Sottosequenza crescente piu` lunga

elementi non necessariamente contigui

- Vogliamo individuare la sottosequenza crescente piu` lunga in una sequenza di numeri. La sottosequenza non deve essere necessariamente formata da elementi contigui nella sequenza input. La sequenza input e` memorizzata in un array A.
- Indichiamo con A[0,…,i] il segmento di A degli elementi con indice da 0 a i.
- Esempio: A=<3 12 9 4 12 5 8 11 13 10>
- La sottosequenza crescente piu` lunga e` <3 4 5 8 11 13>

100

Sottosequenza crescente piu` lunga

elementi non necessariamente contigui

- OPT(i) = lunghezza della sottosequenza crescente piu` lunga che termina in i (l'ultimo elemento della sottosequenza e` A[i])
- Una volta che abbiamo calcolato OPT(i) per ogni i, andiamo a calcolare il massimo di tutti i valori OPT(i) per ottenere la lunghezza della sottosequenza crescente piu` lunga.

101

Sottosequenza crescente piu` lunga

elementi non necessariamente contigui

- Osserviamo che nella sottosequenza crescente piu` lunga che termina in i, l'elemento A[i] e` preceduto

  dalla piu` lunga sottosequenza crescente che termina in un certo j tale che j<i e A[j]<[A[i].

  - Quale di questi j bisogna prendere?
    - Quello per cui la lunghezza della sottosequenza e` massima, cioe` l'indice j per cui si ottiene il massimo valore OPT(j) in questo insieme: { OPT(j): 0 ≤j ≤ i-1 e A[j]<A[i] }

Si ha quindi

OPT(i)=max{OPT(j)+1: 0 ≤j ≤ i-1 e A[j] < A[i] }

102

Sottosequenza crescente piu` lunga

elementi non necessariamente contigui

- Esempio: A=<3 12 9 4 12 5 8 11 13 10>
- per calcolare OPT(9) consideriamo j=0, j=2, j=3, j=5, j=6
- per calcolare OPT(8) consideriamo j=0, j=2, j=3,j=4 j=5, j=6,j=7
- per calcolare OPT(7) consideriamo j=0, j=2, j=3, j=5, j=6
- per calcolare OPT(6) consideriamo j=0, j=3, j=5,
- per calcolare OPT(5) consideriamo j=0, j=3
- per calcolare OPT(4) consideriamo j=0, j=2, j=3
- per calcolare OPT(3) consideriamo j=0
- per calcolare OPT(2) consideriamo j=0
- per calcolare OPT(1) consideriamo j=0
- OPT(0)=1 → OPT(1)=OPT(2)=OPT(3)=2
- OPT(0)=1 e OPT(2)=OPT(3)=2 → OPT(4)=3
- OPT(0)=1 e OPT(3)=2 → OPT(5)= 3
- OPT(0)=1 , OPT(3)=2 , OPT(5)= 3 → OPT(6)=4
- OPT(0)=1 , OPT(2)=OPT(3)=2 , OPT(5)= 3 , OPT(6)=4 → OPT(7)=5
- OPT(0)=1 , OPT(2)=OPT(3)=2 , OPT(4)=3, OPT(5)= 3 , OPT(6)=4, OPT(7)=5→ OPT(8)=6
- OPT(0)=1 e OPT(3)=2 , OPT(5)= 3, OPT(6)=4 → OPT(9)=5

103

Sottosequenza crescente piu` lunga

elementi non necessariamente contigui

Questo algoritmo trova la lunghezza della sottosequenza crescente piu` lunga che termina in i

```
LIS(A,i):
  if i<0 return 0
  if i=0
    P[0]=-1, M[0]=1, return 1
  if M[i]!=empty return M[i]
  M[i]=1
  P[i]=-1      //serve nel caso alla fine M[i]=1
  for( j=0; j<i; j++)
     m=LIS(A,j)
     if (A[j]<A[i] && M[i]<m+1)
       M[i]=m+1
       P[i]=j
  return M[i]
```

M globale
P globale: mi serve per la stampa
P[i]= indice dell'elemento che precede i nella sottosequenza crescente piu` lunga che termina in i

tempo $O(n^2)$

Per trovare la sottosequenza crescente piu` lunga dell'array occorre prima invocare LIS(A,n-1) e poi trovare il massimo dell'array M.

Sottosequenza crescente piu` lunga

elementi non necessariamente contigui

Questo algoritmo stampa la sottosequenza crescente piu` lunga. Indichiamo con max l'indice in cui si trova l'elemento massimo di M, cioe` M[max]=max{OPT(i): 0<=i<=n-1}

```
PrintLIS(A,i):
if i >= 0
    PrintLis(A,P[i])
    print(A[i])
```

prima chiamata con i=max

M e P costruiti in precedenza

Sottosequenza crescente piu` lunga

elementi non necessariamente contigui

Questa e` la versione iterativa dell'algoritmo LIS

```
ITLIS(A)
   n = A.length
   for i=0 to n-1
      M[i]=1
      P[i]=-1
   for (int i = 0; i < n; i++) //ogni iterazione computa OPT(i), i=0,...,n-1
      for (int j = 0; j < i; j++) //ogni iterazione computa OPT(j) , j<i e A[j]<A[i]
         if (A[j] < A[i])
            if M[j] + 1 >M[i]
               M[i]=M[j]+1
               P[i]=j
   max= M[0]
   for i=0 to n-1
      if M[i]>max
         max=A[i]
   return max
```

M[i]=lunghezza sottosequenza crescente piu` lunga che termina in A[i]
P[i]= indice predecessore di A[i] nella sottosequenza crescente piu` lunga che termina in A[i]

106

Sottosequenza crescente piu` lunga

elementi non necessariamente contigui

Questa e` la versione iterativa dell'algoritmo LIS

```
ITLIS(A)
   n = A.length
   for i=0 to n-1
      M[i]=1
      P[i]=-1
   for (int i = 0; i < n; i++) //ogni iterazione computa OPT(i), i=0,...,n-1
      for (int j = 0; j < i; j++) //ogni iterazione computa OPT(j) , j<i e A[j]<A[i]
         if (A[j] < A[i])
            if M[j] + 1 >M[i]
               M[i]=M[j]+1
               P[i]=P[j]
   max= M[0]
   for i=0 to n-1
      if M[i]>max
         max=A[i]
   return max
```

M[i]=lunghezza sottosequenza crescente piu` lunga che termina in A[i]
P[i]= indice predecessore di A[i] nella sottosequenza crescente piu` lunga che termina in A[i]

107

Esercizio

- Si scriva un algoritmo che trova il valore massimo ottenibile con una parentesizzazione completa della seguente espressione: $x_1/x_2/.../x_{n-1}/x_n$
- Una parentesizzazione completa di $x_1/x_2/.../x_{n-1}/x_n$ 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.

Esercizio

- 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

Esercizio

- Soluzione basata sulla programmazione dinamica.
- Sia i<j e sia P(i,...,j) una parentesizzazione della sottoespressione $x_i/x_{i+1}/.../x_j$. Supponiamo che questa parentesizzazione a livello piu` esterno sia formata da una certa parentesizzazione P(i,...,k) di $x_i/x_{i+1}/.../x_k$ e una certa parentesizzazione P(k+1,...,j) di $x_{k+1}/x_{k+2}/.../x_j$, per un certo i≤ k ≤ j-1
- Il costo C(P(i,...,j)) di P(i,...,j) e` quindi C(P(i,...,k)) / C(P(k+1,...,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)).

Esercizio

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

Esercizio

- Ho bisogno anche di una formula per min(i,j)
- Sia P''(i,...,j) la parentesizzazione di $x_i/x_{i+1}/.../x_j$ di costo minimo
- A livello piu` esterno, P''(i,...,j) contiene una certa parentesizzazione P''(i,...,k) di $x_i/x_{i+1}/.../x_k$ e una certa parentesizzazione P''(k+1,...,j) di $x_{k+1}/x_{k+2}/.../x_j$, per un certo intero k compreso tra i e j-1.
- P''(i,...,j) e` di costo minimo se e solo se P''(i,...,k) e` la parentesizzazione di costo minimo di $x_i/x_{i+1}/.../x_k$ e P''(k+1,...,j) e` la parentesizzazione di costo massimo di $x_{k+1}/x_{k+2}/.../x_j$.
  - 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 $x_i/x_{i+1}/.../x_j$ allora dobbiamo calcolare il minimo su tutti i k compresi tra i e j-1 di min(i,k)/MAX(k+1,j).

Esercizio

- Si ha quindi:
- $MAX(i,j)= \max_{\{k:\ i \le k \le j-1\}}\{ MAX(i,k)/min(k+1,j) \}$ se i<j
- $MAX(i,j)= x_i$ se i=j
- $min(i,j)= \min_{\{k:\ i \le k \le j-1\}}\{ min(i,k)/MAX(k+1,j) \}$ se i<j
- $min(i,j)= x_i$ se i=j

Esercizio

```
CatenaDiDivisioni (x) // x array t.c. x[i]=x_i
    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=lung
    for i←1 to n
      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 M[i, j]< vM then M[i, j]←vM
          if m[i, j]> vm then m[i, j]← vm
   return M[1, n]
```

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

$O(n^3)$

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 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]
```