

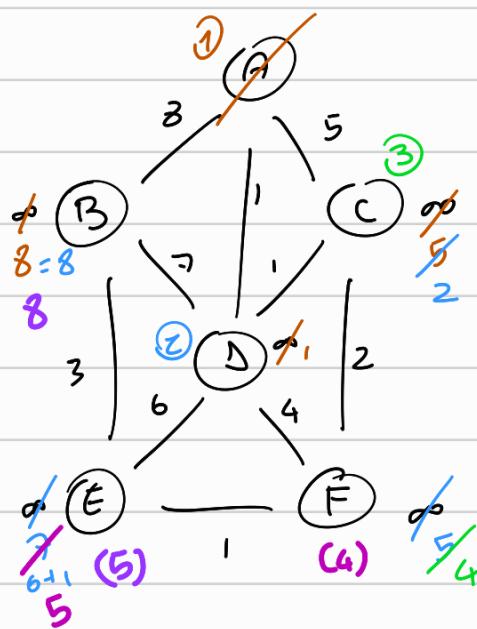
Algoritmo di Dijkstra

/ 15-03

Risolve il problema di trovare (tutti) i cammini minimi dal nodo sorgente

Dato un grafo G pesato, fissato un nodo sorgente s si vogliono trovare tutti i cammini minimi partendo da s .

Esempio



Porto da A
 distanza
 "minimo = somma pesi degli archi minima
 (in un grafo non pesato gli archi pesano 1)

Idea: distanza provvisoria



alla fine distanza minima

Il nodo sorgente ha distanza 0, gli altri all'inizio hanno ∞

Porto da A e guardo gli archi aggiornando la distanza provvisoria

Mi muovo su D e sommo i nuovi pesi con i precedenti

Mi muovo su C , poi F , poi E , poi B (senza tornare sui precedenti)

Estraggo i nodi in base dei pesi minimi.

Ricerca minima, si usa come frangia uno HEAP
 (non presente nella versione originale)

getMin	costo
add	logn

Digustra (G, s) n
 P for each (u nodo in G) $\text{dist}[u] = \text{infinito}$; // orrey distanze provvisorie $\text{elist}[s] = \emptyset$; $\text{parent}[s] = \text{null}$ // parent permette di costituire l'albero $Q = \text{heap}$ nodo
di comuni minimifor each (u nodo in G) $Q.\text{add}(u, \text{dist}[u])$ // tutti i nodi sono in Q (no nodi neri) $\hookrightarrow n \log n$ ($\frac{\text{add} \rightarrow \log n}{\text{for} \rightarrow n}$)while (Q non vuoto) $u = Q.\text{getMin}()$ // estrae nodo d : dist minimafor each (u, v) arco in G \rightarrow se lista adiac. $\mathcal{O}(m)$ come x le visiteif ($\text{dist}[u] + \text{cost}(u, v) < \text{dist}[v]$) // se v vero e' sempre
se v falso (dist già calcolata) $\text{dist}[v] = \text{dist}[u] + \text{cost}(u, v)$ $\text{parent}[v] = u$ Q. change Priority ($v, \text{dist}[v]$) // v ha priorità diminuita $\left(\begin{array}{l} Q \downarrow \rightarrow \log n = m \log n \\ \text{Xn° archi} \rightarrow m \end{array} \right)$ (più in basso nell'albero)// Post: $\text{dist}[u] = d(u)$ Il nodo u ha calcolato lunghezza minima

Nota: - uso heap e distanza infinito - non essenziale

- non serve marcire nodi: bianchi = distanza infinito

grigi = dist. ≠ infinito ma presenti ancora
nello heap

neri = esclusi dallo heap

□ Definizione:

uno heap binario è un albero binario quasi completo tale che:

- ogni nodo v contiene un elemento $\text{elem}(v)$ ed una chiave $\text{chiave}(v)$ presa da un dominio totalmente ordinato
- il valore della chiave in un nodo v è **maggiore** dei valori delle chiavi associate ai suoi due figli

□ Una definizione analoga può essere data per uno heap in cui il valore della chiave in un nodo v è **minore** dei valori delle chiavi associate ai suoi due figli

Correttezza algoritmo Dijkstra con invarianti (alg iterativo)

Dato $d(u)$ = distanza da s a u = lunghezza minima comune da s a u

^{inv}
1) per ogni u "vero" ($u \in Q$)
 $\text{dist}[u] = d(u)$

(cioè abbiamo già calcolato la distanza)

^{inv}
2) per ogni u "non vero" ($u \notin Q$)
 $\text{dist}[u] = d \setminus Q(u)$ lunghezza minima di comune da s a u
d'uno Q formato solo da nodi non veri ($u \notin Q$)

→ per i nodi in Q non abbiamo ancora calcolato la distanza ma
ne abbiamo un'approssimazione = lunghezza minima ma solo tra
comuni formati da nodi veri (prima di visitare altri minimi)

Prove di correttezza

- inv (totale) vale all'inizio :

- all'inizio tutti i nodi sono in Q quindi inv1 vale perché
non ve ne sono

- inv2 : $\text{dist}[u] = \text{lunghezza min di comune formato solo da nodi veri}$
→ non ho comuni al momento, quindi c'è infinito,
(trovare per s per cui vale zero)

- inv vuote e Q vuoto \Rightarrow Post

alla fine i nodi sono tutti veri

→ vale inv1 per tutti (dist definitive calcolate)

- terminazione: l'algoritmo termina, cio' che decrece e' lo step a ogni ciclo
(= n° nodi in Q)

- > preservare
// INV e Q vuota
corpo while
// INV

Prima di un'iterazione vogliamo sia INV1 che INV2

→ estraggo u da dist primo minima
(cioe' $\text{dist}[u] \leq \text{dist}[w] \forall w \in Q$)

ovendolo estratto, u diventa nero, devo far vedere che
 $\text{dist}[u]$ e' lunghezza di qualunque cammino da s a u
(non solo formato da nodi neri)

Sappiamo che esiste un cammino P_i da s a u con lunghezza $\text{dist}[u]$

Per assurdo: supponiamo esiste un altro cammino da s a u di
(lunghezza < P_i)

Due casi:

- questo cammino e' formato solo da nodi neri, ma allora so già che
non puo' essere piu' corto per INV1

- quest. cammino contiene un (primo) nodo non nero.

quindi e' formato come $P_{i-1} P_{i-2}$

con P_{i-1} cammino da s a u formato da soli nodi neri

P_{i-2} cammino da u a u

Ma lunghezza $P_{i+1} \geq \text{dist}[u]$ perché u era a dist. prov. minima

Quindi lunghezza $P_{i-1} \geq$ lunghezza P_i

Inoltre: lunghezza $(P_{i-1} + P_{i-2}) \geq$ lunghezza P_i

(con l'assunzione che non si hanno pesi negativi)

(\hookrightarrow ipotesi fondamentale
x Dijkstra)

\rightarrow inv1 vale ancora dopo l'aggiunta
del nodo

\rightarrow inv2 no, va ripristinato aggiornando le distanze coi nuovi pesi

Note: per provare l'inv1 necessita di inv2, in questo caso detta
inversamente assistente

Il risultato dell'algoritmo:

- distanza come numero $\rightarrow \text{dist}[u]$

- albero dei cammini minimi \rightarrow array parent[i]

Complessità:

• usando matrice di ordi ($\bullet \mathcal{O}(m)$ può arrivare a $\mathcal{O}(n^2)$) = totale

• usando liste di ordi ho in totale $\mathcal{O}(n \cdot m)$

• usando liste di soluzioni ho $\mathcal{O}(m \log n)$, in totale $\mathcal{O}((n+m) \log n)$

Se non usassi lo heap, lista non ordinata:

- edd ha cost. costante
 - get Min ha cost. lineare
-) $t_{\text{tot}} = n^2$

Con lista ordinata

- get Min cost. costante
 - edd e sopravventi lineari
-) $t_{\text{tot}} = n^2$

Note: se il grafo è deesso ($m \approx \Theta(n^2)$) lo heap non conviene

ottengo $\mathcal{O}(n^2 \log n)$