Sciatori

Siano dati:

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

Problema: Trovare un assegnamento sciatore-sci, in modo tale da minimizzare la somme delle differenze 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:

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

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.

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

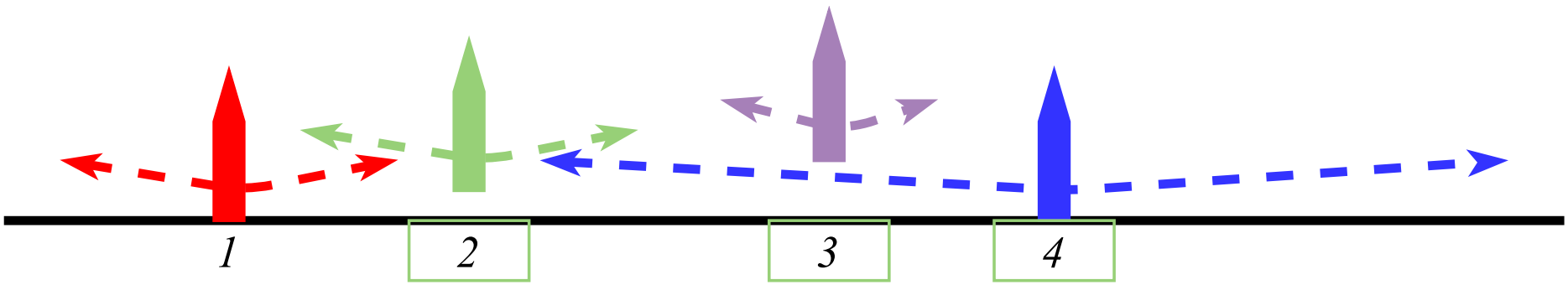
Oltre a calcolare la complessità dell'algoritmo proposto, discutere anche la correttezza, menzionando la tecnica scelta e specificando bene perché tale tecnica può essere applicata in questo caso.

High Line

La High Line è un parco lineare di New York realizzato su una sezione in disuso della ferrovia sopraelevata chiamata West Side Line.

- E' un rettilineo lungo L metri corredato da aiuole e piante. Lungo il parco, esistono n irrigatori.
- L'irrigatore i-esimo è collocato ad una distanza $D[i]$ dall'inizio della High Line e ha un raggio di azione pari a $R[i]$, ovvero innaffia la sezione $[\ D[i] - R[i], D[i] + R[i]\ ]$.

Scrivere un algoritmo che restituisca il minimo numero di irrigatori che vanno attivati per innaffiare l'intera linea $[0, L]$, oppure -1 se ciò è impossibile.



Costo partizione di un vettore

- Il costo $C(i, j)$ di un sottovettore $V[i \dots j]$ di V è pari alla somma dei suoi elementi
  - Esempio: $V = [1, 2, 4, 8, 16, 2]$, $C(2, 4) = 2 + 4 + 8 = 14$.
- Una k-partizione di V è una divisione di V in k sottovettori non vuoti, contigui che coprono totalmente il vettore e non si sovrappongono.
  - Esempio: $V = [2, 3, 7, -7, 15, 2]$, una delle $n - 1$ 2-partizioni è $[2, 3, 7], [-7, 15, 2]$.
- Il costo della k-partizione è il costo massimo dei suoi sottovettori.
  - Nella 2-partizione $[2, 3, 7], [-7, 15, 2]$, il costo dei sottovettori è
    $2 + 3 + 7 = 12$,
    $-7 + 15 + 2 = 10$,
    quindi il costo della 2-partizione è pari a 12.

Costo partizione di un vettore (Versione formale)

- Il costo $C(i,j) = \sum_{t=i}^{j} V[t]$
- k-partizione di V è una divisione di V in k sottovettori contigui $V[j_0+1 \ldots j_1], V[j_1+1 \ldots j_2], \ldots, V[j_{k-1}+1 \ldots j_k]$ con $j_0 = 0, j_k = n$ e $j_t < j_{t+1}, \forall 1 \leq t < k$.
- Il costo della k-partizione è pari a:
$$\max\{C(j_{t-1}+1, j_t) : \forall 1 \leq t \leq k\}$$

Costo partizione di un vettore

Scrivere un algoritmo che prenda in input un vettore V contenente n interi e un intero k tale che $2 \leq k \leq n$ e restituisca il costo della k-partizione di V di costo minimo.

Esempio: $V = [2, 3, 7, -7, 15, 2]$, $k = 3$
$[2, 3, 7], [-7, 15], [2]$ costo 2+3+7=12
$[2, 3], [7][-7, 15, 2]$ costo -7+15+2=10

1. Soluzione per $k = 2$ (facile: $O(n^2)$, meglio in $O(n)$).
2. Soluzione per $k = 3$ (facile: $O(n^3)$, meglio in $O(n^2)$).
3. Soluzione generale (facile: $O(n^k)$, meglio in $O(kn^2)$).

Spoiler alert!

Sciatori

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

- Secondo l'algoritmo, associamo:
  - $H[2] = 10$ con $L[1] = 9$ $(m(2) = 1)$
  - $H[1] = 5$ con $L[2] = 14$ $(m(1) = 2)$

  La differenza totale sarebbe $|9 - 10| + |14 - 5| = 10$.
- Se l'associazione fosse:
  - $H[1] = 5$ con $L[1] = 9$ $(m(1) = 1)$
  - $H[2] = 10$ con $L[2] = 14$ $(m(2) = 2)$

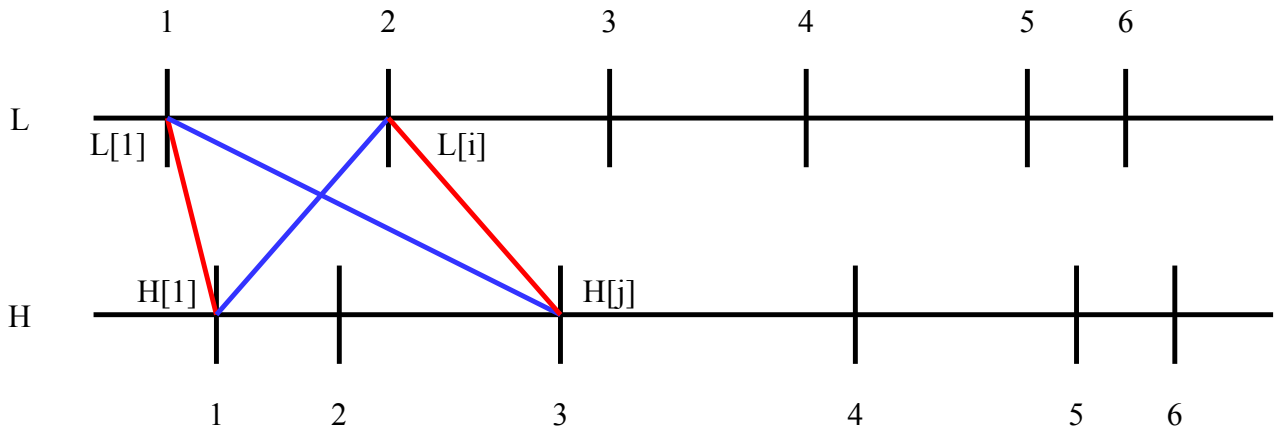
  La differenza totale sarebbe: $|9 - 5| + |14 - 10| = 8$.

Quindi l'algoritmo proposto non è corretto.

Sciatori

Algoritmo greedy corretto

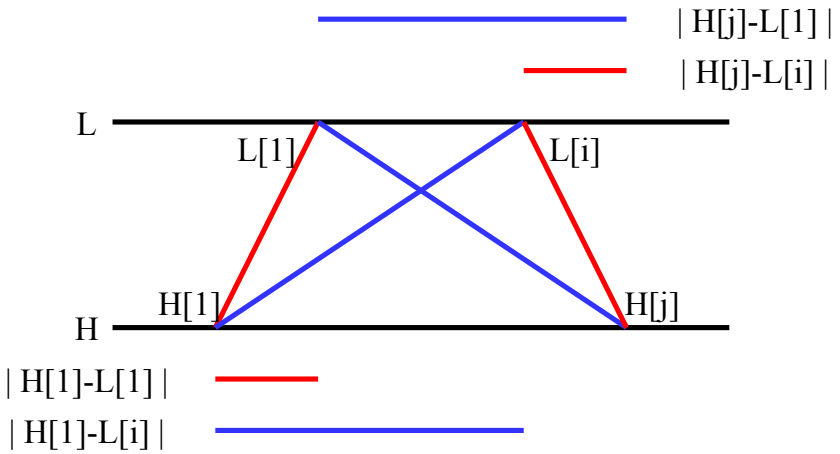
Si assegna lo sci più corto allo sciatore più basso, e si prosegue così fino allo sci più lungo assegnato allo sciatore più alto. Il costo computazionale è quindi dato dall'ordinamento dei due vettori.

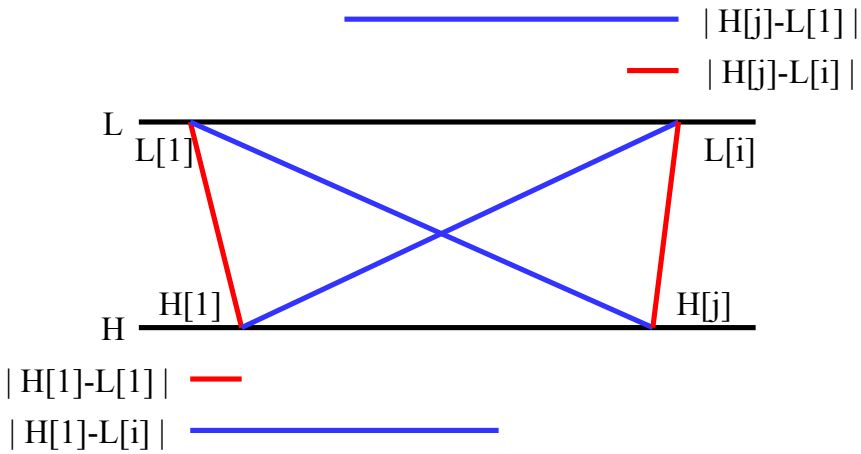


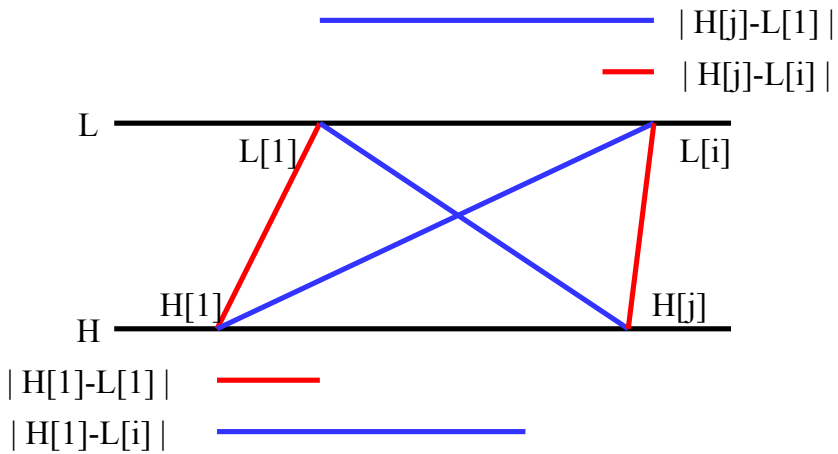
Dobbiamo dimostrare che:

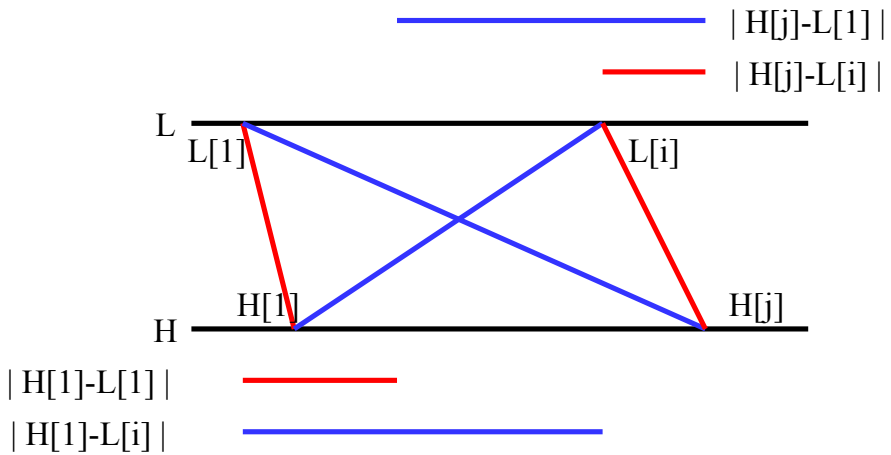
$$|H[1] - L[1]| + |H[j] - L[i]| \leq |H[1] - L[i]| + |H[j] - L[1]|$$

Sciatori


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


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


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


| H[j]-L[1] |
| H[j]-L[i] |
L
L[1]
L[i]
H[1]
H[j]
H
| H[1]-L[1] |
| H[1]-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 $i = 1$
while $i \leq n$ **and** $L \geq V[i]$ **do**
    $L = L - V[i]$
    $i = i + 1$
return $i - 1$

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 cardinalita di S ed quindi è ottima

High Line (Compito 29/06/17)

- Il problema può essere risolto tramite un algoritmo greedy
- Ogni irrigatore definisce un intervallo $[\ D[i] - R[i], D[i] + R[i]\ ]$
- Ordiniamo gli intervalli per estremo di inizio
- La variabile *last* contiene la posizione dell'ultimo punto innaffiato
- All'inizio, $last = 0$, a significare che il prossimo innaffiatore deve coprire questa posizione
- Fra tutti gli irrigatori che raggiungono *last*, si prende quello con estremo di fine più alto
- Si memorizza questo estremo di fine in *last*
- Il problema si riduce al sottoproblema $[last, L]$, dove tutti gli intervalli precedenti non devono più essere considerati perché hanno estremo superiore minore di *last*.

High Line (Compito 29/06/17)

int sprinkles(**int**[ ] D, **int**[ ] R, **int** n)

{ ordina D, R per $D[i] - R[i]$ crescenti }
int $last = 0$ % Ultimo estremo intervallo da intersecare
int $count = 0$ % Numero innaffiatori da restituire
int $i = 0$
while $i \leq n$ **and** $0 \leq last \leq L$ **do**
    int $max = -1$
    while $i \leq n$ **and** $D[i] - R[i] \leq last$ **do**
        $max = \max(D[i] + R[i], max)$
        $i = i + 1$
    $last = max$
    $count = count + 1$
return iif($last \geq L$, $count$, -1)

High Line (Compito 29/06/17)

Il costo dell'algoritmo è lineare nel numero di intervalli, ma l'ordinamento richiede $O(n \log n)$.

Costo partizione di un vettore (Compito 05/06/14)

- Vediamo innanzitutto un approccio generico al problema, estremamente inefficiente per valori grandi di k.
- Proponiamo poi soluzioni efficienti "ad-hoc" per $k = 2$ e $k = 3$, "propedeutiche" alla costruzione di una soluzione generale.
- Infine, vediamo una soluzione generica che può essere utilizzata per qualunque valore di k, basata su programmazione dinamica

Costo partizione di un vettore (Versione $\Theta(n^k)$)

int partitionK(**int**[ ] V, **int** n)

int $minSoFar = +\infty$
for $i_1 = 1$ **to** $n - (k-1)$ **do**
  $tot_1 = \mathsf{sum}(V, 1, i_1)$
  for $i_2 = i_1 + 1$ **to** $n - (k-2)$ **do**
    $tot_2 = \mathsf{sum}(V, i_1 + 1, i_2)$
    for $i_3 = i_2 + 1$ **to** $n - (k-3)$ **do**
      $tot_3 = \mathsf{sum}(V, i_2 + 1, i_3)$
      for $\ldots = \ldots + 1$ **to** $n - (\ldots)$ **do**
        $\ldots$
        for $i_{k-1} = i_{k-2} + 1$ **to** $n - 1$ **do**
          $tot_k = \mathsf{sum}(V, i_{k-1} + 1, n)$
          $minSoFar = \mathsf{min}(minSoFar, \mathsf{max}(tot_1, \ldots, tot_k))$

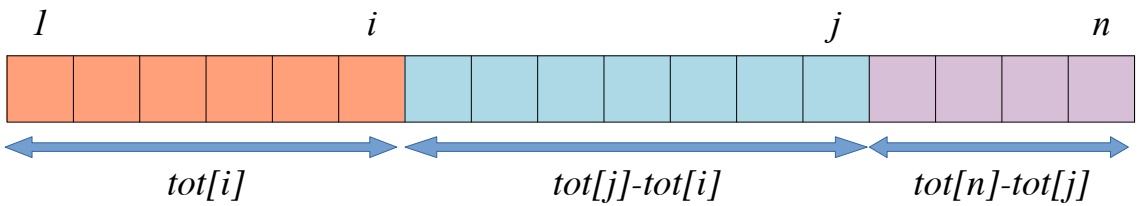
2-partizione

int partition2(**int**[ ] V, **int** n)

int $tot = 0$
for $i = 1$ **to** n **do**
  $tot = tot + V[i]$
int $sumSoFar = 0$
int $minSoFar = +\infty$
for $i = 1$ **to** $n - 1$ **do**
  $sumSoFar = sumSoFar + V[i]$
  $minSoFar = \min(minSoFar, \max(sumSoFar, tot - sumSoFar))$
return $minSoFar$

3-partizione

```
int partition3(int[] V, int n)
int[] tot = new int[1...n]
tot[1] = V[1]
for i = 2 to n do
    tot[i] = tot[i − 1] + V[i]
int minSoFar = +∞
for i = 1 to n − 2 do
    for j = i + 1 to n − 1 do
        int temp = max(tot[i], tot[j] − tot[i], tot[n] − tot[j])
        minSoFar = min(minSoFar, temp)
return minSoFar
```



k-partizione

Sia $DP[i][t]$ il minimo costo associato al sottoproblema di trovare la migliore t-partizione nel vettore $V[1 \dots i]$. Il problema iniziale corrisponde a $DP[n][k]$ – ovvero trovare la migliore k-partizione in $V[1 \dots n]$. Sfruttiamo un vettore di appoggio tot definito come nel caso $k = 3$.

$$DP[i][t] = \begin{cases} +\infty & t > i \\ tot[i] & t = 1 \\ \min_{1 \le j < i} \max(DP[j][t-1], tot[i] - tot[j]) & \text{altrimenti} \end{cases}$$

k-partizione

int partition(**int**[ ] V, **int** n, **int** k)

int[ ][ ] DP = **new int**$[1 \ldots n][1 \ldots k] = \{-1\}$ % Initizialized to -1
int[ ] tot = **new int**$[1 \ldots n]$
$tot[1] = V[1]$
for $i = 2$ **to** n **do**
  $tot[i] = tot[i-1] + V[i]$
return partitionRec(V, tot, DP, n, k)

k-partizione

```
int partitionRec(int[] V, int[] tot, int[][] DP, int i, int t)
if t > i then
    return +∞
else if t == 1 then
    return tot[i]
else if DP[i][t] < 0 then
    int DP[i][t] = +∞
    for j = 1 to i − 1 do
        int cost = max(partitionRec(V, tot, DP, j, t − 1), tot[i] − tot[j])
        DP[i][t] = min(DP[i][t], cost)
return DP[i][t]
```

k-partizione

Costo computazionale:

- La tabella ha dimensione $n \times k$
- Ogni cella richiede tempo $O(n)$ per essere riempita
- Il costo computazionale è quindi $O(kn^2)$