

Note per la Lezione 19

Ugo Vaccaro

Continuiamo con l'applicazione della tecnica Greedy, ricordando che un algoritmo Greedy costruisce una soluzione ad un dato problema *iterativamente in passi*, nel modo seguente:

- Inizia con il trovare una soluzione ad un sottoproblema di piccola taglia;
- Ad ogni passo aggiunge una nuova parte alla soluzione precedentemente computata, fino ad ottenere una soluzione al problema intero;
- La nuova parte di soluzione che viene scelta ad ogni dato passo è quella che, tra *tutte* le possibili parti che si potrebbero aggiungere, risulta essere la migliore (ad es., in un problema di ottimizzazione di massimo, potrebbe essere quella che ci dà il maggior incremento di valore alla soluzione parziale fin'ora calcolata, in un problema di minimo potrebbe essere quella che causa il minor incremento di costo).

Applichiamo la tecnica Greedy al seguente problema.

Input del problema: Supponiamo di avere un *server* (ad es., una CPU, una stampante, un distributore di benzina, o un cassiere di banca, ...) che ha n clienti da servire (che potrebbero essere, rispettivamente, dei job da eseguire, dei file da stampare, o clienti che devono fare il pieno oppure ritirare soldi allo sportello). Per soddisfare la richiesta del cliente i -esimo, il server impiegherà t_i unità di tempo (t_i = tempo di servizio del cliente i).

Output del problema: Calcolare un ordine mediante il quale servire i clienti, in modo da minimizzare il tempo medio che ogni cliente dovrà attendere per veder servita la propria richiesta.

Vediamo un esempio. Supponiamo di avere tre clienti, con tempi di servizio dati da $t_1 = 5$, $t_2 = 10$, $t_3 = 3$. Ci sono $3! = 6$ possibili modi di servire sequenzialmente i clienti, cui corrispondono differenti tempi totali T di attesa per i clienti per vedere servita la propria richiesta, e differenti tempi medi di attesa dei clienti \bar{T} , come riportati in tabella:

Ordine	T	\bar{T}
1 2 3	$5+(5+10)+(5+10+3)=38$	$38/3$
1 3 2	$5+(5+3)+(5+3+10)=31$	$31/3$
2 1 3	$10+(10+5)+(10+5+3)=43$	$43/3$
2 3 1	$10+(10+3)+(10+3+5)=41$	$41/3$
3 1 2	$3+(3+5)+(3+5+10)=29$	$29/3$
3 2 1	$3+(3+10)+(3+10+5)=34$	$34/3$

Intuizione per l'algoritmo Greedy. Innanzitutto notiamo che il tempo medio di attesa \bar{T} di ogni cliente per veder servita la propria richiesta lo si può scrivere come:

$$\bar{T} = \frac{1}{n} \sum_{i=1}^n (\text{tempo di attesa del cliente } i).$$

Dato che n è fissato, minimizzare \bar{T} equivale a minimizzare la somma

$$T = \sum_{i=1}^n (\text{tempo di attesa del cliente } i).$$

Nell'esempio precedente abbiamo anche visto che l'ordine di servizio dei clienti che minimizza T (e quindi \bar{T}) è ottenuto servendo per prima i clienti che necessitano di minor tempo per essere serviti, e poi servendo quelli che necessitano di maggior tempo.

Quest'ultimo fatto vale in generale? Immaginiamo un algoritmo Greedy che costruisce l'ordine di servizio un passo (cliente) alla volta, e supponiamo che dopo aver servito i clienti i_1, \dots, i_m , intendiamo servire il cliente j . La questione è la seguente: come scegliere j allo scopo di ottenere un ordine di servizio con minimo valore T ? Chiediamoci innanzitutto qual è l'incremento che T (= somma dei tempi di attesa dei clienti in coda) subisce in questo passo. È evidente che T passa dal valore A (=somma dei tempi di attesa dei clienti i_1, \dots, i_m) al valore B (=somma dei tempi di attesa dei clienti i_1, \dots, i_m + il tempo di attesa del cliente j) = $A + \sum_{k=1}^m t_{i_k} + t_j$.

Visto che gli algoritmi greedy non “disfano” decisioni precedentemente prese, (e quindi non possiamo cacciare via clienti già messi in coda), allo scopo di minimizzare l'incremento $t_j + \sum_{k=1}^m t_k$ che T subisce, non abbiamo altra scelta che minimizzare t_j , ovvero scegliere tra tutti i clienti non ancora in coda quello che richiede tempo di servizio minore. *Ergo*, conviene servire i clienti nell'ordine di tempo di servizio *crescente*.

Quindi un algoritmo Greedy per il problema in questione può essere il seguente:

1. Ordina i clienti in modo tale che la risultante sequenza dei tempi di servizio t_i sia *non decrescente*.
2. Servi i clienti secondo l'ordine calcolato al passo 1.

La complessità dell'algoritmo è chiaramente $\Theta(n \log n)$. Proviamo che l'algoritmo Greedy produce un ordine di servizi con \bar{T} minimo.

Supponiamo che $P = p_1 p_2 \dots p_n$ sia una generica permutazione degli interi da 1 a n , e sia $s_i = t_{p_i}$. Se i clienti vengono serviti secondo l'ordine P , allora il tempo di servizio richiesto dal cliente i -esimo è s_i , ed inoltre il tempo totale di attesa di tutti i clienti è

$$\begin{aligned} T(P) &= s_1 + (s_1 + s_2) + (s_1 + s_2 + s_3) + \dots + (s_1 + \dots + s_n) \\ &= ns_1 + (n-1)s_2 + (n-2)s_3 + \dots + 2s_{n-1} + s_n \\ &= \sum_{k=1}^n (n-k+1)s_k \end{aligned}$$

Supponiamo che P non ordini i clienti secondo l'ordine di tempo di servizio crescente. Allora, possiamo trovare due interi a e b , con $a < b$ ma $s_a > s_b$. In altre parole, il cliente a -esimo viene servito prima del cliente b -esimo anche se esso ha tempo di servizio maggiore. Sia P' un nuovo ordinamento di servizio, uguale a P tranne che per le posizioni degli p_a e p_b , che vengono scambiate. Secondo il nuovo ordine di servizio P' , adesso il cliente b -esimo viene servito *prima* del cliente a -esimo.

Il tempo totale di attesa di tutti i clienti secondo il nuovo ordinamento P' è

$$T(P') = \sum_{\substack{k=1 \\ k \neq a, b}}^n (n-k+1)s_k + (n-a+1)s_b + (n-b+1)s_a$$

Confrontiamo il vecchio valore $T(P)$ ed il nuovo $T(P')$. Avremo

$$\begin{aligned}
T(P) - T(P') &= \sum_{k=1}^n (n-k+1)s_k - \sum_{\substack{k=1 \\ k \neq a, b}}^n (n-k+1)s_k - (n-a+1)s_b - (n-b+1)s_a \\
&= (n-a+1)s_a + (n-b+1)s_b - (n-a+1)s_b - (n-b+1)s_a \\
&= (b-a)(s_a - s_b) > 0
\end{aligned}$$

Ovvero

$$T(P') < T(P).$$

Quindi...

- Possiamo migliorare *ogni* ordinamento di servizio clienti in cui qualche cliente è servito prima di qualcun altro che richiede un minor tempo di servizio.
- Chi è quindi l'ordinamento *non migliorabile* (ovvero ottimo)? Ovviamente quello che ordina i clienti in modo tale che la risultante sequenza dei tempi di servizio t_i sia *crescente* e li serve in quest'ordine.
- Abbiamo quindi provato che l'algoritmo Greedy ottiene un ordinamento di servizio che minimizza il tempo medio di attesa dei clienti

Il problema appena studiato appare anche in altri contesti. Supponiamo di avere un insieme di n file che vogliamo memorizzare su di un nastro magnetico. La lettura su nastri magnetici è *sequenziale*, ovvero per leggere un dato file occorre prima scorrere tutti i file dall'inizio del nastro, per poi leggere il file cui siamo interessati. Sia $L[1 \dots n]$ un array che lista la lunghezza di ciascun file, ovvero il file i ha lunghezza $L[i]$, per $i = 1, \dots, n$. Se i file sono stati memorizzati sul nastro nell'ordine da 1 a n , allora il tempo per leggere il file k è proporzionale a

$$t(k) = \sum_{i=1}^k L[i].$$

Tale quantità riflette il fatto che prima di poter leggere il file k occorre aver scorso tutti i file precedenti. Supponendo che tutti i file abbiano la stessa probabilità di essere letti, otteniamo che il tempo medio per leggere un arbitrario file è:

$$\frac{1}{n} \sum_{k=1}^n t(k) = \frac{1}{n} \sum_{k=1}^n \sum_{i=1}^k L[i].$$

Se cambiamo l'ordine con cui abbiamo memorizzato i file sul nastro (ovvero, non l'ordine non è più $2, \dots, n$) in generale cambierà il tempo medio di accesso ai file. Sia $\pi(i)$ l'indice del file memorizzato nella posizione i del nastro. Allora, il tempo medio sarà

$$\frac{1}{n} \sum_{k=1}^n \sum_{i=1}^k L[\pi(i)].$$

Il problema è di trovare la permutazione π degli indici che rende tale somma minima. È ovvio che problema è già stato risolto con l'algoritmo precedente.

◇

Programmazione Dinamica contro Algoritmi Greedy

Sia la tecnica Greedy che la tecnica della Programmazione Dinamica si basano criticamente sulla seguente proprietà goduta da certi problemi di ottimizzazione:

Se S è una soluzione ottima ad un problema di ottimizzazione \mathcal{P} , allora le componenti di S sono soluzioni ottime a sottoproblemi di \mathcal{P} .

Tuttavia, esistono differenze tra l'applicabilità delle due tecniche. In generale, possiamo affermare che ogni qualvolta la tecnica Greedy produce la soluzione ottima di un problema di ottimizzazione, allora anche la Programmazione Dinamica produce la soluzione ottima (benchè in generale gli algoritmi Greedy sono più efficienti per quanto riguarda il tempo impiegato) Non vale però il viceversa, ovvero esistono problemi risolvibili ottimalmente dalla Programmazione Dinamica, ma per cui la tecnica Greedy non produce la soluzione ottima.

Vedimo un esempio di tali tipi di problemi: Il problema dello Zaino 0-1

Input: Un insieme A di n oggetti $A = \{1, \dots, n\}$, di valori $v[1], \dots, v[n]$ e pesi $w[1], \dots, w[n]$, rispettivamente. Una capacità di carico totale W .

Output: Un sottoinsieme $S \subseteq \{1, \dots, n\}$, di peso totale

$$\sum_{i \in S} w[i] \leq W$$

e valore totale

$$\sum_{i \in S} v[i]$$

massimo.

Verifichiamo che, in generale, Greedy non produce soluzioni ottime per il problema dello Zaino 0-1. A tale scopo, ricordiamo che in un problema di ottimizzazione in cui occorre massimizzare qualche funzione, gli algoritmi Greedy costruiscono la soluzione per passi, scegliendo ad ogni passo un nuovo elemento da aggiungere alla soluzione parziale finora costruita, tenendo conto di due condizioni:

- che il nuovo elemento da aggiungere preservi la ammissibilità della soluzione parziale ottenuta, ovvero rispetti i vincoli del problema,
- che il nuovo elemento da aggiungere sia quello che, tra tutti gli elementi finora non ancora considerati, procuri il maggior incremento possibile al valore della funzione obiettivo.

Consideriamo la seguente istanza di input: in cui l'insieme di oggetti è $A = \{1, 2, 3\}$, di valori $v[1] = 60\$, v[2] = 100\$, v[3] = 120\$, pesi $w[1] = 10, w[2] = 20, w[3] = 30$, e la capacità di carico totale è $W = 50$.$

Pertanto, l'oggetto 1 vale 6\$ al Kg, l'oggetto 2 vale 5\$ a Kg, e l'oggetto 3 vale 4\$ al Kg. L'algoritmo Greedy ovviamente inizia con lo scegliere l'oggetto 1, poi l'oggetto 2 e poi basta in quanto si è già caricato di 30 Kg, e non può prendere l'oggetto 3 in quanto la capacità di carico totale è pari a 50 Kg. Il valore della soluzione prodotta dall'algoritmo Greedy è 160\$.

Tale soluzione è chiaramente *non ottima*, in quanto se prendessimo gli oggetti 2 e 3 rispetteremmo il vincolo sulla capacità di carico, ed avremmo una soluzione di valore globale pari a 220\$, superiore, pertanto, al valore della

soluzione ottenuta dall'algorithm Greedy. Attraverso la tecnica di Programmazione Dinamica invece possiamo invece determinare il valore della soluzione migliore possibile.

A tale scopo, per ogni $k = 1, \dots, n$ e $w = 1, \dots, W$, definiamo $c[k, w]$ come il valore di una soluzione ottima per il sottoproblema di scegliere un sottoinsieme di valore massimo in $\{1, \dots, k\}$, e capacità di carico totale w .

Per ottenere una equazione di ricorrenza per $c[k, w]$, effettuiamo la solita osservazione: per ottenere la soluzione ottima, possiamo decidere di *non prendere* oppure di *prendere* l'ultimo oggetto k .

- Se decidiamo di non prenderlo, avremo una soluzione di valore $c[k - 1, w]$.
- Se decidiamo di prenderlo, avremo una soluzione di valore $c[k - 1, w - w[k]] + v[k]$.

Pertanto

$$c[k, w] = \max\{c[k - 1, w], c[k - 1, w - w[k]] + v[k]\},$$

con le condizioni iniziali $c[1, w] = v[1]$ se $w \geq w[1]$, e $c[1, w] = 0$ se $w < w[1]$.

Esercizio: scrivere pseudocodice per il calcolo di $c[n, W]$, sia in forma iterativa che ricorsiva usando memoization.

Notiamo che la tecnica Greedy risolve invece in maniera ottima una variante del problema

Problema dello zaino frazionario: stesse assunzioni del problema dello Zaino 0-1, ma adesso di ogni oggetto i ne possiamo prendere una qualsivoglia frazione $x_i \leq 1$.

Un generico algorithm specificherà, quindi, le frazioni x_1, \dots, x_n che intende prendere di ogni oggetto sotto la condizione che $\sum_{i=1}^n w[i]x_i \leq W$. Il valore della sua soluzione sarà invece $\sum_{i=1}^n v[i]x_i$.

Supponiamo che gli oggetti siano stati ordinati in modo tale che

$$\frac{v[1]}{w[1]} \geq \frac{v[2]}{w[2]} \geq \dots \geq \frac{v[n]}{w[n]}.$$

Effettuiamo anche la naturale assunzione che $\sum_{i=1}^n w[i] > W$, altrimenti la soluzione al problema è banale: prendi tutti gli n oggetti!

Un possibile algorithm Greedy per il problema dello zaino frazionario può essere:

Greedy-Zaino_frazionario($v[1], \dots, v[n], w[1], \dots, w[n], W$)

```

1. FOR (i=1, i<n+1, i=i+1) {
2.    $x_i = 0$ 
3. }
4.  $\bar{W} = W, j = 1$  (% $\bar{W}$  è la capacità di carico residua)
5. WHILE ( $\bar{W} > 0$ ) {
6.   IF ( $w[j] \leq \bar{W}$ ) {
7.      $x_j = 1$ 
8.   } ELSE {
9.      $x_j = \bar{W}/w[j]$ 
10.  }
11.   $\bar{W} = \bar{W} - x_j w[j],$ 
12.   $j = j + 1$ 
13. }
```

L'algoritmo `Greedy-Zaino_frazionario` produce una soluzione ottima al problema dello zaino frazionario.

Prova. Poichè abbiamo assunto che $\sum_{i=1}^n w[i] > W$, alla fine dell'algoritmo varrà $\overline{W} = 0$. Pertanto esiste un indice j per cui $1 = x_1 = \dots = x_{j-1} > x_j \geq x_{j+1} = \dots = 0$. Consideriamo una generica soluzione ottima al problema, che prende frazioni y_1, \dots, y_n degli oggetti. Per tale soluzione deve per forza valere che $\sum_{i=1}^n w[i]y_i = W$ (altrimenti potremmo prendere un altro pò di qualche oggetto, incrementando il valore della soluzione). Sia k il più piccolo intero tale che $y_k < 1$, ed ℓ il più piccolo intero $k < \ell$ tale che $y_\ell > 0$ (un tale ℓ deve per forza esistere altrimenti la soluzione descritta dalle y_i è uguale a quella greedy ed allora non vi è nulla da provare). Definiamo una nuova soluzione, in tutto e per tutto uguale a quella descritta dalle y_i , tranne che per y_k il quale viene *umentato* di un valore pari a $\epsilon/w[k]$, e per y_ℓ che *decrementato* di un valore pari a $\epsilon/w[\ell]$, con $\epsilon = \min\{v[k](1 - y_k), v[\ell]y_\ell\} > 0$. É facile verificare (per esercizio) che: (1) la nuova soluzione rispetta i vincoli sulla capacità di carico; (2) il suo valore non è inferiore a quello della soluzione descritta dalle y_i .

Inoltre, nella nuova soluzione o vale che y_k è diventato pari ad 1, oppure che y_ℓ è diventato 0. Iterando il ragionamento, ad ogni passo otteniamo una soluzione di valore non inferiore a quello della soluzione ottima descritta dalle y_i , ma che assomiglia sempre di più alla soluzione dell'algoritmo greedy. Al termine del procedimento otterremo proprio la soluzione greedy, e quindi anche la prova che essa è ottima.