Problem set 3
Esercizio 3

Alessandro Straziota

Esercizio 3. Stiamo pianificando un viaggio dalla località A alla località B. Idealmente ci piacerebbe fare delle tappe da 200 km al giorno, ma questo potrebbe non essere possibile perché i punti dove possiamo sostare per la notte sono solo nelle posizioni $a_1 < a_2 < \cdots < a_n$, dove $a_i \in \mathbb{N}$ indica la distanza in km dell'i-esimo punto di sosta dalla località di origine A.

Per esempio, supponiamo che B è a 600 km da A. Se $a_1 = 100$, $a_2 = 200$, $a_3 = 400$ allora possiamo sostare in a_2 e a_3 e fare esattamente tre tappe da 200 km al giorno. Ma se $a_1 = 150$, $a_2 = 300$, $a_3 = 450$ allora non possiamo fare tappe da 200 km al giorno e dobbiamo scegliere fra varie opzioni: per esempio, potremmo (1) fare due soste, in a_1 e a_3, oppure (2) fare tutte e tre le soste in a_1, a_2 e a_3, o una delle altre opzioni. Per quantificare il nostro grado di soddisfazione rispetto alle varie opzioni supponiamo di usare questa misura: se in un giorno facciamo x km, allora paghiamo un *costo* pari a $(x - 200)^2$. La soluzione (1) quindi, in cui faremmo tre tappe, di cui due da 150 km e una da 300 km, avrebbe un costo totale di $2(150 - 200)^2 + (300 - 200)^2 = 15$mila; la soluzione (2) invece, in cui faremmo quattro tappe da 150 km l'una, avrebbe un costo totale di $4(150 - 200)^2 = 10$mila.

Progettare un algoritmo efficiente che prende in input i numeri $a_1, a_2, \ldots, a_n$ (per semplicità, supponiamo che l'ultimo punto di sosta a_n è la destinazione B) e restituisce i punti di sosta in cui fermarsi per minimizzare il costo totale.

L'algoritmo progettato in questione è un algoritmo di *programmazione dinamica*.
Innanzitutto data una istanza $a_1 < a_2 < ... < a_n$, dove ogni $a_i \in \mathbb{N}$ rappresenta la distanza di un punto di sosta dal punto iniziale A e a_n rappresenta la distanza tra A e il punto di arrivo B, possiamo dire che una soluzione ammissibile è un insieme $S \subseteq \{1, ..., n\}$ dove ogni $i \in S$ rappresenta il fatto che bisogna sostare nel punto di sosta distante a_i da A.

Data ora una soluzione ammissibile S definiamo una *funzione costo*

$$COST(S) = \sum_{i=1}^{|S|-1} (|a_{i+1} - a_i| - 200)^2$$

A questo punto ciò che si desidera è trovare una soluzione ammissibile tale che *minimizzi* il suo costo associato.

Per trovare una soluzione ottima è necessario calcolare prima il valore di una eventuale soluzione ottima per l'istanza. In ordine, se consideriamo il primo punto distante a_1 possiamo dire che è conveniente sostare in quel posto oppure no. Analizzando i due possibili casi:

- se conviene fermarsi nel punto a_1 allora il costo della soluzione ottima sarà $(a_1 - 200)^2$ più il costo del sottoproblema dove il punto di partenza è il punto con distanza a_1

- se **non** conviene fermarsi nel punto a_1 allora esisterà certamente un punto più conveniente in cui fermarsi, a distanza a_k dalla partenza (con $1 < k \leq n$)

Notiamo quindi che possiamo generalizzare questi due passaggi nella semplice ricerca in un certo $k \leq n$ che minimizzi la quantità $(a_k - 200)^2$ per poi sommarci il valore della soluzione ottima per una istanza che va dal punto a_k in poi.
Più formalmente poniamo il valore $a_0 = 0$ (ovvero la distanza tra A e se stesso) possiamo dire che una soluzione ottima avrà costo

$$OPT(0, n) = \min_{0<k\leq n} \{(|a_k - a_0| - 200)^2 + OPT(k, n)\}$$

In forma più generale avremo che

$$OPT(i, j) = \begin{cases} \min_{i<k\leq j}\{(|a_k - a_i| - 200)^2 + OPT(k, j)\} & \text{se } i < j \\ 0 & \text{se } i = j \end{cases}$$

Una volta trovato il valore dell'ottimo è possibile effettuare una *ricerca* su quali valori possono appartenere a una soluzione S ottima.

Alla luce di quanto osservato, i seguenti pseudocodici descriveranno tutti i metodi per costruire la soluzione ottima al problema

Algorithm 1: OPT

Data: $a_0, a_1, ..., a_n$ tali che $a_i \in \mathbb{N}$, $a_0 < a_1 < ... < a_n$ e $a_0 = 0$

Result: Il valore dell'ottimo $OPT(0, n) = \min COST(S)$

begin

- $n' \leftarrow n + 1$;
- Sia C la matrice $n' \times n'$;
- **for** $1 \leq i \leq n'$ **do**
  - $C_{i,i} \leftarrow 0$;
- **for** $2 \leq s \leq n'$ **do**
  - **for** $1 \leq i \leq n' - s$ **do**
    - $j \leftarrow i + s$;
    - $C_{i,j} \leftarrow \min_{i<k\leq j}\{(|a_k - a_i| - 200)^2 + C_{k,j}\}$;
- **return** C;

Algorithm 2: ALG

Data: $a_0, a_1, ..., a_n$ tali che $a_i \in \mathbb{N}$, $a_0 < a_1 < ... < a_n$ e $a_0 = 0$

Result: La sequenza ottimale $a'_1, ..., a'_k$ che indica dove effettuare le soste, con $k \leq n$

begin

- `GLOBAL` $C \leftarrow OPT(a_0, a_1, ..., a_n)$;
- **return** $proceduraRicorsiva(0,\ n)$;

Algorithm 3: proceduraRicorsiva

Data: due indici $i \leq j$

Result: La sequenza ottimale $a'_1, ..., a'_k$ che indica dove effettuare le soste

begin

  if $i = j$ **then**

    return *"STOP"*;

  $k \leftarrow i + 1$;

  while $C_{i,j} \neq C_{i,k} + C_{k,j}$ **do**

    $k \leftarrow k + 1$;

  return a_k, *proceduraRicorsiva*$(k,\ j)$;

Analisi del costo di OPT L'algoritmo OPT sostanzialmente costruisce una matrice grande $n \times n$. In effetti vengono considerate e calcolate solamente metà delle caselle della matrice, però dato che $\frac{n^2}{2} \in O(n^2)$ possiamo affermare che bisogna calcolare un numero di caselle dell'ordine di $O(n^2)$.
C'è però da osservare che il costo per il calcolo di una singola casella non è costante, bensì lineare in n. Infatti, per ognuna di queste caselle, bisogna calcolare e ricercare il minimo tra una serie di k valori, dove però k è al più pari a n.
Quindi possiamo concludere che l'algoritmo OPT ha complessità $O(n^3)$.

Analisi complessiva di ALG L'algoritmo ALG alla fine esegue una chiamata all'algoritmo OPT (che sappiamo avere complessità $O(n^3)$) e una alla procedura ausiliaria *proceduraRicorsiva*.
Questa procedura ausiliaria a sua volta esegue al suo interno un ciclo **while** più una chiamata ricorsiva, finché non si arriverà a un caso base in cui avrà in input una istanza di un solo elemento. È possibile dimostrare il costo complessivo di tutte le chiamate ricorsive è $O(n)$.
Consideriamo la seguente equazione di ricorrenza

$$
\begin{aligned}
T(n) &= T(n-k_1) + O(k_1) \\
&= T(n-k_1-k_2) + O(k_1+k_2) \\
&\vdots \\
&= T(n - \sum_i k_i) + O\Big(\sum_i k_i\Big) \qquad (1) \\
&\vdots \\
&= T(1) + O(n) = O(n) \qquad (per \sum_i k_i = n-1)
\end{aligned}
$$

Conclusione: l'algoritmo ALG ha complessità $O(n^3)$.

Implementazione In seguito l'implementazione in linguaggio Python dell'algoritmo *ALG* e di tutte le rispettive procedure utili al calcolo della soluzione ottima

```python
def ALG(a):

    a.append(0)
    a.sort()

    C = OPT(a)

    return ["START"] + procedura_ricorsiva(a, C, 0, len(a)-1)

def OPT(a):

    n = len(a)

    C = [ [float("inf")]*n for _ in range(n) ]

    for i in range(n):
        C[i][i] = 0

    for s in range(1,n):
        for i in range(n-s):
            j = i + s
            '''i < k <= j'''
            ks = list(range(i+1,j+1))

            '''OPT(i,j) = min{ (abs(a_i - a_k) - 200)^2 + OPT(k,j) }'''
            C[i][j] = min(list(map(lambda k:  (abs(a[i] - a[k]) - 200)**2 + C[k][j], ks)))

    return C

def procedura_ricorsiva(a, C, i, j):

    if i == j:
        return ["STOP"]

    k = i+1
    while C[i][j] != C[i][k] + C[k][j]:
        k += 1

    return [a[k]] + procedura_ricorsiva(a, C, k, j)

print( ALG([0,0,0,0,150,300,450,600]) )
```