

Esercizi svolti di Progettazione di algoritmi
Facoltà di Ingegneria dell'informazione, informatica e statistica
Dipartimento di informatica
Anno Accademico 2022-2023

Esercitazione 8



SAPIENZA
UNIVERSITÀ DI ROMA



Es 1. Calcolo dell'ottimo momento di acquisto/vendita

Testo

In una sequenza $S = (a_1, a_2 \dots, a_n)$ di interi positivi, l'intero a_i rappresenta il prezzo di una certa merce fra i giorni, si vuole sapere qual'è il giorno i in cui conviene comprare la merce ed il giorno j , con $j \geq i$, in cui conviene rivenderla in modo da massimizzare il profitto. In altre parole siamo interessati a conoscere la coppia (i, j) con $i \leq j$ per cui risulta massimo il valore $a_j - a_i$. Descrivere un algoritmo che risolve il problema in $O(n)$ tempo.



Es 1. Calcolo dell'ottimo momento di acquisto/vendita

Idea

Ecco un algoritmo che risolve il problema in tempo lineare $O(n)$:

1. Inizializza le variabili `max_profit` a 0, `buy_day` e `sell_day` a 1.
2. Per ogni giorno `current_day` da 2 a `n`:
 - a. Se il prezzo della merce in `current_day` è inferiore al prezzo della merce nel giorno `buy_day`, imposta `buy_day` su `current_day`.
 - b. Altrimenti, calcola il profitto potenziale (`potential_profit`) come la differenza tra il prezzo della merce in `current_day` e il prezzo della merce nel giorno `buy_day`. Se `potential_profit` è maggiore di `max_profit` e `current_day` è maggiore o uguale a `buy_day`, imposta `max_profit` su `potential_profit` e `sell_day` su `current_day`.
3. Restituisci la coppia (`buy_day`, `sell_day`) come risultato.



Es 1. Calcolo dell'ottimo momento di acquisto/vendita

Soluzione

```
def find_best_days(prices):  
    n = len(prices)  
    max_profit = 0  
    buy_day = 0  
    sell_day = 0  
    for current_day in range(n):  
        if prices[current_day] < prices[buy_day]:  
            buy_day = current_day  
            sell_day = current_day  
        elif prices[current_day] - prices[buy_day] > max_profit:  
            max_profit = prices[current_day] - prices[buy_day]  
            sell_day = current_day  
    return (buy_day, sell_day)
```



Es 1. Calcolo dell'ottimo momento di acquisto/vendita

Esecuzione

```
buy_day, sell_day = find_best_days([7, 1, 5, 3, 6, 4])  
print("Giorno di acquisto: {}; Giorno di vendita: {}".format(buy_day, sell_day))  
-> Giorno di acquisto: 1; Giorno di vendita: 4  
buy_day, sell_day = find_best_days([7, 6, 4, 3, 1])  
print("Giorno di acquisto: {}; Giorno di vendita: {}".format(buy_day, sell_day))  
-> Giorno di acquisto: 4; Giorno di vendita: 4  
buy_day, sell_day = find_best_days([3, 1, 2, 5, 4, 7])  
print("Giorno di acquisto: {}; Giorno di vendita: {}".format(buy_day, sell_day))  
-> Giorno di acquisto: 1; Giorno di vendita: 5
```



Es 2. Massima sotto-matrice di tutti 1 in una matrice di 0 e 1

Testo

Data una matrice quadrata binaria M di dimensione $n \times n$ si vuole sapere qual'è il massimo m per cui la matrice quadrata $m \times m$ di soli uni risulta sottomatrice di M .
Descrivere un algoritmo che, data la matrice M , risolve il problema in tempo $O(n^3)$.



Es 2. Massima sotto-matrice di tutti 1 in una matrice di 0 e 1

Soluzione

```
def matriceUni(M):  
    m = len(M)  
    U = [[y for y in x] for x in M]  
    ri, rj = -1, -1  
    dmax = 0  
    for i in range(m):  
        for j in range(m):  
            if U[i][j]:  
                dmax, ri, rj = 1, i, j  
    d = 2  
    while d == dmax + 1:  
        for i in range(0, m-d+1):  
            for j in range(0, m-d+1):  
                if U[i][j]>0 and U[i+1][j]>0 and U[i][j+1]>0 and U[i+1][j+1]>0:  
                    dmax, U[i][j], ri, rj = d, d, i, j  
            else:  
                U[i][j] = 0  
        d = d+1  
    return dmax, ri, rj
```



Es 2. Massima sotto-matrice di tutti 1 in una matrice di 0 e 1

Esecuzione

M = [[1,0,1,1,1], [1,1,1,1,1], [1,1,1,0,1], [1,1,1,1,1], [1,1,0,1,1]] print(matriceUni(M)) -> (3, 1, 0)	M = [[0,0,0,0,0], [0,0,1,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]] print(matriceUni(M)) -> (1, 1, 2)	M = [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]] print(matriceUni(M)) -> (0, -1, -1)	M = [[1,1,0,0], [1,1,1,1], [0,1,1,1], [0,1,1,1]] print(matriceUni(M)) -> (3, 1, 1)	M = [[1,1,1,1], [1,1,1,1], [1,1,1,1], [1,1,1,1]] print(matriceUni(M)) -> (4, 0, 0)
---	---	---	---	---