



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dispense del Corso di Laboratorio di Fondamenti di Informatica II e Lab

Massimiliano Corsini, Federico Bolelli

Esercitazione 04: Backtracking

Ultimo aggiornamento: 08/04/2020

Backtracking: Torre di Cartoni

- Esercizio 1 (Torre di Cartoni):

All'interno di un magazzino ci sono n cartoni. Ogni cartone possiede un peso in grammi, un'altezza in centimetri e un limite massimo di peso che può sostenere sopra di sé, anch'esso espresso in grammi. Si definisca la seguente struttura dati nel file `torrecartoni.h`:

```
typedef struct {  
    unsigned p; // Peso  
    unsigned a; // Altezza  
    unsigned l; // Limite  
} cartone;
```

Backtracking: Torre di Cartoni

Nel file `torrecartoni.c` si implementi la definizione della procedura ricorsiva `TorreCartoni`:

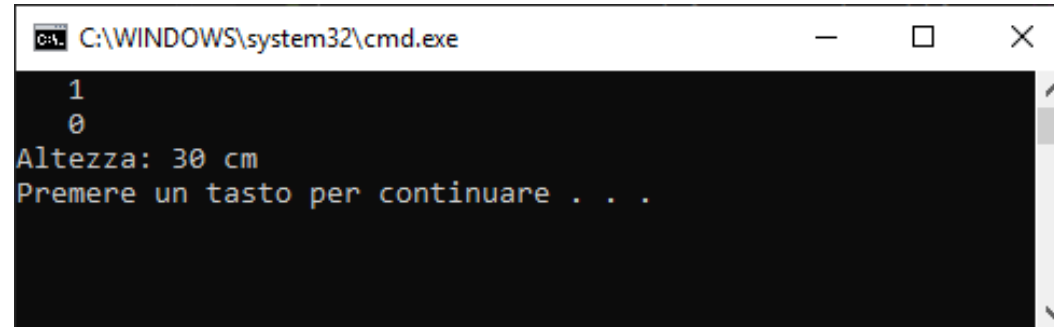
```
void TorreCartoni(cartone *c, int n);
```

Dato un array di n cartoni, la funzione deve individuare la configurazione che massimizzi l'altezza di una pila di cartoni, rispettando il vincolo che nessun cartone abbia sopra di sé un peso superiore al limite consentito.

N.B. è possibile che esistano più soluzioni ottime, se ne consideri solo una. Ovviamente, non è detto che tutti i cartoni del magazzino possano essere impilati.

Backtracking: Torre di Cartoni

- La funzione deve stampare su standard output la soluzione ottima trovata, utilizzando il formato dell'esempio che segue.
- Con $c = \{ \{ .p=10, .a=20, .l=40 \}, \{ .p=10, .a=10, .l=8 \}, \{ .p=9, .a=3, .l=5 \} \}$ l'output dovrà essere il seguente:



```
C:\WINDOWS\system32\cmd.exe
1
0
Altezza: 30 cm
Premere un tasto per continuare . . .
```

- In questo caso la torre ottima ha altezza 30 cm ed è formata da due pacchi: quello di indice 0 alla base e quello di indice 1 in cima.

Backtracking: Torre di Cartoni

- In pratica ogni cartone viene rappresentato da un numero che corrisponde al suo indice nel vettore c . Dall'alto verso il basso, il primo indice rappresenta la testa della torre, l'ultimo la base.
- Si consiglia di utilizzare una funzione ausiliaria per la risoluzione dell'esercizio.
- Ricordatevi di visualizzare lo spazio delle soluzioni prima di procedere con l'implementazione.
- Quante sono le scelte possibili ad ogni passo?
- **Attenzione perché l'ordine con cui posiziono i cartoni nella torre è importante!**
- **Suggerimento**: costruite la torre a partire dalla cima!

Backtracking: Stazioni di Servizio

- Esercizio 2 (Stazioni di Servizio):

Giovanni deve percorrere m chilometri in motocicletta. Prima di partire si segna la posizione delle n stazioni di servizio s_0, s_1, \dots, s_{n-1} presenti lungo il percorso. Tali posizioni sono identificate dalle distanze (in chilometri) d_0, d_1, \dots, d_{n-1} dove d_0 è la distanza dal punto di partenza alla stazione s_0 , e per $i = 1, \dots, n - 1$, d_i è la distanza fra le stazioni s_{i-1} e s_i . Inoltre, per $i = 0, \dots, n - 1$, $p[i]$ indica il prezzo (al litro) del carburante nella stazione s_i . La motocicletta consuma 0.05 litri per chilometro e ha un serbatoio di 30 litri inizialmente pieno. Giovanni decide di riempire totalmente il serbatoio ogni volta che si ferma in una stazione di servizio.

Backtracking: Stazioni di Servizio

Nel file `stazioniservizio.c` si implementi la definizione della procedura ricorsiva `StazioniServizio`:

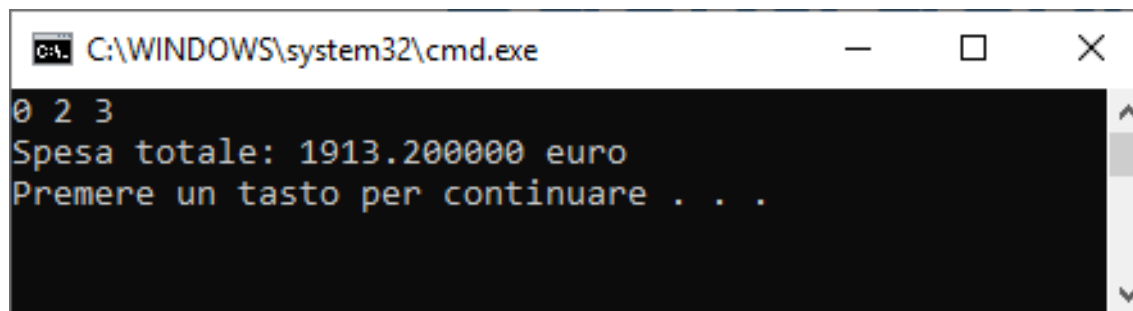
```
void StazioniServizio(double m, int n, double *d, double *p);
```

Dati i km totali da percorrere `m` e gli array delle distanze e dei prezzi `d` e `p`, la funzione deve individuare in quali delle stazioni di servizio Giovanni deve fermarsi per minimizzare la spesa per il carburante pur percorrendo tutti gli `m` km.

Si ignorino i litri di carburante che rimangono nel serbatoio al termine del viaggio.

Backtracking: Stazioni di Servizio

- La funzione deve stampare su standard output la soluzione ottima (se esiste), ovvero la sequenza di stazioni in cui occorre fermarsi per spendere il meno possibile e percorrere gli m km. Il formato dell'output dovrà essere lo stesso dell'esempio che segue. Date le seguenti stazioni:
 - 0: km 260.0000, prezzo 35.0000
 - 1: km 284.0000, prezzo 35.0000
 - 2: km 308.0000, prezzo 33.0000
 - 3: km 332.0000, prezzo 29.0000
 - 4: km 356.0000, prezzo 23.0000
- e dato $m=1540$, la funzione dovrà stampare:



```
C:\WINDOWS\system32\cmd.exe
0 2 3
Spesa totale: 1913.200000 euro
Premere un tasto per continuare . . .
```

- Nel caso in cui il problema non ammetta soluzione visualizzare in output la stringa "Non esistono soluzioni".

Modalità di Consegna

- Per questa esercitazione dovreste consegnare tutti e due gli esercizi utilizzando un sistema di sottomissione online simile a quello che avete usato durante il corso di Fondamenti di Informatica I.
- **Collegatevi al sito <https://aimagelab.ing.unimore.it/OLJ2/esami> e fate il login utilizzando le vostre credenziali shibboleth di UNIMORE.**
- Selezionate *Esercitazione Backtracking II*, aprite il link dell'esercizio di cui volete fare la sottomissione e incollate il codice nei box relativi ai rispettivi file. **Non dovete caricare il `main()`.**
- Assicuratevi tuttavia di scrivere il `main()` in Visual Studio per verificare se quello che avete fatto funziona prima di caricare la soluzione.