



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

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

Esercitazione 05: Algoritmi Greedy

Ultimo aggiornamento: 03/04/2019

Algoritmi Greedy: Monete 1/3

- Esercizio 1 (Monete):

Nel file `monete.c` si implementi la definizione della funzione `Monete`:

```
int* Monete(int *t, int size, int b);
```

La funzione accetta come parametri un array `t` di valori che rappresentano i tagli di monete disponibili (ad esempio 50, 20, 10, 5, 2 e 1 centesimo/i), la sua dimensione `size`, e un budget `b` espresso in centesimi. La funzione deve trovare il numero minimo intero di monete necessarie per formulare il budget, allocare dinamicamente un array delle stesse dimensioni di `t` dove scrivere il quantitativo di monete di ogni tipo necessarie per raggiungere il budget.

Algoritmi Greedy: Monete 1/3

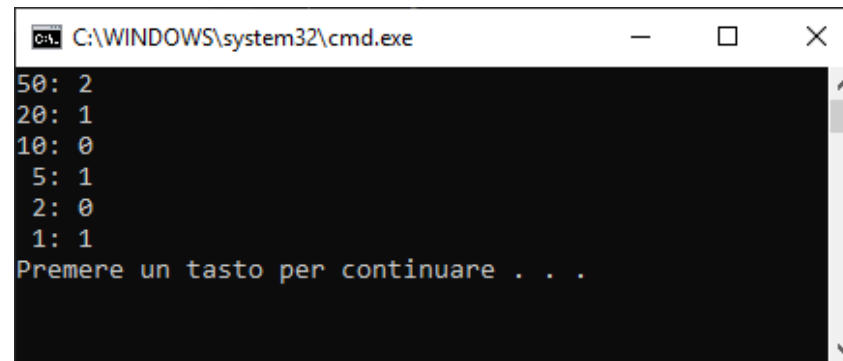
- La funzione deve quindi restituire l'array precedentemente allocato o `null` se il budget `b` è minore o uguale a 0.
- Si utilizzi a tale scopo un algoritmo greedy basato sull'opportuna funzione di costo.
- Si supponga di disporre di infinite monete per ogni taglio e che i tagli disponibili permettano di costruire il budget.
- Si assuma che l'array `t` preso in input dalla funzione sia ordinato in maniera decrescente.

Algoritmi Greedy: Monete 3/3

Nel `main()`, chiamare la funzione `monete` e mostrare su standard output la soluzione, ovvero la sequenza di monete da utilizzare. Ad esempio, con

$$t = \{50, 20, 10, 5, 2, 1\} \text{ e } b = 126$$

L'output dovrà essere il seguente:



```
C:\WINDOWS\system32\cmd.exe
50: 2
20: 1
10: 0
5: 1
2: 0
1: 1
Premere un tasto per continuare . . .
```

In questo caso la soluzione greedy ci dice che il numero minimo di monete necessarie per costruire il budget è 5, due pezzi da 50c, uno da 20c, uno da 5c e uno da 1c.

Algoritmi Greedy: Gioielli 1/3

- Esercizio 2 (Gioielli):

Nel file `gioielli.c` si implementi la definizione della procedura `Gioielli`:

```
void Gioielli(const char* filename, float b);
```

dove `filename` è il nome di un file di testo e `b` rappresenta il budget. La procedura deve leggere da file i gioielli disponibili e selezionare quelli da comprare (un gioiello si deve comprare per intero senza frazionamenti) in modo da massimizzare il peso complessivo dei gioielli acquistati rispettando il budget. Un gioiello si può acquistare una sola volta. Si utilizzi a tale scopo un algoritmo greedy con *l'opportuna funzione di costo*.

Algoritmi Greedy: Gioielli 2/3

Un gioiello è identificato dalla seguente struttura dati:

```
typedef struct {  
    int codice;  
    float peso;  
    float prezzo;  
} Gioiello;
```

dove `codice` identifica il gioiello, `peso` il suo peso in grammi e `prezzo` il suo prezzo di vendita in euro. Il file memorizza i dati per righe. Ogni riga del file contiene il codice, il peso e il prezzo di un singolo gioiello separati da spazi:

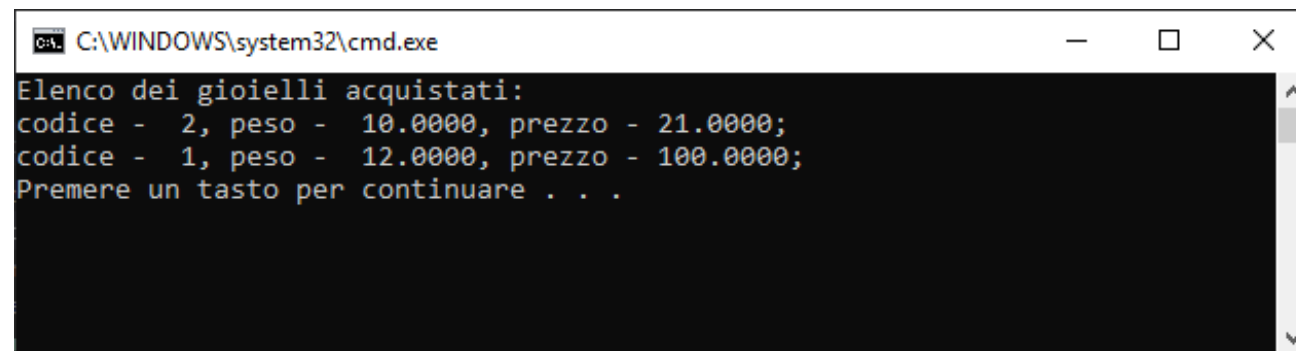
```
<codice> <peso> <prezzo>
```

Algoritmi Greedy: Gioielli 3/3

Al termine dell'esecuzione dell'algoritmo greedy, la procedura deve visualizzare su standard output i gioielli che devono essere acquistati. Ad esempio, dato budget 121 e il file di gioielli:

```
1 12 100
2 10 21
3 25 120
```

l'output dovrà essere:



```
C:\WINDOWS\system32\cmd.exe
Elenco dei gioielli acquistati:
codice - 2, peso - 10.0000, prezzo - 21.0000;
codice - 1, peso - 12.0000, prezzo - 100.0000;
Premere un tasto per continuare . . .
```

In questo caso la soluzione greedy non corrisponde all'ottimo assoluto.