

STRUTTURE DATI PER INSIEMI DISGIUNTI

Serve per mantenere una collezione $S = \{S_1, \dots, S_k\}$ di insiemi dinamici disgiunti; ciascun insieme è identificato da un rappresentante, che è un elemento dell'insieme. Può essere scelto con una regola prestabilita oppure senza particolari condizioni.

Operazioni

Indicando con x e y degli oggetti rappresentanti:

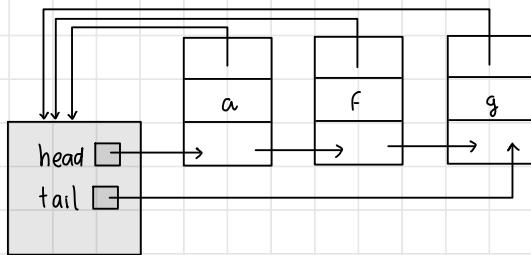
- $\text{Make-set}(x)$ crea un nuovo insieme
- $\text{Union}(x, y)$ unisce gli insiemi dinamici che contengono x e y
- $\text{Find-set}(x)$ restituisce un puntatore al rappresentante dell'insieme

Liste concatenate

Ciascun insieme è rappresentato da una lista concatenata. L'oggetto di ciascun insieme ha una **head** che punta al primo oggetto della lista e una **tail** che punta all'ultimo.

Ogni oggetto è fatto da:

- un valore
- puntatore al rappresentante
- puntatore al successivo



Per le operazioni di Make-set e Find-set non ci sono particolarità di implementazione e richiedono un tempo $O(1)$.

Una Union richiede invece un tempo $O(n^2)$, perché si richiede di aggiornare il puntatore di ogni oggetto della lista assorbita e si fa su n oggetti.

Euristica dell'unione pesata

Si supponga che ogni lista memorizzi anche la propria lunghezza, e si supponga di operare sempre la lista più piccola a quella più Lunga.

In questo modo, una sequenza di m operazioni Make-set , Union e Find-set , n delle quali sono Make-set , impiega un tempo $O(m \cdot n \log n)$.

TEOREMA Utilizzando l'euristica precedente, una sequenza di m operazioni Make-set, Union e Find-set, n delle quali sono Make-set, impiega un tempo $O(m+n\log n)$.

DIMOSTRAZIONE Ogni operazione Union unisce due insiemi disgiunti, quindi sono al più $n-1$ operazioni.

Considerando l'oggetto x , ogni volta che viene aggiornato si trova sempre nell'insieme più piccolo:

1° aggiornamento almeno 2 elementi
2° aggiornamento almeno 4 elementi
...
 $\lceil \log k \rceil$ almeno k elementi

Dato che ci sono al più n elementi, il puntatore di ciascun oggetto è stato aggiornato al più $\lceil \log n \rceil$ volte in tutte le operazioni Union.

Quindi il tempo totale è $O(n\log n)$.

Le altre operazioni impiegano $O(1)$ e sono in numero $O(m)$: il tempo totale è quindi la loro somma $O(m+n\log n)$.

Foreste di insiemi disgiunti

Implementazione più veloce, in cui si rappresentano gli insiemi con alberi radicati, dove ogni nodo contiene un elemento e ogni albero rappresenta un insieme.

In ogni foresta, ogni elemento punta al padre, il nodo radice contiene il rappresentante ed è padre di sé stesso.

Euristiche

Introdotte per consentire una rappresentazione con foreste più asintoticamente ottimali.

Unione per rango

Fare in modo che la radice dell'albero con meno nodi punti alla radice dell'albero con più nodi.

Il rango è un limite superiore per l'altezza del nodo.

Nell'unione per rango, la radice con il rango più piccolo viene fatta puntare alla radice con il rango più grande durante un'operazione di Union.

Compressione del cammino

Viene utilizzata durante le operazioni Find-Set per fare in modo che ciascun nodo nel cammino di ricerca punti direttamente alla radice.

La compressione del cammino non cambia i ranghi.

Pseudocodice per foreste

Per l'implementazione, bisogna tenere traccia dei ranghi, mantenendo il valore intero $x.rank$ per ogni nodo x , che rappresenta il numero di archi nel cammino semplice più lungo fra x e una foglia sua discendente.

Usando entrambe le euristiche, richiede tempo $O(m+2(m,n))$, con $2(m,n) \leq 4$, rendendo praticamente lineare in m la complessità

SISTEMI DI INDEPENDENZA

$\langle E, F \rangle$ è un sistema di indipendenza se:

- E è un insieme finito ($|E| < \infty$)
- F è una famiglia di sottoinsiemi di E ($F \subseteq \mathcal{P}(E)$)
tale che $\forall A \in F, \forall B \subseteq A$ vale che $B \in F$

Concetto chiave: i sottoinsiemi di sottoinsiemi sono a loro volta ammissibili.

Esempi di sistemi di indipendenza

\bar{E} un sistema di indipendenza.

- E finito
- $F = \mathcal{P}(E)$
- $E \leq V$ spazio vettoriale, un insieme finito di vettori
- $F = \{A \subseteq E \mid A \text{ formato da vettori linearmente indipendenti}\}$
- $E = \{K_1, \dots, K_n\}$ $\text{ING}: E \rightarrow \mathbb{N}_+$
- $F = \{A \subseteq E \mid \text{ING}(A) \leq H\}$
- $E = \text{insieme degli archi di } G = (V, E) \text{ non orientato}$
- $F = \{A \subseteq E \mid (V, A) \text{ è una foresta e un sottografo}\}$

Sistema indotto

Sia $\langle E, F \rangle$ un sistema di indipendenza e sia $C \subseteq E$. Si definisce $\langle C, F_C \rangle$ la coppia in cui $F_C = \{A \in F \mid A \subseteq C\}$

La coppia $\langle C, F_C \rangle$ è un s.i. ed è chiamata sottosistema o sistema indotto da C .

Siano $\langle E, F \rangle$ e $C \subseteq E$. Sia $A \in F$. Allora $A \cap C \in F_C$, infatti $A \cap C \subseteq A$ e quindi $A \cap C \in F$
Ma vale anche che $A \cap C \subseteq C$ e quindi $A \cap C \in F_C$

Massimale

Sia $\langle E, F \rangle$ un s.i., $A \in F$ è massimale sse $\forall e \in E \setminus A \quad A \cup \{e\} \notin F$, cioè non è estendibile.

$A \in F$ massimale in $\langle E, F \rangle \Rightarrow A \cap C$ massimale in $\langle C, F_C \rangle$

PROBLEMA DI MASSIMO ASSOCIATO AD UN S.I. PESATO

Si supponga $\langle E, F \rangle$ un s.i. e $w: E \rightarrow R_+$ una funzione valore con $w(A) = \sum_{x \in A} w(x)$
Si definiscono:

istanza $\langle E, F \rangle$, $w: E \rightarrow R_+$

soluzione $M \in F$ massimale di valore massimo $w(M) = \max_{A \in F} \{w(A)\}$

Si associa l'algoritmo general purpose che può non dare l'ottimo, ma che sicuramente ritorna sempre un sottoinsieme massimale:

Greedy-max(E, F, w):

```
S = ∅  
Q = E  
Q = Sort(Q)  
For i = 1 to n.  
  if  $S \cup \{Q[i]\} \in F$   
    S =  $S \cup \{Q[i]\}$   
return S
```

Ha un costo $O(n \log n + n \cdot C(n)) = n \log n$
Con $C(n)$ il costo, presupposto costante,
per verificare se l'insieme è in F

MATROIDI

Un s.i. $\langle E, F \rangle$ è un matroide sse vale la seguente proprietà di scambio:

$\forall A, B \in F$ con $|B| > |A| \exists b \in B \setminus A$ t.c. $A \cup \{b\} \in F$

\hookrightarrow sostituibile con $|B| = |A| + 1$

Cioè, per ogni coppia, deve esistere un elemento che se aggiunto da B ad A ciò che si ottiene è ancora parte del s.i., per questo detta "di scambio".

TEOREMA $\langle E, F \rangle$ è un matroide sse $\forall C \subseteq E$ gli insiemi massimali di $\langle C, F_C \rangle$ hanno la stessa cardinalità.

DIMOSTRAZIONE Per assurdo non hanno uguale cardinalità, si supponga $|B| > |A|$. Ma allora A è estendibile quindi non massimale. Ne consegue che $|A| = |B|$.

TEOREMA DI RADO

Sia $\langle E, F \rangle$ un S.I., le seguenti proposizioni sono equivalenti:

- 1) $\langle E, F \rangle$ è un matroide sse
- 2) $\forall w: E \rightarrow \mathbb{R}_+$, Greedy-Max(E, F, w) risolve il problema di massimo associato, ovvero calcola un sottoinsieme massimale di peso massimo.

LEMMA $\langle E, F \rangle$ è un matroide sse $\forall c \subseteq E$, gli insiemi massimali di (C, F_C) hanno la stessa cardinalità.

DIMOSTRAZIONE Per dimostrare che da 1 consegue 2, si assuma $\langle E, F \rangle$ come matroide e scelta arbitrariamente $w: E \rightarrow \mathbb{R}_+$ mostriamo che Greedy-Max(E, F, w) risolve il pb di max associato.

Sia $S = \{a_1, \dots, a_p\}$ la soluzione fornita con $w(a_1) \geq \dots \geq w(a_p)$.

Per come è fatto G.max , sappiamo che S è massimale. Sia $S' \subseteq F$ un qualunque sottoinsieme massimale in F , vogliamo dimostrare che $w(S) \geq w(S')$, ossia che S è soluzione del pb di max associato.

Siccome S' è massimale ed $\langle E, F \rangle$ è un matroide, vale che $|S'| = |S| = p$.

Siano quindi b_1, \dots, b_p gli elementi S' con $w(b_1) \geq \dots \geq w(b_p)$ e si assume $S \cap S' = \emptyset$.

Consideriamo i 2 seguenti casi mutualmente esclusivi:

- 1) $\forall i \in \{1, \dots, p\} w(a_i) \geq w(b_i)$. In tal caso si ottiene che $w(S) \geq w(S')$ TRUE
- 2) $\exists i \in \{1, \dots, p\} w(a_i) < w(b_i)$. Si: a K il più piccolo indice i , vale allora che:

$$w(a_K) < w(b_K) \leq w(b_{K-1}) \leq w(a_{K-1})$$

$\downarrow \quad \downarrow \quad \downarrow$
per $i = K$ ordinat; per $i = K-1$ non vale *

Si definisca $C = \{e \in E \mid w(e) \geq w(b_K)\}$. Si ha che:

- $b_1, \dots, b_K \in C$
- $a_1, \dots, a_{K-1} \in C$ ma $a_K \notin C$

Pertanto $|S' \cap C| > |S \cap C|$.

Si ottiene una contraddizione in quanto $S \cap C$ e $S' \cap C$ dovrebbero essere la stessa cardinalità, in quanto essendo S e S' massimali in $\langle E, F \rangle$ allora $S \cap C$ e $S' \cap C$ sono massimali in $\langle C, F_C \rangle$ essendo $\langle E, F \rangle$ un matroide allora lo è anche $\langle C, F_C \rangle$.

In conclusione, si può dimostrare solo il caso 1, quindi la conseguenza è verificata.

DIMOSTRAZIONE Per dimostrare che da 2 consegue 1, si dimostri che dalla negazione di 1 consegue la negazione di 2.

Si assume che $\langle E, F \rangle$ non è un matroide, si deve verificare che, per almeno un peso, non funziona, ovvero $\exists w: E \rightarrow R_+$ tale che Greedy-max(E, F, w) non restituisce l'ottimo

Siccome $\langle E, F \rangle$ non è un matroide, allora $\exists C \subseteq E$, $\exists A, B \in F_C$ massimali con $|A| > |B| = p$

Si definisce $w: E \rightarrow R_+$, $\forall e \in E$:

$$w(e) = \begin{cases} p+2 & \text{se } e \in B \\ p+1 & \text{se } e \in B \setminus A \\ 1 & \text{altrimenti} \end{cases}$$

Sicuramente B dà gli elementi di peso massimo, infatti $w(B) = p(p+2)$

L'algoritmo restituisce quindi B come massimale.

Ma $w(A) = \text{almeno } p+1$ ciascuno dei quali ha peso almeno $p+1$ ($p+2$ oppure $p+1$).

Da cui $w(A) \geq (p+1)(p+1) = p^2 + 2p + 1 > p(p+2) = p^2 + 2p$

Ne consegue che G.max non sta calcolando l'ottimo, perché A è un candidato migliore di B che non è un ottimo.

ALGORITMO GREEDY

La tecnica Greedy è contraria a quella dinamica: prima si effettua una scelta vincente sul sottoproblema, e poi si risolve

Si sceglie quindi il valore massimo come prima scelta, e poi si risolve sui sottoproblemi rimanenti.

TEOREMA Sia (V, E) non orientato e sia (V, C) una foresta con $C \subseteq E$, allora il numero di alberi è: $|V| - |C|$.

DIMOSTRAZIONE sia T il numero di alberi che compongono la foresta (V, C) e siano:

• v_i n° di vertici albero i nella foresta (V, C)

• e_i n° di archi albero i nella foresta (V, C)

$$|V| = |\mathbb{E}| + t$$

Allora $|C| = \sum_i e_i = \sum_i (v_i - 1) = |V| - t$ da cui $|C| = |V| - t$ e infine $t = |V| - |C|$.

TEOREMA Sia (V, E) un grafo non orientato e sia $\langle E, F \rangle$ un sistema di indipendenza con $F = \{A \subseteq E \mid (V, A) \text{ è una foresta}\}$.

Allora $\langle E, F \rangle$ è un matroide, ovvero vale la proprietà di scambio.

DIMOSTRAZIONE Siano $A, B \in F$ con $|A| < |B|$ arbitrariamente scelti, si deve dimostrare che $\exists b \in B \setminus A$ t.c. $A \cup \{b\} \in F$

Si ha che:

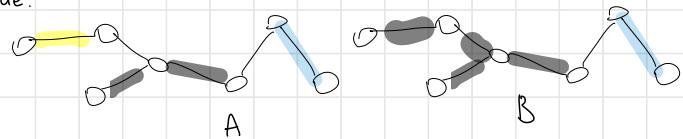
- (V, A) è una foresta con $t = |V| - |A| \rightarrow$ ha più alberi (eq. meno archi)
- (V, B) è una foresta con $t = |V| - |B| \rightarrow$ ha più archi

Quindi $|V| - |B| < |V| - |A|$, (V, B) deve contenere un qualche albero T i cui vertici sono in differenti alberi di (V, A) .

Siccome T è连通的, allora $\exists (u, v)$ in T t.c. u e v si trovano in alberi differenti di (V, A) .

Di conseguenza (u, v) può essere aggiunto ad A senza creare cicli.

Ne consegue che $\langle E, F \rangle$ è un matroide.



■ PROBLEMA DI MINIMO

Si riconduce al problema di massimo, ordinando in senso crescente i valori w associati.

Si può, invece, anche operare una proporzione dei pesi del tipo:

- Sia $K = \max\{w(e) \mid e \in E\}$
- $w'(e_i) = K - w(e_i) \quad \forall i - \text{arco}$

Ma, per semplicità, si opera la seguente equivalenza per evitare la definizione di un'altra funzione peso:

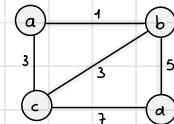
$$\text{Equivalenti} \begin{cases} \text{Greedy-Max}(E, F, w') \rightarrow S & [Q \text{ decrescente}] \\ \text{Greedy-Min}(E, F, w) \rightarrow S' & [Q \text{ crescente}] \end{cases}$$

ALBERO DI COPERTURA MINIMO

Dato $G = (V, E)$ un grafo non orientato e connesso e pesato sugli archi da $w : E \rightarrow \mathbb{R}_+$, un albero di copertura minima di G , o albero ricoprente minimo MST, è una coppia (V, T) con $T \subseteq E$ tale che (V, T) sia aciclico e connesso, i.e. un albero.

Inoltre $w(T) = \min\{w(A) : A \subseteq E \text{ è un albero}\}$

Per essere tale, deve coprire tutti i vertici del grafo G .



G è non orientato, e T è, essendo albero, anche una foresta. Dunque esiste un algoritmo Greedy che fornisce un ottimo.

Di conseguenza, Greedy-min fornisce un MST, perché sicuramente fornisce il peso minimo, e in più è quello massimale.

Inoltre, in quanto connesso, ricopre tutti i vertici sicuramente.

TEOREMA di definizione ad albero. $G = (V, E)$ grafo non orientato. G è un albero se:

- è anche connesso e aciclico
- è connesso e $|E| = |V| - 1$
- è aciclico e massimale, i.e. se si aggiunge un arco qualsiasi il grafo che ne deriva contiene un ciclo

Arco sicuro

Dati un grafo connesso non orientato e pesato $G = (V, E)$, un sottoinsieme A dell'insieme T di archi di un MST e un qualsiasi taglio che rispetti A , allora un arco leggero (u, v) del taglio è sicuro per A , cioè $A \cup \{(u, v)\} \subseteq T$.

Corollario Dati un grafo connesso non orientato e pesato $G = (V, E)$, un sottoinsieme A degli archi di MST, una componente连通的 $C = (V_C, E_C)$ di $G_A = (V, A)$, allora un arco leggero (u, v) del taglio $(V_C, V - V_C)$ è un arco sicuro per A .

Algoritmo generico

Generic-MST (G, w)

```
S = ∅  
while S non forma un MST  
    trova un arco  $(u, v)$  sicuro per S  
    S = S ∪ {(u, v)}  
return S
```

ALGORITMO DI KRUSKAL

In Kruskal, l'arco che si aggiunge ad S , considerato sicuro, è uno di peso minimo tra quelli che collegano una componente连通的 di (v, s) ad una diversa componente连通的 di (v, s) .

Pseudocodice

Si appoggia all'algoritmo di Greedy-Min.

Greedy-Min(E, F, w):

$S = \emptyset$

$Q = E$

$Q = \text{Sort}(Q)$ \& crescente

for each $v \in V$

make-set(v)

for each $(u, v) \in Q$

if $S \cup \{(u, v)\} \subseteq F \iff \text{FIND-SET}(u) \neq \text{FIND-SET}(v)$

$S = S \cup \{(u, v)\}$

union(u, v)

return S

$O(|E| \log |E|)$

Union $\Rightarrow O(m)$

$m = |V| + O(|E|)$

↳ Union e find-set
↳ make-set

Siccome G è connesso $|E| \geq |V| - 1$

$|V| \leq |E| + 1$

$|V| = O(|E|)$ quindi $m = O(|E|)$

Quindi $O(|E| \log |E|) + O(m) \equiv O(|E| \log |E|) + O(|E|) \Rightarrow O(|E| \log |E|)$

Ma $|E| \leq |V|^2$, $\log |E| < \log |V|^2 < 2 \log |V|$. Quindi si può abbassare a $O(|E| \log(|V|))$.

■ ALGORITMO DI PRIM

L'idea alla base è quella di aggiungere un arco sicuro fintantoché esiste un taglio e degli archi tra V' a V .

Taglio

$(V', V \setminus V')$ con $V' \subseteq V$ è una partizione di vertici tc. $\{V', V \setminus V'\}$ è una partizione rispetto a un sottoinsieme di archi $S \subseteq E$ se nessun arco di S attraversa il taglio.

(u, v) attraversa $(V', V \setminus V')$ se $u \in V'$ e $v \in (V \setminus V')$ o viceversa.

$(u, v) \in E$ che attraversa un taglio è leggero sse ha peso minimo tra gli archi che attraversano il taglio.

Pseudocodice

$\forall v$ peso dell'arco di cammino minimo che collega $u \in V'$

$V \setminus V'$ memorizzato in un MIN-HEAP, ovvero una coda con priorità

$V \in (V \setminus V')$ ha attributi:

Key minimo valore tra i pesi degli archi che collegano vertici di (V', S) a V , ∞ se non ce ne sono

- π predecessore di v all'interno dell'albero, qualora v entrasse nell'albero.

MST-Prim(G, W, r)

for each v in V do

$v.Key = \infty$

$v.\pi = \text{NIL}$

$r.Key = 0$

$Q = V$

 while $Q \neq \emptyset$ do:

$v = \text{extract_min}(Q)$

 for each $z \in \text{adj}[v]$

 if $z \in Q$ and $w(z, v) < z.Key$

$z.Key = w(z, v)$

$z.\pi = v$

|

|

|

$O(|V|)$

|

$O(|E|)$

|

$O(|E|)$

|

$O(|E| \log |V|)$

|

$O(|V|)$

$O(\log |V|)$

|

$O(|E| \log |V|)$

|

$\} O(|V| \log |V|)$

|

$O(|E| \log |V|)$

|

$O(|E| \log |V|)$

Dalle notazioni della complessità, si ottiene: $O(|V|\log|V| + |E|\log|V|) \equiv O(|E|\log|V|)$ poiché $|V| \leq |E| + 1$.

Non c'è differenza computazionale tra i due algoritmi di costruzione di MST.

■ ALGORITMO DI DIJKSTRA

Algoritmo per il calcolo dei cammini minimi da una sorgente unica, ovvero il vertice $s \in V$, detto vertice sorgente. Restituisce quindi $\forall v \in V$ diverso da s , il cammino di peso minimo che inizia in s e termina in v .

Effettivamente, si costruisce un albero $G' = (V', E')$ di cammini minimi con radice il vertice s tale che:

- V' è l'insieme dei vertici raggiungibili da s
- G' è un albero di radice s
- $\forall v \in V'$, l'unico cammino minimo da s a v è un cammino minimo da s a v in G .

Scomposizione

Il cammino minimo dalla sorgente s al vertice v , rappresentato da $\delta(s, v)$ può essere scomposto in $\delta(u, v) = \delta(s, u) + w(u, v)$, dove u è il predecessore di v nel cammino.

Limite superiore

Quindi dato un qualsiasi arco (u, v) si ha che: $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Tecnica del rilassamento

Si aggiungono ad ogni vertice v i due attributi d e π che indicano un limite superiore e il predecessore.

Prima dell'esecuzione:

- $d = 0$ per s
- $d = \infty$ per ogni vertice v , tranne s
- $\pi = \text{NIL}$ per ogni vertice v

Alla fine, invece, indicheranno rispettivamente il peso del cammino minimo da s a v e il predecessore di v nel cammino minimo.

La tecnica definisce il seguente algoritmo di rilassamento.

Relax(u, v, w):

```
| if  $r.d > u.d + w(u, v)$ 
|    $r.d = u.d + w(u, v)$ 
|    $v.\pi = u$ 
```

Pseudocodice

Dato l'algoritmo di rilassamento, si può definire:

Dijkstra(G, w, s):

```
| For each  $v \in V$ :
```

```
|    $v.d = \infty$ 
```

```
|    $v.\pi = \text{NIL}$ 
```

```
|    $s.d = 0$ 
```

```
|    $V = \emptyset$ 
```

```
|    $Q = \text{priority\_queue}(v)$ 
```

```
| while  $Q \neq \emptyset$ :
```

```
|    $u = \text{extract\_min}(Q)$ 
```

```
|    $V = V \cup \{u\}$ 
```

```
|   For each  $v \in \text{adj}[u]$ :
```

```
|     Relax( $u, v, w$ )
```

Prova di correttezza

TEOREMA Sia $\langle v_1 = s, v_2, \dots, v_n \rangle$ la sequenza di vertici estratti da Q in un'esecuzione. Quando il vertice v_k viene estratto, allora $v_k.d = \delta(s, v_k)$.

LEMMA Se $v.d = \delta(s, v)$ a qualche passo dell'esecuzione, allora sicuramente $v.d$ rimarrà tale per il resto dell'esecuzione.

DIMOSTRAZIONE Il teorema è vero per $v = s$, infatti $s.d = 0$ e $\delta(s, s) = 0$.

Si dimostri anche per un v_k , generico vertice estratto da Q ma non ancora aggiunto a S :

$$\underbrace{\langle v_1, v_2, \dots, v_{k-1}, v_k, v_{k+1}, \dots, v_n \rangle}_S$$

In questa configurazione, sicuramente $v_k.d \leq v_i.d$ con $k+1 \leq i \leq n$. Inoltre il teorema è vero fino a v_{k-1} , si deve dimostrare vero per v_k , cioè $v_k.d = \delta(S, v_k)$.

Per costruzione, il cammino minimo da S a v_k comprende, eccetto v_k , solo vertici in S .

Si ponga per assurdo $v_k.d > \delta(S, v_k)$, allora il cammino minimo da S a v_k include anche vertici in $V-S$.

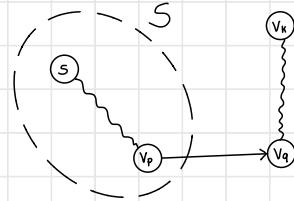
Quindi $v_p.d = \delta(S, v_p)$ per ipotesi

$\Rightarrow v_q.d = \delta(S, v_q)$ quando vengono rilassati

gli archi di v_p

$\Rightarrow \delta(S, v_q) < \delta(S, v_k)$ poiché non esistono archi di peso zero

$\Rightarrow v_q.d < v_k.d$



Ma è una contraddizione con il fatto che v_k viene estratto prima di v_q che non appartiene a S .

BREADTH FIRST SEARCH

Distanza

Dato $G = (V, E)$ la distanza di u da v è il minimo numero di archi da percorrere percorrere per andare da v a u .

Visita in ampiezza

Di un grafo non orientato, a partire da un vertice sorgente s :

- all'inizio viene visitata la sorgente s
- vengono poi visitati tutti gli adiacenti di s
- in seguito, si visitano gli adiacenti degli adiacenti
si prosegue così

In altre parole, si visitano dopo s tutti i vertici a distanza 1, 2, 3... da s .

Proprietà

- Visita tutti e soli i vertici raggiungibili da s
- Ogni vertice del grafo viene visitato al più una volta
- Permette di trovare la distanza, in archi, da s di tutti i vertici raggiungibili dalla sorgente

Color code

Lo stato di un vertice è codificato in colori:

- bianco vertice non visitato
- grigio vertice visitato, adiacenti non completamente visitati
- nero vertice visitato, adiacenti completamente visitati

Utilizza una coda Q , che contiene solo vertici grigi:

- un vertice viene inserito in Q non appena viene visitato
- un vertice visitato rimane in Q finché non viene estratto per esplorare i suoi adiacenti
- quando tutti gli adiacenti di un vertice risultano visitati, il vertice diventa nero
- la visita termina quando Q è vuota

Pseudocodice

BFS (G, s):

```

For each  $v \in V \setminus \{s\}$  do
    color [ $v$ ] = W
     $d[v] = \infty$ 
     $\pi[s] = \text{NIL}$ 
color [ $s$ ] = G
 $d[s] = 0$ 
 $\pi[s] = \text{NIL}$ 
 $Q = \emptyset$ 
enqueue( $Q, s$ )

```

$O(|V|)$

tempo assunto costante
per queste operazioni

```

while  $Q$  is not  $\emptyset$  do
     $v = \text{head}(Q)$ 
    For each  $u \in \text{adj}[v]$  do
        if color [ $u$ ] is W then
            color [ $u$ ] = G
            enqueue( $Q, u$ )
             $d[u] = d[v] + 1$ 
             $\pi[u] = v$ 
    color [ $v$ ] = B
    dequeue( $Q$ )

```

$O(|E|) \rightarrow$ costo ispezione liste
di adiacenza di
dimensione $2|E|$

Albero della BFS

Sfruttando i predecessori trovati dal codice, si genera un albero $T = (V_T, E_T)$:

- $V_T = \{v \in V \mid \text{color}[v] = B\}$
- $E_T = \{(\pi[v], v) \mid v \in V_T\}$

Complessità

Nel caso peggiore è $O(|V| + |E|)$

Output

L'algoritmo produce 3 liste:

- d distanza di ogni nodo dal nodo sorgente
- π predecessore del nodo
- color colore di ogni nodo (raggiunto / non raggiunto)

Variazioni di BFS

1 Dato un grafo, contare i vertici raggiungibili da un determinato vertice u .

Count-Reachable(G, u):

BFS(G, u)

$c = 0$

for each $v \in V \setminus \{u\}$ do

if $d[v] \neq \infty$ then

$c = c + 1$

return c

2 Stampare l'elenco dei vertici che hanno distanza pari da un vertice u .

Even-distance(G, u)

BFS(G, u)

for each $v \in V \setminus \{u\}$ do :

if $d[v] \% 2 = 0$ and $d[v] \neq \infty$ do :

print v

3 Stampare il percorso minimo, in numero di archi, dal vertice u al vertice v .

Min-path(G, u, v)

if $u = v$ then print u

else

BFS(G, u)

if $\pi[v] = \text{NIL}$ then :

print "non raggiungibile"

else print-path(v)

Print-path(w) :

if $\pi[w] \neq \text{NIL}$:

print-path($\pi[w]$)

print w

4 Verifica se un grafo è connesso.

Is_connected (G) :

```
u = random(V)
BFS(G, u)
For each v ∈ V do:
    if d[v] = ∞ then:
        return false
return true
```

5 Modificare il BFS per limitare la visita a distanza al più K.

```
if d[u] < K then
    color[u] = G
    enqueue(Q, u)
else
    color[u] = B
```

6 Verificare se un grafo è un albero

Ricorda: un grafo connesso è privo di cicli se $|E| = |V| - 1$

Is-tree (G)

```
if Is_connected(G) and |E| = |V| - 1 then
    return true
else
    return false
```

7 Ricostruire l'albero della visita.

BFS-tree (G, s)

```
BFS(G, s)
VT = ∅, ET = ∅
For each v ∈ V do:
    if color[v] is B then:
```

```
add v to VT
for each v ∈ VT \ {s} do
    add (π[v], v) to ET
return (VT, ET)
```

DEPTH FIRST SEARCH

È la visita in profondità di un grafo da un vertice sorgente s.

- visita s
- visita il primo adiacente a_1 di s, poi visita il primo adiacente a_2 di a_1 , e così via...
- quando raggiunge un vertice che non ha adiacenti da visitare risale al predecessore e la visita riparte, se possibile, da un altro adiacente di tale predecessore
- ogni volta che non ci sono adiacenti da visitare, si risale al predecessore
- quando la visita risale alla sorgente s e non ha più adiacenti da visitare, si sceglie una nuova sorgente e la visita riparte
- La visita termina quando non ci sono più vertici disponibili per essere scelti come nuova sorgente

I vertici hanno associato un colore come in BFS.

Strutture di base

- Color vettore dei colori
 - π vettore dei predecessori
 - d vettore dei tempi di scoperta (v diventa grigio)
 - f vettore dei tempi di completamento (v diventa nero)
- $\left. \begin{matrix} \text{intervallo temporale di } v \\ \text{ } \end{matrix} \right\}$

Teorema delle parentesi

TEOREMA Dopo una visita in profondità, uno solo dei 3 casi si può verificare per due vertici u e v:

- 1) $[d[u], f[u]]$ contiene $[d[v], f[v]]$, v è discendente di u in un albero della visita
- 2) $[d[u], f[u]]$ contiene $[d[v], f[v]]$, u è discendente di v in un albero della visita
- 3) $[d[u], f[u]]$ e $[d[v], f[v]]$ sono disgiunti, u e v non sono discendenti l'uno dell'altro in un albero della visita

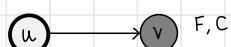
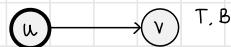
DIMOSTRAZIONE se $d[u] < d[v]$ e $d[v] < f[u]$ verranno ispezionati tutti gli archi uscenti da v prima di riprendere l'ispezione degli archi uscenti da u, quindi v è discendente di u.
Se $f[v] < f[u]$, segue che $[d[u], f[u]]$ contiene $[d[v], f[v]]$.

Se $d[u] < d[v]$ e $f[u] < d[v]$ sicuramente $d[u] < f[u]$ e $d[v] < f[v]$, quindi $d[u] < f[u] < d[v] < f[v]$. Nessuno dei due vertici è stato scoperto mentre l'altro era grigio, nessuno dei due è discendente dell'altro, quindi $[d[u], f[u]]$ e $[d[v], f[v]]$ sono disgiunti.

Se $d[u] > d[v]$, si ripete scambiando il ruolo dei vertici.

Etichettatura degli archi

- D'albero T (u, v) t.c. v è bianco quando l'arco viene esplorato
 - $\hookrightarrow u$ diventa predecessore di v
- All'indietro B (u, v) t.c. v è grigio quando l'arco viene esplorato
 - $\hookrightarrow u$ non è predecessore di v
 - $\hookrightarrow v$ è antenato di u
- In avanti F (u, v) t.c. v è nero e $d[u] < d[v]$
 - $\hookrightarrow u$ non è predecessore di v
 - $\hookrightarrow u$ è antenato di v
- trasversale C (u, v) t.c. v è nero e $d[u] > d[v]$
 - $\hookrightarrow u$ non è predecessore di v
 - $\hookrightarrow u$ non è antenato di v
 - $\hookrightarrow u$ e v possono essere in due diversi alberi



Pseudocodice

DFS(G):

```

For each  $v \in V$  do
  color [ $v$ ] = W
   $\pi$  [ $v$ ] = NIL
   $d$  [ $v$ ] = 0
   $f$  [ $v$ ] = 0
time = 0
For each  $v \in V$  do
  if color [ $v$ ] is W then
    DFS_VISIT ( $G, v$ )
  
```

DFS_VISIT (G, u):

```

time = time + 1
 $d$  [ $u$ ] = time
color [ $u$ ] = G
for each  $v \in \text{adj}[u]$  do:
  if color [ $v$ ] = W then
     $\pi$  [ $v$ ] =  $u$ 
    DFS_VISIT ( $G, v$ )
color [ $u$ ] = B
time = time + 1
 $f$  [ $u$ ] = time
  
```

Complessità

L'inizializzazione costa $O(|V|)$. La visita viene chiamata al più una volta per ogni vertice v del grafo, quindi $O(|V|)$ chiamate.

Complessivamente su tutte le chiamate di visita, il costo di ispezione delle liste di adiacenza è $O(|E|)$.

Nel caso peggiore la complessità è $O(|V| + |E|)$.

Pseudocodice con etichettatura

Modifica al codice precedente evidenziato:

```
if color[v] = W then
    (u,v) → "arco T"
    π[v] = u
    DFS_VISIT(G,v)

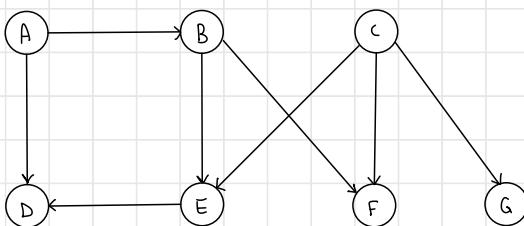
else
    if color[v] = G then
        (u,v) → "arco B"
    else
        if d[u] < d[v] then
            (u,v) → "arco F"
        else
            (u,v) → "arco C"
```

Ordinamento topologico

Si consideri un grafo $G=(V,E)$ orientato aciclico.

L'ordinamento topologico è un elenco dei vertici $T = \langle v_1, v_2, \dots, v_n \rangle$ tale che per ogni arco (v_i, v_j) si ha che v_i viene prima di v_j in T .

Eseguire una DFS nella quale l'operazione di visita consiste nell'aggiungere il nodo in testa ad una lista "at finish time".



$$T = \langle C, G, A, B, F, E, D \rangle$$

Si modifica il codice DFS come segue.

DFS(G):

```
| for each  $v \in V$  do  
|   color [ $v$ ] = W  
|    $\pi [v] = \text{NIL}$   
|    $d[v] = 0$   
|    $f[v] = 0$   
| time = 0  
| S = empty stack  
| for each  $v \in V$  do  
|   if color [ $v$ ] is W then  
|       DFS_VISIT ( $G, v$ )
```

DFS_VISIT(G, u):

```
| time = time + 1  
|  $d[u] = \text{time}$   
| color [ $u$ ] = G  
| for each  $v \in \text{adj}[u]$  do:  
|   if color [ $v$ ] = W then  
|        $\pi [v] = u$   
|       DFS_VISIT ( $G, v$ )  
| color [ $u$ ] = B  
| push (S,  $u$ )  
| time = time + 1  
|  $f[u] = \text{time}$ 
```

E si stampa con:

```
while not isEmpty(S) do:  
    print top(S)  
    pop(S)
```

DFS non orientato

Un grafo non orientato, dopo una visita in profondità, ha solo archi T (di albero) e archi B (all'indietro).

Essendo non orientato, il predecessore di u compare nella lista degli adiacenti di u e quindi non si deve considerare, soprattutto in vista della etichettatura degli archi.

Segue lo pseudocodice con la clausola di cui sopra.

$\text{DFS}(G)$:

```
for each  $v \in V$  do
    color [ $v$ ] = W
     $\pi$  [ $v$ ] = NIL
    d [ $v$ ] = 0
    f [ $v$ ] = 0
time = 0
for each  $v \in V$  do
    if color [ $v$ ] is W then
        DFS_VISIT ( $G, v$ )
```

$\text{DFS_VISIT}(G, u)$:

```
time = time + 1
d [ $u$ ] = time
color [ $u$ ] = G
for each  $v \in \text{adj}[u] \setminus \{\pi[u]\}$  do
    if color [ $v$ ] = W then
        print "arco T"
         $\pi$  [ $v$ ] =  $u$ 
        DFS_VISIT ( $G, v$ )
    else
        print "arco B"
color [ $u$ ] = B
time = time + 1
f [ $u$ ] = time
```

DIMOSTRAZIONI VARIE

Lis / Lics

TEOREMA Sia X una sequenza di m numeri interi e sia X_i un suo prefisso di lunghezza i con $1 \leq i \leq m$.

Sia Z^i una tra le più lunghe sottosequenze crescenti di X_i e che termina con x_i .

Allora vale che $Z^i = Z^*|x_i$ con $Z^* \in W_i$ e $|Z^*| = \max\{|W| : W \in W_i\}$ con W_i insieme delle sottosequenze crescenti di X_i che finiscono con x_i , a cui è possibile concatenare x_i .

DIMOSTRAZIONE Per assurdo.

Si supponga che $Z^*|x_i$ non sia soluzione del problema i -esimo.

Allora, relativamente alla soluzione Z' :

$$1) Z^i = Z'|x_i$$

$$2) |Z'| > |Z^*| \text{ necessariamente}$$

Sia z' l'ultimo elemento di Z' , allora $z' < x_i$

Sia $h < i$ tale che $x_h = z'$

Per definizione di W_i , si ha che $Z' \in W_i$, infatti Z' è sottosequenza crescente di X_h con $n_h < n_i$.

Ma contraddizione con il punto 2) e l'ipotesi $|Z^*| = \max\{|W| : W \in W_i\}$

Wis / Hateville

$$\begin{aligned} \text{TEOREMA } S_i = & \begin{cases} S_{i-1} & \text{se } i \notin S_i \\ S_{p(i)} \cup \{i\} & \text{se } i \in S_i \end{cases} \quad \text{inoltre } V(S_i) = \begin{cases} V(S_{i-1}) & \text{se } i \notin S_i \\ V(S_{p(i)}) + V(i) & \text{se } i \in S_i \end{cases} \end{aligned}$$

DIMOSTRAZIONE Nel caso $i \notin S_i$, per assurdo.

Se S_{i-1} non fosse la soluzione del problema i -esimo, allora $S_i \neq S_{i-1}$.

Inoltre $V(S_i) > V(S_{i-1})$ per definizione: la soluzione massimizza i valori.

Dato che $i \notin S_i$, allora $S_i \subseteq \{1, \dots, i-1\} = X_{i-1}$, con $\text{COMP}(S_i) = T$ per definizione.

Quindi S_i è candidato soluzione di X_{i-1} , la cui soluzione è S_{i-1} .

Ma allora S_{i-1} non è soluzione di X_{i-1} , che è assurdo per definizione.

Nel caso $i \in S_i$, per assurdo.

Se $S_{p(i)} \cup \{i\}$ non fosse la soluzione del problema i -esimo, allora $S_i \neq S_{p(i)} \cup \{i\}$.

Inoltre $V(S_i) > V(S_{p(i)}) + v_i$. Allora $S_i = S' \cup \{i\}$ e $\text{COMP}(S_i) = T$, con $S' \subseteq \{1, \dots, p(i)\}$.

Quindi S' è candidato soluzione per $p(i)$.

Si può allora scrivere $V(S' \cup \{i\}) > V(S_{p(i)}) + v_i$, $V(S') + v_i > V(S_{p(i)}) + v_i$,

$V(S') > V(S_{p(i)})$ assurdo per definizione perché $S_{p(i)}$ non è sol. del problema $p(i)$.

LCS

TEOREMA Sia (i, i) un generico problema di LCS con $i > 0 \wedge i > 0$, per ogni sottoproblema più piccolo (i, \bar{i}) sia $Z^{(i, \bar{i})}$ una sua soluzione.

Sia $Z_k^{(i, i)} = \langle z_1, \dots, z_k \rangle$ una soluzione del sottoproblema (i, i) . Allora:

- 1) Se $x_i = y_i$ allora $z_k = x_i \wedge Z_{k-1}^{(i, i)} = Z^{(i-1, \bar{i}-1)}$
- 2) Se $x_i \neq y_i \wedge z_k \neq x_i$ allora $Z_k^{(i, i)} = Z^{(i-1, i)}$
- 3) Se $x_i \neq y_i \wedge z_k \neq y_i$ allora $Z_k^{(i, i)} = Z^{(i, \bar{i}-1)}$

DIMOSTRAZIONE Per punti altrimenti esce un mappassone (cit. chef Barbieri)

1) Per assurdo $x_i = y_i$, allora: $(z_k \neq x_i \vee z_k \neq y_i)^{1a} \vee (Z_{k-1}^{(i, i)} \neq Z^{(i-1, \bar{i}-1)})^{1b}$

1a) se $z_k \neq x_i$ allora $Z_k^{(i, i)}|_{x_i}$ è una LCS di x_i e y_i .

Quindi $Z_k^{(i, i)}$ non è soluzione del problema (i, i) perché se $x_i = y_i$, con $z_k \neq x_i$, aggiungendo x_i si ottiene una sottosequenza più lunga, che è assurdo.

se $z_k \neq y_i$ allora $Z_k^{(i, i)}|_{y_i}$ è una LCS di x_i e y_i .

Quindi $Z_k^{(i, i)}$ non è soluzione del problema (i, i) perché se $x_i = y_i$, con $z_k \neq y_i$, aggiungendo y_i si ottiene una sottosequenza più lunga, che è assurdo.

1b) Se $Z_{k-1}^{(i, i)} \neq Z^{(i-1, \bar{i}-1)}$ allora $Z_{k-1}^{(i, i)}$ non è soluzione del problema $(i-1, \bar{i}-1)$.

Si pone allora che W sia soluzione di $(i-1, \bar{i}-1)$.

Si ha $|W| > k-1$ e $W|_{x_i}$ è sottosequenza sia di x_i che di y_i e ha $|W|_{x_i}| > k$.

Allora $Z_k^{(i, i)}$ non è soluzione del problema (i, i) , che è assurdo.

2) Per assurdo $Z_k^{(i, i)}$ non è soluzione del sottoproblema $(i-1, \bar{i})$.

Sia allora W la soluzione a tale sottoproblema, avrà allora lunghezza $> k$.

Ma W è anche sottosequenza di x_i, y_i e di $x_{i-1}, y_{\bar{i}}$

$Z_k^{(i, i)}$ allora non può essere soluzione del problema (i, i) , che è assurdo.

3) Per assurdo $Z_k^{(i, i)}$ non è soluzione del sottoproblema $(i, \bar{i}-1)$

Sia allora W la soluzione a tale sottoproblema, avrà allora lunghezza $> k$.

Ma W è anche sottosequenza di x_i, y_i e di $x_i, y_{\bar{i}-1}$

$Z_k^{(i, i)}$ allora non può essere soluzione del problema (i, i) , che è assurdo.

Arco Sicuro

T è l'insieme di archi di un MST che non contiene l'arco (u, v) .

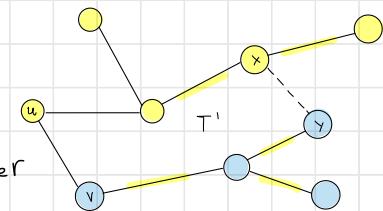
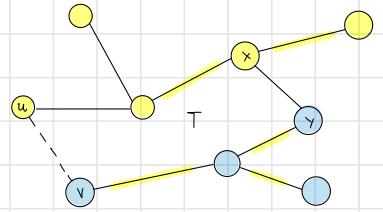
$A \subseteq T$ è un sottoinsieme di archi dell'MST ed esiste un taglio che rispetta A .

Dato che T è connesso, esiste un cammino semplice p in T che va da w a v . Ma (w, v) attraversa il taglio, quindi esiste un (x, y) di p che attraversa il taglio.

Per ipotesi, $A \subseteq T$ non contiene (u, v) e (x, y) , e per costruzione $T' = (T \setminus \{(x, y)\}) + \{(u, v)\}$, e deve essere un MST. Sicuramente si ha che $w(u, v) \leq w(x, y)$, si ha quindi che $w(T') \leq w(T)$.

Inoltre si ha che $A \subseteq T'$ sicuramente, ma anche $A \cup \{(u, v)\} \subseteq T'$

Di conseguenza, poiché T' è un MST, (u, v) è un arco sicuro per A .



Corollario arco sicuro

Il taglio $(V_C, V - V_C)$ rispetta A e (u, v) è un arco leggero per questo taglio. Quindi, (u, v) è sicuro per A .