Minimum Spanning Tree

Nei problemi `MST` (Minimum Spanning Tree o albero di supporto a peso minimo), dato un grafo non orientato $G = (V, E)$ con pesi w_{ij} per ogni $(i, j) \in E$, si vuole determinare tra tutti gli alberi di supporto $T = (V, E_T)$ del grafo quello con peso complessivo $\sum_{(i,j)\in E_T} w_{ij}$ minimo.

Modello per Esempio 3

Il problema nell'Esempio 3 può essere modellato come problema MST dove:

- i nodi del grafo rappresentano i computer;
- gli archi del grafo rappresentano i potenziali collegamenti diretti tra i diversi computer;
- i pesi w_{ij} degli archi rappresentano i costi dei collegamenti.

Risolvere il problema di mettere in rete i computer a un costo minimo per i collegamenti equivale a risolvere un problema MST sul grafo appena descritto.

Nel seguito vedremo delle procedure di risoluzione per questo problema.

Algoritmo greedy per MST

- **Inizializzazione** Si ordinino tutti gli $m = \mid E \mid$ archi del grafo in ordine non decrescente rispetto al peso, cioè

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{m-1}) \leq w(e_m).$$

  Si ponga $E_T = \emptyset$ e $k = 1$.
- **1** Se $\mid E_T \mid = \mid V \mid -1$, STOP e si restituisce l'albero $T = (V, E_T)$ come soluzione. Altrimenti si vada al Passo 2.
- **2** Se e_k non forma cicli con gli archi in E_T, si ponga $E_T = E_T \cup \{e_k\}$. Altrimenti si lasci E_T invariato.
- **3** Si ponga $k = k + 1$ e si ritorni al Passo 1.

Correttezza algoritmo

Supponiamo per assurdo che esista un albero di supporto $T' = (V, E_{T'})$ a peso minimo con peso inferiore a $T = (V, E_T)$, ovvero quello restituito dall'algoritmo greedy. Indichiamo con e_h l'arco a peso più piccolo tra quelli in $E_T \setminus E_{T'}$.

Andiamo ad aggiungere e_h a $E_{T'}$. In tal caso si forma esattamente un ciclo che deve contenere almeno un arco $e_r \notin E_T$ (gli archi in E_T formano un albero di supporto e quindi non possono generare cicli).

Si deve avere che $w_{e_r} \geq w_{e_h}$, altrimenti l'algoritmo greedy avrebbe selezionato e_r al posto di e_h (e_r *non* forma cicli con gli archi in E_T selezionati fino al momento in cui viene inserito e_h se e_h è il primo degli archi in E_T che non fanno parte di $E_{T'}$).

Ma allora possiamo togliere e_r e sostituirlo con e_h in modo da ottenere un albero di supporto a peso non superiore rispetto a T'.

Iterando questo ragionamento, possiamo sostituire tutti gli archi in $E_{T'} \setminus E_T$ con archi in E_T senza mai aumentare il peso di T' fino a riottenere l'albero T con peso non superiore a T', il che contraddice l'ipotesi iniziale.

Complessità algoritmo

L'operazione più costosa, almeno nel caso di grafi densi, ovvero con un numero di archi $O(\mid V \mid^2)$, è quella di ordinamento degli archi secondo il costo non decrescente, che richiede $O(\mid E \mid \log(\mid E \mid))$ operazioni.

Il ciclo che segue è di lunghezza non superiore a $\mid E \mid$ e, utilizzando opportune strutture dati, il numero di operazioni da esso richieste non supera $O(\mid E \mid \log(\mid E \mid))$.

Abbiamo quindi il seguente risultato.

L'algoritmo greedy ha complessità $O(\mid E \mid \log(\mid E \mid))$*.*

Esempio

Si risolva il problema `MST` sul grafo completo $G = (V, E)$ con $V = \{a_1, a_2, a_3, a_4\}$ e pesi

$$w_{a_1a_2} = 2 \;\; w_{a_1a_3} = 3 \;\; w_{a_1a_4} = 8 \;\; w_{a_2a_3} = 4 \;\; w_{a_2a_4} = 7 \;\; w_{a_3a_4} = 5$$

Risultato

Chiamiamo *foresta di supporto* di un grafo G un grafo parziale $F = (V, E_F)$ di G privo di cicli. In particolare, un albero di supporto è una foresta con una sola componente connessa.

Teorema *Indichiamo con* $(V_1, E_1), \ldots, (V_k, E_k)$ *le componenti connesse di una foresta di supporto* $F = (V, E_F)$ *del grafo* G*. Sia* (u, v) *un arco a peso minimo tra quelli con un solo estremo in* V_1*. Allora, tra tutti gli alberi di supporto a peso minimo tra quelli contenenti* $\cup_{i=1}^k E_i$*, ce ne è almeno uno che contiene* (u, v)*.*

Dimostrazione

Per assurdo si supponga che ci sia un albero di supporto $T = (V, E_T)$ con $E_T \supseteq \cup_{i=1}^{k} E_i$ e di peso minore rispetto a tutti quelli che contengono (u, v), dove ipotizziamo $u \in V_1$ e $v \notin V_1$.

Aggiungiamo (u, v) a E_T. In tal caso si forma esattamente un ciclo che oltre a (u, v), deve contenere anche un altro arco (u', v') con $u' \in V_1$ e $v' \notin V_1$ (altrimenti il ciclo che parte da $u \in V_1$ non potrebbe chiudersi).

Per come è definito (u, v), si deve avere che $w_{uv} \leq w_{u'v'}$.

Se ora togliamo (u', v') otteniamo un albero di supporto a peso non superiore a T, che contiene tutti gli archi in $\cup_{i=1}^{k} E_i$ (quindi anch'esso a peso minimo tra gli alberi che contengono tali archi) e che contiene anche (u, v), il che contraddice l'ipotesi iniziale.

Algoritmo MST-1

- **Inizializzazione** Scegli un nodo $v_1 \in V$. Poni $U = \{v_1\}$, $E_T = \emptyset$ e

$$c(v) = v_1 \quad \forall\, v \in V \setminus \{v_1\}.$$

(nel corso dell'algoritmo $c(v)$ conterrà, per i nodi in $V \setminus U$, il nodo in U più vicino a v).

- **1** Se $U = V$, STOP.
- **2** Seleziona

$$\bar{v} \in \arg\min_{v \in V \setminus U} w_{vc(v)}$$

($\bar{v}$ è il nodo in $V \setminus U$ più vicino a un nodo in U)

- **3** Poni $U \cup \{\bar{v}\}$ e $E_T = E_T \cup \{(\bar{v}, c(\bar{v}))\}$.
- **4** Per ogni $v \in V \setminus U$, se

$$w_{v\bar{v}} < w_{vc(v)}$$

poni $c(v) = \bar{v}$. Ritorna al Passo 1.

Correttezza algoritmo

Inizialmente abbiamo la foresta con $V_1 \equiv U = \{v_1\}$, $V_i = \{v_i\}$ $i = 2, \ldots, n$, con tutti gli $E_i = \emptyset$.

Alla prima iterazione si inserisce l'arco (v_1, v_{j_1}), $j_1 \neq 1$, a peso minimo tra quelli con un solo estremo in $U \equiv V_1$ e quindi, in base al teorema visto, tale arco farà parte dell'albero di supporto a peso minimo tra tutti quelli contenenti $\cup_{i=1}^{n} E_i = \emptyset$ e quindi in realtà l'albero di supporto a peso minimo tra tutti quelli del grafo.

Con l'aggiunta di questo arco, le due componenti connesse (V_1, E_1) e (V_{j_1}, E_{j_1}) si fondono in un'unica componente connessa con nodi $U = \{v_1, v_{j_1}\}$ e l'insieme di archi $E_T = \{(v_1, v_{j_1})\}$, mentre le altre componenti connesse non cambiano. Abbiamo cioè le componenti connesse

$$(U, E_T), \qquad (V_i, \emptyset) \quad i \in \{2, \ldots, n\} \setminus \{j_1\}.$$

Continua

Alla seconda iterazione andiamo a selezionare il nodo v_{j_2} e il relativo arco $(v_{j_2}, c(v_{j_2}))$ con il peso minimo tra tutti quelli con un solo estremo in U.

In base al teorema, l'arco $(v_{j_2}, c(v_{j_2}))$ farà parte di un albero di supporto a peso minimo *tra tutti quelli che contengono l'unione di tutti gli archi delle componenti connesse*, che si riduce a E_T.

Ma poichè E_T contiene il solo arco (v_1, v_{j_1}) che, come dimostrato precedentemente fa parte di un albero di supporto a peso minimo, possiamo anche dire che l'arco aggiunto farà parte di un albero di supporto a peso minimo tra tutti quelli possibili.

Continua

Quindi andiamo a inserire in U il nodo v_{j_2} e in E_T l'arco $(v_{j_2}, c(v_{j_2}))$ e avremo quindi le nuove componenti connesse

$$(U, E_T), \qquad (V_i, \emptyset) \quad i \in \{2, \ldots, n\} \setminus \{j_1, j_2\}.$$

Osservando che l'unione degli archi delle componenti connesse coincide sempre con E_T e che E_T contiene solo archi che fanno parte di un albero di supporto a peso minimo, possiamo iterare il ragionamento garantendo in questo modo che quando $U = V$ (e $\mid E_T \mid = \mid V \mid -1$), (V, E_T) sia un albero di supporto a peso minimo per il grafo G.

Complessità algoritmo

L'algoritmo MST-1 richiede un numero di operazioni $O(\mid V \mid^2)$.

Dimostrazione Il numero di iterazioni è pari a $\mid V \mid -1$. In ogni iterazione dobbiamo trovare un minimo tra un numero di valori pari a $\mid V \setminus U \mid$ e aggiornare (eventualmente) i valori c per i nodi in $V \setminus U$.

Dal momento che $\mid V \setminus U \mid \leq \mid V \mid$, abbiamo complessivamente un numero di operazioni pari a $O(\mid V \mid^2)$.

Nota bene

Si può dimostrare che questo algoritmo ha complessità ottima per `MST` almeno per grafi densi.

Infatti, per tali grafi non possiamo aspettarci di fare meglio di $O(\mid V \mid^2)$: la sola operazione di lettura dei dati di input (i pesi degli archi) richiede già $O(\mid V \mid^2)$.

Algoritmo MST-2

- **Inizializzazione** Poni $E_T = \emptyset$ e sia $\mathcal{C} = \{S_1, \ldots, S_n\}$ con $S_i = \{v_i\}$ per ogni $i = 1, \ldots, n$ ($\mathcal{C}$ nel corso dell'algoritmo conterrà la collezione di componenti connesse di (V, E_T) ed essendo inizialmente E_T vuoto, viene inizializzata con n componenti, una per ciascun nodo del grafo). Poni

$$componente[v_j] = S_j \quad j = 1, \ldots, n.$$

  ($componente[v]$ restituisce la componente connessa a cui appartiene il nodo v durante l'algoritmo).

- **1** Se $\mid \mathcal{C} \mid = 1$, allora STOP.
- **2** Per ogni $S_i \in \mathcal{C}$, poni $min[i] = \infty$.

- **3** Per ogni $(u, v) \in E$, siano $S_i = componente[u]$ e $S_j = componente[v]$. Se $i \neq j$, allora:
  - $w_{uv} < min[i] \Rightarrow shortest[i] = (u, v)$, $min[i] = w_{uv}$;
  - $w_{uv} < min[j] \Rightarrow shortest[j] = (u, v)$, $min[j] = w_{uv}$;

  (si noti che $shortest[i]$ indica l'arco a peso minimo tra quelli con un solo nodo in S_i).
- **4** Per tutti gli $S_i \in \mathcal{C}$ poni

$$E_T = E_T \cup \{shortest[i]\}.$$

- **5** Poni $\mathcal{C}$ pari all'insieme di componenti connesse di (V, E_T) e aggiorna i valori $componente[v]$ per ogni $v \in V$. Torna al Passo 1.

Correttezza e complessità

Non dimostriamo la correttezza dell'algoritmo (lo si può fare per esercizio tenendo conto che anche questa è basata sul teorema visto).

Vediamo invece la complessità dell'algoritmo.

L'algoritmo MST-2 richeide un numero di operazioni $O(\mid E \mid \log(\mid V \mid))$.

Dimostrazione

In ogni iterazione dell'algoritmo abbiamo:

- al Passo 3 un ciclo su tutti gli archi del grafo dove per ogni arco dobbiamo eseguire dei confronti e aggiornare (eventualmente) i valori $shortest$ per i due nodi dell'arco. Quindi, questo ciclo richiede un numero di operazioni $O(\mid E \mid)$;
- il Passo 4 con l'aggiunta di un numero di archi non superiore a $O(\mid V \mid)$ richiede un numero di operazioni di ordine non superiore a $O(\mid V \mid)$.

Osservando che lo sforzo per il Passo 4 è dominato da quello per il Passo 3 (nei grafi in cui sia presente almeno un albero di supporto si deve avere che il numero di archi non può essere inferiore a $\mid V \mid -1$), in una singola iterazione il numero di operazioni è dell'ordine di $O(\mid E \mid)$.

Numero iterazioni

Ci si arresta quando $|\mathcal{C}| = 1$. Quello che vogliamo mostrare è che $|\mathcal{C}|$, inizialmente pari a $|V|$, viene almeno dimezzato a ogni iterazione.

In effetti a ogni iterazione dell'algoritmo una componente connessa contiene *almeno* due componenti connesse dell'iterazione precedente, visto che ogni componente connessa dell'iterazione precedente è unita, con l'aggiunta dell'arco $shortest[j]$, a un'altra componente di tale iterazione.

Se a ogni iterazione dimezziamo (almeno) $|\mathcal{C}|$ arriveremo a $|\mathcal{C}| = 1$ in al più $\log(|V|)$ iterazioni, da cui il risultato che si voleva dimostrare.

Nota bene

Si noti che per grafi densi con $\mid E \mid = O(\mid V \mid^2)$ questa complessità è peggiore di quella di MST-1, ma se il numero di archi scende sotto l'ordine $O(\mid V \mid^2 / \log(\mid V \mid))$ l'algoritmo MST-2 ha prestazioni migliori di MST-1.

Nota bene -2

Tutti e tre gli algoritmi visti per il problema MST sono algoritmi costruttivi, senza revisione delle decisioni passate.

Infatti si parte sempre da una soluzione incompleta (il grafo con i nodi originari ma privo di archi) e a ogni iterazione si aggiungono uno o più archi, fino ad arrivare a costruire un albero di supporto.

Shortest Path

Nei problemi SHORTEST PATH (cammino a costo minimo) dato un grafo orientato $G = (V, A)$ con costo (distanza) d_{ij} per ogni $(i, j) \in A$ e dati due nodi $s, t \in V$, $s \neq t$, vogliamo individuare un cammino elementare orientato da s a t di costo minimo.

Modello per Esempio 4

Il problema nell'Esempio 4 può essere modellato come problema SHORTEST PATH dove:

- i nodi del grafo rappresentano le località;
- gli archi del grafo rappresentano le strade che collegano tra loro alcune località;
- i valori d_{ij} degli archi rappresentano i tempi di percorrenza delle strade.

Risolvere il problema di individuare il percorso con il minimo tempo di percorrenza tra due località fissate equivale a risolvere un problema SHORTEST PATH sul grafo appena descritto.

Nel seguito vedremo un paio di procedure di risoluzione per questo problema.

Algoritmi per SHORTEST PATH

Per questo problema presenteremo due algoritmi:

- Algoritmo di Dijkstra: valido solo se $d_{ij} \geq 0 \ \forall \ (i,j) \in A$. Restituisce i cammini minimi tra un nodo fissato $s \in V$ e tuuti gli altri nodi del grafo.
- Algoritmo di Floyd-Warshall: valido anche per distanze negative *a patto che non ci siano cicli di lunghezza negativa*. Restituisce i cammini minimi tra tutte le coppie di nodi del grafo *se il grafo non contiene cicli a costo negativo*. In quest'ultimo caso restituisce un ciclo a costo negativo.

Nel seguito si supporrà sempre che $d_{ij} = +\infty$ per ogni $(i,j) \notin A$ e indicheremo con n la cardinalità di V.

Algoritmo di Dijkstra

- **Inizializzazione** Poni

$$W = \{s\}, \quad \rho(s) = 0, \quad e(s) = -$$

e per ogni $y \in V \setminus \{s\}$

$$\rho(y) = d_{sy} \quad e(y) = s.$$

- **1** Se $W = V$, allora STOP, altrimenti vai al Passo 2.
- **2** Sia

$$x \in \arg\min\{\rho(y) \ : \ y \notin W\}.$$

- **3** Poni $W = W \cup \{x\}$ e per ogni $y \notin W$ aggiorna $e(y)$ e $\rho(y)$ come segue

$$e(y) = \begin{cases} e(y) & \text{se } \rho(y) \leq \rho(x) + d_{xy} \\ x & \text{altrimenti} \end{cases}$$

$$\rho(y) = \min\{\rho(y), \rho(x) + d_{xy}\}.$$

- **4** Ritorna al Passo 1.

Correttezza algoritmo

Se $d_{ij} \geq 0 \; \forall \; (i,j) \in A$, si dimostra che:

- per ogni $y \in V$, il valore $\rho(y)$ rappresenta a ogni iterazione la lunghezza del cammino minimo da s a y *passando solo attraverso nodi in* W, mentre in $e(y)$ è memorizzato il nodo che precede immediatamente y in tale cammino (per il nodo s, nodo di partenza, si usa l'etichetta $-$ per indicare che non è preceduto da altri nodi);
- quando il nodo x viene inserito in W al Passo 3, il valore $\rho(x)$ rappresenta la distanza minima tra s e x. Il cammino minimo può essere ricostruito procedendo a ritroso a partire dall'etichetta $e(x)$.

Dimostrazione

La dimostrazione dei due punti si fa per induzione.

Le due cose sono ovviamente vere inizialmente, quando $W = \{s\}$.

Supponiamo ora che siano vere a un'iterazione e mostriamo che sono vere anche a quella successiva, distinguendo tra i due punti.

Punto 1

Dato $y \notin W$, il cammino minimo da s a y può:

- non passare dal punto x e in tal caso, per l'ipotesi induttiva essere pari a $\rho(y)$;
- essere un cammino in cui y è preceduto immediatamente da x e quindi in tal caso avere lunghezza $\rho(x) + d_{xy}$.

Quindi, il cammino minimo da s a y passando solo attraverso nodi in W è di lunghezza pari a

$$\min\{\rho(y), \rho(x) + d_{xy}\}.$$

Nota bene

Non può accadere che il cammino minimo da s a y passante solo per nodi in W abbia x in una posizione che non è quella che precede immediatamente y, ovvero non può avere la seguente forma

$$s \to \cdots \to x \to \cdots \to t \to \cdots \to y,$$

con $t \in W$.

Infatti, essendo già $t \in W$, per l'ipotesi induttiva il cammino minimo da s a t (che non contiene x), è di lunghezza non maggiore al cammino

$$s \to \cdots \to x \to \cdots \to t,$$

e possiamo quindi sostituire quest'ultimo cammino con quello di lughezza $\rho(t)$, non passante per x.

Punto 2

Per l'ipotesi induttiva, il valore $\rho(x)$ è la lunghezza del cammino minimo da s a x *passando solo attraverso nodi in* W.

Ipotizziamo per assurdo che esista un cammino da s a x di lunghezza inferiore a $\rho(x)$ che passi attraverso nodi $\notin W$ e sia z il primo di tali nodi:

$$s \to \cdots \to z \to \cdots \to x.$$

Se interrompiamo tale cammino a z, otteniamo un cammino da s a z con le seguenti caratteristiche:

- passa solo attraverso nodi in W e quindi ha lunghezza non inferiore a $\rho(z)$;
- essendo tutte le distanze non negative, dobbiamo avere che tale cammino ha lunghezza non superiore alla lunghezza del cammino da s a x passando per z e quindi, per ipotesi, strettamente inferiore a $\rho(x)$.

Allora, dobbiamo avere $\rho(z) < \rho(x)$ che però contraddice la regola di scelta di x al Passo 2.

Complessità algoritmo

L'algoritmo di Dijkstra richiede un numero di operazioni $O(n^2)$*.*

Dimostrazione Ci sono n iterazioni. In ciascuna di queste si deve calcolare il minimo tra $\mid V \setminus W \mid \leq n$ valori e per ogni nodo in $V \setminus W$ confrontare due valori. Dunque, a ogni iterazione eseguiamo al più $O(n)$ operazioni e complessivamente eseguiamo $O(n^2)$ operazioni.

Quindi, l'algoritmo ha complessità polinomiale.

Esempio

Si applichi l'algoritmo di Dijkstra a un grafo $G = (V, A)$ con $V = \{a, b, c, d\}$, con nodo di partenza $s \equiv a$ e con la seguente matrice di distanze:

$$D = \begin{bmatrix} * & 3 & 12 & 16 \\ 9 & * & 18 & 7 \\ 5 & * & * & 3 \\ 8 & * & 1 & * \end{bmatrix}$$

Operazione di triangolazione

Data una matrice $n \times n$ di distanze R, per un dato $j \in \{1, \ldots, n\}$ chiamiamo *operazione di triangolazione* il seguente aggiornamento della matrice R:

$$R_{ik} = \min\{R_{ik}, R_{ij} + R_{jk}\} \quad \forall\, i, k \in \{1, \ldots, n\} \setminus \{j\}.$$

Algoritmo di Floyd-Warshall

- **Inizializzazione** Per $i \neq j$ poni $R_{ij} = d_{ij}$. Poni $R_{ii} = +\infty$ per ogni $i = 1, \ldots, n$. Definisci la matrice $n \times n$ con componenti E_{ij} inizialmente tutte pari a $-$. Poni $j = 1$.
- **1** Esegui l'operazione di triangolazione con j fissato e aggiorna E come segue

$$E_{ik} = \begin{cases} j & \text{se } R_{ik} > R_{ij} + R_{jk} \\ E_{ik} & \text{altrimenti} \end{cases}$$

- **2** Se $j = n$ o esiste $R_{ii} < 0$, allora STOP, altrimenti poni $j = j + 1$ e vai al Passo 1.

Terminazione

Nel caso di distanze $d_{ij} \geq 0$, la condizione di arresto $R_{ii} < 0$ non potrà mai verificarsi e si dimostra che i valori R_{ij} danno la lunghezza del cammino minimo da i a j per ogni $i \neq j$, mentre le etichette E_{ij} consentono di ricostruire tali cammini minimi.

Nel caso di distanze negative, se non interviene la condizione di arresto $R_{ii} < 0$, allora anche qui gli R_{ij} danno la lunghezza del cammino minimo da i a j.

Se invece a una certa iterazione si verifica la condizione $R_{ii} < 0$, questa indica la presenza di un ciclo a costo negativo nel grafo. In tal caso, anche ignorando la condizione di arresto $R_{ii} < 0$, non possiamo garantire che al momento della terminazione con $j = n$ gli R_{ij} diano la lunghezza del cammino minimo da i a j.

Complessità algoritmo

L'algoritmo di Floyd-Warshall richiede un numero di operazioni $O(n^3)$.

Dimostrazione Ci sono n iterazioni. In ciascuna di queste si deve eseguire un'operazione di triangolazione che richiede un numero di operazioni pari a $O(n^2)$. Quindi, complessivamente eseguiamo $O(n^3)$ operazioni.

Dunque, anche questo algoritmo ha complessità polinomiale.

Esempio

Si consideri il problema con le seguenti distanze:

$$D = \begin{bmatrix} * & * & * & 1 \\ 2 & * & 1 & * \\ * & * & * & * \\ * & -4 & 3 & * \end{bmatrix}$$

Domanda

Esistono algoritmi di complessità polinomiale in grado di restituire soluzioni ottime del problema in presenza di cicli negativi?

Tale problema risulta essere difficile: non sono noti algoritmi polinomiali per esso e presumbilmente non ne esistono.

Nota bene

Gli algoritmi di Dijkstra e di Floyd-Warshall possono essere visti sono algoritmi di raffinamento locale.

Infatti, in entrambi i casi, data una coppia di nodi, si parte da una soluzione ammissibile (il cammino costituito dall'arco diretto tra i due nodi) e a ogni iterazione tale cammino viene aggiornato nel caso se ne trovi uno di lunghezza inferiore.