

# CoderFarm - Corso base

## Lezione 7

Carlo Collodel, Francesco Cerroni

15 dicembre 2022

# Esercizi delle OIS!

f1

## Testo del problema

Vengono dati i tempi di completamento di *Hamilton* e *Verstappen* per ogni giro di una gara che conta complessivamente  $N$  giri. Calcolare **chi ha vinto la gara** (è garantito che non ci siano pareggi) e **chi ha completato un giro nel minor tempo possibile** (è garantito che esista un tempo minore di completamento di un giro).

Questo problema è semplicemente di implementazione:

```
#include <iostream>

using namespace std;
const int INF = 1e9; /* uso un numero grande per inizializzare il min */

int main() {
    int n;
    cin >> n;

    int somma_h = 0, somma_v = 0, min_h = INF, min_v = INF;
    /* leggo i tempi di Hamilton */
    for (int i = 0; i < n; ++i) {
        int x; cin >> x;
        somma_h += x;
        min_h = min(min_h, x);
    }
    /* leggo i tempi di Verstappen */
    for (int i = 0; i < n; ++i) {
        int x; cin >> x;
        somma_v += x;
        min_v = min(min_v, x);
    }

    cout << (somma_h < somma_v ? "Hamilton" : "Verstappen") << endl;
    cout << (min_h < min_v ? "Hamilton" : "Verstappen") << endl;
    return 0;
}
```

# Esercizi delle OIS!

multiplication

## Testo del problema

Dato un numero  $K \leq 10^8$ , trova il numero di multipli consecutivi di  $K$  che è necessario "scrivere su un foglio" in modo da ottenere (nella rappresentazione in base 10) tutte le cifre possibili (0, 1, ..., 9) almeno una volta.

# multiplication

## Osservazione importante

Quando sommo un generico numero  $Y$  a un generico numero  $X$ , se i due addendi hanno lo stesso numero di cifre, ottengo un riporto di al più 1 sulla cifra più significativa (quella più a sinistra).

### Esempio

Se volessi sommare 8 e 4, otterrei 12 (ho un riporto).

Se volessi sommare 19919 e 77777, otterrei 97696 (no riporto).

Se volessi sommare 99999 con 99999, otterrei 199998 (c'è il riporto).

# multiplication

## Soluzione

In al più 10 multipli di  $K$  avrò almeno una cifra di riporto (otterrò sicuramente la cifra 1).

Ripeto questo ragionamento 9 volte e sono sicuro di avere tutte le cifre da 1 a 9.

E lo 0? Questo caso è garantito entro i primi 10 multipli di  $K$ , la dimostrazione è lasciata come esercizio per voi!

La soluzione è quindi semplicemente continuare a costruire multipli di  $K$  finché non otteniamo tutte le cifre desiderate, visto che abbiamo dimostrato che il numero di operazioni necessarie è molto piccolo.

# Esercizi delle OIS!

lights

## Testo del problema

Ci sono  $N$  luci disposte in una riga, con colori da 0 a  $C - 1$ , è garantito che c'è almeno una luce per ogni colore distinto.

Trova la lunghezza del più piccolo segmento contiguo di luci che contiene almeno una volta tutti i colori da 0 a  $C - 1$ .

# lights

## Soluzione quadratica

Itero per ogni indice  $i, j \in [0, N - 1]$  ( $i \leq j$ ) e controllo che l'intervallo  $[i, j]$  contenga almeno una volta tutti i colori diversi.

- ▶  $i$  può assumere tutti i valori in  $[0, N - 1]$ , complessità  $\mathcal{O}(N)$
- ▶  $j$  può assumere tutti i valori in  $[i, N - 1]$ , complessità  $\mathcal{O}(N)$
- ▶ Complessità totale:  $\mathcal{O}(N^2)$



# lights

Soluzione  $N \log N$

Uso la ricerca binaria sulla **lunghezza dell'intervallo da considerare**: se esiste un intervallo di lunghezza  $K \neq N$  tale che è presente almeno una luce per ogni colore distinto, allora sicuramente ne esisterà uno lungo  $K + 1$  e così via...

- ▶ Ricerca binaria su  $N$  lunghezze possibili, complessità  $\mathcal{O}(\log N)$
- ▶ Gli intervalli di lunghezza  $K$  sono al più  $N$ , complessità  $\mathcal{O}(N)$
- ▶ Complessità totale:  $\mathcal{O}(N \log N)$

Questa soluzione basterebbe per i *constraints* del problema.

# lights

Osservazione aggiuntiva

Se fisso  $j$  e considero tutti gli intervalli che terminano in  $j$ , posso notare che tra questi **esiste un solo intervallo con lunghezza minima**.

Trovare questo intervallo (partendo da un intervallo valido, ma non ottimale) è semplice, basta "avanzare" con l'indice sinistro  $i$  finchè nell'intervallo ci sono ancora tutti i colori distinti di luci.

# lights

## Altra osservazione

Se ho un intervallo ottimale  $O_1 = [i_1, j_1]$  (che quindi finisce in  $j_1$ ), dato un altro intervallo ottimale  $O_2 = [i_2, j_2]$  con  $(j_1 < j_2)$  avrò sicuramente  $i_1 \leq j_1$ .

Basti pensare al fatto che "stiamo aggiungendo" elementi a destra per passare da  $O_1$  a  $O_2$ , quindi per ottenere un intervallo valido con lunghezza minima vorrò solo rimuovere elementi a sinistra quando possibile!

# lights

## Two pointers

Questa tipologia di algoritmi è detta **Two Pointers**, consiste nel tenere due indici  $i$  e  $j$  (con  $i \leq j$ ) e far avanzare (e solo avanzare!)  $i$  e  $j$  in un determinato modo, considerando ad ogni cambio di indice l'intervallo  $[i, j]$ .

### Complessità

A prima vista sembra che gli algoritmi di questo tipo siano quadratici di complessità sugli indici, tuttavia  $i, j$  possono solo **aumentare!**

Dato che  $i, j < N$ , la complessità è lineare negli intervalli, perché ogni indice viene incrementato al massimo  $N$  volte.

Posso mantenere un *array delle frequenze* per ogni colore di luce (e il numero di colori distinti nel mio intervallo) e a ogni cambio di indice aggiornarlo (aggiornarli) in  $\mathcal{O}(1)$ .

Complessità finale:  $\mathcal{O}(N)$

# Esercizi delle OIS!

scoazze

## Testo del problema

Vengono dati  $N$  bidoni della spazzatura con le loro capacità  $C_i$ . Per  $K$  giorni ci sono delle *query* della forma  $(T_j, Q_j)$  dove  $Q_j$  sacchi di spazzatura vengono aggiunti al bidone con indice  $T_j$ . Ogni notte è possibile svuotare un intervallo contiguo di bidoni della spazzatura, pagando la somma dello spazio non usato dei bidoni da liberare.

Trova il costo minimo tale che i bidoni non sforino mai la capacità e che dopo l'ultimo giorno, tutti i bidoni siano vuoti.

Il problema sembra difficile a prima vista, e i vincoli sui numeri ( $N, K \leq 200'000$ ) non aiutano!

- ▶ Osserva il syllabus (il numero di libri arancioni): per questo problema è solo 1, non servono tecniche "avanzate" per risolverlo.
- ▶ Considera i subtask e prova a risolverli: i subtask sono spesso costruiti per aiutare ad andare verso alla soluzione finale, per questo problema soffermatevi in particolare sui **Subtask 3 e 4**.
- ▶ Prova a notare pattern o cercare soluzioni alternative negli esempi del problema!

Ogni "tipologia" di spazzatura è si trova almeno una volta nelle query

Se un bidone viene svuotato perché alla successiva query che lo coinvolge supererebbe la capacità, allora il bidone viene prima svuotato, poi verrebbe aggiunta la quantità che avrebbe fatto sfiorare il bidone non vuoto.

Questo implica che:

- ▶ Se un bidone viene svuotato per non superare la capacità almeno una volta, allora all'ultimo giorno il bidone sarà sicuramente non vuoto.
- ▶ Se un bidone viene riempito almeno una volta (ma non sfiorerebbe mai), allora all'ultimo giorno il bidone sarà sicuramente non vuoto.

Queste osservazioni e i limiti del Subtask 3 ci permettono di elaborare una strategia:

- ▶ Se alla  $j$ -esima query il bidone  $T_j$  sforerebbe, svuoto **solo quel bidone** il giorno prima.
- ▶ All'ultimo giorno, dopo tutte le query, svuoto **tutti i bidoni con un'unica query**, le condizioni di prima mi assicurano che all'ultimo giorno, con questa strategia, pagherò il prezzo ottimale.



A ogni bidone  $i$  non vengono aggiunta più di  $C_i$  sacchi di spazzatura

La condizione equivalente è: "non sono mai costretto a svuotare un bidone perché supererebbe la capacità massima".

Per me è ottimale svuotare ogni bidone **il giorno dopo dell'ultima query che riguarda quel bidone** (e svuotare solo quel bidone).

Proviamo ad unire le soluzioni dei Subtask 3 e 4:

- ▶ Se devo svuotare un bidone perché sforerebbe alla prossima query riguardante quel bidone, svuoto **solo quel bidone appena possibile**.
- ▶ Se è passato il giorno che contiene l'ultima query che riguarda un bidone, allora svuoterò **solo quel bidone** appena avrò tempo.

# Greedy

Che cos'è?

La soluzione di *scoazze* è un esempio degli algoritmi detti **Greedy** (in inglese: "avido").

Gli algoritmi Greedy sono particolari perché le scelte ottimali sono sempre quelle **locali** (la scelta che faccio ora non influenza la scelta che avrei fatto dopo).

Vediamo un esempio:

## Scheduling di eventi

Sono dati  $N$  eventi, ognuno con un orario di inizio  $A_i$  e un orario di fine  $B_i$ . L'obiettivo è partecipare al massimo numero di eventi con la limitazione che 2 eventi sono compatibili se e solo se il primo ha un orario di fine strettamente minore dell'orario di inizio del successivo (non posso partecipare a due eventi diversi allo stesso tempo).

# Greedy

## Scheduling di eventi

Leggendo un problema come questo ci sono 3 possibili soluzioni (che comprendono un "ottimo" locale) che possono venire in mente:

- ▶ Processare gli eventi in base all'orario di inizio e scegliere ad ogni step il più breve
- ▶ Processare gli eventi in base all'orario di inizio e scegliere ad ogni step quello che inizia prima
- ▶ Processare gli eventi in base all'orario di fine e scegliere ad ogni step quello che termina prima

Le prime 2 soluzioni sono **sbagliate** e per dimostrarlo basta cercare qualche controesempio (esercizio per voi!). La terza invece è corretta.

# Greedy


## Scheduling di eventi

### Dimostrazione della correttezza della soluzione

Iniziamo l'algoritmo e cerchiamo l'evento che finisce prima. Scegliendo questo evento come il primo a cui partecipare implica che alcuni eventi (potenzialmente nessuno) vengono esclusi, ma se avessimo scelto un evento che termina dopo il numero di eventi ancora disponibili non sarebbe aumentato. A questo punto scartiamo tutti gli eventi non più disponibili e ripetiamo lo stesso ragionamento con quelli rimanenti.

# Greedy

## Scheduling



```
int main() {
    int N;
    vector<int> A(N), B(N);

    //lettura dell'input e ordinamento in base alla fine

    int ris = 1; // numero massimo di eventi a cui partecipare
    int limite = B[0]; // orario di fine dell'ultimo evento scelto
    for (int i = 1; i < N; i++) {
        if (A[i] > limite) { // controlla se l'evento i è valido
            ris++; // aumento degli eventi selezionati
            limite = B[i]; // aggiornamento dell'orario di fine
        }
    }

    cout << ris << "\n"; // output del risultato
}
```

# Greedy

## Problema del resto

### Problema

Devi raggiungere una certa somma di denaro  $S$  (espressa in centesimi) e hai a disposizione tutte le taglie di monete:  $\{1, 2, 5, 10, 20, 50, 100, 200\}$ . Qual è il minimo numero di monete necessarie?

# Greedy

## Problema del resto

La soluzione in questo caso è molto semplice: scelgo ad ogni step la più grande taglia di moneta non maggiore della quantità di denaro restante. La correttezza è facile da dimostrare ma è abbastanza macchinoso come processo, se siete interessati potete trovarla sul libro "CP Handbook"<sup>1</sup>.

---

<sup>1</sup><https://cses.fi/book/book.pdf>



# Greedy

## Problema del resto

Supponiamo invece di avere un insieme di taglie generico, la soluzione funziona ancora?

### Controesempio

Sia l'insieme delle taglie  $\{1, 3, 4\}$  e la somma da raggiungere 6. Applicando l'algoritmo otteniamo come soluzione 2 monete da 1 e una da 4, che non è ottimale. La soluzione corretta in questo caso è 2 monete da 3.

Per risolvere il problema nel caso generale bisogna ricorrere alla *programmazione dinamica*.

# Altri esercizi per casa!

- ▶ Coin Change [https://training.olinfo.it/#/task/ois\\_coinchange/statement](https://training.olinfo.it/#/task/ois_coinchange/statement)
- ▶ Missing coing sum <https://cses.fi/problemset/task/2183>
- ▶ Lotteria di quadri  
[https://training.olinfo.it/#/task/abc\\_quadri/statement](https://training.olinfo.it/#/task/abc_quadri/statement)

# Fine

Ci vediamo alla prossima lezione!

- ▶ **E-Mail:** `base@coderfarm.it`
- ▶ **Discord:** T.B.D.