Programmazione dinamica (VII parte)

Progettazione di Algoritmi a.a. 2022-23

Matricole congrue a 1

Docente: Annalisa De Bonis

128

Moltiplicazione di una catena di matrici

- Input: una sequenza di n matrici $A_1, A_2, A_3, \ldots, A_n$, *compatibili due a due rispetto al prodotto*
  - Due matrici A e B sono compatibili rispetto al prodotto se il numero di colonne di A è uguale al numero di righe di B
- Obiettivo: vogliamo calcolare il prodotto delle n matrici in modo da minimizzare il numero di moltiplicazioni.
  - Data una matrice mxn A e una matrice nxp B la matrice AxB è una matrice mxp e per calcolare ciascuna delle mp entrate di AxB abbiamo bisogno di moltiplicare una riga di A per una colonna di B → n moltiplicazioni scalari per ciascuna entrata di AxB→ mnp moltiplicazioni scalari.
  - La moltiplicazione tra matrici è associativa per cui possiamo scegliere l'ordine in cui moltiplichiamo le matrici parentesizzando opportunamente la catena di matrici

Moltiplicazione di una catena di matrici

- Consideriamo le tre matrici
  - A: 100×1 vettore riga
  - B: 1×100 vettore colonna
  - C: 100×1 vettore riga
- Numero di moltiplicazioni per diverse parentesizzazioni:
  - $((A \cdot B) \cdot C) \rightarrow (100 \times 1 \times 100)+(100 \times 100 \times 1)= 20000$
    - prima $100 \times 1 \times 100$ moltiplicazioni per $A \cdot B$ e poi $100 \times 100 \times 1$ moltiplicazioni per $(A \cdot B) \cdot C$
  - $(A \cdot (B \cdot C)) \rightarrow (1 \times 100 \times 1)+ (100 \times 1 \times 1) = 200$
    - prima $1 \times 100 \times 1$ moltiplicazioni per $B \cdot C$ e poi $100 \times 1 \times 1$ moltiplicazioni per $A \cdot (B \cdot C)$

130

Moltiplicazione di una catena di matrici

- Per i<j, una parentesizzazione P(i,…,j) del prodotto $A_i \cdot A_{i+1} \cdots A_j$ consiste nel prodotto di due parentesizzazioni (P(i,…,k) · P(k+1,…j))
  - P(i,…,k) e P(k+1,…j) sono le due parentesizzazioni a livello piu` esterno
- Sia $A(i \dots j)= A_i \cdot A_{i+1} \cdots A_j$ una sottosequenza della catena di moltiplicazioni $A_1 \cdot A_2 \cdot A_3, \dots \cdot A_n$.
- Supponiamo che P(i . . . j) sia una parentesizzazione ottima di A(i . . . j) e siano P(i . . . k) e P(k + 1 . . . j) le parentesizzazioni a livello piu` esterno in P(i . . . j) .
- Sottostruttura ottimale: se P(i . . . j) è una parentesizzazione ottima di A(i . . . j) allora le due parentesizzazioni P(i . . . k) e P(k + 1 . . . j) sono ottime per le sottosequenze A(i . . . k) e A(k + 1 . . . j) rispettivamente.

131

Moltiplicazione di una catena di matrici

OPT(i,j): minimo numero di moltiplicazioni scalari per calcolare il prodotto A(i . . . j)

Caso i = j. In questo caso (base) OPT(i,j)=0

Caso i < j.

- Supponiamo di sapere che la parentesizzazione ottima per A(i,…,j) sia formata a livello piu` esterno dal prodotto delle parentesizzazioni di A(1,..,k) e A(k+1,….,n). Per la sottostruttura ottimale si ha:
  OPT(i,j)= OPT(i,k) + OPT(k + 1,j) + $c_{i-1} \cdot c_k \cdot c_j$
  - c_{i-1} = numero di righe della matrice A_i
  - c_i = numero di colonne della matrice A_i
  - la matrice ($A_i \cdot . . . \cdot A_k$) ha dimensioni $c_{i-1} \times c_k$
  - la matrice ($A_{k+1} \cdot . . . \cdot A_j$) ha dimensioni $c_k \times c_j$

  → costo per moltiplicare la matrice ($A_i \cdot . . . \cdot A_k$) con ($A_{k+1} \cdot . . . \cdot A_j$) è $c_{i-1} \cdot c_k \cdot c_j$
- Siccome non conosciamo il valore di k nella soluzione ottima, computiamo il valore OPT(i,k) + OPT(k + 1,j) + $c_{i-1} \cdot c_k \cdot c_j$ per ogni k tra 1 e j-1 e scegliamo il piu` piccolo di questi valori.

132

Moltiplicazione di una catena di matrici

Dai due casi precedenti si ha la seguente formula di ricorrenza:

OPT(i,j)=0 se i=j

$$OPT(i,j)= \min_{i \le k < j} \{OPT(i,k) + OPT(k + 1,j) + c_{i-1} \cdot c_k \cdot cj\} \qquad \text{se } i < j$$

133

Esercizio

```
MoltiplicazioneMatrici (c) // c array t.c. c[i] = ci= #colonne Ai = = #righe Ai+1
 n=length(c)
 for i=1 to n
     M[i, i]=0
 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=lung
   for i=1 to n-lung+1
      j=i+lung-1
      M[i, j]=∞
      for k=i toj–1
           M = M[i, k]+ M[k+1, j] + c[i-1] c[k] c[j]
           if M < M[i, j] then M[i, j]=M
   return M[1, n]
```

$O(n^3)$

vengono calcolati 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)

134

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

135

Esercizio

- Formalizzazione problema:
- Input: In input riceviamo un intero positivo V, un intero positivo n, n interi positivi $a_1,\ldots,a_n$ a due a due distinti ed n valori reali $v_1,\ldots,v_n$
- 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.

136

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

137

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 $1 \leq i \leq j-1$ e $a_j - a_i \geq a_j/V$
- 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

138

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 dda celle contigue.

Esempio abcdada ha soluzione adda

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

OPT(i,i)=1

Se i<j:

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)}

139

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]
```

142