Algoritmi e Strutture Dati

Anno Accademico: 2021/2022

Dimostrazioni

Algoritmi su grafi e Teorema Fondamentale della NP-Completezza

---

13 Agosto 2022

Matteo Grazioso 884055

Indice

1 Teorema fondamentale dei Minimum Spanning Tree — MST 1
1.1 Enunciato 1
1.1.1 Corollario 1
1.2 Dimostrazione 1
1.2.1 Cosa si intende dimostrare 1
1.2.2 Procedimento 1

2 Algoritmo di Kruskal 2
2.1 Enunciato 2
2.2 Dimostrazione 2
2.2.1 Cosa si intende dimostrare 2
2.2.2 Procedimento 2

3 Algoritmo di Prim 3
3.1 Enunciato 3
3.2 Dimostrazione 3
3.2.1 Cosa si intende dimostrare 3
3.2.2 Procedimento 3

4 Algoritmo di Dijkstra 4
4.1 Enunciato 4
4.2 Dimostrazione 4
4.2.1 Cosa si intende dimostrare 4
4.2.2 Procedimento 4

5 Algoritmo di Bellman-Ford 6
5.1 Enunciato 6
5.2 Dimostrazione 6
5.2.1 Cosa si intende dimostrare 6
5.2.2 Procedimento 6

6 Algoritmo di Floyd-Warshall 8
6.1 Enunciato 8
6.2 Dimostrazione 8
6.2.1 Cosa si intende dimostrare 8
6.2.2 Procedimento 8

7 Teorema Fondamentale della NP-completezza 10
7.1 Enunciato 10
7.2 Dimostrazione 10
7.2.1 Cosa si intende dimostrare 10
7.2.2 Procedimento 10

1 Teorema fondamentale dei Minimum Spanning Tree — MST

1.1 Enunciato

Sia $G = (V, E, w)$ un grafo non orientato, pesato e connesso.
Siano:

a. $A \subseteq E$ appartenente ad un MST;
b. $(S, V \backslash S)$ un taglio che rispetta A;
c. $(u, v) \in E$, arco leggero che attraversa il taglio

Allora:
(u, v) è sicuro per A (appartiene ad un MST)

1.1.1 Corollario

Sia $G = (V, E, w)$ un grafo non orientato, pesato e connesso.
Siano:

a. $A \subseteq E$ contenuto in qualche MST;
b. C è una componente connessa di $G_A = (V, A)$;
c. (u, v), arco leggero che collega C ad un'altra componente connessa (c.c.) di G_A

Allora:
(u, v) è sicuro per A (appartiene ad un MST)

1.2 Dimostrazione

1.2.1 Cosa si intende dimostrare

$T^{'} \mid A \cup \{(u, v)\} \in T^{'}$ è un MST

1.2.2 Procedimento

Sia $T \subseteq E$ un MST che contenga A.

1. $(u, v) \in T$ sicuramente $A\{(u, v)\} \subseteq T$;
2. $(u, v) \notin T$ allora crea un ciclo;
3. Pongo $T^{'} = (T \cup \{(u, v)\}) \setminus \{(x, y)\}$ dove (x, y) attraversa il taglio;
4. Affermo che $T^{'}$ è un MST;

Quindi:
$W(T) \leq W(T^{'})$ perchè T è MST;
$W(T^{'}) \leq W(T)$ perchè $T^{'}$ è MST;

$W(T^{'}) = W(T) + w(u, v) - w(x, y) \leq W(T)$ perchè $w(u, v) - w(x, y) \leq 0$

Allora

$$W(T) = W(T^{'})$$

2 Algoritmo di Kruskal

MST su grafi NON orientati e connessi.

2.1 Enunciato

```
Kruskal_MST(G, w)
    A = empty_set
    for each v in V[G]                  // theta (n)
        make_set(u)
    sort(E[G])                          // O(n log(n))

    for each (u,v) in E                 // theta (m)
        if(find_set(u) <> find_set(v))  // theta ( l o g  (m))
            union(u,v)
            A U {(u, v)}
    return A
```

$T(Kruskal) = m\ log(m)$. m domina perché essendo connesso assumiamo che $m \geq n - 1$.

2.2 Dimostrazione

2.2.1 Cosa si intende dimostrare

L'insieme A restituito in output dall'algoritmo è un MST

2.2.2 Procedimento

La correttezza si dimostra attraverso il corollario del Teorema MST.
Nella fase di inizializzazione di Kruskal, creo un set per ogni vertice, che sono banalmente delle componenti connesse e l'insieme $A = \emptyset$ è anch'esso banalmente contenuto in un MST.

L'ordinamento in ordine non decrescente in base al peso degli archi fa sì, insieme all'istruzione `if (findset(u) <> findset(v)`, che venga sempre estratto un arco leggero tra due diverse componenti connesse (che verranno poi unite dall'istruzione `union(u,v)`.
Per il corollario questi archi saranno sicuri per A e l'insieme restituito in output sarà un MST.

3 Algoritmo di Prim

MST su grafi NON orientati e connessi.

3.1 Enunciato

```
Prim_MST(G, w, r)
    Q = V[G]                          // Q coda di priorita' heap binario
    for each u in Q                            // theta (n)
        key[u] = infty
        \pi[u] = NIL

    key[r] = 0
    while Q <> emptyset                        // theta (n)
        u = extract_min(Q)                     // O (log(n))
        for each v in adj[u]                   // 2m volte totali
            if (v in Q and key[v] > w(u, v))
                key[v] = w(u, v)               // O (log(n))
    return A
```

In cui viene ritornato $A = \{(u, \pi[u]) \in E \mid u \in V \setminus \{r\}\}$

$T(Prim) = O(n + n\ log(n) + m\ log(n))$. m domina quindi $O(m\ log(n))$.

3.2 Dimostrazione

3.2.1 Cosa si intende dimostrare

L'insieme A restituito in output dall'algoritmo è un MST

3.2.2 Procedimento

La ricerca dei vertici adiacenti a u e la verifica che essi non appartengano a Q garantisce che gli archi "estratti" rispettino il taglio $S, Q \setminus S$.
L'assegnamento del campo *key* dei vertici adiacenti v farà sì che il successivo vertice ad essere estratto da Q sarà il vertice v con valore *key* minore, ed essendo questo valore proprio il peso dell'arco (u, v) sarà anche un arco leggero.
Per il teorema, $A \cup \{(u, v)\}$ farà parte di un Minimum Spanning Tree.

4 Algoritmo di Dijkstra

Cammino minimo semplice da s a tutti i vertici di un grafo *ORIENTATO*.

Richiede $Pesi \geq 0$ e grafo connesso.

4.1 Enunciato

```
Dijkstra(G, w, s)
    Q = V[G]
    Init_ss(G, s)        // come Prim tutto a infty\NIL tranne s che e' 0\NIL
    S = emptyset
    while Q <> emptyset                // theta (n)
        u = extract_min(Q)   // O (log(n)) se heap binario, theta (n) se array
        for each v in adj[u]           // m volte totali
            relax(u, v, w(u, v))       // O (log(n)) se heap, theta (1) se array
    return (d, G_\pi)                  // d vettore di stime, G_\pi grafo dei
predecessori
```

```
relax(G, u, v, w(u,v))
    if d[v] > d[u]+w(u,v)
        d[v] = d[u]+w(u,v)
```

$T(Dijkstra) =$

◇ Heap:

- **Grafo sparso**: $m \approx n = O(n\ log(n))$
- **Grafo denso**: $m \approx n^2 = O(n^2\ log(n))$

◇ Array:

- **Grafo sparso**: $m \approx n = O(n^2)$
- **Grafo denso**: $m \approx n^2 = O(n^2)$

4.2 Dimostrazione

4.2.1 Cosa si intende dimostrare

Alla fine dell'algoritmo: $d[u] = \delta(s,u)\ \forall u \in V$.
G_π è un albero di cammini minimi.

4.2.2 Procedimento

Dimostriamo solamente la prima parte, per farlo utilizzeremo le proprietà di Disuguaglianza triangolare:

$$\delta(s,v) \leq \delta(s,u) + w(u,v)$$

Limite inferiore: $\delta(s,u) \leq d[u]$ in ogni momento dell'esecuzione;

Assenza di cammino: $\forall v \in V$ non raggiunto da s, $\delta(s,u) = \infty$;

Convergenza: in breve, se si ha un cammino minimo da s a u, dopo la prima relax (u, v, w(u , v)) avrò che $d[v] = \delta(s, v)$;

Procedendo con la dimostrazione:
Per assurdo $\exists\ u \in V|$ al momento dell'estrazione da Q, $d[u] \neq \delta(s, u)$ e questo accade per la prima volta.

1. $u \neq s$ perchè dopo la init_ss sicuramente $d[s] = \delta(s, s)$;
2. Quindi, al momento di estrarre u, l'insieme S dei vertici già estratti sarà $\neq \emptyset$ perchè almeno vi sarà il vertice s;
3. Faccio un disegno con l'insieme S contenente i vertici s, x collegati da un cammino minimo. Disegno l'insieme $Q = V \backslash S$ con i vertici y, u collegati da un cammino minimo. Disegno l'arco (x, y).
   (a) **Caso 1:** s non raggiunge u, ma allora $\delta(s, u) = \infty = d[u]$ *dalla init*
   (b) **Caso 2:** cammino minimo tra s, u
4. Al momento di estrarre x, per ipotesi $d[x] = \delta(s, x)$
5. Dopo la relax (u, v, w(u, v)), $d[y] = \delta(s, y)$ per proprietà della convergenza;
6. Ora però poniamo che Dijkstra estragga il vertice u, sarà quindi vero che $d[u] \leq d[y]$
7. $\delta(s, y) \leq \delta(s, u)$. Per ipotesi $s, \ldots, y$ era un cammino minimo; Essendo da y a u *pesi* ≥ 0 allora per certo $\delta(s, y) \leq \delta(s, u)$
8. $\delta(s, u) \leq d[u]$ **limite inferiore**

Mettendo assieme i pezzi e si cerca di far risultare che in realtà $d[u] = \delta(s, u)$

- Dal punto 8 si ottiene: $\delta(s, u) \leq d[u]$
- Dal punto 6 si ottiene: $d[u] \leq d[y]$
- Dal punto 5 si ottiene: $d[y] = \delta(s, y)$
- Dal punto 7 si ottiene: $\delta(s, y) \leq \delta(s, u)$

Si conclude la dimostrazione perchè $\delta(s, u) \leq d[u] \leq \delta(s, u)$ quindi $d[u] = \delta(s, u)$, configurando un assurdo.

5 Algoritmo di Bellman-Ford

Funziona come l'algoritmo di Dijkstra però funziona anche con pesi negativi e scopre i cicli negativi restituendo false.

5.1 Enunciato

```
Bellman_Ford(G, w, s)
    Init_ss(G, s)                    // theta (n)
    for i=1 to |V[G]|-1              // theta (n-1)
        for each (u, v) in E[G]      // theta (m)
            relax(u, v, w(u, v))     // O (1)
    for each (u, v) in E[G]          // theta (m)
        if d[v] > d[u]+w(u, v)
            return false
    return true, d, \pi
```

$T(Bellman - Ford) = \theta(n + (n-1)m + m) = \theta(n \cdot m)$. Pertanto per i grafi sparsi si ha $\theta(n^2)$, mentre per i grafi densi si ha $\theta(n^3)$.

5.2 Dimostrazione

5.2.1 Cosa si intende dimostrare

Se $d[u] = \delta(s,u)\ \forall u \in V$ l'algoritmo restituisce *true*.
G_π è un albero di cammini minimi.

5.2.2 Procedimento

1. Dimostro $d[u] = \delta(s,u)\ \forall u \in V$.
   Se $u \in V$,allora $\delta(s,u) = +infty$ se non è raggiungibile $\in R$. (-$\infty nonpossibile).\delta(s,u) \in R$, allora esiste almeno un cammino minimo tra s,u. Pongo p cammino semplice minimo tra s,u, quindi *il numero di archi di* $p \leq n-1$ e questo spiega anche il perchè delle $n-1$ iterazioni,cioè perchè è un cammino minimo semplice.
   A questo punto applichiamo ripetutamente la proprietà della convergenza.
   Con un disegno, si mostra che i ripetuti $n-1$ cicli di relax su TUTTI gli archi setteranno i vari $d[x] = \delta(s,x)$.
2. Dimostro che alla fine l'algoritmo ritorna TRUE Utilizzo la proprietà triangolare $\delta(s,v) \leq \delta(s,u) + w(u,v)$.
   I 2 casi sono:
   - $\forall (u,v) \in E:\ d[v] \leq d[u] + w(u,v) \to TRUE$
   - $\exists (u,v) \in E:\ d[v] > d[u] + w(u,v) \to FALSE$

   Banalmente alla fine sarà proprio $d[v] \leq d[u] + w(u,v)$.

   A questo punto pongo l'ipotesi $\exists$ *ciclo negativo raggiunto da* s.
   La tesi è che l'algoritmo di Bellman-Ford ritorni FALSE, ma per assurdo pongo che ritorni TRUE, quindi esisterà un ciclo negativo $C \equiv \langle x_0, \ldots, x_q \rangle$ raggiunto da s con $x_0 = x_q$.
   Allora
   $$\sum_{i=1}^{q} w(x_{i-1}, x_i) < 0$$

e avremo che

$$\forall i = 1 \ldots q : \; d[x_i] \leq d[x_{i-1}] + w(x_{i-1}, x_i)$$

(perché per ipotesi ritorna TRUE); che equivale a dire che:

$$\sum_{i=1}^{q} d[x_i] \leq \sum_{i=1}^{q} d[x_{i-1}] \leq \sum_{i=1}^{q} w(x_{i-1}, x_i)$$

Semplificando

$$d[x_q] \leq d[x_0] + \sum_{i=1}^{q} w(x_{i-1}, x_i)$$

ma considerando che $x_0 = x_q$ allora

$$0 \leq \sum_{i=1}^{q} w(x_{i-1}, x_i)$$

verificandosi così un assurdo.

6 Algoritmo di Floyd-Warshall

Funziona come l'algoritmo di Dijkstra però funziona anche con pesi negativi e scopre i cicli negativi restituendo false. L'algoritmo, in assenza di cicli negativi, restituisce una matrice n*n contenente i cammini minimi tra tutti i vertici del grafo.

6.1 Enunciato

W è una matrice $n \cdot n$ dove per ogni coppia di vertici i, j con $i \neq j$,la cella i, j della matrice è inizializzata a $w(i, j)$ se esiste un arco tra i due vertici, oppure ∞. Se $i = j$,la cella i, j sarà $= 0$.

```
1 Floyd_Warshall(W)
2     n = rows(W)
3     D^0 = W
4     for k = 1 to n
5         for i = 1 to n
6             for j = 1 to n
7                 d_{ij}^{(k)} = min ( d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, d_{ij}^{(k-1)})
8     return D^(n)
```

Si trascrive a seguire la riga 7:

$$d_{ij}^{(k)} = min(d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, d_{ij}^{(k-1)})$$

$T(Floyd\_Warshall) = \theta(n^3)$.

6.2 Dimostrazione

6.2.1 Cosa si intende dimostrare

Dalla matrice D^{k-1} posso ottenere D^k.

6.2.2 Procedimento

Si parte dalla constatazione che per trovare il minimo di un insieme X si può dividerlo in due parti Y, Z e trovare il $min(min(y), min(z))$.
Allo stesso modo posso immaginare i cammini che vanno da i a j divisi in due gruppi:

1. passanti per il vertice k;
2. non passanti per il vertice k.

Si pone la matrice $\widehat{D_{i,j}^{(k)}}$ come la matrice contenente $\{p \in p\widehat{D^{(k)}}$ passante per $k\}$. A questo punto si afferma che

$$D^{(k)} = \widehat{D^{(k)}} \ \cup \ d^{(k-1)}$$

. Ora si può concludere che

$$d_{ij}^{(k)} = min(\widehat{D^{(k)}}, \ d^{(k-1)})$$

Nel caso di $d^{(k-1)}$ ho già tutte le informazioni disponibili, mentre nel caso di $\widehat{D^{(k)}}$ no; però si sa che i cammini in questa matrice possono essere pensati come

$$\widehat{D^{(k)}} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

ma a questo punto si termina la dimostrazione perchè si è già a conoscenza di tutte le informazioni. Quindi,

$$d_{ij}^{(k)} = min(d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, d_{ij}^{(k-1)})$$

7 Teorema Fondamentale della NP-completezza

7.1 Enunciato

$$P \cap NPC \neq \emptyset \rightarrow P = NP$$

7.2 Dimostrazione

7.2.1 Cosa si intende dimostrare

Se $P = NP$ allora $P \subseteq NP$ e $NP \subseteq P$

7.2.2 Procedimento

Se l'Intersezione non è vuota, allora $\exists Q \in P$ tale che appartiene a NP.
Per la proprietà degli NPC però sappiamo anche che

$$\forall\, p^{'} \in NP,\ p^{'} \leq_P P$$

quindi quindi anche Q è riducibile polinomialmente ad un NPC e pertanto anche tutto l'insieme P lo è.