Sciatori

Siano dati:

- n sciatori di altezza $H[0], \ldots, H[n-1]$;
- n paia di sci di lunghezza $L[0], \ldots, L[n-1]$.

Problema: trovare un assegnamento sciatore-sci che minimizzi la somma delle differenze assolute fra l'altezza degli sciatori e la lunghezza degli sci a loro assegnati.

In altre parole, se allo sciatore i è assegnato il paio di sci $m(i)$ (matching), minimizzare la seguente quantità:

$$\sum_{i=0}^{n-1} |H[i] - L[m(i)]|$$

Sciatori

Soluzione proposta

Identifichiamo la coppia sciatore-sci la cui differenza è minore, assegniamo lo sci allo sciatore e ci riduciamo ad un problema più piccolo, con uno sciatore in meno e uno sci in meno.

Dimostrare la correttezza dell'algoritmo proposto oppure trovare un controesempio.

Sfilatino alla Nutella

Nella sagra di Hateville, è stato realizzato lo sfilatino alla Nutella più lungo del mondo, lungo L centimetri. Ora si tratta di realizzare un altro record: il maggior numero di persone servite con lo stesso sfilatino.

Alla sagra sono presenti n persone, dove la persona i-esima chiede un *segmento* di sfilatino lungo $V[i]$ centimetri; secondo il regolamento, ogni richiesta va servita esattamente, ovvero se la persona i verrà servita, riceverà il segmento richiesto. Tutte le lunghezze sono intere positive.

Scrivere un algoritmo che restituisca il numero massimo di persone che possono essere servite con lo sfilatino. Non è necessario utilizzare tutto lo sfilatino.

Oltre a calcolare la complessità dell'algoritmo proposto, discutere anche la correttezza, menzionando la tecnica scelta e specificando bene perché tale tecnica può essere applicata in questo caso.

Vettori ordinati

Si scriva un algoritmo che preso in input n e k, restituisca il numero totale di vettori distinti di lunghezza n, contenenti valori interi compresi fra 1 e k, ordinati dalla relazione $\leq$. Si discuta la correttezza e la complessità dell'algoritmo proposto.
Ad esempio, dati $n = 4$ e $k = 3$, questi sono i possibili vettori ordinati:

$$[1,1,1,1],[1,1,1,2],[1,1,1,3],[1,1,2,2],[1,1,2,3],$$
$$[1,1,3,3],[1,2,2,2],[1,2,2,3],[1,2,3,3],[1,3,3,3],$$
$$[2,2,2,2],[2,2,2,3],[2,2,3,3],[2,3,3,3],[3,3,3,3]$$

e quindi il valore da restituire è 15.

Spoiler alert!

Sciatori

Si consideri il seguente input: $H = [5, 10]$, $L = [9, 14]$.

- Secondo l'algoritmo, associamo:
  - $H[1] = 10$ con $L[0] = 9$ $(m(1) = 0)$
  - $H[0] = 5$ con $L[1] = 14$ $(m(0) = 1)$

  La differenza totale sarebbe $|9 - 10| + |14 - 5| = 10$.
- Se l'associazione fosse:
  - $H[0] = 5$ con $L[0] = 9$ $(m(0) = 0)$
  - $H[1] = 10$ con $L[1] = 14$ $(m(1) = 1)$

  La differenza totale sarebbe: $|9 - 5| + |14 - 10| = 8$.

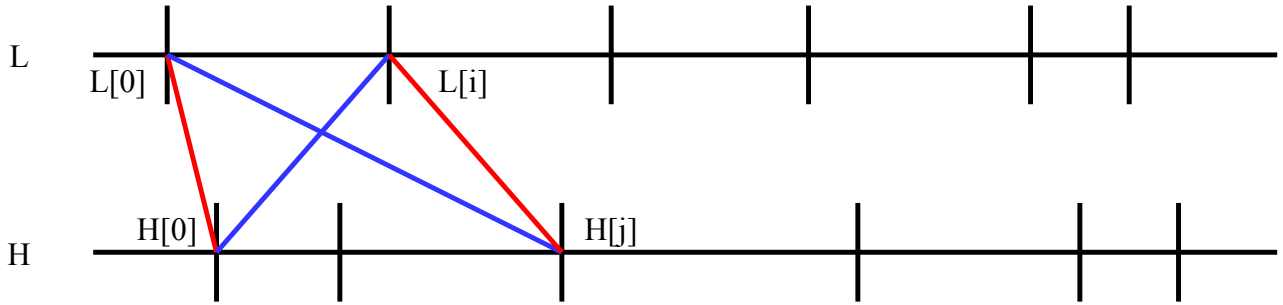
Quindi l'algoritmo proposto non è corretto.

Sciatori

Algoritmo corretto: ordinare sciatori e sci e assegnare lo sci i-esimo allo sciatore i-esimo

Sciatori

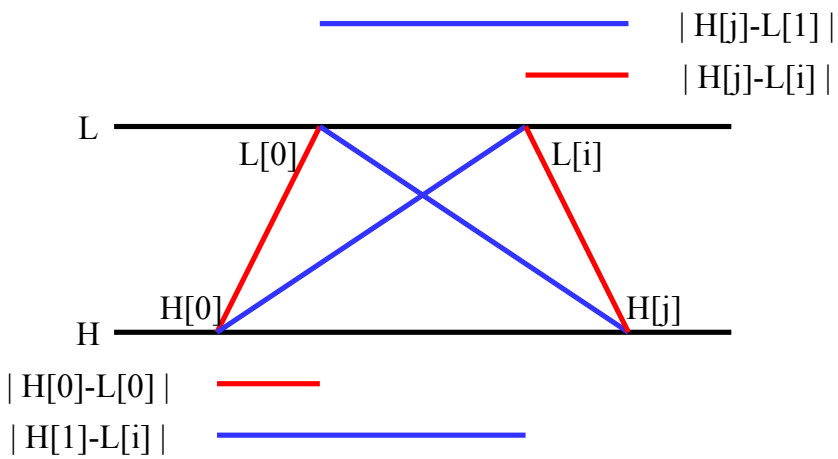
Assumendo che il vettore sia ordinato, supponiamo che lo sci 0 non sia assegnato allo sciatore 0 (connessioni rosse); quindi lo sci 0 è assegnato a uno sciatore j e lo sciatore 0 è assegnato a uno sci i (connessioni blu).

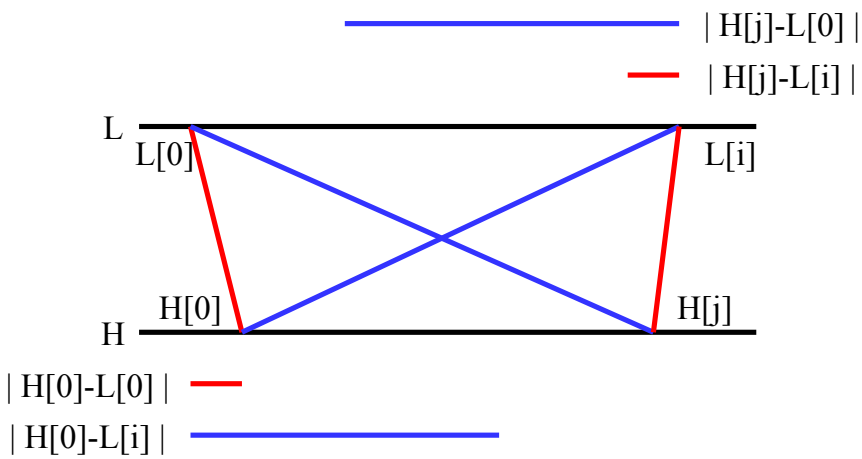


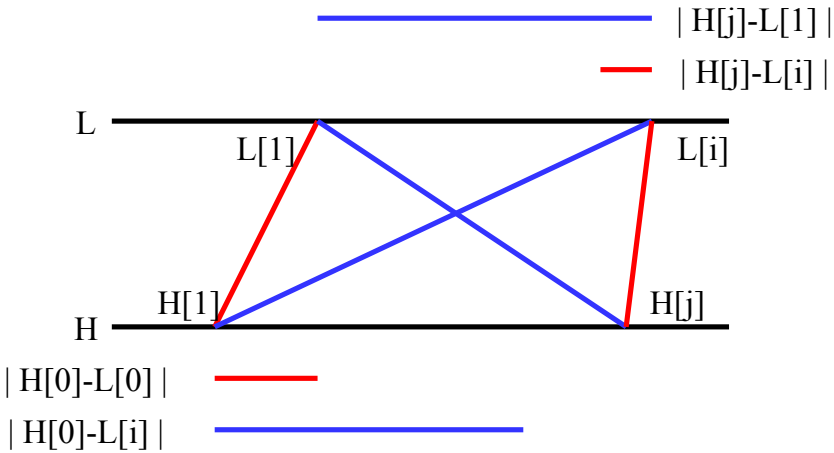
Dobbiamo dimostrare che:

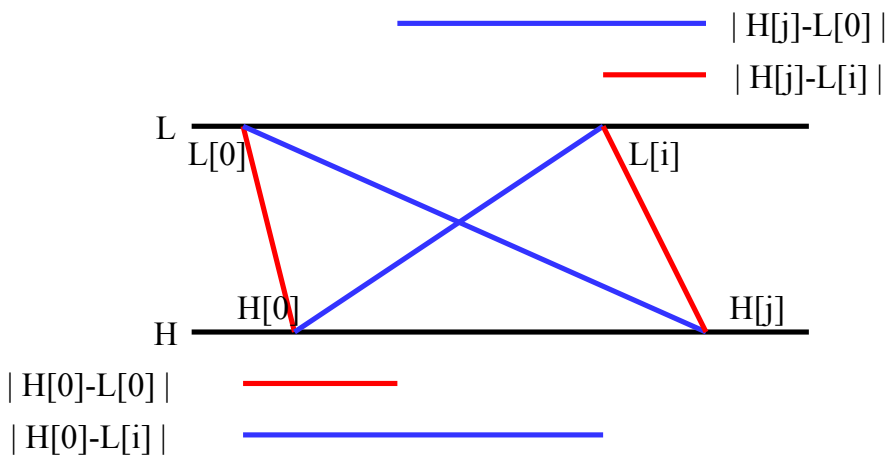
$$|H[0] - L[0]| + |H[j] - L[i]| \leq |H[0] - L[i]| + |H[j] - L[0]|$$

Sciatori


| H[j]-L[1] |
| H[j]-L[i] |
L
L[0]
L[i]
H[0]
H[j]
H
| H[0]-L[0] |
| H[1]-L[i] |


| H[j]-L[0] |
| H[j]-L[i] |
L
L[0]
L[i]
H[0]
H[j]
H
| H[0]-L[0] |
| H[0]-L[i] |


| H[j]-L[1] |
| H[j]-L[i] |
L
L[1]
L[i]
H[1]
H[j]
H
| H[0]-L[0] |
| H[0]-L[i] |


| H[j]-L[0] |
| H[j]-L[i] |
L
L[0]
L[i]
H[0]
H[j]
H
| H[0]-L[0] |
| H[0]-L[i] |

Sfilatino alla Nutella (Compito 31/05/13)

Il problema può essere risolto con tecnica greedy. Ordiniamo i segmenti per lunghezza crescente. Procediamo quindi a tagliare prima il segmento più corto, poi quello successivo e così via finché possibile (cioè fino a quando la lunghezza residua ci consente di ottenere un altro segmento).

---

int maxSegments(**int**[ ] V, **int** n, **int** L)

---

sort(V, n)
int $i = 0$
while $i < n$ **and** $L \geq V[i]$ **do**
    $L = L - V[i]$
    $i = i + 1$
return i % Soddisfatte le richieste $0 \ldots i-1$, quindi i in totale

---

Sfilatino alla Nutella (Compito 31/05/13)

Sottostruttura ottima

- Sia S una soluzione ottima, $x \in S$ un elemento di S
- $S - \{x\}$ è una soluzione ottima per il sottoproblema $L - V[x]$ che non consideri il segmento x
- Se così non fosse, esisterebbe una soluzione S' tale che $|S'| > |S - \{x\}|$.
- Allora si potrebbe ottenere una soluzione $S' \cup \{x\}$ di cardinalità superiore a S, assurdo

Sfilatino alla Nutella (Compito 31/05/13)

Proprietà greedy

- Sia S una soluzione ottima che non contiene il segmento più corto m
- Sia m' il segmento più corto in S
- La soluzione $S - \{m'\} \cup \{m\}$ ha la stessa cardinalità di S, soddisfa i vincoli ed è quindi ottima

Vettori ordinati

Sia $DP[i][t]$ il numero di vettori ordinati di lunghezza i, contenenti valori compresi fra 1 e t. $DP[i][t]$ può essere calcolato in maniera ricorsiva come segue:

$$DP[i][t] = \begin{cases} 1 & i = 0 \\ 0 & i > 0, t = 0 \\ \sum_{j=1}^{t} DP[i-1][j] & i > 0, t > 0 \end{cases}$$

La soluzione si trova in posizione $DP[n][k]$.
Ovviamente, questo richiede una tabella $O(nk)$, per calcolare ogni elemento delle quale saranno necessarie $O(k)$ operazioni, per un costo totale di $O(nk^2)$.

Vettori ordinati

```
int sortedPermRec(int i, int t, int[][] DP)
if i == 0 then
    return 1
if DP[i][t] < 0 then
    DP[i][t] = 0
    for j = 1 to t do
        DP[i][t] = DP[i][t] + sortedPermRec(i − 1, j, DP)
return DP[i][t]
```

Vettori ordinati

int sortedPerm(**int** n, **int** k)

int[ ][ ] $DP =$ **new int**$[0 \dots n][0 \dots k]$
for $i = 1$ **to** n **do**
    for $t = 1$ **to** k **do**
        $DP[i][t] = -1$
return sortedPermRec(n, k, DP)

Vettori ordinati

Una soluzione alternativa, più efficiente, calcola $DP[i][t]$ nel modo seguente:

$$DP[i][t] = \begin{cases} t & i = 1 \\ 0 & t = 0 \\ DP[i-1][t] + DP[i][t-1] & \text{altrimenti} \end{cases}$$

Vettori ordinati

sortedPerm(**int** n, **int** k)

int[ ][ ] $DP =$ **new int**$[0 \ldots n][0 \ldots k]$
for $i = 1$ **to** n **do**
  $DP[i][0] = 0$
for $j = 1$ **to** k **do**
  $DP[1][j] = j$
for $i = 2$ **to** n **do**
  for $t = 1$ **to** k **do**
    $DP[i][t] = DP[i-1][t] + DP[i][t-1]$
return $DP[n][k]$

Ovviamente, questo richiede una tabella $O(nk)$, per calcolare ogni elemento delle quali saranno necessarie $O(1)$ operazioni, per un costo totale di $O(nk)$.