       1->4->3->
1       4       2       1->4->
1       5       9       1->4->5->
```

Obiettivo (2)

```
Lettura del file I:\CorsiUniversitari\AlgoritmiStruttureDati\19_20\es5\grafo2.txt


Tempo di esecuzione: 0.000000 sec

Src     Nodo    Costo    Percorso ottimo

1       2       3       1->11->3->2->
1       3       2       1->11->3->
1       4       2       1->4->
1       5       3       1->11->5->
1       6       4       1->18->9->6->
1       7       3       1->7->
1       8       3       1->18->9->8->
1       9       2       1->18->9->
1       10      3       1->11->3->10->
1       11      1       1->11->
1       12      3       1->12->
1       13      2       1->11->13->
1       14      2       1->14->
1       15      3       1->15->
1       16      3       1->11->16->
1       17      2       1->17->
1       18      1       1->18->
1       19      3       1->11->13->19->
1       20      3       1->18->20->
1       21      3       1->11->3->21->
1       22      3       1->14->22->
1       23      3       1->4->23->
1       24      4       1->7->24->
1       25      4       1->18->9->26->25->
1       26      3       1->18->9->26->
```