



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

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

Esercitazione 03: Backtracking

Ultimo aggiornamento: 20/03/2019

Backtracking: Ombrelloni 1/3

- Esercizio 1 (Ombrelloni):

Alcuni amici (k) trascorrono una giornata al mare. Giunti in spiaggia decidono di affittare un ombrellone ciascuno: tutti vogliono affittarne uno in prima fila, senza però essere vicini tra loro. La prima fila, contenente n ombrelloni, è tutta libera. Nel file `ombrelloni.c` si implementi la definizione della procedura ricorsiva `Ombrelloni`:

```
void Ombrelloni(int k, int n, unsigned s, bool *v,  
               unsigned cnt, unsigned *n_slz)
```

Backtracking: Ombrelloni 2/3

La procedura accetta i seguenti parametri:

- k : il numero di ragazzi da posizionare;
- n : il numero di posti disponibili in prima fila;
- s : la posizione attuale, ovvero a che livello dell'albero di backtrack si trova la funzione corrente;
- v : un array binario che indica lo stato degli ombrelloni in prima fila (ad esempio 1 = occupato, 0 = libero). All'inizio dovranno essere tutti liberi:

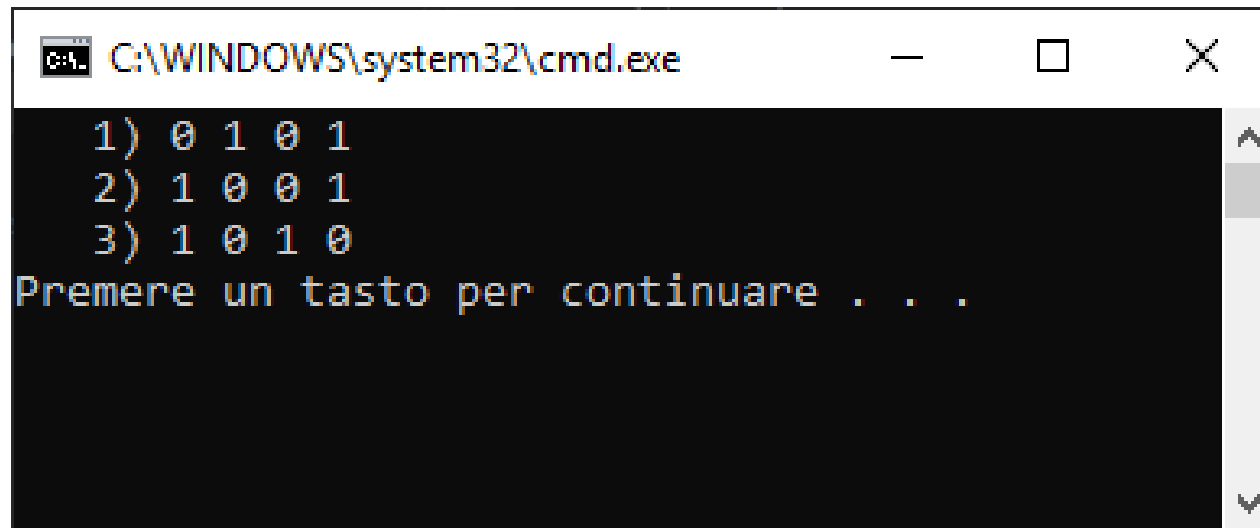
| | | | | | | |
|---|---|---|---|-----|-----|---|
| 0 | 1 | 2 | 3 | | n-1 | n |
| 0 | 0 | 0 | 0 | ... | 0 | 0 |

- cnt : un contatore che memorizza il numero di ragazzi posizionati nella soluzione corrente;
- n_slz : il numero totale di soluzioni trovate ;

N.B. la funzione di backtracking potrebbe trovare la soluzione anche senza utilizzare il parametro cnt .

Backtracking: Ombrelloni 3/3

La procedura ricorsiva deve visualizzare su standard output tutti i possibili modi in cui è possibile posizionare i k ragazzi nella fila di n ombrelloni. Ad esempio, con $k = 2$ e $n = 4$ l'output dovrà essere il seguente:



```
C:\WINDOWS\system32\cmd.exe

1) 0 1 0 1
2) 1 0 0 1
3) 1 0 1 0
Premere un tasto per continuare . . .
```

Backtracking: Babbo Natale 1/4

- Esercizio 2 (Babbo Natale):

Ogni anno che passa, Babbo Natale fatica sempre più a caricare la slitta dei regali che ha una portata massima di p kg. Per aiutarlo, nel file `babbonatale.c` si implementi la definizione della procedura ricorsiva `BabboNatale` che deve individuare tra n regali di peso `pacchi[i]` quali caricare per massimizzarne il numero totale, senza sforare la portata. La procedura deve avere il seguente prototipo:

```
void BabboNatale(int p, int const *pacchi, int n, unsigned s,  
                bool *curr, bool *best, unsigned *max, unsigned cnt,  
                int sum)
```

Backtracking: Babbo Natale 2/4

La procedura accetta i seguenti parametri:

- `p`: portata massima della slitta;
- `pacchi`: array dei pesi dei regali disponibili;
- `n`: dimensione dell'array `pacchi`;
- `s`: la posizione attuale, ovvero a che livello dell'albero di backtrack si trova la funzione corrente;
- `cur`: un array binario che indica i regali attualmente caricati nella soluzione corrente (ad esempio 1 = caricato, 0 = NON caricato);
- ...

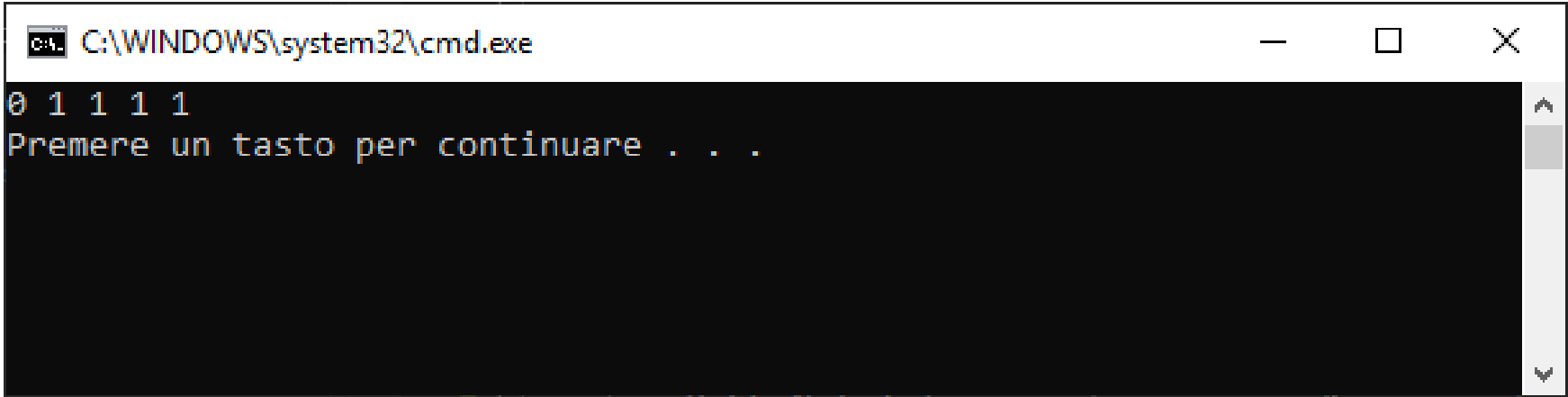
Backtracking: Babbo Natale 3/4

- ...
- **best**: un array binario che indica i regali caricati nella miglior soluzione fino ad ora trovata (ad esempio 1 = caricato, 0 = NON caricato);
- **max**: numero di regali caricati nella soluzione **best**;
- **cnt**: numero di regali caricati nella soluzione **cur**;
- **sum**: somma dei pesi dei regali caricati nella soluzione **cur**;

N.B. la funzione di backtracking potrebbe trovare la soluzione anche senza utilizzare i parametri **max**, **cnt** e **sum**.

Backtracking: Babbo Natale 4/4

Mostrare su standard output la soluzione, ovvero la sequenza di regali che occorre caricare. Ad esempio, con $p = 20$ e $\text{pacchi} = \{ 10, 11, 1, 3, 3 \}$ l'output dovrà essere il seguente:



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