Colautti Michea – Julian Cummaudo

DTi – 2024/2025 SA

Figure panini

Sommario

[Introduzione al problema 2](#_Toc185323951)

[Implementazione algoritmo 2](#_Toc185323952)

[Pseuodocodice 3](#_Toc185323953)

[Punti importanti 4](#_Toc185323954)

[Complessità 5](#_Toc185323955)

[Test 6](#_Toc185323956)

[Problemi riconosciuti 6](#_Toc185323957)

[Conclusioni 7](#_Toc185323958)

# Introduzione al problema

In questo progetto, dato un numero  N  di giorni e un numero  F  di figurine, insieme ai relativi prezzi di acquisto e di vendita per ciascun giorno, il programma dovrebbe calcolare il massimo fatturato potenziale ottenibile.

I dati di input vengono forniti tramite file di testo nel formato seguente:

2 3

5 2 7 2 9 5

7 7 8 6 12 10

Nel caso sopra,  N = 2  (giorni) e  F = 3  (figurine). Ogni riga rappresenta un giorno, mentre ogni coppia di colonne fornisce il prezzo di acquisto e il prezzo di vendita per una specifica figurina in quel giorno. Ad esempio, per il primo giorno, la sequenza “5 2 7 2 9 5” indica:

* Figurina 1: prezzo di acquisto = 5, prezzo di vendita = 2
* Figurina 2: prezzo di acquisto = 7, prezzo di vendita = 2
* Figurina 3: prezzo di acquisto = 9, prezzo di vendita = 5

Inoltre a questo meccanismo ci sono stati imposti alcuni vincoli:

* 1 ≤ N ≤ 3000
* 1 ≤ N ≤ 3000
* Vi ≤ Ai 🡪 Ovvero il prezzo di vendita per un giorno i deve essere minore o uguale al prezzo di acquisto

## Implementazione algoritmo

Prima di implementare il codice vero e proprio abbiamo implementato delle funzioni di supporto, utili alla lettura e all’estrapolazione dei dati. Eccole riassunte brevemente.

//Funzione utile alla lettura del file e all'allocazione dimamica delle matrici

void readFile(const char\* fileName, int\*\* data, int\* size);

//Funzione utile all'inizailizzazione dei valori N e F

void initializeValues(int\* data, int\* days, int\* tradingCards);

//Funzione utile al riempimento della matrice con i giorni e i prezzi di vendita e acquisto

int\*\* createMatrix(int\* data, int days, int cols);

Il resto del codice è stato poi sviluppato interamente nel main.

## Pseuodocodice

Questo è un flowchart base del funzionamento del programma:

Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

La parte importante del programma è però quella che avviene a partire dal punto “allocate dp array”, per questo abbiamo anche scritto un meta-codice, più chiaro.

NB: Nelle istanze fornite, a contrario di quello che dicono le specifiche, la prima colonna indica il prezzo di acquisto, la seconda quello di vendita.

Inizio

// Inizializzazione delle variabili

[...]

// Controllo dei valori

[...]

// Allocazione matrice dinamica

[...]

// Assegnazione dei prezzi di acquisto e vendita

[...]

// Inizializzazione della tabella DP

dp[0][0] ← 1.0 // Inizio con 1 CHF

For ogni carta j da 1 a F fai

dp[0][j] ← 0.0 // Nessuna carta posseduta inizialmente

endFor

// Calcolo della tabella DP

for ogni giorno i da 1 a N fai

// Gestione del denaro liquido nel giorno i

dp[i][0] ← dp[i - 1][0] // Denaro del giorno precedente

if ogni carta j da 1 a F fai

cash\_from\_selling ← dp[i - 1][j] \* sell\_price[i - 1][j - 1]

if cash\_from\_selling > dp[i][0] allora

dp[i][0] ← cash\_from\_selling

endIf

endIf

// Gestione delle carte possedute nel giorno i

for ogni carta j da 1 a F fai

dp[i][j] ← dp[i - 1][j] // Carte del giorno precedente

card\_bought ← dp[i][0] / buy\_price[i - 1][j - 1]

if card\_bought > dp[i][j] allora

dp[i][j] ← card\_bought

endIf

endFor

endFor

// Calcolo del capitale finale

Stampa(dp[N][0] con due decimali)

Fine

## Punti importanti

Il punto più importante, che vale la pena esplorare, è il calcolo della tabella dp. Possiamo dividere questo processo in 5 punti, tendendo conto che l’obbiettivo finale è determinare il capitale massimo.

1. **Struttura DP:**
   1. dp[i][0]: Denaro liquido al giorno i.
   2. dp[i][j]: Numero massimo di carte j possedute al giorno i.
2. **Aggiornamento del Denaro Liquido (**dp[i][0]**):**
   1. Inizia con il denaro del giorno precedente: dp[i][0] = dp[i-1][0].
   2. Considera la vendita di ogni tipo di carta posseduta il giorno precedente:
   3. Calcola cash\_from\_selling = dp[i-1][j] \* sell\_price[i-1][j-1].
   4. Se cash\_from\_selling aumenta dp[i][0], aggiorna il denaro liquido.
3. **Aggiornamento delle Carte Possedute (**dp[i][j]**):**
   1. Mantieni il numero di carte del giorno precedente: dp[i][j] = dp[i-1][j].
   2. Valuta l’acquisto di nuove carte con il denaro disponibile:
   3. Calcola card\_bought = dp[i][0] / buy\_price[i-1][j-1].
   4. Se card\_bought supera dp[i][j], aggiorna il numero di carte possedute.
4. **Iterazione Giornaliera:**
   1. Ripeti i passaggi sopra per ogni giorno, aggiornando la tabella DP giorno per giorno
5. **Risultato Finale:**
   1. Il capitale finale è dp[N][0], che rappresenta il denaro liquido al termine dell’ultimo giorno.

## Complessità

Dalla nostra analisi riuslta che l’algoruitmo ha una complessita temporale di **O(N × F)**, dove:

* **N** è il numero di giorni.
* **F** è il numero di tipi di carte da collezione.

Questo perché le iterazioni prinicpali richiedono entrambi N×F, questo perché si scorre ogni giorno e ogni carta.

# Test

Abbiamo testato il codice con tutte le istanze: grazie alla modularità del nostro codice è bastato cambiare una linea nel main, contente il nome del file, con il nome dell’istanza che volevamo testare.

const char\* fileName = "../instances/instance\_1\_10.txt";

Di seguito una tabella di alcune tempistiche, in alcuni casi simbolici:

Immagine che contiene testo, Carattere, ricevuta, schermata

Descrizione generata automaticamente

Per registrare le tempistiche abbiamo compreso tutto l’algoritmo, a partire dall’inizializzazione delle matrici.

I test riportati sono stati eseguiti su MacBook Pro M2 Max, 32 GB RAM, 1TB SSD.

## Problemi riconosciuti

Abbiamo riscontrato un piccolo problema con l’istanza “3000\_1”, il nostro algoritmo restituisce il valore **12752006544119.77**, mentre stando alle istanze l’output dovrebbe essere **12752006544119.75.**

C’è quindi un errore di **0.2.** È probabilmente dovuto ai calcoli fatti con l’aritmetica floating-point.

# Conclusioni

Grazie a questo progetto abbiamo potuto applicare, forse per la prima volta, un approccio strettamente legato alla struttura dell’algoritmo. In passato spesso il focus dei progetti era il risultato, mentre in questo caso abbiamo percepito che la maggior parte del lavoro era produrre un buon algoritmo. Nel nostro caso abbiamo optato per un approccio Bottom-Up, ma siamo sicuri che si poteva risolvere anche in altri modi.

Inoltre è stato stimolante applicare i concetti di progettazione appresi ed essere più consapevoli del codice scritto.

Siamo piuttosto soddisfatti del nostro lavoro e pensiamo di aver raggiunto l’obbiettivo finale.