Visite, ordinamento topologico e componenti connesse

Visita in Ampiezza Grafi: Dimostrazione $d[v] = \delta(s, v)$

Dimostrazione $d[v] \geq \delta(s, v)$**.** Dato che l'albero dei predecessori π contiene solo archi appartenenti a G, il cammino da s a v è un cammino che appartiene anche a G, quindi la lunghezza del cammino da s a v nell'albero è maggiore o uguale alla distanza tra s e v.
Dimostrazione $d[v] \leq \delta(s, v)$**.** Definiamo l'insieme dei vertici a distanza k dalla sorgente nel grafo come $V_k = v \in V | \delta(s, v) = k$ (v_0 contiene solo la sorgente).
Nel caso base, $d[v_0] \leq \delta(s, v)$ (distanza di s da sè stesso: $0 \leq 0$). Sia $v \in V_k$: allora $\delta(s, v) = k$ (per definizione).
Con $k > 0$ (passo), esisterà almeno un vertice w tale che $\delta(s, w) = k - 1$ e $(w, v) \in E$, ovvero un arco che va da v a w. Definiamo l'insieme dei vertici appartenenti a V_{k-1} con arco entrante in v come $U_{k-1} = w \in V_{k-1} | (w, v) \in E$. Tra questi, sia u il primo vertice di U_{k-1} ad essere scoperto ed inserito nella coda: per politica FIFO, u sarà anche il primo ad essere estratto dalla coda. Quando guarderò i vertici adiacenti a u, v sarà ancora bianco (perchè più lontano), e v verrà inserito nell'albero come figlio di u, con $d[v] = d[u] + 1$. Inoltre, per ipotesi induttiva, $d[u] \leq k - 1$.
Quindi, quando inseriremo v nell'albero:

- $d[v] = d[u] + 1$
- ma $d[u] \leq k - 1$, quindi $d[v] \leq (k - 1) + 1$
- $d[v] \leq k$

Ordinamento Topologico: Correttezza algoritmo astratto

Teorema. *ord è un'ordinamento topologico di G. Denotiamo con ord_i e G' (copia del grafo G) il contenuto di ord e G' all'iterazione i-esima: ad ogni istante del ciclo, non può esserci nessun cammino in G che porta da un vertice in G' ad uno in ord ("all'indietro").*

Dimostrazione. Caso base: con $i = 0$, la condizione è banalmente verificata (*ord* non contiene vertici). Passo induttivo: ad un istante k, non c'è alcun cammino in G dai vertici G'_k ai vertici in ord_k, quindi ad un istante $k + 1$, non c'è alcun cammino in G dai vertici G'_{k+1} ai vertici in ord_{k+1}.
Al passo k-esimo, si sceglie un vertice sorgente u in G'_k: non ci sono cammini in G che raggiungono u passando solo per i vertici di G'_k. Per ipotesi induttiva, aggiungendo u ad ord_k all'istante k, all'istante $k + 1$ non ci sarà alcun cammino in G dai vertici di G'_{k+1} ai vertici in ord_{k+1}. □

Ordinamento Topologico: Teorema dell'ordinamento topologico

Teorema (Ordinamento topologico)**.** *In una qualunque visita DFS, $f[v] < f[u]$ per ogni arco $\langle u, v \rangle$.*

Dimostrazione. Supponiamo per assurdo che si abbia $f[u] < f[v]$; quindi, apro e chiudo u e poi apro e chiudo v, oppure apro u e v e poi chiudo u e v. Il primo caso è impossibile, perchè u non può diventare nero prima che v diventi grigio, ossia prima che tutti i suoi adiacenti siano stati scoperti. Il secondo caso è impossibile perchè u sarebbe discendente di v e l'arco $\langle u, v \rangle$ sarebbe un arco all'indietro, ma il grafo è aciclico. □

Componenti Fortemente Connesse: Lemma del cammino fortemente connesso

Lemma (Cammino fortemente connesso)**.** *Se due vertici x, y di un grafo sono in una stessa cfc, allora nessun cammino tra di essi può abbandonare tale cfc.*

Dimostrazione. Sia z tale che $x \to z$ e $z \to y$. z è banalmente raggiungibile da x per ipotesi ($x \to z$); siccome x, y appartengono alla stessa cfc, esisterà un cammino $y \to x$. Esiste anche $z \to y$ per ipotesi. Quindi, per concatenazione, esisterà anche un cammino $z \to y \to x$. □

Componenti Fortemente Connesse: Teorema del sottoalbero fortemente connesso

Teorema (Sottoalbero fortemente connesso)**.** *In una qualunque DFS di un grafo G orientato, tutti i vertici di una cfc vengono collocati in uno stesso sottoalbero.*

Dimostrazione. Sia r il primo vertice di una data cfc che viene scoperto dalla DFS: da r sono raggiungibili tutti gli altri vertici della cfc (per definizione). Poichè r è il primo vertice ad essere stato scoperto, al momento della sua scoperta tutti gli altri vertici della cfc saranno bianchi. Per il lemma precedente, tutti i cammini da r agli altri vertici della cfc conterranno solo vertici bianchi che fanno parte della cfc. Allora, per il **teorema del cammino bianco**, tutti i vertici appartenenti alla cfc di r saranno discendenti di r nell'albero DFS. □

Correttezza dell'algoritmo di Kosaraju

Per dimostrare la correttezza dell'algoritmo, ci si avvale del **teorema del sottoalbero fortemente connesso** e dei lemmi seguenti.

Lemma (2)**.** *Un grafo orientato e il suo trasposto hanno le stesse cfc.*

Lemma (3)**.** *Sia A^T un albero ottenuto con la visita in profondità G^T, considerando i vertici in ordine decrescente dei tempi di fine visita su G, e sia u la sua radice. Per ogni vertice v discendente di u in A^T, v e u appartengono alla stessa cfc.*

Dimostrazione. Dimostriamo che ogni discendente di u in A^T è anche un discendente di u in un albero della foresta costruita dalla DFS su G. La dimostrazione è fatta per assurdo.
Consideriamo un cammino sull'albero A^T a aprtire dalla radice u; sia v il primo vertice sul cammino per cui il lemma non vale (cioè v non è discendente di u nella visita di G) e sia w il suo predecessore sul cammino. L'enunciato è valido per w: $d[u] \leq d[w] < f[w] \leq f[u]$.
Siccome la visita DFS di G^T considera i vertici in ordine decrescente di fine visita, vale $f[v] < f[u]$. Per il **teorema delle parentesi**, se v non è discendente di u nella prima visita, deve valere $d[v] < f[v] < d[u] < f[u]$. Ma questo è impossibile, in quanto v è adiacente a w in G, e la visita di v non può terminare prima che sia iniziata la visita di un suo adiacente.
Quindi in G esiste un cammino da u a v. Siccome v è discendente di u in A^T, esiste anche un cammino da u a v in G^T, e quindi da v a u in G. □

Greedy

Greedy Intervalli Disgiunti: Dimostrazione di Correttezza

Definiamo le invarianti (S è l'insieme di intervalli esaminati ad un passo intermedio k):

- **Max**: la sequenza $A_1, A_2, \ldots, A_k$ di intervalli disgiunti scelti è, per l'insieme S, una **sequenza massimale**
- **PrimaMax**: la sequenza $A_1, A_2, \ldots, A_k$ è, fra le sequenze massimali, quella che finisce prima
- **PrimaVisti**: ogni intervallo $\in S$ termina prima della fine di qualunque intervallo $\notin S$

L'invariante da dimostrare è **Max**.

Dimostrazione. **Caso base**. Inizialmente, S è l'insieme vuoto, la sequenza massimale per S è la sequenza vuota. Tutte e tre le invarianti sono vere.
Passo. Sia A l'intervallo che termina prima tra quelli ancora da esaminare (ossia fra quelli $\notin S$). Consideriamo allora l'insieme $S' \equiv S \cup \{A\}$ e cerchiamo di stabilire qual è per S' la sequenza massimale (**Max**) che finisce prima (**PrimaMax**).
Caso 1: A inizia **prima** della fine della sequenza $A_1, A_2, \ldots, A_k$, quindi A interseca A_k; quindi, ogni sequenza avente A come ultimo elemento non può avere più di k elementi. Non inserendo A nella soluzione, l'invariante Max si mantiene.
Caso 2: A inizia **dopo** la fine della sequenza $A_1, A_2, \ldots, A_k$; allora, la sequenza $A_1, A_2, \ldots, A_k, A$:

- è per S' una sequenza massimale, perchè una sequenza massimale per S' non può avere più di $k+1$ elementi
- è per S' la sequenza massimale che finisce prima, perchè in S' non ci sono sequenze massimali non includenti A

Anche in questo caso, le invarianti si mantengono. □

Greedy Moore: Dimostrazione di Correttezza

L'algoritmo di Moore è corretto, si dimostra per induzione.

Dimostrazione. **Invariante**. Sia S l'insieme di tutti i job finora esaminati:

1. *Sol* è uno **scheduling massimale** $L_1, L_2, \ldots, L_k$ di job di S che rispetta le scadenze, cioè quello con il numero massimo di elementi
2. *Sol* è, fra tutti gli scheduling massimali di job di S, quello di durata totale minima
3. *Sol* è ordinato per tempi di scadenza crescenti
4. ogni job non in S ha una scadenza posteriore o uguale alle scadenze dei job in S

Passo. Sia t_k l'istante di fine dello scheduling $Sol = L_1, L_2, \ldots, L_k$. Sia L il primo job non ancora esaminato, cioè non in S, che ha scadenza prima di tutti gli altri job non esaminati. Siano s la scadenza di L e d la durata di L. Considerando l'insieme $S' = S \cup \{L\}$, si possono verificare due casi:

1. L aggiunto ad S è eseguibile entro la sua scadenza
2. L non è eseguibile

Caso 1. Lo scheduling così ottenuto è uno scheduling massimale. $\{L\}$ ha cardinalità 1; quindi, se esistesse uno scheduling per $S \cup \{L\}$ con più di $k+1$ elementi, vi sarebbe uno scheduling per S con più di k elementi, il che è assurdo. Inoltre $S \cup \{L\}$ è di durata minima.
Caso 2. Lo scheduling $L_1, L_2, \ldots, L_k, L$ non è una soluzione. Se L è il processo di durata massima, sostituendolo ad un job già presente S, si otterrebbe uno scheduling di durata maggiore o uguale (violando punto 2). Se in S esiste un job L_{max} di durata maggiore di L, eliminando L_{max} ed aggiungendo L ad S, si ottiene uno scheduling di ancora k elementi, di durata minore (minima).
Quindi, $L_1, L_2, \ldots, L_k$ è uno scheduling massimale e di durata minima. □

Greedy Huffman: Dimostrazione di Correttezza

L'algoritmo restituisce un albero di Huffman, che rende minima la lunghezza media di codifica.

Foresta di Huffman. La foresta di Huffman per un alfabeto C con funzione di frequenza f è una foresta i cui elementi $T_1, T_2, \ldots, T_n$ sono sottoalberi di un albero di Huffman T per quell'alfabeto.

Invariante. La foresta $\{T_1, T_2, \ldots, T_n\}$ costruita dall'algoritmo al passo generico è una foresta di Huffman per C e f, cioè esiste un albero di Huffman T di cui gli alberi $T_1, T_2, \ldots, T_n$ sono sottoalberi.

Caso base. Prima dell'esecuzione del ciclo, l'invariante vale banalmente, in quanto gli alberi sono tutti nodi singoli, cioè foglie corrispondenti ai caratteri dell'alfabeto C.

Passo. Assumiamo che prima della k-esima iterazione del ciclo l'invariante valga, cioè che la foresta $F = \{T_1, T_2, \ldots, T_n\}$ sia una foresta di Huffman per il dato alfabeto C. Mostriamo che dopo il (k+1)-esimo passo dell'iterazione, che fonde i due alberi T_a e T_b aventi due frequenze minime in un nuovo albero T_{ab}, la nuova foresta $F - \{T_a, T_b\} \cup \{T_{ab}\}$ è ancora una foresta di Huffman.

Dimostrazione. Mostriamo che fra tutti gli alberi di codifica di cui $T_1, T_2, \ldots, T_n$ sono sottoalberi, vi è un albero (T''') la cui lunghezza media di codifica $L(T''')$ è minima e di cui T_{ab} è un sottoalbero. Quindi l'albero T_{ab} può essere inserito nella foresta al posto di T_a e T_b: la foresta risultante $F - \{T_a, T_b\}$ è ancora una foresta di Huffman.
Per ipotesi induttiva, esiste un albero T' di cui gli alberi $T_1, \ldots, T_a, \ldots, T_b \ldots, T_n$ sono sottoalberi, dove T_a e T_b sono, nella foresta al passo considerato, i due alberi di frequenze minime, $f(T_a)$ e $f(T_b)$. Mostriamo che esiste un albero di Huffman T''' avente T_{ab} come sottoalbero.
Consideriamo, fra i nodi interni di T' non appartenenti alla foresta F, cioè fra i nodi che nel passo considerato non sono ancora stati creati, quello di profondità massima. Sia esso z. Come ogni nodo interno, z deve avere due sottoalberi figli non nulli, siano T_x e T_y.
Poichè T_a e T_b sono, al passo considerato, i due alberi di pesi minimi, assumendo $f(T_a) \leq f(T_b)$ e $f(T_x) \leq f(T_y)$, abbiamo $f(T_a) \leq f(T_x)$ e $f(T_b) \leq f(T_y)$. Poichè z è un nodo di profondità massima, le radici degli alberi T_x e T_y si trovano in T' a profondità d non inferiore a quelle di T_a e T_b, siano d_1 e d_2: quindi, $d_1 \leq d$ e $d_2 \leq d$.
Scambiamo di posizione T_a e T_x per ottenere un albero di lunghezza media non superiore (analogamente per T_b e T_y).
Ma T' è un albero di Huffman, cioè avente $L(T')$ minimo. Quindi deve essere $L(T''') = L(T')$, e anche T''', che ha T_{ab} come sottoalbero, è un albero di Huffman. Dunque, la foresta $F - \{T_a, T_b\} \cup \{T_{ab}\}$ è ancora una foresta di Huffman. □

Programmazione dinamica e UnionFind

MSI: Sottostruttura ottima

Teorema (Sottostruttura ottima MSI)**.** *Se S_i è una soluzione ottima per P_i (problema ristretto ai primi i nodi), allora vale una delle seguenti affermazioni:*

- $V_i \notin S_i$ *e* S_i *è anche una soluzione ottima per* P_{i-1}
- $V_i \in S_i$ *e* $S_i - \{V_i\}$ *è una soluzione attima per* P_{i-2}

Dimostrazione. Se S_i è una soluzione ottima per P_i, allora sono possibili due casi.

Caso 1. Nel primo caso, S_i è anche una soluzione ammissibile per il problema P_{i-1}, in quanto non contiene V_i. Se non fosse ammissibile per P_{i-1}, esisterebbe una soluzione S'_{i-1} con peso maggiore di S_i, ma tale soluzione sarebbe una soluzione anche per P_i e sarebbe migliore di S_i, contraddicendo l'ipotesi che S_i sia la soluzione ottima per P_i.

Caso 2. Nel secondo caso, V_i fa parte di S_i, quindi $S' = S_i - \{V_i\}$ è una soluzione ottima per il problema P_{i-2}. V_{i-1} non può appartenere a S_i per la condizione di indipendenza sulle soluzioni. Per il

problema P_{i-2}, S' è una soluzione ottima: se non lo fosse, esisterebbe una soluzione S'' per P_{i-2} di peso maggiore di S'. Allora $S'' \cup \{V_i\}$ sarebbe una soluzione per P_i e avrebbe peso maggiore di S_i (assurdo). □

Corollario. *Se S_{i-1} è una soluzione ottima per P_{i-1} e S_{i-2} è una soluzione ottima per P_{i-2}, allora la soluzione per P_i è la soluzione di peso massimo tra S_{i-1} e $S_{i-2} \cup \{V_i\}$.*

LCS: saper spiegare come si arriva all'algoritmo, partendo dalla sottostruttura ottima, casi e sottoproblemi

Siano $S1 : a_1, \ldots, a_m$ e $S2 : b_1, \ldots, b_n$ e $S3 : c_1, \ldots, c_k$; sono possibili due casi.

Caso 1. Nel primo caso, $a_m = b_n$. a_m sarà contenuto in $S3$, e occuperà l'ultima posizione di $S3$; quindi $c_k = a_m = b_n$. Dato che gli ultimi elementi della sequenze iniziali già appartengono alla lcs, si possono non considerare nel confronto con il resto delle sequenze. Quindi $S3 = lcs(S1_{m-1}, S2_{n-1}) + \{a_m\}$.

Caso 2. Nel secondo caso, $a_m \neq b_n$. In questo caso, possiamo dure che a_m o b_n non saranno contenuti in $S3$. Quindi, $S3$ sarà la sequenza più lunga tra $lcs(S1 - \{a_m\}, S2)$ e $lcs(S1, S2 - \{b_n\})$.

Per memorizzare la soluzione di tutti i sottoproblemi, si usa una **matrice** (LCS) di $m + 1$ righe e $n + 1$ colonne. La casella $LCS[i, j]$ contiene la più lunga sottosequenza comune dei due segmenti iniziali di lunghezze rispettive i e j. La casella 0-esima contiene la più lunga sottosequenza dei prefissi vuoti.

Union Find: Analisi ammortizzata con il metodo dei crediti

Il **metodo dei crediti** viene usato per determinare il costo ammotizzato di una sequenza di operazioni, senza andare nel dettaglio. 1 credito vale $\mathcal{O}(1)$ passi di esecuzione. Le funzioni meno costose depositano crediti sugli oggetti, ed i crediti possono essere prelevati dalle funzioni più costose. Il costo ammortizzato è dato dalla somma di tutti i costi diviso per il numero di operazioni.

Vediamo n makeSet e $n - 1$ makeUnion analizzati con il metodo dei crediti:

- **Numero massimo di cambi padre**. Ad ogni makeUnion, se una foglia cambia padre, il nuovo insieme che la contiene sarà grande almeno il doppio di quello di partenza. Quindi la foglia dopo k cambi di padre apparterrà ad un insieme di 2^k elementi. Siccome il numero totale di elementi è n, $2^k \leq n$, quindi $k \leq \log_2 n$.
- **Deposito crediti**. Assegniamo ad ogni makeSet un costo aggiuntivo di $\log_2 n$ crediti. Con n makeSet, si accumulano $n \log_2 n$ crediti.
- **Costo union**. Ogni union consuma 1 credito per il costo dell'operazione d cambio del padre. Poichè i cambi di padre sono limitati a $\log_2 n$ per foglia, si consumano un totale di $n \log_2 n$ crediti.

Costo di m find, n makeSet e $n - 1$ makeUnion: $\mathcal{O}(m + n \log_2 n)$.

Algoritmi su grafi

Dijkstra: Dimostrazione di Correttezza

Proprietà (1)**.** *Un sottocammino di un cammino minimo è un **cammino minimo**.*

Proprietà (2)**.** *Siano S l'insieme di vertici già considerati dalla visita e D l'insieme dei vertici ancora da considerare:*

2.1. il $d[v]$ di ogni vertice in S non viene più modificato

2.2. tutti i predecessori dei nodi nella coda di priorità sono in S

2.3. per ogni nodo (eccetto s), $d[v] \neq \infty$ se e solo se il predecessore di quel nodo non è nullo

2.4. per ogni nodo (eccetto s), $d[v] \neq \infty$ implica $d[v] = d[\pi[v]] + W(\pi[v], v)$

Proprietà (3)**.** *Se un cammino ha distanza diversa da ∞, esiste un cammino tra due nodi nel grafo.*

Dimostrazione. Per ipotesi induttiva, esiste un cammino da s a $\pi[u]$. Allora, il cammino da s a $\pi[u]$ più l'arco $(\pi[u], u)$ costituisce un cammino da s a u.
Supponiamo, per assurdo, che tra s e u vi sia almeno un cammino $s \equiv v_1 \to v_2 \to \cdots \to v_{k-1} \to v_k \equiv u$ e che u venga estratto dalla coda con $d[u] = \infty$. v_{k-1} è già stato estratto quindi $d[u] = d[v_k] = d[v_{k-1}] + W(v_{k-1}, v_k)$. Ma $W(v_{k-1}, v - k) < \infty$ e $d[u] = \infty$, quindi $d[v_{k-1}] = \infty$. Questo può essere iterato per ciascun vertice del cammino, contraddicendo $d[s] = 0$. □

Il predicato $\forall t \in S : d[t] = \delta(s, t)$ è un'invariante del ciclo while.

Dimostrazione. **Caso base**. Il predicato è vero perchè all'inizion S è vuoto.

Passo. Dimostriamo che per il nuovo vertice u estratto da D, $d[u] = \delta(s, u)$.

Caso 1. Il vertice estratto da D ha $d[u] \neq \infty$. Sia $\pi[u] = r \neq NULL$ per proprietà 2.3. Sappiamo allora che r è nell'albero dei cammini minimi e che $d[u] = d[r] + W(r, u)$ (per proprietà 2.4).
Supponiamo, per assurdo, che tra s e u esista un cammino di peso minore di $d[u]$: esso deve contenere un arco che tra un vertice in s e uno in D, poniamo siano x il vertice in S e y quello in D. Questo cammino può essere visto come la concatenazione di tre cammini $(s \rightsquigarrow x \rightsquigarrow y \rightsquigarrow u)$. Se $s \rightsquigarrow x \rightsquigarrow y \rightsquigarrow u$ è minimo, anche $s \rightsquigarrow x \rightsquigarrow y$ è minimo, quindi $d[y] = \delta(s, y)$. Si ottiene quindi $d[y] + W(y \rightsquigarrow u) \geq d[y] \geq d[u]$, perchè u è stato estratto e ha $d[u]$ minimo.
Quindi $W(s, \ldots x, \ldots y, \ldots u) = d[y] + W(y \rightsquigarrow u)$ non può essere minore di $d[u]$.

Caso 2. Se u è estratto con $d[u] = \infty$, allora (proprietà 3) non esiste alcun cammino tra s e u. □

Minimo Albero Ricoprente: Lemma del Taglio

Un **taglio** è una partizione dell'insieme V di tutti i nodi del grafo in due parti non vuote, S e $V - S$. Si dice che un arco (u, v) **attraversa il taglio** se i suoi estremi u e v appartengono uno ad una parte ed uno all'altra.

Sia A un insieme di archi appartenenti ad un MAR di un grafo G. Consideriamo un taglio non attraversato da alcun arco di A; siano S e $V - S$ le sue due parti. Sia (u, v) l'arco di peso minimo fra tutti gli archi del grafo che attraversano il taglio (chiamato **arco leggero**): allora (u, v) appartiene a un MAR che estende A, cioè l'insieme $A \cup \{(u, v)\}$ è anch'esso un sottoinsieme di un MAR del grafo G.

Proprietà (1)**.** *Se in un albero libero si elimina un arco, si ottengono due alberi.*

Proprietà (2)**.** *Se si connettono due alberi, si ottiene un albero.*

Dimostrazione. Dato un grafo G, siano:

- A: insieme di archi di G che supponiamo siano appartenenti ad uno stesso MAR di G
- $(S, V - S)$: un taglio che non taglia nessun arco di A (ma taglia archi di G)
- (u, v): arco di peso minimo fra quelli di G tagliati dal taglio $(S, V - S)$

Un MAR di G che estende A è per definizione un albero di peso minimo fra tutti gli alberi ricoprenti di G che contengono A. Un albero ricoprente AR contenente A deve connettere tutti i nodi di G, quindi deve contenere un cammino tra u e v. Poichè u e v si trovano da parti opposte del taglio, un tale cammino deve contenere almeno un arco (x, y) che attraversa il taglio (possono anche coincidere). Se nell'albero AR sostituiamo l'arco (x, y) con l'arco (u, v), si ottiene ancora un albero ricoprente di peso totale minore o uguale al peso di AR. □

Minimo Albero Ricoprente: Teorema dell'unicità del MAR

Teorema (Unicità del MAR)**.** *Se i pesi degli archi sono tutti distinti, il MAR è unico.*

Dimostrazione. Per assurdo, supponiamo che G abbia due minimi alberi ricoprenti distinti, $M1$ e $M2$. Poichè sono distinti, esiste almeno un arco in uno dei due alberi che non appartiene anche all'altro. Sia e l'arco di peso minimo che appartiene solo ad uno dei due MAR (in questo caso, $M1$). Se si aggiunge e ad $M2$, si crea un ciclo C. Poichè $M1$ non contiene cicli, nel ciclo c'è almeno un arco e' che non appartiene a $M1$: questo arco ha peso maggiore di e in quanto abbiamo scelto e come arco di peso minimo che appartiene as uno dei due alberi ma non all'altro.
Togliendo e' dal ciclo, si ottiene un albero $M'2$ diverso da $M2$; $M'2$ ha un peso minore di $M2$, fatto che contraddice l'ipotesi iniziale. □

Prim: Dimostrazione di Correttezza

Sia S l'albero di visita costruito. S conterrà i nodi definitivi (quelli nell'albero) e $V - S$ contiene i nodi non definitivi (non nell'albero).

Invarianti:

- **IS**: tutti gli archi dell'albero S appartengono ad un qualche minimo albero ricoprente dell'intero grafo G
- **ID**: per ogni nodo x non definitivo, $d[x]$ è il peso dell'arco più leggero che collega x ad un nodo nero

Dimostrazione. **Caso base**. Dopo la prima iterazione, c'è solo un nodo definitivo, s; l'albero S non contiene nessun arco; ogni nodo x adiacente a s ha distanza $d[x]$ uguale al peso dell'arco (s, x) e ogni altro nodo ha distanza uguale a ∞. Quindi, **IS** e **ID** sono soddisfatti.

Passo. Scegliamo un taglio che separi i nodi neri dagli altri: sicuramente non taglia nessun arco di S. Scegliamo u con $d[u]$ minore tra i nodi non neri. S è sottoinsieme di un MAR, e per il lemma del taglio (y, u) appartiene ad un MAR di G che estende S (IS si mantiene). Per ripristinare ID, basta controllare se il nuovo nodo u avvicina qualche suo adiacente ad S. □

Kruskal: Dimostrazione di Correttezza

Invariante Gli archi in A definiscono una foresta che è sottoinsieme di un certo MAR.

Caso base L'invariante è banalmente vero all'istante iniziale, quando A non contiene nessun arco.

Passo

Caso 1 Si sceglie (u, v) come l'arco più leggero non ancora considerato. Nel primo caso, u e v appartengono allo stesso albero; l'aggiunta di (u, v) creerebbe un ciclo, quindi viene scartato.

Caso 2 Nel secondo caso, u e v appartengono a due alberi distinti T_1 e T_2. Consideriamo un taglio che non tagli nessun arco di A, e che abbia T_1 e T_2 da parti opposte, tagliando l'arco (u, v). L'arco (u, v) è l'arco di peso minimo fra tutti gli archi che attraversano il taglio. Quindi, per il lemma del taglio, sappiamo che $(u, v) \cup A$ è un sottoinsieme di un MAR.

Bellman-Ford: Dimostrazione di Correttezza

Sia $\delta(s, v_k)$ a distanza da s a v_k.

Definiamo l'invariante **KPATH**: dopo la k-esima iterazione del ciclo esterno, si ha $d[v_k] = \delta(s, v_k)$ per ogni nodo v_k per cui il cammino minimo da s a v_k è composto al più da k archi.

Dimostrazione. **Base**. per $k = 0$, per il cammino da s a s, $d[s] = 0$ e $d[u] = \infty$ per tutti gli altri nodi u. **Passo**. Dimostriamo che KPATH sia vero anche dopo $k + 1$ iterazioni. Sia v_{k+1} un nodo il cui cammino minimo $s \rightsquigarrow v_k \rightsquigarrow v_{k+1}$ è composto da $k + 1$ archi. Sappiamo che $\delta(s, v_{k+1}) = \delta(s, v_k) + W(v_k, v_{k+1})$ e $d[v_k] = \delta(s, v_k)$. Quindi: $\delta(s, v_{k+1}) = d[v_k] + W(v_k, v_{k+1})$.

Al passo $k + 1$, ci sono due casi:

- $d[vk + 1]$ viene aggiornata, allora $d[vk + 1] = \delta(s, v_{k+1})$
- $d[vk + 1]$ non viene aggiornata, poichè $d[vk + 1]$ è il peso di un cammino da s a v_{k+1} in G, $d[vk + 1] \geq \delta(s, v_{k+1})$, e poichè non è stata aggiornata, $d[vk + 1] = \delta(s, v_{k+1})$.

I cammini restituiti dall'algoritmo sono effettivamente quelli minimi. □

Floyd-Warshall: saper spiegare come si arriva all'algoritmo, partendo dalla definizione di distanze k-vincolate

Denotiamo i vertici del grafo come $v1, v2, \ldots, v_n$. Per un k fissato con $1 \leq k \leq n$ definiamo: **cammino minimo k-vincolato** tra x e y (denotato con π^k_{xy}) il cammino che va da x a y di costo minimo tra tutti quelli che non contengono i vertici $\{v_{k+1}, \ldots, v_n\}$ e **distanza k-vincolata** (denotata con d^k_{xy}), che è il peso W se π^k_{xy} esiste, ∞ altrimenti.

Un cammino minimo k-vincolato è un cammino minimo in un **grafo k-vincolato** G_k: varrà quindi la proprietà della sottostruttura ottima e sarà possibile applicare tecniche di programmazione dinamica.

Per ogni $1 \leq k \leq n$ e per ogni coppia x, y definiamo l'equazione ricorsiva:

$$d^k_{xy} = min(d^{k-1}_{xy}, d^{k-1}_{xv_k} + d^{k-1}_{v_k y})$$

In poche parole, preso un k, si va a vedere se la concatenazione dei cammini minimi da x a v_k e da v_k a y ha peso minore del cammino da x a y e si verifica la disuguaglianza triangolare: se **non** si verifica, si applica un rilassamento facendo passare il cammino minimo attraverso v_k. Abbiamo due casi:

- $v_k \notin \pi^k_{xy}$: allora π^k_{xy} è anche un cammino minimo $(k-1)$ vincolato; se così non fosse, esisterebbe un cammino minimo $(k-1)$ vincolato di peso minore, ma questo sarebbe anche un cammino k-vincolato con peso minore di π^k_{xy} (assurdo)
- $v_k \in \pi^k_{xy}$: i sottocammini da x a v_k e da v_k a y sono cammini minimi k-vincolati (per sottostruttura ottima); poichè non contengono internamente v_k, essi sono anche cammini $(k - 1)$ vincolati.

Complessità e approssimazione

Perché si sospetta che $P \subset NP$ e $NP \subset PSPACE$

Dimostrazione che il problema dell'Halt limitato è NP completo

Un problema NP-completo è quello dell'**Halt limitato**: dato un programma X ed un intero k, verificare se esiste un input per cui X termina in al massimo k passi.

Appartiene ad NP perchè la fase di verifica può essere effettuata in tempo polinomiale simulando $X(i)$ per k passi.

Ongi problema appartenente ad NP è riconducibile polinomialmente ad halt limitato.

Sia $Prob \in NP$, allora esiste un programma C che verifica un certificato in tempo polinomiale $p(n)$. Allora è possibile costruire un programma C' che va in loop ogniqualvolta la risposta di C sia negativa, e termina altrimenti. È possibile riscrivere $Prob$ come il problema dell'halt limitato, con C' come programma X e $p(n)$ come k. L'halt limitato restituirà 1 se esistono dei dati per cui C' termina in $p(n)$ passi, 0 altrimenti.

Il problema dell'halt limitato è NP e NP-hard, quindi è NP-completo.

Dimostrazione che il problema dell'Halt è indecidibile

Esistono una serie di problemi definiti **indecidibili**: non solo non si conoscono algoritmi per risolverli, ma è stato dimostrato che tali algoritmi non possono esistere, nemmeno utilizzando tempo e spazio infiniti.

Il problema dell'halt è un problema indecidibile.

Definire un programma $halt(P, i)$ che restituisca 1 se il programma $Prog$ (con un certo input i) termina in un numero finito di passi, 0 altrimenti.

Se potessimo scrivere halt, potremmo scrivere un programma G come:

Algoritmo 1 G(prog)

```
if halt(Prog,Prog) then
    loop
else return 0
end if
```

$G(G)$ termina solo se $G(G)$ non termina, che è una contraddizione.

Algoritmo Approssimato per Copertura Vertici: saper spiegare perché è un algoritmo 2-approssimato

APPROX-COVER ha un fattore di approssimazione di 2; si dice che è **2-approssimato**.

Dimostrazione. Sia A l'insieme di archi (u, v) scelti all'inizio di ogni ciclo. Per coprire gli archi in A, una copertura ottima deve contenere almeno un vertice di ciascun arco in A. Due archi in A non possono avere un vertice in comune, perchè una volta scelto (u, v) tutti gli archi incidenti ad u o v sono rimossi da E'. Pertanto, non ci sono due archi in A coperti dallo stesso vertice e $|A|$ è un limite inferiore per la dimensione di una copertura ottima $C*$. Ma $|C| = 2|A|$, poichè nel risultato metto sia u che v , quindi

$$|C| = 2|A| \leq 2|C^*|$$

□

Algoritmo Approssimato per TSP: saper spiegare perché è un algoritmo 2-approssimato

Sia H^* il circuito ottimale per l'insieme di vertici in considerazione. Rimuovere un qualsiasi arco dal circuito crea un albero ricoprente T. Quindi, il peso del MAR computato dall'algoritmo fornisce un lower bound per il costo del circuito ottimale:

$$c(T) \leq c(H^*)$$

Un cammino completo su T (circumnavigazione, sia W) contiene ogni vertice due volte. Quindi si ha:

$$c(W) = 2c(T)$$

La disuguaglianza vista prima implica che:

$$c(W) \leq 2c(H^*)$$

Quindi, il costo di W è 2-approssimato.