




FIGURE PANINI



Colautti Michea – Julian Cummaudo

DTI – 2024/2025 SA

Sommario

Introduzione al problema	2
Implementazione algoritmo	2
Pseudocodice	Errore. Il segnalibro non è definito.
Punti importanti	5
Complessità	5
Test	6
Problemi riconosciuti	6
Conclusioni	7

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 deve 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 sono stati imposti alcuni vincoli:

- $1 \leq N \leq 3000$
- $1 \leq F \leq 3000$
- $V_i \leq A_i \rightarrow$ 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, sono state implementate delle funzioni di supporto, utili alla lettura e all'estrapolazione dei dati. Eccole riassunte brevemente.

```
//Funzione utile alla lettura del file e all'allocazione dinamica delle matrici
void readFile(const char* fileName, int** data, int* size);

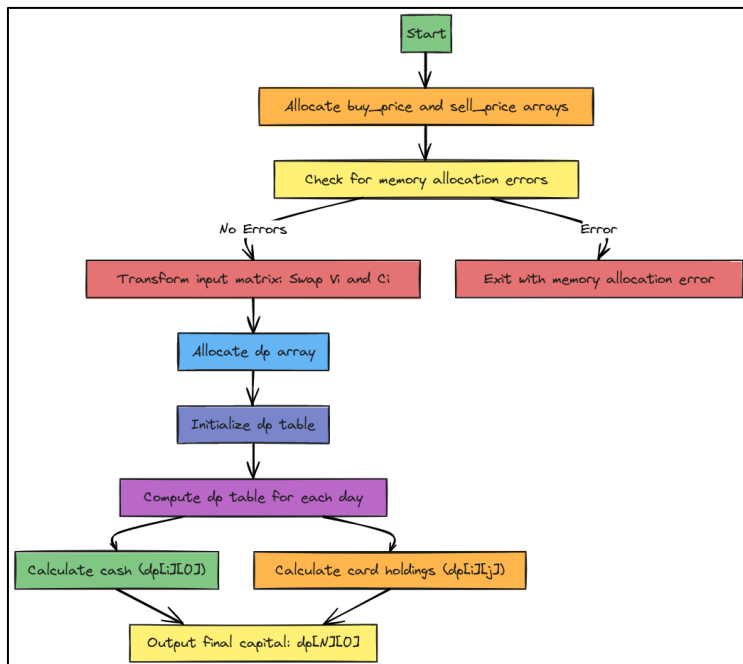
//Funzione utile all'inizializzazione dei valori N e F
void initializeValues(int* data, int* days, int* tradingCards);

//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.

Pseudocodice

Questo è un flowchart base del funzionamento del programma:



La parte importante del programma è però quella che avviene a partire dal punto “allocate dp array”, nella pagina seguente è quindi presente il meta-codice del set di istruzioni “Compute dp table for each day”.

Un fatto da tenere a mente leggendo il codice, è che nelle istanze fornite, a contrario di quello che dicono le specifiche, la prima colonna indica il prezzo di acquisto, la seconda invece 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

```

Una volta inizializzata la matrice “dp” è possibile quindi svolgere il calcolo del miglior profitto, in due for innestati dentro un terzo.

```

// 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. È possibile dividere questo processo in 5 punti, tenendo conto che l'obiettivo finale resta quello di determinare il capitale massimo.

1. **Struttura DP:**
 - a. $dp[i][0]$: Denaro liquido al giorno i .
 - b. $dp[i][j]$: Numero di carte j possedute al giorno i , per questo esercizio, valgono anche valori con la virgola.
2. **Aggiornamento del Denaro Liquido ($dp[i][0]$):**
 - a. Iniziare con il denaro del giorno precedente: $dp[i][0] = dp[i-1][0]$.
 - b. Considerare la vendita di ogni tipo di carta posseduta il giorno precedente:
 - c. Calcolare $cash_from_selling = dp[i-1][j] * sell_price[i-1][j-1]$.
 - d. Se $cash_from_selling$ aumenta $dp[i][0]$, aggiornare il denaro liquido.
3. **Aggiornamento delle Carte Possedute ($dp[i][j]$):**
 - a. Mantenere il numero di carte del giorno precedente: $dp[i][j] = dp[i-1][j]$.
 - b. Valutare l'acquisto di nuove carte con il denaro disponibile:
 - c. Calcolare $card_bought = dp[i][0] / buy_price[i-1][j-1]$.
 - d. Se $card_bought$ supera $dp[i][j]$, aggiornare il numero di carte possedute.
4. **Iterazione Giornaliera:**
 - a. Ripetere i passaggi sopra per ogni giorno, aggiornando la tabella DP di conseguenza
5. **Risultato Finale:**
 - a. Il capitale finale è $dp[N][0]$, che rappresenta il denaro liquido al termine dell'ultimo giorno.

Complessità

Dalle analisi condotte risulta che l'algoritmo ha una complessità temporale di $O(N \times F)$, dove:

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

Dal codice si nota il ciclo for più esterno, che si ripete N volte, e al suo interno, in successione, due cicli for che invece si ripetono F volte. La complessità risulta quindi $O(N*2F)$, equivalente e più corretto dire $O(N*F)$.

Test

È stato testato il codice con tutte le istanze: grazie alla modularità del codice è bastato cambiare una stringa per ogni istanza che si è desiderato testare

```
const char* fileName = "../instances/instance_1_10.txt";
```

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

Istanza	Tempo [ms]	Motivazione
10_1	0.001	Caso base
10_100	0.018	Caso con F maggiore
100_100	0.131	Caso con N e F maggiori
3000_1	0.125	Caso massimo

Per registrare le tempistiche è stato 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

È stato riscontrato un piccolo problema con l'istanza "3000_1", l'algoritmo restituisce il valore **12752006544119.77**, mentre stando alle istanze l'output dovrebbe essere **12752006544119.75**.

È quindi presente un errore di **0.2**, probabilmente dovuto ai calcoli fatti con l'aritmetica floating-point. In nessun'altra istanza è stato notato un errore per cui abbiamo deciso di non indagare oltre, per un errore dello 0.000000000000001568% presente in una sola istanza.

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'obiettivo finale.