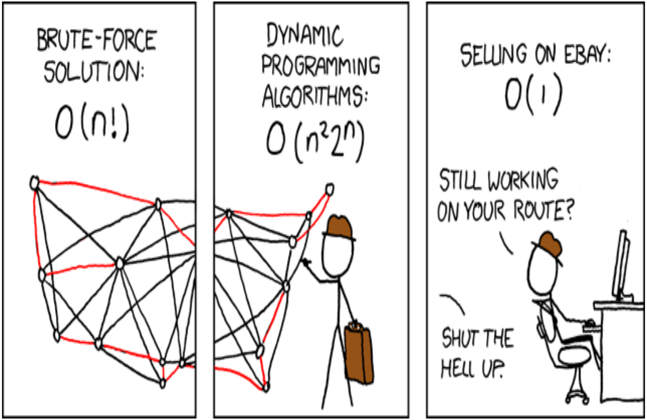
Algoritmi e strutture dati

Grafi e percorsi minimi tra tutte le coppie di vertici



Menú di questa lezione

In questa lezione affrontiamo di nuovo il concetto di percorso minimo su un grafo pesato e diretto, e cerchiamo soluzioni per il problema di trovare i percorsi minimi tra tutte le coppie di nodi in maniera simultanea.

Percorsi minimi tra tutte le coppie di vertici

In quest'ultimo blocco, affrontiamo il problema: dato un grafo $G = (V, E, W)$ pesato, diretto e connesso, vogliamo calcolare il peso del **percorso minimo per ogni copppia di vertici**. Osserviamo in primo luogo che se aggiungiamo l'ipotesi di non avere archi negativi, possiamo risolvere questo problema applicando $|V|$ volte l'algoritmo *Dijkstra*. In caso di grafi densi, questo porterebbe ad un algoritmo $\Theta(|V|^3)$, mentre in caso di grafi sparsi ad uno $\Theta(|E| \cdot |V| \cdot log(|V|))$. In caso di archi negativi, peró saremmo costretti a usare *Bellman-Ford*, per una complessitá $\Theta(|V|^4)$. Ci domandiamo se possiamo fare meglio di cosi, in questo caso generale (quindi, con possibili archi negativi ma non cicli negativi).

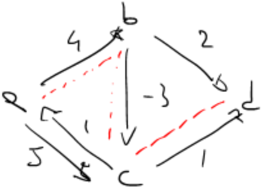
Percorsi minimi tra tutte le coppie di vertici

Il primo problema che dobbiamo affrontare è quello della rappresentazione. Fino adesso, abbiamo usato con successo una rappresentazione a **liste di adiacenza**, che oltre ad essere comoda e naturale ci ha permesso di utilizzare l'analisi aggregata nella maggioranza dei casi. Ma il problema dei percorsi minimi tra tutte le coppie è sostanzialmente diverso da quello con sorgente singla, e in un certo modo richiede in maniera naturale una rappresentazione **matriciale** del grafo. Dunque in questa parte faremo la seguente assunzione: un grafo G è rappresentato dalla sola matrice W di pesi. Cosa ci aspettiamo come risultato di un algoritmo che risolve questo problema? Poichè la rappresentazione è matriciale, ci aspettiamo due matrici come risultato: in una troveremo un'indicazione sul percorso (**matrice dei predecessori**, denotata con Π), e nell'altra, troveremo i pesi calcolati (denotata con D).

Rappresentazione matriciale di $G=(V,E,W)$

W	1	2	3	4
1	∅	4	5	Nil
2	Nil	∅	-3	2
3	1	Nil	∅	1
4	Nil	Nil	Nil	∅

A



Da

I cammini minimi si rappresentano con matrici D, Π

a → 1
b → 2
c → 3
d → 4

D

	1	2	3	4
1	0			2
2		0		
3			0	
4			∞	0

Π

	1	2	3	4
1	1		2	3
2		2		
3			3	
4			Nil	4

i)

Indice di c

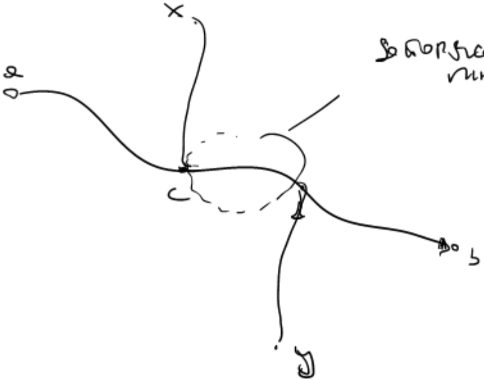
Percorsi minimi tra tutte le coppie di vertici

Senza cicli negativi, rappresentare G con la matrice dei pesi W significa assumere che per ogni coppia i, j, W contene **nil** se non c'è un arco tra i e j, 0 se $i = j$, e il peso dell'arco tra i e j altrimenti. Osserviamo che potrebbero esserci **self loops** di peso positivo: questi vengono ignorati perchè non hanno alcun ruolo nei cammini minimi. La matrice D di pesi calcolati si comporterà esattamente nello stesso modo, e la matrice Π dei predecessori, invece, avrà, per ogni coppia i, j, **nil** se $i = j$ oppure se non esiste un percorso tra i e j, e k se, per qualche percorso minimo, da i a j, k precede j su di esso.

Programmazione dinamica

Questo problema ci permette di introdurre una nuova tecnica, alternativa al divide and conquer e alla strategia greedy, chiamata **programmazione dinamica**. La programmazione dinamica, come la strategia greedy, è pensata per risolvere in maniera efficiente un problema di ottimizzazione; trovare i percorsi minimi, cosí come trovare un albero di copertura minimo, è un problema di ottimizzazione. Nel divide and conquer, l'approccio ricorsivo serve a risolvere **sottoproblemi separati tra loro**; nella programmazione dinamica serve a risolvere **sottoproblemi comuni**. In questo caso, il percorso minimo tra a e b potrebbe (o no) passare per c; ma se è cosi, io devo sapere qual è il percorso minimo tra a e c e quello tra c e b. Se il percorso minimo tra a e d passasse, anche lui, per c, quello tra a e c mi servirebbe di nuovo. Una caratteristica importante della programmazione dinamica è che tutta la complessità di progettazione si accumula nella progettazione iniziale: le dimostrazioni di correttezza si riducono ad osservare che l'algoritmo segue il modello dinamico che è stato pensato.

[illegible]



x
a
c
b
y

[illegible]

L'algoritmo della moltiplicazione di matrici

Cominceremo con un approccio di programmazione dinamica che servirá a introdurre le idee principali, basato sulla moltiplicazione di matrici. Facciamo le seguenti osservazioni: primo, è conveniente pensare ai vertici come numerati da 1 a $|V|$ (e quindi li indicheremo con indici $i, j, \ldots$); secondo, come giá sappiamo, nessun percorso minimo tra due vertici puó avere piú di $|V| - 1$ archi (altrimenti sarebbe un ciclo). In termini di notazione, useremo le matrici cosi: invece di $W[i,j]$ usiamo W_{ij}, e lo stesso varrá per D e Π.

Sappiamo già che i percorsi minimi tra due vertici sono caratterizzati da una sottostruttura ottima. Troveremo che questa è proprio una condizione essenziale per l'applicazione della programmazione dinamica. In quale modo possiamo vedere questa sottostruttura perchè sia utile adesso? Per esempio, così: definiamo l_{ij}^{m} come **il peso del cammino minimo tra il vertice i e il vertice j che si puó costruire usando al piú m archi**. Chiaramente, $l_{ij}^{1} = W_{ij}$ per ogni coppia di vertici i e j, ed il problema che affrontiamo consiste nel capire come trovare l^{m} a partire da l^{m-1}.

L'algoritmo della moltiplicazione di matrici

Minimo tra tutte le deviazioni possibili

Non è difficile convincersi che:

$$l_{ij}^m = min\{l_{ij}^{m-1}, min_k\{l_{ik}^{m-1} + W_{kj}\}\}$$

Infatti, se **mi permetto** di usare un arco in piú, devo stabilire se mi conviene: questo accadrá se, e solo se, tra tutti i vertici, ne esiste uno tale che è migliore di quello attuale come predecessore di j. Siccome $W_{jj} = 0$ sempre, possiamo semplificare in:

$$l_{ij}^m = min_k\{l_{ik}^{m-1} + W_{kj}\}$$

L'algoritmo della moltiplicazione di matrici

Questa uguaglianza si puó applicare in maniera iterativa finchè si ottiene il percorso minimo. Quando dovremmo fermarci? Sappiamo che in assenza di cicli negativi, ogni percorso minimo non puó contare con piú di $|V| - 1$ archi. Dunque si ha che $l_{ij}^{|V|-1} = l_{ij}^{|V|} = l_{ij}^{|V|+1} = \ldots$. Questa sará la condizione di stop. Applichiamo la formula per calcolare, dunque, una matrice L, di grado $|V|$, esattamente $|V| - 1$ volte, partendo dalla matrice iniziale che è data dal valore 0 sulla diagonale, e ∞ altrimenti. Queste matrici si chiameranno $L^1 = W, L^2, \ldots, L^{|V|}$, e l'ultima conterrá i pesi dei cammini minimi (e la chiameremo D per rispettare la notazione introdotta).

L'algoritmo della moltiplicazione di matrici: versione lenta

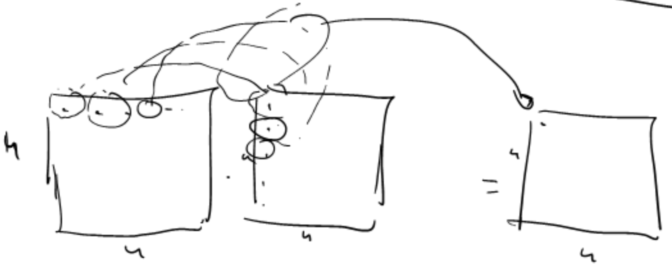
proc ***ExtendShortestPaths*** (L, W)
- $n = L.rows$
- **let** L' **be a new matrix**
- **for** $i = 1$ **to** n
  - **for** $j = 1$ **to** n
    - $L'_{ij} = \infty$
    - **for** $k = 1$ **to** n
      - $L'_{ij} = min\{L'_{ij}, L_{ik} + W_{kj}\}$
- **return** $i + 1$

proc ***SlowAllPairsMatrix*** (W)
- $n = L.rows$
- $L^1 = W$
- **for** $m = 2$ **to** $n - 1$
  - $L^m = ExtendShortestPaths(L^{m-1}, W)$
- **return** L^{n-1}

L'algoritmo della moltiplicazione di matrici: versione lenta

La **correttezza** di questo algoritmo è immediata perchè dipende dalla formula che abbiamo visto. Sebbene noi abbiamo calcolato solo la matrice dei pesi dei cammini minimi, $L^{|V|}$, è possibile modificare l'algoritmo per calcolare anche la matrice dei predecessori. La sua **complessitá** è facile da calcolare: quattro cicli di lunghezza $|V|$, per un totale di $\Theta(|V|^4)$. Come possiamo migliorare questo risultato, sapendo che la moltiplicazione di matrici è associativa?

All Pairs Shortest Paths con moltiplicazione di matrici



$$A \cdot B = C$$

$$e^{m-1} \qquad W \qquad e^{m}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

$\sum \rightarrow \min$

$\cdot \rightarrow +$

RICORDA: la moltiplicazione di matrici è **ASSOCIATIVA**

$$A(BC) = (AB)C$$

L'osservazione principale è: abbiamo bisogno unicamente di $L^{|V|}$, e non di tutte le altre. Grazie all'associatività dell'operazione di moltiplicazione, questa osservazione ci permette di usare il principio del **quadrato iterativo**. Questo ci permetterà di portare uno dei quattro cicli da lineare a logaritmico.

Nella versione SLOW facevamo così:

$$L^1$$
$$L^2 = L^1 W = WW$$
$$L^3 = L^2 W = WWW$$
$$L^4 = L^3 W = L^2 L^2$$

L'algoritmo della moltiplicazione di matrici: versione veloce

proc *ExtendShortestPaths* (L, W)

- $n = L.rows$
- **let** L' **be a new matrix**
- **for** $i = 1$ **to** n
  - **for** $j = 1$ **to** n
    - $L'_{ij} = \infty$
    - **for** $k = 1$ **to** n
      - $L'_{ij} = min\{L'_{ij}, L_{ik} + W_{kj}\}$
- **return** $i + 1$

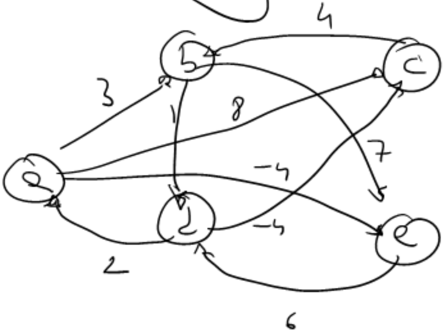
proc *FastAllPairsMatrix* (W)

- $n = L.rows$
- $L^1 = W$
- $m = 1$
- **while** $(m < n - 1)$
- **do**
  - $L^{2 \cdot m} = ExtendShortestPaths(L^m, L^m)$
  - $m = 2 \cdot m$
- **return** L^m

L'algoritmo della moltiplicazione di matrici: versione veloce

Nuovamente, la **correttezza** è ovvia. La **complessitá**, in questo caso, diventa $\Theta(|V|^3 \cdot log(|V|))$. Questo si deve al fatto che uno dei quattro cicli che c'erano precedentemente è stato convertito in una ricerca binaria: invece di esplorare tutte le matrici dalla prima alla $|V|$-esima, si procede attraverso i quadrati $(1{,}2{,}4{,}8{,}\ldots)$ fino a **superare** $|V|$; poichè tutte le matrici di indice superiore al $|V|$-esimo sono uguali, possiamo restituirne una qualsiasi.

ESERCIZIO $\boxed{145}$



$e \to d$	alt	b	:	4
"	"	c	:	∞
"	"	d	:	∞
"	"	e	:	$\boxed{2}$

$\boxed{A}$

$$L^2_{ed} = L^2_{54} \; \text{EXTEND}(L^1, W) = \text{EXTEND}(W, W) = 2$$

= peso cammino da 5 a 4 (e a d) con al massimo 2 archi

Algoritmo di *Floyd-Warshall*

Nell'algoritmo di *Floyd-Warshall*, che migliora i due risultati precedenti, utilizzeremo idee molto simili; la differenza fondamentale è la caratterizzazione della sottostruttura ottima. Robert Floyd, Bernard Roy, e Stephen Warshall, tra il 1959 e il 1962, sono accreditati per questo algoritmo.

La caratterizzazione della sottostruttura ottima in questo caso è fatta attravero i vertici invece che attraverso gli archi. Siano i e j due vertici qualsiasi, e consideriamo un percorso minimo $i \rightsquigarrow j$. Consideriamo, adesso, tutti i vertici diversi da i e da j, che compaiono su questo percorso: li chiamiamo vertici **intermediari**. In G, per un percorso minimo $i \rightsquigarrow j$ i vertici intermediari possono essere qualsiasi vertice nell insieme $\{1, \ldots, |V|\}$. Ma in generale possiamo chiederci qual è il percorso minimo tra i e j limitandoci a **pescare** i vertici intermediari in una qualunque sottoinsieme $\{1, \ldots, k\}$, con $k \leq |V|$. Usiamo questa notazione: $i \rightsquigarrow_k j$ per indicare un percorso minimo tra i e j dove tutti gli intemediari appartengono al sottoinsieme $\{1, \ldots, k\}$.

Algoritmo di *Floyd-Warshall*

Quindi, $i \rightsquigarrow_{|V|} j = i \rightsquigarrow j$. L'idea di *Floyd-Warshall* è che possiamo arrivare a $i \rightsquigarrow j$ esaminando tutti i percorsi minimi $i \rightsquigarrow_1 j$, $i \rightsquigarrow_2 j$, ... Infatti, la relazione tra $i \rightsquigarrow_{k-1} j$ e $i \rightsquigarrow_k j$ è semplice: k non è un intermediario di $i \rightsquigarrow j$, allora $i \rightsquigarrow_{k-1} j = i \rightsquigarrow_k j$; se, invece, k è un intermediario di $i \rightsquigarrow j$, allora $i \rightsquigarrow_k j$ é composto da $i \rightsquigarrow_{k-1} k$ e da $k \rightsquigarrow_{k-1} j$.

proc ***Floyd-Warshall*** (W)

$$
\left\{
\begin{array}{l}
n = W.rows \\
D^0 = W \\
\textbf{for } k = 1 \textbf{ to } n \\
\left\{
\begin{array}{l}
\textbf{let } D^k \textbf{ be a new matrix} \\
\textbf{for } i = 1 \textbf{ to } n \\
\left\{
\begin{array}{l}
\textbf{for } j = 1 \textbf{ to } n \\
\quad D^k_{ij} = min\{D^{k-1}_{ij}, D^{k-1}_{ik} + D^{k-1}_{kj}\}
\end{array}
\right.
\end{array}
\right.
\end{array}
\right.
$$

Ciò che avevo prima

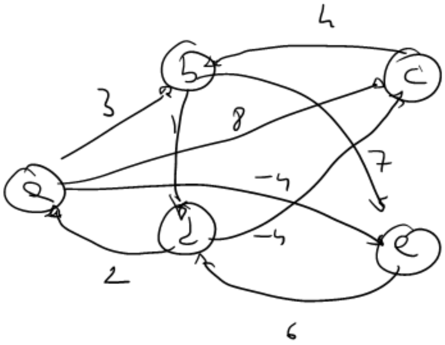
Così posso ottenere se posso anche da k

sui vertici e non sugli archi

Algoritmo di *Floyd-Warshall*

La **correttezza** di *Floyd-Warshall* si basa sulle osservazioni precedenti. La **complessità** di *Floyd-Warshall* è, chiaramente, $\Theta(|V|^3)$, e la sua terminazione è ovvia.

Esercizio 146



$D^0 = W$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-4	0	∞
5	∞	∞	∞	6	0

D^1

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2			0	
5	∞				0

. . . . . .

$\boxed{1B}$

Conclusione

La programmazione dinamica è una delle tecniche più utili in algoritmica. Le sue ramificazioni sono innumerevoli, dalla teoria dei linguaggi formali, a quella degli automi, a, appunto, il mondo dell'ottimizzazione. Questo conclude il nostro percorso negli algoritmi e strutture dati.