La programmazione dinamica

Sebastiano Vigna

21 marzo 2006

1 Introduzione

La programmazione dinamica è una tecnica algoritmica che consente di affrontare problemi apparentemente intrattabili (cioè dotati di un'apparente complessità esponenziale).

Per capire il meccanismo dietro alla programmazione dinamica, possiamo metterla al confronto con il *divide et impera*: in quest'ultimo caso, un problema viene spezzato in due o più sottoproblemi, e la soluzione del problema originale viene costruita a partire dalle soluzioni dei sottoproblemi (ottenute, in genere, in modo ricorsivo). Un classico esempio di *divide et impera* è il Quicksort: gli elementi della lista da ordinare vengono divisi in due blocchi, quelli più piccoli e quelli più grandi di un pivot, e l'algoritmo viene chiamato ricorsivamente sui due blocchi.

Ci sono però casi in cui il *divide et impera* non è applicabile perché *non sappiamo come ottenere i sottoproblemi*: detto altrimenti, il problema non contiene abbastanza struttura da permettere di decidere come spezzarlo in più parti.

In questo caso entra in gioco la programmazione dinamica: si procede a calcolare le soluzioni di *tutti* i sottoproblemi possibili, e a partire da sottosoluzioni si ricavano nuove sottosoluzioni, fino a risolvere il problema originale.

È difficile, se non impossibile, dare una definizione più specifica della programmazione dinamica. L'aspetto più complesso nella costruzione di un algoritmo di programmazione dinamica è l'analisi della struttura interna del problema, e la sua riduzione a sottoproblemi. Questa fase è molto delicata, perché una frantumazione eccessiva può condurre a un numero di sottoproblemi troppo grande, ma d'altra parte senza un numero sufficiente di sottosoluzioni non sarà possibile ricostruire quella desiderata.

Una volta fatta questa analisi, viene costruita una tabella che contiene in ogni suo elemento una sottosoluzione (o i dati sufficienti a ricostruirla). Una volta che la tabella è riempita, è facile desumere la soluzione del problema originale.

2 Un primo esempio

Un primo esempio (il più semplice possibile, in effetti) che dovrebbe risultare chiarificante è il problema dello *zaino*[1]. Abbiamo uno zaino che regge un peso P, e un insieme T di tipi di oggetti. A ogni tipo $t \in T$ corrispondono un peso $p(t)$ e un valore $v(t)$, entrambi interi positivi. Vogliamo riempire lo zaino non superando P con il peso complessivo degli oggetti, ma allo stesso tempo massimizzando la somma dei valori degli oggetti nello zaino. Per ogni tipo è disponibile una fornitura illimitata di oggetti.

È chiaro che in linea di principio potremmo provare a enumerare tutti gli insiemi di oggetti che stanno nello zaino, e cercare quello con valore massimo. Se però i tipi sono molti, e la capacità dello zaino grande, il numero di soluzioni da esaminare diventa improponibile.

Ci accorgiamo però della seguente relazione tra una soluzione del nostro problema, e la soluzione dello stesso problema per uno zaino più piccolo (cioè meno resistente): se ho una soluzione ottima (cioè il cui valore è massimo tra tutte le soluzioni possibili) per uno zaino che regge P, e tolgo dalla soluzione un oggetto qualsiasi di tipo t, ottengo una soluzione ottima per uno zaino che regge $P - p(t)$. Infatti, se per

[1] In realtà, questa è una versione semplificata dall'assunzione che la fornitura di oggetti sia illimitata.

assurdo esistesse una soluzione migliore con peso inferiore a $P - p(t)$, potrei aggiungerle un oggetto di tipo t e ottenere così una soluzione migliore per il problema originale (il che è impossibile, avendo assunto che la soluzione fosse ottima).

Questo significa che se conosciamo la soluzione ottima per uno zaino di peso Q, possiamo ottenere nuove soluzioni per zaini di grandezza superiore aggiungendo un oggetto di tipo t, per ogni tipo t in T. In particolare, se ho una soluzione di valore V per uno zaino di peso Q, allora so che esiste una soluzione di valore $V + v(t)$ per uno zaino di peso $P + p(t)$, per ogni t in T. *Tutte le soluzioni ottime si ottengono in questo modo.*

Possiamo quindi risolvere il nostro problema come segue: consideriamo un vettore s che contiene nella posizione i il valore di una soluzione ottima per uno zaino di peso i, con $0 \leq i \leq P$. Chiaramente all'inizio $s_0 = 0$ (la soluzione ottima per uno zaino che non può contenere niente). Adesso esaminiamo gli elementi di s a partire da 0. Se stiamo esaminando l'elemento di indice i, possiamo assumere che s_i contenga il valore della soluzione ottima. A questo punto possiamo costruire nuove soluzioni provando ad aggiungere un oggetto di ogni tipo alla soluzione corrente (tanto gli oggetti sono disponibili in quantità arbitraria), e per ogni nuova sottosoluzione andiamo a registrare il risultato nel vettore s (chiaramente solo se migliora i valori trovati sinora).

Per esempio, supponiamo di avere tre tipi di oggetti, con peso e valore dati nella figura 1. Lo zaino dato

Tipo	Peso	Valore
1	5	2
2	3	3
3	7	8

Figura 1: Tipi di oggetti con relativi pesi e valori.

ha $P = 20$. All'inizio, il vettore s is presenta come segue:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Al primo passo, esaminiamo la soluzione di posto 0, e aggiungiamo un oggetto dei tre tipi possibili. Le nuove soluzioni che otteniamo vanno sostituite a quelle vecchie solo se sono migliori (in questo caso succede sempre):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	3	0	2	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0

Notate che abbiamo anche *propagato* il valore ottimo dalla posizione 0 alla posizione 1, dato che se possiamo ottenere un certo valore con uno zaino che regge i lo possiamo ottenere anche con uno zaino che regge $i + 1$.

Passiamo ora a esaminare la soluzione di posto 1; arriviamo così a

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	3	3	2	2	8	8	0	0	0	0	0	0	0	0	0	0	0	0

e quindi, esaminando la soluzione di posto 2, a

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	3	3	3	2	8	8	8	0	0	0	0	0	0	0	0	0	0	0

L'ultimo passaggio presenta alcuni fenomeni degni di nota: innanzitutto, la soluzione ottima per il peso 2 *non* è stata propagata sul peso 3, dato che ve ne era già una migliore. In secondo luogo, abbiamo sostituito la soluzione ottima per uno zaino di capacità 5, che è così passata da 2 a 3.

Le cose ora cominciano a farsi interessanti. Esaminando la posizione di posto 3 andiamo infatti a costruire soluzioni con più di un oggetto, ottenendo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	3	3	3	6	8	8	8	11	0	0	0	0	0	0	0	0	0	0

Per esempio, ci siamo accorti che è meglio mettere due elementi con peso 3 che non uno di peso 5 in uno zaino da 6.

È facile a questo punto completare la tabella: basta continuare la scansione fino a riempire l'elemento di posto 20.

2.1 Considerazioni sul tempo di esecuzione

Perché la soluzione proposta dovrebbe essere meglio di una ricerca esaustiva? In effetti, in generale non è meglio di una soluzione esaustiva, ma può esserlo *se P e $|T|$ non sono troppo grandi*. Infatti, il numero di aggiornamenti effettuato sulla tabella è proprio dell'ordine di $P|T|$.

Queste considerazioni vanno fatte in generale quando si cerca di affrontare un problema tramite programmazione dinamica: bisogna accertarsi che il numero di sottoproblemi che si vuole risolvere non sia troppo grande e che, comunque, il tempo necessario alla loro soluzione non sia troppo lungo.

3 Complichiamoci la vita

L'esempio proposto è particolarmente semplice perché è possibile ridurre il problema modificando *un solo parametro* (il peso trasportabile dallo zaino). Andiamo a porre il problema in maniera leggermente diversa, e vediamo come, a prezzo di una sofisticazione maggiore, la programmazione dinamica permetta ancora di uscirne illesi.

Invece di avere un insieme di tipi di oggetti disponibili in quantità illimitata, abbiamo ora un insieme *finito* di oggetti E, ciascuno dotato di peso e valore. Dobbiamo nuovamente riempire lo zaino nel miglior modo possibile.

Non possiamo però più utilizzare la riduzione precedente: infatti, se tolgo un oggetto e da una soluzione ottima per uno zaino che porta P, *non è detto che quanto rimane sia una soluzione ottima per uno zaino che porta $P - p(e)$*. Infatti, utilizzando e potrei ottenere una soluzione migliore per quel peso, e a questo punto non potrei aggiungerlo nuovamente: la dimostrazione per assurdo non funziona più.

Dobbiamo quindi trovare un modo più furbo di ridurre il problema e, in questo caso, non c'è altro modo di farlo se non *riducendolo rispetto a* due *parametri*. Consideriamo un ordine arbitrario degli m oggetti che formano E; abbiamo quindi una lista $e_1, e_2, \ldots, e_m$. Vogliamo trovare il valore di una soluzione ottima per tutti gli zaini di peso inferiore a P e *per tutti gli insiemi di oggetti $e_1, e_2, \ldots, e_j$ con $j \leq m$*.

Supponiamo di avere infatti una soluzione ottima di valore V per uno zaino di peso P che utilizza gli oggetti $e_1, e_2, \ldots, e_j$. Ci sono due possibilità: o la lista utilizza l'ultimo oggetto e_j, oppure no. Nel primo caso so di avere una soluzione ottima di valore $V - v(e_j)$ per uno zaino di peso $P - p(e_j)$ usando la lista di oggetti $e_1, e_2, \ldots, e_{j-1}$; nel secondo, so di avere una soluzione ottima per uno zaino di peso P con la lista di oggetti $e_1, e_2, \ldots, e_{j-1}$. In ogni caso, uno dei parametri che definisce il sottoproblema viene ridotto.

È chiaro che a questo punto un vettore non sarà sufficiente: avremo bisogno di una matrice $s_{i,j}$ che contiene nella posizione di indici i e j il valore della sottosoluzione ottima per uno zaino di peso i utilizzando i primi j oggetti di E (chiaramente, $0 \leq i \leq P$ e $0 \leq j \leq m$). Quando avremo riempito tutta la tabella, la posizione $s_{P,m}$ conterrà la soluzione ottima del problema originale.

Chiaramente, la prima riga e la prima colonna sono riempite con zeri: se lo zaino non può contenere nulla, o non ci sono oggetti disponibili, il valore ottimo è zero. A questo punto esaminiamo colonna per colonna la matrice, facendo crescere l'indice j. Per ogni riga i, se troviamo una soluzione di valore v propaghiamo la soluzione nella colonna successiva sulla stessa riga con valore v (questo corrisponde a non utilizzare l'elemento j), e inoltre registriamo l'esistenza di una soluzione di valore $v + v(e_j)$ sulla riga $i + p(e_j)$ (purché, naturalmente, $i + p(e_j) \leq P$).

In figura 2 vengono elencati cinque oggetti con relativo peso e valore[2]. Vogliamo riempire in maniera ottima uno zaino con $P = 10$. Cominciamo con la nostra matrice inizializzata come descritto:

[2] Attenzione: non si tratta più di *tipi* di oggetti!

Oggetto	Peso	Valore
1	3	3
2	2	4
3	5	4
4	2	1
5	6	7

Figura 2: Oggetti con relativi pesi e valori.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0

Esaminiamo ora la colonna 0, propagando le soluzioni nella colonna 1 e aggiungendo quelle risultanti dall'aggiunta dell'elemento 1:

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	3	0	0	0	0
4	0	3	0	0	0	0
5	0	3	0	0	0	0
6	0	3	0	0	0	0
7	0	3	0	0	0	0
8	0	3	0	0	0	0
9	0	3	0	0	0	0
10	0	3	0	0	0	0

Passiamo ora all'elemento 2:

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	4	0	0	0
3	0	3	4	0	0	0
4	0	3	4	0	0	0
5	0	3	7	0	0	0
6	0	3	7	0	0	0
7	0	3	7	0	0	0
8	0	3	7	0	0	0
9	0	3	7	0	0	0
10	0	3	7	0	0	0

Come potete vedere, nelle soluzioni con peso inferiore a 5 risulta più conveniente utilizzare il secondo elemento (tra i primi due). Continuiamo:

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	4	4	0	0
3	0	3	4	4	0	0
4	0	3	4	4	0	0
5	0	3	7	7	0	0
6	0	3	7	7	0	0
7	0	3	7	8	0	0
8	0	3	7	8	0	0
9	0	3	7	8	0	0
10	0	3	7	11	0	0

Manca ancora il penultimo oggetto...

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	4	4	4	0
3	0	3	4	4	4	0
4	0	3	4	4	5	0
5	0	3	7	7	7	0
6	0	3	7	7	7	0
7	0	3	7	8	8	0
8	0	3	7	8	8	0
9	0	3	7	8	9	0
10	0	3	7	11	11	0

...e infine l'ultimo:

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	4	4	4	4
3	0	3	4	4	4	4
4	0	3	4	4	5	5
5	0	3	7	7	7	7
6	0	3	7	7	7	7
7	0	3	7	8	8	8
8	0	3	7	8	8	11
9	0	3	7	8	9	11
10	0	3	7	11	11	12

3.1 Considerazioni sul tempo di esecuzione

Questa soluzione richiede una discreta quantità di memoria (ordine di Pm interi), e ordine di Pm operazioni. Di nuovo, purché P e m non siano troppo grandi, il tempo richiesto è enormemente inferiore a tentare i 2^m sottoinsiemi di oggetti possibili.

In realtà, guardando bene l'algoritmo è chiaro che per riempire la colonna di indice j ci occorrono solo dati contenuti nella colonna di indice $j-1$: possiamo quindi implementare l'algoritmo utilizzando *una sola colonna*, dato che aggiorniamo sempre dati su righe di indice superiore a quella esaminata.

4 Ricostruire le soluzioni

Fin qui tutto bene: ma cosa accadrebbe se il problema posto ci richiedesse di *emettere la soluzione ottima*, anziché semplicemente calcolarne il valore?

In generale non è detto che trovare il valore di una soluzione ottima sia facile quanto trovare effettivamente la soluzione. Spesso è però possibile tenere traccia incrementalmente (cioè ogni volta che si risolve un nuovo sottoproblema) di un frammento di informazione che permette poi di ricostruire la soluzione. A volte la soluzione è addirittura desumibile dalla tabella dei valori ottimi dei sottoproblemi.

Questo è, ad esempio, il caso di zaino: se infatti esaminiamo una riga della tabella da destra a sinistra, troveremo una variazione in valore in presenza di inserimenti di nuovi oggetti. Attenzione però: quando non è presente nessuna variazione questo non significa che l'oggetto non venga incluso, perché potrebbe venire incluso in una sottosoluzione da cui poi viene costruita quella corrente.

Quindi, per ricostruire correttamente la soluzione ottima occorre inizialmente esaminare da destra a sinistra l'ultima riga. Troviamo la prima variazione tra la colonna 4 e la colonna 5, quindi l'oggetto 5 fa parte della soluzione, che viene a questo punto ridotta alla soluzione del problema che utilizza i primi quattro oggetti e peso $10 - p(5) = 4$. Continuiamo quindi sulla riga 4, dove troviamo immediatamente un'altra variazione; quindi, anche l'oggetto 4 viene utilizzato, e la sottosoluzione da esaminare è ora quella per il peso $4 - p(4) = 2$ che utilizza i primi tre oggetti. Vediamo che non c'è variazione tra la colonna 2 e la colonna 3, quindi l'oggetto 3 non viene usato: continuiamo sulla stessa riga, e scopriamo una variazione tra la colonna 1 e la colonna 2, quindi l'oggetto 2 è stato utilizzato. Concludiamo sulla riga $2 - p(2) = 0$, dove troviamo uno zero. La soluzione ottima utilizza quindi gli oggetti 2, 4 e 5.

In altri casi una quantità più o meno grande di informazione deve essere mantenuta a parte; è il caso, ad esempio, del problema "Il negozio di fiori", delle IOI 1999.

5 Palindrome

Armati della conoscenza acquisita, proviamo a risolvere il seguente problema, proposto alle IOI 2000: data una stringa di al più 5000 caratteri, qual è il numero minimo di caratteri che occorre inserire per renderla palindroma?[3]

Denotiamo con $f(s)$ il minimo numero di caratteri che rende s palindroma, e notiamo le seguenti proprietà:

1. $f(\mathtt{a}s\mathtt{a}) = f(s)$
2. $f(\mathtt{a}s\mathtt{b}) = \min\{\, f(\mathtt{a}s) + 1,\, f(s\mathtt{b}) + 1 \,\}$

Vale a dire: se i due caratteri all'estremità della stringa sono identici, possiamo scartarli e semplicemente cercare di rendere palindroma in modo ottima la stringa rimanente. Se invece i caratteri all'estremità della stringa sono diversi, possiamo cercare di rendere palindrome le stringhe risultanti dalla cancellazione di una delle estremità, aggiungere 1 al numero di caratteri necessari (dal momento che dobbiamo reinserire il carattere cancellato) e prendere il risultato migliore. Questa proprietà ha una dimostrazione decisamente non banale, ma è ragionevole pensare che possa funzionare.

Ora, tutto questo puzza di programmazione dinamica: infatti, per calcolare $f(s)$ basta conoscere $f(s')$ per qualunque sottostringa[4] s' di s di lunghezza $l-1$ o $l-2$, dove l è la lunghezza di s. Quindi un modo di affrontare il problema è quello di calcolare f su *tutte* le sottostringhe di s, definite a partire dalla loro posizione iniziale i e dalla loro lunghezza j (chiaramente non tutte le coppie di indici i e j sono valide).

Apparentemente questa scelta ci pone di fronte a un problema insormontabile: la matrice risultante sarebbe composta da 5000×5000 interi. Ciononostante, guardando più da vicino si può notare che per riempire la riga j ci occorrono solo i dati presenti sulle righe $j-1$ e $j-2$. Quindi, se anche immaginiamo l'algoritmo su una matrice 5000×5000, potremo poi implementarlo con soli tre vettori da 5000 elementi utilizzati in maniera circolare.

A questo punto, il codice che risolve il problema, per quanto surrealmente corto, è (modulo inizializzazione, lettura dei file ecc.):

```
for(j=2; j<=N; j++)
   for(i=0; i<=N-j; i++) {
      if (parola[i] == parola[i+j-1]) ott[(j+2)%3][i] = ott[j%3][i+1];
      else ott[(j+2)%3][i] = 1+min(ott[(j+1)%3][i], ott[(j+1)%3][i+1]);
   }
```

Qui `N` è la lunghezza della stringa in input, contenuta in `parola`, e la matrice `ott` contiene tre righe da 5000 elementi utilizzate circolarmente. Notate che le prime due righe della matrice sono inizializzate automaticamente, dato che una parola di lunghezza inferiore a due è sempre palindroma.

6 Un esempio particolarmente difficile

Descriviamo ora i meccanismi di programmazione dinamica alla base della soluzione di "Una striscia di terra", proposto alle IOI 1999. In questo caso la soluzione del problema è veramente diabolica, perché la parte dinamica non consiste nella costruzione delle sottosoluzioni ottime, bensì nella costruzione di una matrice di informazioni da cui si ottiene per esame esaustivo l'insieme delle sottosoluzioni ottime.

[3] Una stringa è *palindroma* se è uguale alla sua trasposta, cioè se è la stessa se letta da sinistra e destra o da destra a sinistra: per esempio, "alla" e "ara" sono palindrome.

[4] Con "sottostringa" intendiamo qui una sottosequenza di caratteri consecutivi in s.

Il problema è presto spiegato: avete a disposizione una striscia di terra di dimensioni massime 700×700, e vi viene fornita un'altezza per ogni coppia di coordinate. Dovete individuare un sottorettangolo di ampiezza al più 100 e di area massima in cui la variazione tra la massima e la minima altezza è minore o uguale a un parametro C, anch'esso dato in input. C è compreso tra 0 e 10.

Il fatto che il valore di C sia così fortemente limitato fa subito pensare alla programmazione dinamica, ma d'altra parte la limitazione arbitraria sull'ampiezza della striscia suona molto strana. In effetti, un approccio standard che cerca di definire sottoproblemi seguendo parametri naturali (per esempio, la lunghezza o la larghezza della sottosoluzione) è destinato a fallire. Il problema fondamentale è che non si vede come mettere insieme due sottosoluzioni, dato che le loro variazioni in altezza, sebbene limitate, una volta combinate potrebbero eccedere il limite richiesto.

Quindi, utilizziamo un approccio complemamente diverso: esaminiamo l'input riga per riga e manteniamo in maniera dinamica una matrice $M_{i,d}$ che dice, per ogni colonna i, qual è il numero massimo di righe di cui si può salire mantenendo la variazione in altezza con l'elemento della riga corrente nell'intervallo $[0-d, C-d]$, con $d = 0, 1, \ldots, C$.

La matrice è gestibile in maniera dinamica perché a fronte di una nuova riga è sufficiente fare scorrere la lista di $C+1$ valori di M corrispondenti a una colonna di un numero di posti uguale alla differenza tra l'altezza corrente e quella precedente, e riempire il resto con uni (lo scorrimento è verso l'alto o verso il basso a seconda del segno della differenza). Inserire un uno è sempre possibile, dato che significa che possiamo utilizzare solo l'elemento sulla riga corrente.

Una volta che la matrice è riempita, per ogni larghezza $l \leq 100$ è possibile stabilire la massima altezza di un rettangolo che soddisfa i vincoli e sta sopra alla riga corrente. Fissiamo una colonna i e un valore di d, e supponiamo che il vettore a contenga le altezze della riga corrente. Sappiamo di quanto possiamo al più salire nella colonna i mantenendo la variazione in $[0-d, C-d]$ andando a leggere $M_{i,d}$. Poi ci spostiamo su $i-1$, andiamo a vedere la massima ascesa possibile leggendo $M_{i,d+a_{i-1}-a_i}$ (sempre che non debordiamo da M) e la minizziamo con quella corrente, e così via. Si noti che la condizione imposta implica che anche gli elementi nella colonna $i-1$ saranno corretti, perché le loro altezze saranno nell'intervallo

$$[a_{i-1} - (d + a_{i-1} - a_i), a_{i-1} + C - (d + a_{i-1} - a_i)] = [a_i - d, a_i + C - d]$$

esattamente come quelli nella colonna i. Ogni volta massimizziamo l'area del rettangolo ottenuto con quella corrente.

Se l'indice $d + a_{i-1} - a_i$ deborda da M significa che l'altezza dell'elemento nella colonna $i-1$ della riga corrente è già al di fuori dell'intervallo possibile di variazione, e quindi non è possibile estendere ulteriormente il rettangolo.

Ricapitolando: costruiamo per ogni riga le sottosoluzioni ottime che stanno sopra alla riga stessa. Per costruire incrementalmente le soluzioni, però, aggiorniamo in maniera dinamica una matrice che contiene informazioni sulla variazione in altezza di ogni colonna. A partire da questa matrice, le sottosoluzioni sono costruite con un esame esaustivo di tutte le possibilità compatibili con la matrice stessa.