Palestra di algoritmi day 10

17/01/2023

Abbiamo visto che quando abbiamo un input molto basso, possiamo provare tutte le combinazioni di soluzioni possibili, ossia usare un approccio “brute force”

A volte però non è necessario usare questo approccio, in quanto esistono tecniche migliori e più veloci.

Oggi parleremo di una di queste.

Algoritmi greedy

Greedy = ingordo

La tecnica greedy si basa sul fare sempre, ad ogni passo, la scelta migliore. Queste scelte migliori ci porteranno ad avere la soluzione migliore.

Detta così non ha molto senso, quindi vediamo un esempio.

Resto

Immaginiamo di avere a disposizione illimitate monete di ogni tipo.

(2€, 1€, 50c, 20c, 10c, 5c, 2c, 1c)

Come facciamo a dare il resto usando il minor numero di monete possibili?

Esempio: devo dare un resto di 3,25€

Soluzione

La soluzione migliore è data da

1 x 2€

1 x 1€

1 x 20c

1 x 5c

Come siamo arrivati a questa soluzione? Prendendo sempre la moneta più grande possibile.

Esercizio

Problema “Gelati costosi (gelato)” sul sito delle OII.

https://training.olinfo.it/#/task/pre-egoi-gelato/statement

Piccola deviazione sulle funzioni

Una funzione è un blocco di codice che prende degli argomenti in input e dà un risultato (sia input che output possono non esserci).

Esempio1:

```
void hello(){ //void è il tipo di valore ritornato. In questo caso la funzione non ritorna niente
        cout << “hello”;
}
```

Se nel main scrivo:
hello();
Verrà eseguito il blocco di codice dentro alla funzione hello

Esempio 2

```
void stampaNumero(int a){ //questa volta la funzione prende in input un intero
	cout << a;
}
int main(){
	stampaNumero(3);
	int x = 4;
	stampaNumero(x);
}
```

NOTA BENE

```
void stampaInteroPiuUno(int a){
        a = a+1;
        cout << a;
}
```

```
int main(){
        int a = 3;
        stampaIntero(a);
}
```

La variabile a dentro al main è diversa dalla variabile a dentro alla funzione.

La funzione crea una copia delle variabili passate come input.

Quindi la funzione non modifica a.

Il cout nel main stampa sempre 3.

Per fare in modo che la funzione modifichi la variabile anche nel main, useremo variabili globali (dichiarate prima del main, ricordate?). Ci sono altri metodi che però vedrete durante le lezioni di informatica, non con me.

```
void incrementaA(){
	a = a+1;
}

int A = 0;

int main(){
	incrementaA();
	cout << A;
}
```

Funzioni che ritornano un valore

```
int somma(int a, int b){
	return a + b;
}

int main(){
	int x = 3;
	int y = 4;
	cout << somma(x,y);
}
```

Sintassi giusta per una funzione

```
//Prima del main scriviamo solo l'"intestazione"
int somma(int a, int b);

int main(){
        …
}

//Dopo il main mettiamo la funzione per intero
int somma(int a, int b){
        return a+b;
}
```

Esercizio 2

Rispetta i versi (disuguaglianze)

https://training.olinfo.it/#/task/disuguaglianze/statement

Esercizio 3

Nimbus3000, su github

Vector

Dichiarazione: **vector<int> vett;** (posso anche mettere float, char, …)

Inizializzazione: **vector<int> vett = {1, 2, 3, 4, 5}** (come array normale)

Aggiungere un elemento: **vett.push_back(2)** //aggiunge 2 in fondo al vector

Sapere la dimensione: **vett.size()**

Per scorrere un vector:

```
int n = vett.size();
for(int i=0; i<n; i++){
        cout << vett[i] << " ";
}
```

Pair

Dichiarazione: **pair<int, int> coppia;**

Assegnare valori:
coppia.first = 3;
coppia.second = 4;

Vector di pair

```
vector < pair<int,int>> vett;


for(int i=0; i<vett.size(); i++){
        cout << vett[i].first << “ “ << vett[i].second << “\n”;
}
```

Ordinare i vector

```
vector<int> vett;

sort(vett.begin(), vett.end());
```

Ordinare vector di pair

vector <pair <int, int>> vett;

Per ordinare in base al primo elemento, uso sort normalmente

sort(vett.begin(), vett.end());

E per ordinare in base al secondo elemento?

```
bool sortBySecond(const pair<int,int> &a, const pair<int,int> &b) {
    return (a.second < b.second);
}
```

Nel main:

```
sort(vett.begin(), vett.end(), sortBySecond);
```