Capitolo 10

Minimo albero di copertura

Definizione 10.1. Dato un grafo $G = (V, E)$ non orientato e connesso, un *albero di copertura* di G è un sottoinsieme $T \subseteq E$ tale che il sottografo (V, T) è un albero libero.

Osserviamo quindi che un albero di copertura è un sottografo di G che contiene tutti i suoi vertici ed è sia aciclico che connesso. Dalla Proprietà 9.1 sappiamo che $|T| = |V| - 1$.

Sia definita una *funzione peso* $w : E \to \mathbb{R}$; definiamo il *costo* di un albero di copertura come la somma dei pesi degli archi che lo compongono:

$$w(T) = \sum_{e \in T} w(e)$$

Definizione 10.2. Dato un grafo $G = (V, E)$ non orientato, connesso e pesato sugli archi, un *albero di copertura minimo* di G è un albero di copertura di G di costo minimo. Abbrevieremo il nome in MST, dall'inglese *Minimum Spanning Tree*).

Le definizioni viste finora possono essere estese al caso in cui G non sia connesso: in tal caso si parla di *minima foresta ricoprente*.

Lemma 10.1. *Sia $G = (V, E)$ un grafo connesso e non orientato. Sia $T \subseteq E$ un albero di copertura. Siano $(u, v) \in E \setminus T$ e p un cammino semplice da u a v con tutti gli archi in T (esiste in quanto T è connesso). Sia (x, y) un arco in p. Allora $(T \setminus \{(x, y)\}) \cup \{(u, v)\}$ è un albero di copertura.*

Dimostrazione. Sia $T' = (T \setminus \{(x, y)\}) \cup \{(u, v)\}$. Poiché $(u, v) \notin T$, deduciamo che (u, v) non fa parte del cammino p, quindi $(u, v) \neq (x, y)$, da cui $|T'| = (|T| - 1) + 1 = |V| - 1$. Grazie alla Proprietà 9.3, per provare che (V, T) rappresenta un albero libero ci basta provare che (V, T) è connesso. Consideriamo allora il cammino semplice p e dividiamolo nei sottocammini $p = p_1 \langle x, y \rangle p_2$, dove:

- p_1 è un cammino da u a x che non contiene (x, y);
- p_2 è un cammino da y a v che non contiene (x, y).

Per mostrare che (V, T) è connesso, prendiamo due nodi qualsiasi $a, b \in V$ e dimostriamo che esiste un cammino da a a b fatto di archi in T'. Poiché (V, T) è connesso, allora esiste un cammino q da a a b fatto da archi in T. Ora, possiamo avere due casi.

1. Se q non contiene l'arco (x, y), allora q è un cammino anche in $T \setminus \{(x, y)\}$ e dunque anche in T'. Il cammino q è quindi un cammino da a a b in T'.
2. Se q contiene l'arco (x, y), allora possiamo scomporre q in $q = q_1 \langle x, y \rangle q_2$ dove:
   - q_1 è un cammino da a a x che non contiene (x, y);
   - q_2 è un cammino da y a b che non contiene (x, y).

   Definiamo il cammino[1] $q' = q_1, \overleftarrow{p}_1, \langle u, v \rangle, \overleftarrow{p}_2, q_2$. È immediato verificare che q' è cammino da a a b in T'.

[1] Dato il cammino $p =< x_0, x_1, \ldots, x_k >$ con $\overleftarrow{p}$ intendiamo il cammino $< x_k, x_{k-1}, \ldots, x_0 >$, ovvero il cammino p percorso in senso inverso.

Concludiamo quindi che a e b sono connessi in T'.

Il sottografo (V, T') è connesso e T' contiene esattamente $|V| - 1$ archi, quindi è aciclico grazie alla Proprietà 9.3: dunque T' è un albero di copertura. □

10.1 Teorema fondamentale

Definizione 10.3. Un *taglio* $(S, V \setminus S)$ di un grafo non orientato $G = (V, E)$ è una partizione di V.

Definizione 10.4. Un *arco attraversa* il taglio $(S, V \setminus S)$ se uno dei suoi estremi si trova in S e l'altro in $V \setminus S$.

Definizione 10.5. Un *taglio rispetta* un insieme A di archi se nessun arco di A attraversa il taglio.

Definizione 10.6. Un arco che attraversa un taglio si dice *leggero* se ha peso pari al minimo tra i pesi di tutti gli archi che attraversano tale taglio.

Definizione 10.7. Sia A sottoinsieme di un qualche albero di copertura minimo; un arco si dice *sicuro* per A se può essere aggiunto ad A e quest'ultimo continua ad essere sottoinsieme di un albero di copertura minimo.

Teorema 10.1 (fondamentale)**.** *Sia $G = (V, E, w)$ un grafo non orientato, connesso e pesato; sia inoltre:*

- *$A \subseteq E$ contenuto in qualche albero di copertura minimo;*
- *$(S, V \setminus S)$ un taglio che rispetta A;*
- *$(u, v) \in E$ un arco leggero che attraversa il taglio.*

Allora (u, v) è sicuro per A.

Dimostrazione.

Ipotesi: esiste un albero di copertura minimo $T \subseteq E$ tale che $A \subseteq T$.

Tesi: dobbiamo trovare un albero di copertura minimo $T' \subseteq E$ tale che $A \cup \{(u, v)\} \subseteq T'$.

Poiché $(S, V \setminus S)$ rispetta A e (u, v) attraversa il taglio, allora $(u, v) \notin A$. Possiamo distinguere due casi:

1. se $(u, v) \in T$, allora $A \cup \{(u, v)\} \subseteq T$ che è albero di copertura minimo.
2. se $(u, v) \notin T$, dal momento che T è connesso, esiste un cammino semplice p in T che va da u a v. Poiché (u, v) attraversa il taglio, significa che u e v stanno da parti opposte rispetto al taglio. Quindi esiste almeno un arco (x, y) di p che attraversa il taglio. Sia $T' = (T \setminus \{(x, y)\}) \cup \{(u, v)\}$. Grazie al Lemma 10.1 T' è un albero di copertura. Abbiamo quindi:
   (a) $(u, v) \in T'$;
   (b) Per ipotesi sappiamo che $A \subseteq T$. Inoltre (x, y) attraversa il taglio ed il taglio rispetta A, quindi $(x, y) \notin A$. Da ciò segue che $A \subseteq T \setminus \{(x, y)\}$. Quindi a maggior ragione $A \subseteq (T \setminus \{(x, y)\}) \cup \{(u, v)\} = T'$. Concludiamo che $A \cup \{(u, v)\} \subseteq T'$. Quindi T' è un albero di copertura contenente $A \cup \{(u, v)\}$.
   (c) Verifichiamo il peso di T':

$$w(T') = w(T) - w(x, y) + w(u, v)$$

   dal momento che (x, y) attraversa il taglio e (u, v) attraversa il taglio ed è leggero abbiamo $w(x, y) \geq w(u, v)$ e quindi $w(u, v) - w(x, y) \leq 0$. Concludiamo che $w(T') \leq w(T)$ Dal momento che T è MST per ipotesi, segue che $w(T') = w(T)$.

Dunque anche T' è MST e contiene $A \cup \{(u, v)\}$. □

Corollario 10.1. *Sia $G = (V, E, w)$ un grafo non orientato, connesso e pesato; siano:*

- $A \subseteq E$ *contenuto in un albero di copertura minimo;*
- C *componente connessa della foresta* $G_A = (V, A)$*;*
- $(u, v) \in E$ *arco leggero che connette* C *ad un'altra componente connessa di* G_A*.*

Allora (u, v) *è sicuro per* A*.*

Dimostrazione. è sufficiente applicare il teorema fondamentale considerando il taglio $(C, V \setminus C)$; poiché:

- il taglio rispetta A;
- (u, v) è leggero per il taglio;

si hanno le ipotesi del Teorema 10.1, dal quale si ottiene che (u, v) è sicuro per A. □

10.2 Algoritmo di Kruskal

L'algoritmo sfrutta il Corollario 10.1 e mantiene, istante per istante, una *foresta* contenuta in qualche MST. Per poter gestire gli insiemi disgiunti che rappresentano le varie componenti connesse della foresta, occorre usare una struttura dati appropriata. La tecnica utilizzata consiste nel partizionare l'insieme V in k classi tali che:

- $\bigcup_{i=1}^{k} V_i = V$
- $i \neq j \Rightarrow V_i \cap V_j = \emptyset$

e rapppresentare ogni classe con un elemento della classe stessa. Devono essere permesse le seguenti operazioni:

`makeSet(v)`: crea una classe il cui unico elemento è v e lo elegge come rappresentante della stessa (costo $O(1)$);

`findSet(v)`: restituisce il rappresentante della classe cui v appartiene (costo $O(1)$);

`union(u, v)`: unisce le classi di u e v (partendo da una partizione generata da n chiamate a `makeSet`. Si possono definire delle strutture ottimizzare tali che se si eseguono k `union`, il costo totale di tutte le operazioni è $O(k \cdot \log k)$).

algoritmo `mstKruskal` (*grafo pesato* G) $\rightarrow$ *MST*

```
1 A = ∅
2 for each u ∈ V do
3   makeSet(u)
4 ordina gli archi di E in modo non decrescente rispetto al peso
5 for each (u, v) ∈ E do
6   if(findSet(u) != findSet(v)) then
7     A = A ∪ {(u, v)}
8     union(u, v)
9 return A
```

Invariante di ciclo: $A \subseteq MST$.

Complessità

1. $O(1)$

2-3. $O(n)$

4. $O(m \cdot \log m)$

5-8. in tutte le iterazioni, si eseguono in totale:

   6. 2 `findSet` per ogni arco $\rightarrow O(2 \cdot m)$

7. A viene aggiornato $n-1$ volte (diventa un albero) $\rightarrow O(n)$
8. $n-1$ `union` $\rightarrow O(n \cdot \log n)$

Totale: $O(1) + 2 \cdot O(n) + O(m \cdot \log m) + O(2 \cdot m) + O(n \cdot \log n) = O(m \cdot \log m) = O(m \cdot \log n)$, poiché la connessione del grafo ci dice che $n = O(m)$ e in generale abbiamo $m = O(n^2)$.

Esempio di esecuzione

Consideriamo il grafo in Figura 10.2. Assumiamo che gli archi siano ordinati nel modo seguente: *AB, CD, BD, BC, AD, AC*. Inizialmente tutti i nodi sono isolati. Consideriamo, nell'ordine, gli archi che vengono selezionati:

1. AB: poiché i nodi A e B non sono ancora collegati, l'arco viene selezionato;
2. CD: i nodi C e D non sono nella stessa componente, dunque l'arco viene selezionato;
3. BD: i nodi B e D non sono nella stessa componente, quindi l'arco viene scelto (il MST è ora pronto);
4. BC: i nodi B e C stanno già nella stessa componente, dunque l'arco non viene selezionato;
5. AD: i nodi A e D stanno già nella stessa componente, perciò l'arco viene scartato;
6. AC: i nodi A e C stanno già nella stessa componente, quindi l'arco non viene scelto.

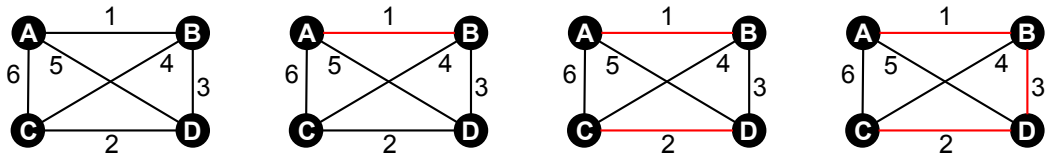

Figura 10.1: Simulazione dell'esecuzione dell'algoritmo di Kruskal

10.3 Algoritmo di Prim

L'algoritmo di Prim è fondato direttamente sul Teorema 10.1 fondamentale degli alberi di copertura minimi. Tale algoritmo mantiene, istante per istante, un *albero* contenuto in qualche MST. L'algoritmo, inoltre, richiede in ingresso un nodo $r \in V$ che sarà la radice dell'albero di copertura. Viene utilizzata una struttura dati del tipo *coda a min-priorità* per gestire l'insieme dei vertici Q non ancora inclusi nell'albero di copertura. L'insieme $v \setminus Q$ contiene i vertici già inseriti nell'albero. Per ciascun vertice $u \in Q$ vengono tenuti aggiornati i seguenti attributi:

- `key[u]` è la chiave di priorità: se u è adiacente a un vertice dell'albero $V \setminus Q$ il valore della chiave `key[u]` è pari al minimo tra i pesi di *tutti* gli archi che collegano u ai nodi dell'albero $V \setminus Q$, altrimenti il valore di `key[u]` è infinito.
- π`[u]` memorizza il predecessore di u nell'albero generato dall'algoritmo.

Si può ricostruire l'albero finale utilizzando i nodi e le informazioni sui predecessori:

$$A = \{(u, \pi[u]) \in E : u \in (V \setminus \{r\})\}$$

algoritmo `mstPrim` (*grafo pesato G, radice r*) $\rightarrow$ *MST*

```
1 Q: coda di priorita contenente tutti i vertici in V
2 for each u ∈ Q do
3   key[u] = ∞
4 key[r] = 0
5 π[r] = NIL
6 while(Q != ∅) do
7   u = extractMin(Q)
```

```
8     for each v ∈ Adj[u] do
9       if(v ∈ Q && w(u, v) < key[v]) then
10        key[v] = w(u, v)
11        π[v] = u
12  return A = {(u, π[u]) ∈ E : u ∈ (V - {r})}
```

Correttezza: segue direttamente dal teorema fondamentale (ad ogni iterazione, basta considerare il taglio $(V - Q, Q)$.

Complessità

1-5. $O(n)$

7. n estrazioni di costo $O(\log n) \to O(n \cdot \log n)$

8-11. consideriamo tutti i vertici adiacenti a quello estratto: l'operazione più costosa è il decremento della chiave, del costo $O(\log n)$: per ogni vertice $u \in Q$, si paga:

$$\sum_{v \in Adj[u]} O(\log n) = deg(u) \cdot O(\log n)$$

Considerando tutti i vertici, si ha:

$$\sum_{u \in V} \delta(u) \cdot O(\log n) = 2 \cdot m \cdot O(\log n) = O(m \cdot \log n)$$

12. costo della ricostruzione $O(n)$

Totale: $O(n) + O(n \cdot \log n) + O(m \cdot \log n) + O(n) = O(m \cdot \log n)$, poiché $m \geq n - 1$.

Esempio di esecuzione

Per ciascun nodo, è indicata la coppia (*chiave, predecessore*) associata ad ogni iterazione del ciclo; i nodi marcati in nero sono quelli ancora presenti in Q, il nodo rosso è quello estratto nell'iterazione considerata e quelli blu sono i nodi già estratti da Q ed esaminati.

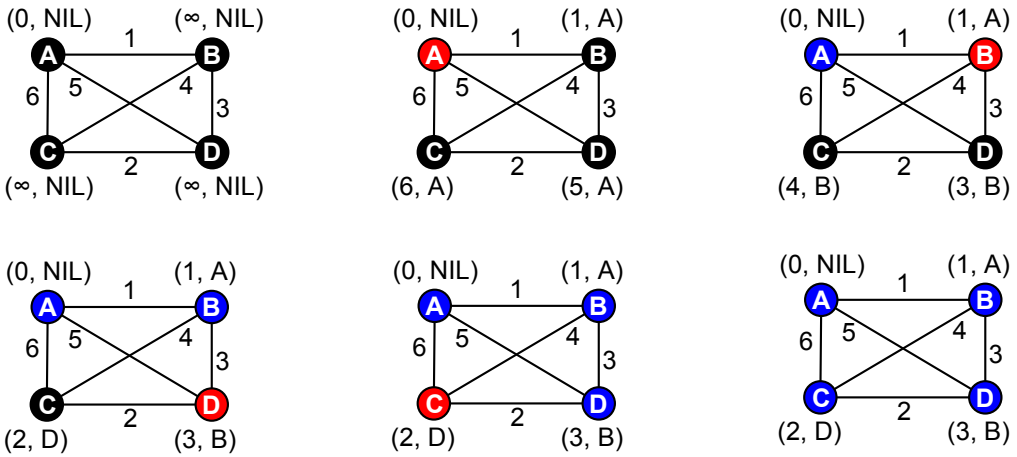

Figura 10.2: Simulazione dell'esecuzione dell'algoritmo di Prim (nodo sorgente a)

L'albero restituito dall'algoritmo, ricostruito a partire dai dati dell'ultima figura, è quello costituito dagli archi *AB, BD* e *CD*.