Un esercizio di Programmazione dinamica

Luciano Gualà

Sommario

In queste note si discute un esercizio presentato durante il corso. L'idea è quella di fornire un ulteriore esempio di applicazione di una tecnica di progettazione algoritmica – quella della *programmazione dinamica* – molto potente (ma difficile da padroneggiare). La soluzione, quindi, è preceduta da una discussione informale sul problema e da considerazioni che guidano verso la progettazione di un algoritmo risolutivo efficiente.

Esercizio Il signor Valter Bianchi, dopo aver fatto un bel gruzzoletto vendendo cristalli non proprio legali, ha deciso di diversificare la sua attività e si è messo a vendere della roba – anch'essa non proprio legale – che per comodità chiameremo stecca di cioccolata. La stecca di cioccolata può essere venduta tutta intera o può essere spezzata in segmenti più piccoli da vendere separatamente. La lunghezza della stecca di cioccolata è di L centimetri, con L intero. Si assuma che nello spezzare la stecca la lunghezza dei pezzi ottenuti (in centimetri) debba essere ancora un numero intero. Per esempio un pezzo lungo 3 centimetri può essere venduto così, spezzato in tre pezzi da 1 centimetro o in due pezzi: uno da 2 centimetri e l'altro da 1 centimetro (mentre non si possono fare due pezzi da mezzo centimetro e due centimetri e mezzo). Il guadagno che il signor Valter Bianchi riesce a fare se vende un pezzo lungo t centimetri è $G(t)$, $t = 1, 2, \ldots, L$. Progettare un algoritmo di programmazione dinamica che aiuti il signor Valter Bianchi a guadagnare il più possibile. La complessità temporale dell'algoritmo deve essere polinomiale in L.

Un esempio di istanza e soluzione è data in Figura 1.

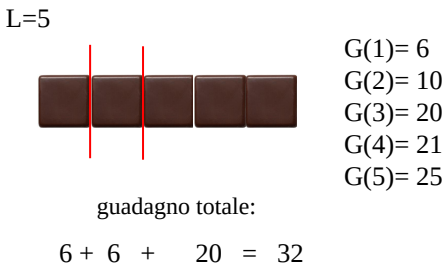

Figura 1: Un esempio di istanza con $L = 5$. La soluzione ottima, ovvero il modo di spezzare la cioccolata per ottenere guadagno massimo è mostrato in rosso.

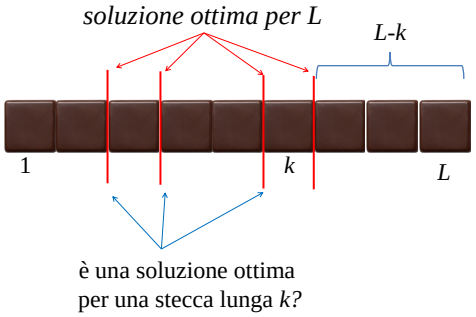

Figura 2: Struttura della soluzione ottima per L.

Ragionare sulla struttura della soluzione cercata. La chiave per progettare un algoritmo di programmazione dinamica è essenzialmente individuare il giusto insieme di sottoproblemi. Fare questo è un punto di arrivo che passa per la comprensione della struttura combinatorica del problema in esame[1]. In particolar modo, quello che bisogna davvero capire è la struttura della soluzione ottima cercata, in modo da poterla esprimere in funzione di soluzioni ottime per sottoproblemi opportuni (e più "piccoli"). Proviamo a farlo in questo caso.

Immaginiamo di avere la soluzione ottima, il modo migliore di spezzare la stecca di cioccolata di lunghezza L. Sia k l'ultima spezzata (se le spezzate le pensiamo ordinate da sinistra a destra) della soluzione ottima (si veda la Figura 2). Dato k, è chiaro che il guadagno ottenuto dalla soluzione ottima è uguale al guadagno ottenuto dall'ultimo pezzo di cioccolata (che ha lunghezza $L-k$) più il guadagno relativo ai pezzi in cui è poi ridiviso il pezzo di lunghezza k. Ora, la domanda cruciale è: il modo in cui questo pezzo di lunghezza k è ridiviso è la soluzione ottima per una stecca di cioccolata lunga k? Se ci si pensa un po' è facile convincersi che la risposta a questa domanda è sì. Infatti, se le spezzate della soluzione ottima per L a sinistra di k non fossero quelle ottime per una stecca lunga k, allora esisterebbe un modo migliore di spezzare il pezzo da k e guadagnare di più dal quel pezzo, il che implicherebbe che potrei guadagnare di più anche dal pezzo lungo L semplicemente usando questa soluzione alternativa per il pezzo lungo k e vendendo il pezzo lungo $L-k$ tutto intero. Ma questo non è possibile perché la soluzione che stiamo considerando è una soluzione ottima per un pezzo lungo L. Insomma: la solita argomentazione di tipo *cut&paste*, taglia e cuci.

Abbiamo quindi espresso la soluzione ottima in questo modo:

$$\text{guadagno ottimo per } L = G(L-k) + \text{guadagno ottimo per } k.$$

Abbiamo fatto un passo avanti nella comprensione della struttura della soluzione ottima. Restano però due punti da chiarire. Il primo è: ma come possiamo conoscere k? La risposta a questa domanda è semplice: non possiamo. Però possiamo "indovinarlo". L'ultima spezzata dell'ottimo infatti può trovarsi in $O(L)$ posizioni; ovvero k può essere uguale a $L-1$, o $L-2$,.., o 1, o 0 (nel senso che la soluzione ottima consiste nel vendere tutta intera la stecca da L). L'idea è quindi quella di provare tutte queste possibilità e prendere la migliore che, quindi, sarà la soluzione ottima. Questo è l'aspetto enumerativo della programmazione

[1] Si guardino anche le slide proposte a lezione sulla programmazione dinamica, sezione Ancora sul ruolo dei sottoproblemi - breve discussione con avvertimenti.

dinamica. Fare questo vuol dire di fatto considerare la seguente relazione:

$$\text{guadagno ottimo per } L = \max_{k=0,1,\ldots,L-1} (G(L-k) + \text{guadagno ottimo per } k)\,.$$

Il secondo punto da chiarire è: come definire i sottoproblemi? Una volta intuita la struttura della soluzione, ovvero la relazione che lega la soluzione ottima all'ottimo di altri problemi, tutto diventa più semplice. Se si analizza la relazione di sopra, è chiaro che la soluzione ottima per L è definita in funzione della soluzione ottima per k, con $k < L$. Inoltre, se si re-itera il ragionamento, la soluzione ottima per k sarà definita in funzione della soluzione ottima per k', con $k' < k$. Abbiamo quindi un sottoproblema distinto per ogni valore di $k = 0, 1, ..., L$.

Una soluzione. Siamo ora pronti per descrivere una algoritmo di programmazione dinamica per il problema. Partiamo, chiaramente, dalla definizione dei sottoproblemi.

Per ogni $j = 0, 1, \ldots, L$, definiamo

$\texttt{Opt}(j)$: il guadagno massimo ottenibile da una stecca di cioccolata lunga j.

A questo punto, dobbiamo capire se i sottoproblemi che abbiamo definito vanno bene, ovvero se tutto si incastra in modo da ottenere un algoritmo di programmazione dinamica. Essenzialmente, è necessario rispondere alle seguenti domande:

1. I sottoproblemi sono pochi? Visto che dovremo risolverli tutti, e per ogni sottoproblema dovremo spendere almeno tempo contante, il numero di sottoproblemi fornisce una delimitazione inferiore al tempo speso dall'algoritmo. Se questo numero è troppo grande (per esempio il numero di problemi è confrontabile con la complessità dell'algoritmo banale che risolve il problema per enumerazione) i sottoproblemi individuati non vanno bene. In questo caso, comunque, il numero di sottoproblemi è $O(L)$, quindi lineari nella dimension dell'istanza.

2. Una volta risolti tutti i sottoproblemi ho risolto il mio problema? La soluzione del problema deve essere infatti esplicitamente o implicitamente derivabile dalle soluzioni dei sottoproblemi. In questo caso, chiaramente, questo è vero: la soluzione cercata è $\texttt{Opt}(L)$.

3. E' possibile definire la soluzione del sottoproblema generico in funzione di sottoproblemi più piccoli? Questo è di fatto il banco di prova, il momento in cui si capisce se si sono individuati i giusti sottoproblemi. Se si è capita la struttura della soluzione ottima, comunque, la maggior parte del lavoro per rispondere a questa domanda è stato già fatto. Infatti, tutto quello che abbiamo detto per la soluzione ottima del nostro problema vale per ogni sottoproblema j-esimo. Quindi possiamo dire che:

$$\texttt{Opt}(j) = \max_{k=0,1,\ldots,j-1} (G(j-k) + \texttt{Opt}(k)).$$

4. In che ordine è possibile guardare/risolvere i sottoproblemi? Quali sono i casi base? Generalmente questo punto è facile da risolvere. La domanda soggiacente è: come definire i casi base e in che ordine risolvere i sottoproblemi in modo che la formula generica del punto precedente risulti corretta e applicabile? Nel nostro caso è semplice. L'unico caso base è $\texttt{Opt}(0) = 0$. E i sottoproblemi vanno quindi considerati nell'ordine $j = 0, 1, \ldots, L$.

A questo punto, l'algoritmo di programmazione dinamica è semplicissimo. Lo pseudocodice è riportato per completezza. La complessità dell'algoritmo è $O(L^2)$: ogni iterazione del ciclo `for` corrisponde ad uno degli $O(L)$ sottoproblemi, ed ogni sottoproblema può essere risolto in tempo $O(L)$.

Algorithm 1: `Cioccolata`(G, L)

```
Opt(0)=0;
for j = 1 to L do
    Opt(j) = max_{k=0,1,...,j-1}(G(j − k) + Opt(k))
return Opt(L);
```

Un ulteriore esercizio. Queste note si concludono con il testo di un altro esercizio (svolto nella stessa lezione). Si invitano gli studenti a elaborare e formalizzare personalmente una soluzione in ogni dettaglio.

Esercizio (*il signor Marche va in vacanza*)
Il signor Marche sta pianificando i suoi n giorni di vacanza fra Roma e Firenze. Avendo a disposizione un budget limitato, vuole trovare un piano che gli faccia spendere il meno possibile. Ha raccolto i seguenti dati. Passare il giorno i-esimo a Roma gli costerebbe r_i euro, mentre a Firenze spenderebbe f_i euro. Ogni giorno può decidere se restare nella città in cui è o spostarsi nell'altra. Spostarsi di città, però, ha un costo fisso di α euro. Inoltre, dopo aver controllato i costi dei voli, si è accorto che arrivare/partire a/da Roma o Firenze è indifferente. Più formalmente, un piano è una sequenza $x = x_1 x_2 \dots x_n$, dove per ogni $i = 1, \dots, n$, $x_i \in \{R, F\}$ specifica se il signor Marche passa il giorno i-esimo a Roma (R) o Firenze (F). Il costo di un piano x è dato dalla somma dei costi giornalieri più gli eventuali spostamenti, ovvero $\sum_{i=1}^{n} c(x_i) + \sum_{i=2}^{n} s_i(x)$, dove $c(x_i) = r_i$ se $x_i = R$, altrimenti (se $x_i = F$) $c(x_i) = f_i$, mentre $s_i(x) = \alpha$ se $x_{i-1} \neq x_i$, 0 altrimenti.

Progettare un algoritmo di programmazione dinamica che calcoli il costo di una vacanza di costo minimo.