Programmazione dinamica (VIII parte)

Progettazione di Algoritmi a.a. 2023-24

Matricole congrue a 1

Docente: Annalisa De Bonis

Parentesizzazione di valore massimo

- Si scriva un algoritmo che trova il valore massimo ottenibile con una parentesizzazione completa della seguente espressione: $x_1/x_2/\ldots/x_{n-1}/x_n$
- Una parentesizzazione completa di $x_1/x_2/\ldots/x_{n-1}/x_n$ si ottiene racchiudendo ciascun '/' insieme alle due sottoespressioni a cui esso si applica tra una coppia di parentesi. La coppia di parentesi piu` esterna puo` essere omessa.
- Ad esempio: le parentesizzazioni complete di 24/6/2 sono

  I. (24/6)/2=2,   II. 24/(6/2)=8. La II produce il valore massimo
- Un approccio potrebbe essere quello di considerare tutti i possibili modi di parentesizzare l'espressione e di calcolare il valore dell'espressione risultante.
  - Questo approccio e` inefficiente perche` il numero di parentesizzazioni complete e` esponenziale.

Parentesizzazione di valore massimo

- Il costo $C(P)$ di una parentesizzazione P e` il valore dell'espressione quando le divisioni sono eseguite nell'ordine dettato dalle parentesi nella parentesizzazione.
- Ad esempio: 24/6/2

  I. (24/6)/2=2, II. 24/(6/2)=8

  2 e` il costo della I parentesizzazione; 8 e` il costo della II parentesizzazione

Parentesizzazione di valore massimo

- Soluzione basata sulla programmazione dinamica.
- Sia i<j e sia P(i,…,j) una parentesizzazione della sottoespressione $x_i/x_{i+1}/\dots/x_j$. Supponiamo che questa parentesizzazione a livello piu` esterno sia formata da una certa parentesizzazione P(i,…,k) di $x_i/x_{i+1}/\dots/x_k$ e una certa parentesizzazione P(k+1,…,j) di $x_{k+1}/x_{k+2}/\dots/x_j$, per un certo $i \le k \le j-1$
- Il costo $C(P(i,\dots,j))$ di P(i,…,j) e` quindi $C(P(i,\dots,k)) / C(P(k+1,\dots,j))$
- Esempio: la parentesizzazione ((100/5)/20)/(15/3) contiene a livello piu` esterno le parentesizzazioni ((100/5)/20) e (15/3) mentre (100/5)/(20/(15/3)) contiene a livello piu` esterno le parentesizzazioni (100/5) e (20/(15/3)).

Parentesizzazione di valore massimo

- $MAX(i, j)$ = costo massimo di una parentesizzazione per la sottoespressione $x_i/x_{i+1}/\ldots/x_j$
- $min(i,j)$ = costo minimo di una parentesizzazione per la sottoespressione $x_i/x_{i+1}/\ldots/x_j$
- Sia $P'(i,\ldots,j)$ la parentesizzazione di $x_i/x_{i+1}/\ldots/x_j$ di costo massimo
- A livello piu\` esterno, $P'(i,\ldots,j)$ contiene una certa parentesizzazione $P'(i,\ldots,k)$ di $x_i/x_{i+1}/\ldots/x_k$ e una certa parentesizzazione $P'(k+1,\ldots,j)$ di $x_{k+1}/x_{k+2}/\ldots/x_j$, per un certo intero k compreso tra i e j-1.
- $P'(i,\ldots,j)$ e\` di costo massimo se e solo se $P'(i,\ldots,k)$ e\` la parentesizzazione di costo massimo di $x_i/x_{i+1}/\ldots/x_k$ e $P'(k+1,\ldots,j)$ e\` la parentesizzazione di costo minimo di $x_{k+1}/x_{k+2}/\ldots/x_j$.
  - si ha quindi $MAX(i,j) = MAX(i,k) / min(k+1,j)$ per un certo k, $i \leq k \leq j-1$
  - cioe\` $MAX(i,j)=max_{\{i<=k<j\}}\{MAX(i,k)/min(k+1,j)\}$.
- Siccome non sappiamo in corrispondenza di quale indice k compreso tra i e j-1 si ottiene la parentesizzazione di costo massimo di $x_i/x_{i+1}/\ldots/x_j$ allora dobbiamo calcolare il massimo su tutti i k compresi tra i e j-1 di $MAX(i,k)/min(k+1,j)$.

Parentesizzazione di valore massimo

- Ho bisogno anche di una formula per $min(i,j)$
- Sia $P''(i,\ldots,j)$ la parentesizzazione di $x_i/x_{i+1}/\ldots/x_j$ di costo minimo
- A livello piu\` esterno, $P''(i,\ldots,j)$ contiene una certa parentesizzazione $P''(i,\ldots,k)$ di $x_i/x_{i+1}/\ldots/x_k$ e una certa parentesizzazione $P''(k+1,\ldots,j)$ di $x_{k+1}/x_{k+2}/\ldots/x_j$, per un certo intero k compreso tra i e j-1.
- $P''(i,\ldots,j)$ e\` di costo minimo se e solo se $P''(i,\ldots,k)$ e\` la parentesizzazione di costo minimo di $x_i/x_{i+1}/\ldots/x_k$ e $P''(k+1,\ldots,j)$ e\` la parentesizzazione di costo massimo di $x_{k+1}/x_{k+2}/\ldots/x_j$.
  - si ha quindi che $min(i,j) = min(i,k) /MAX(k+1,j)$
- Siccome non sappiamo in corrispondenza di quale indice k compreso tra i e j-1 si ottiene la parentesizzazione di costo minimo di $x_i/x_{i+1}/\ldots/x_j$ allora dobbiamo calcolare il minimo su tutti i k compresi tra i e j-1 di $min(i,k)/MAX(k+1,j)$.
- cioe\` $min(i,j)=min_{\{i<=k<j\}}\{min(i,k)/MAX(k+1,j)\}$.
- Caso base: $min(i,j) =Max(i,j)=x_i$ se $i=j$

Esercizio

M[i,j]=MAX(i,j)
m[i,j]=min(i,j)

```
CatenaDiDivisioni (x) // x array t.c. x[i]=xi
 n=length(x)
 for i=1 to n
     M[i, i]=m[i, i]=x[i]
 for lung=2 to n //ogni iterazione calcola M[i,j] ed m[i,j] per
     //i,j tali che i<j e j-i=1=lung
   for i=1 to n-lung+1
       j=i+lung-1
       M[i, j]=0
       m[i, j]=∞
       for k=i toj-1
           vM = M[i, k]/m[k+1, j]
           vm = m[i, k]/M[k+1, j]
           if vM > M[i, j] then M[i, j]=vM
           if vm < m[i, j] then m[i, j]= vm
 return M[1, n]
```

$O(n^3)$

vengono calcolati M[i,j] e m[i,j]
per ogni (i,j), con i<j, in questo ordine:
(1,2),(2,3), … ,(n-1,n)
(1,3),(2,4),…,(n-2,n)
…
(1,n-1),(2,n)
(1,n)

133

Esercizio

- Dovete organizzare una festa ed invitare un insieme di persone nel gruppo dei vostri amici. Di ciascun amico conoscete l'eta` e a ciascuno di essi avete attribuito un valore che indica quanto vi e` gradita la presenza di quella persona alla festa. Volete selezionare gli invitati in modo che le eta` delle persone invitate differisca almeno di un certo numero di anni e che la somma dei valori degli invitati sia la piu` alta possibile. Avete deciso che presi due qualsiasi invitati le loro eta` $a_i$ e $a_j$ devono differire almeno di. max$\{a_i ,a_j \}$/V (parte intera inferiore), dove V e` un intero positivo che dipende dal tipo di festa. Supponiamo che le eta` siano a due a due distinte.
- Formalizzare il problema come problema computazionale
- Fornire una relazione di ricorrenza per computare il valore della soluzione ottima
- Scrivere un algoritmo per computare il valore della soluzione ottima e un algoritmo per stampare la soluzione ottima

134

Esercizio

- Formalizzazione problema:
- Input: In input riceviamo un intero positivo V, un intero positivo n, n interi positivi $a_1,\dots,a_n$ a due a due distinti ed n valori reali $v_1,\dots,v_n$
- Obiettivo: Individuare un sottoinsieme S delle n persone in modo che
  1. per ogni coppia (i,j) di persone in S si abbia $|a_i - a_j| \geq \max\{a_i, a_j\}/V$ (parte intera inferiore della frazione)
  2. e che $\sum_{i \in S} v_i$ massima , il massimo e` calcolato considerando tutti gli insiemi S che soddisfano 1.

135

Esercizio

- Greedy: Potremmo pensare di esaminare le persone in ordine crescente di eta`.
- Sia V=5 e consideriamo le persone 1,2,3 con eta` 11 12 34 e valori 40 80 2: se ordino in modo crescente rispetto alle eta` ottengo la soluzione {1,3}. Se prendo 1 poi ho max(11,12}/5>2 e 12-11=1 per cui non posso prendere 2. Posso prendere 3 perche' max{11,34}/5<7 e 34-11=23. Il valore delle soluzione è 42 mentre la soluzione ottima e` {2,3} con valore 82.
- Greedy: Potremmo pensare di esaminare le persone in modo non crescente rispetto ai valori.
- Sia V=5 e consideriamo le persone 1,2,3 con eta` 50 41 52 e valori 40 39 2: se ordino in modo non crescente rispetto ai valori ottengo la soluzione {1} con valore 40 perche` dopo aver scelto 1 non posso scegliere nessun altro in quanto max{41,50}/5=10 ma 50-41=9 e max{50,52}/5>10 ma 52-50=2.
- La soluzione ottima e` {2,3} con valore 41. Dopo aver preso 2 posso prendere anche 3 perche' max{52,41}/5<11 e 52-41=11

136

Esercizio

- OPT(j)= valore della soluzione ottima per le j eta` piu` piccole

ordiniamo le eta` in modo che $a_1 < a_2 < \ldots < a_n$. NB $a_1 > 4$

- definiamo d(j) : l'indice i piu` grande tale che $1 \le i \le j-1$ e $a_j - a_i \ge a_j/V$
- caso in cui nella soluzione ottima c'e` un invitato di eta` j
- in questo caso sicuramente le persone con indice d(j)+1,…,j-1 non possono essere nella soluzione
  - $OPT(j)=OPT(d(j))+v_j$
- caso in cui nella soluzione ottima non c'e` un invitato di eta` j
  - OPT(j)=OPT(j-1)
- quindi se j>0 → $OPT(j)=\max(OPT(d(j))+v_j, OPT(j-1)\}$
- se j=0 → OPT(j)=0

137

Esercizio

Supponiamo di avere un array di lettere e di voler trovare la sottosequenza palindroma piu` lunga al suo interno. La sottosequenza non è formata necessariamente da celle contigue. Esempio abcdbda ha soluzione adbda

Definiamo OPT(i,j)=sottosequenza palindroma piu` lunga in A[i]…A[j]

OPT(i,i)=1,

OPT(i,i+1)=2 se A[i]=A[i+1] e OPT(i,i+1)=0 se A[i] !=A[i+1]

Se i<j-1:

Se A[i]=A[j] allora la sottosequenza piu` lunga comprende A[i] e A[j] e la sottosequenza piu` lunga per A[i+1]…A[j-1]→ OPT(i,j)=2+OPT(i+1,j-1)

Se A[i]!=A[j] allora la sottosequenza piu` lunga da i a j sicuramente non contiene uno tra A[i] e A[j].

Se non contiene A[j] allora la soluzione ottima è quella per A[i]…A[j-1]

Se non contiene A[i] allora la soluzione ottima è quella per A[i+1]…A[j]

→ OPT(i,j)=max{OPT(i+1,j),OPT(i,j-1)}

138

Esercizio 27 cap. 6

- I proprietari di una pompa di carburante devono confrontarsi con la seguente situazione:
- Hanno un grande serbatoio che immagazzina gas; il serbatoio puo` immagazzinare fino ad L galloni alla volta.
- Ordinare carburante e` molto costoso e per questo essi vogliono farlo raramente. Per ciascun ordine pagano un prezzo fisso P in aggiunta al costo della carburante.
- Immagazzinare un gallone di carburante per un giorno in piu` costa c dollari per cui ordinare carburante troppo in anticipo aumenta i costi di immagazzinamento.
- I proprietari del distributore stanno progettando di chiudere per una settimana durante l'inverno e vogliono che per allora il serbatoio sia vuoto.
- In base all'esperienza degli anni precedenti, essi sanno esattamente di quanto carburante hanno bisogno. Assumendo che chiuderanno dopo n giorni e che hanno bisogno di g_i galloni per ciascun giorno i=1,…,n e che al giorno 0 il serbatoio e` vuoto, dare un algoritmo per decidere in quali giorni devono effettuare gli ordini e la quantita` di carburante che devono ordinare in modo da minimizzare il costo.

Esercizio 27 cap. 6: Soluzione

- Supponiamo che il giorno 1 i proprietari ordinino carburante per i primi i-1 giorni: $g_1+g_2+\ldots+g_{i-1}$
- Il costo di questa operazione si traduce in un costo fisso di P piu` un costo di $c(g_2+2g_3+3g_4\ldots+(i-2)g_{i-1})$
- Sia OPT(d) il costo della soluzione ottima per il periodo che va dal giorno d al giorno n partendo con il serbatoio vuoto
- La soluzione ottima da d ad n include sicuramente un ordine al giorno d per un certo quantitativo di carburante. Sia f il giorno in cui verra` fatto il prossimo ordine.
- La quantita ordinata il giorno d deve essere $g_d+g_{d+1}+\ldots+g_{f-1}$ (con $g_d+g_{d+1}+\ldots+g_{f-1} \leq L$)
- Il costo connesso a questo ordine e` $P+c(g_{d+1}+2g_{d+2}+3g_{d+3}\ldots+(f-1-d)g_{f-1})$
- Il costo della soluzione ottima da d ad n se il secondo ordine in

questo intervallo avviene al tempo f e` $P+\sum_{i=d}^{f-1} c(i-d)g_i + OPT(f)$

$$OPT(d) = P + \min_{f>d:\sum_{i=d}^{f-1} g_i \leq L} \sum_{i=d}^{f-1} c(i-d)g_i + OPT(f)$$

Esercizio 27 cap. 6: Soluzione

Possiamo scrivere un algoritmo iterativo che computa le soluzioni a partire da d = n fino a d=1 e memorizza le soluzioni in un array

```
Input: n, L, g1,…,gn
A[d,p]  //contiene la somma dei gi per i=d,…,p e p t.c. questa
       //somma <=L
S[d,p]   //contiene la somma dei gi(i-d) per i=d,…p
M[d] //contiene soluzione ottima da d ad n

For d = 1 to n
  A[d,d]=gd
  S[d,d]=0

For d = n-1 to 1{                              O(n²)
   min= large_value
   For  f=d+1 to n  {
      If A[d,f-2]+gf-1<=L{
         A[d,f-1]=A[d,f-2]+gf-1
          S[d,f-1] =S[d,f-2] + (f-1-d)gf-1
          If P+S[d,f-1]+ M[f]<min
            min= P+c*S[d,f-1]+ M[f]
        M[d]=min} //fine if esterno
      Else  break }//ha computato l'ottimo per giorni da d ad n
  } //fine for esterno
return M[1]
```

141

141