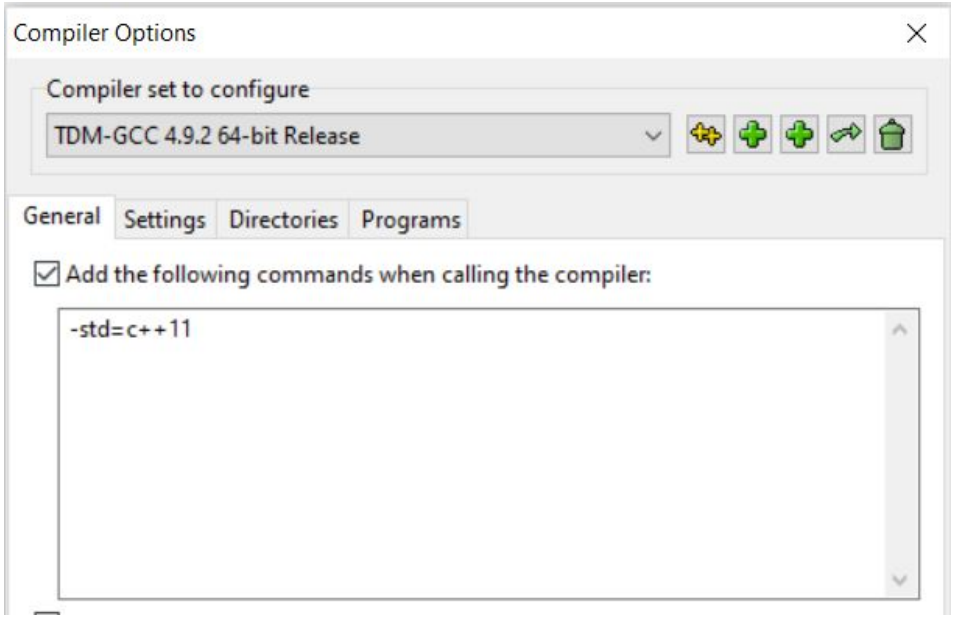
Palestra di algoritmi

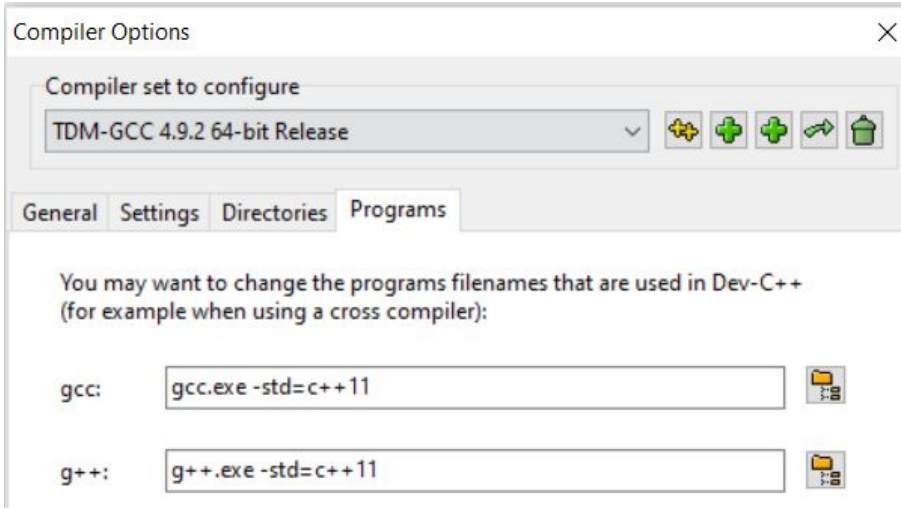
Gruppo B - 14/11/2023

Ambiente di sviluppo - configurare DevC++

Tools -> Compiler Options -> General
Cliccare la spunta e aggiungere “-std=c++11”



Tools -> Compiler Options -> Programs



Template di programma

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    ifstream in("input.txt");
    ofstream out("output.txt");

    in.close();
    out.close();
    return 0;
}
```

Nota: per andare a capo usate preferibilmente “\n”, non endl, perchè rallenta il programma

Easy3

Dati N numeri interi, si vuole trovare la somma pari massima tra tutte le coppie possibili di numeri distinti. Stampare -1 se non ci sono somme pari.

Ad esempio, se abbiamo la sequenza:

1 8 2 3 10 4 5

Allora la somma pari massima sarà data da 10+8 = 18

Come potremmo risolverlo???

Soluzione 1 - brute force

Con un doppio ciclo **proviamo tutte le coppie possibili**, salvando quella massima tra quelle pari.

```
//prendiamo in input l'array
int massimo = -1; //se non ci sono somme pari, dobbiamo stampare -1
for(int i=0; i<N; i++) {
	for(int j=i+1; j<N; j++) {
		if((vet[i]+vet[j])%2==0) { //se troviamo una coppia pari
			if(vet[i]+vet[j] > massimo) //che è più grande del massimo fin'ora
				massimo = vet[i]+vet[j]; //aggiorniamo il massimo
		}
	}
}
out << massimo;
```

Che risultato ci dà questa soluzione?

75 / 100

012	Correct	Output is correct	0.313s	572 KiB
013	Correct	Output is correct	0.719s	656 KiB
014	Correct	Output is correct	0.250s	656 KiB
015	Not correct	Execution timed out	1.098s	624 KiB
016	Not correct	Execution timed out	1.071s	572 KiB
017	Not correct	Execution timed out	1.023s	572 KiB
018	Not correct	Execution timed out	1.098s	572 KiB
019	Not correct	Execution timed out	1.095s	700 KiB

Correttezza e velocità

Per quanto è importante che il nostro programma sia corretto e dia l'output giusto, a volte questo non basta.

Tutti i problemi delle olimpiadi hanno un tempo massimo in cui il programma deve risolvere il problema.

Vediamo con un esempio come questo si applica alla realtà dei fatti.

Possiamo fare meglio?

SI

Soluzione più veloce

Alla fine la somma maggiore sarà data dai numeri più grandi, no?

Ci basta trovare i numeri più grandi dell’array che formano una coppia pari (quindi o due numeri pari o due numeri dispari) e avremo la nostra somma massima.

```
int N, i, num;
int p1, p2, d1, d2;
p1 = p2 = d1 = d2 = -1;

in >> N;

for(i=0;i<N;i++) {
    in >> num;
    if (num%2==0) {
        if(num>p1) {
            p2 = p1;
            p1 = num;
        } else if(num>p2) {
            p2 = num;
        }
    } else {
        if(num>d1) {
            d2 = d1;
            d1 = num;
        } else if(num>d2) {
            d2 = num;
        }
    }
}
```

```
int ris1 = -1;
int ris2 = -1;

if(p1!=-1 && p2!=-1) {
    ris1 = p1+p2;
}
if(d1!=-1 && d2!=-1) {
    ris2 = d1+d2;
}

int ris = ris1;

if(ris1>ris2){
    ris = ris1;
} else {
    ris = ris2;
}

out << ris;
```

Problemi greedy

La tecnica che abbiamo usato è una tecnica greedy, diversamente dal primo tentativo che era un brute force.

Un **algoritmo greedy** è un algoritmo che **a ogni passo sceglie la soluzione migliore in quel momento**, senza “preoccuparsi” delle conseguenze a lungo termine. Se si riesce a dimostrare che prendere le soluzioni migliori ad ogni passo porterà ad avere la soluzione migliore in assoluto, allora l’approccio greedy è corretto.

Nel nostro caso, sapevamo che scegliere i numeri più grandi ad ogni passo ci porterà ad avere i numeri più grandi alla fine, e quindi la somma più grande possibile.

Ordinamento

Per alcuni algoritmi greedy, potrebbe essere utile ordinare i dati prima.

Per ordinare un array potete usare questa funzione:

sort(vet, vet+N)

Dove vet è il nome del vostro vettore e N è il numero di elementi del vettore (N può anche essere più piccolo del numero di elementi totali).

Esempio ordinamento

```
#define MAX 10
int main(){
	int N = 5;
	int vet[MAX]; //il vettore ha 10 elementi
	for(int i=0; i<N; i++) { in >> vet[i]; } //riempiamo solo i primi 5
	sort(vet, vet+N); //ordiniamo solo i primi 5, perchè gli altri non sono inizializzati
}
```

Es:
Vettore non inizializzato: ? ? ? ? ? ? ? ? ? ?
Vettore inizializzato: 10 1 32 45 8 ? ? ? ? ?
Vettore dopo l'ordinamento: 1 8 10 32 45 ? ? ? ? ?

Complessità computazionale

Abbiamo quindi detto che i nostri programmi devono essere veloci, ma non abbiamo mai discusso di come possiamo stabilire se un algoritmo è veloce.

Come possiamo misurare il tempo impiegato da un programma a terminare?

Il tempo reale dipende da troppi parametri:

- linguaggio di programmazione utilizzato
- codice generato dal compilatore
- processore, memoria (cache, primaria, secondaria)
- sistema operativo, processi attualmente in esecuzione

Insomma, dobbiamo trovare un modo più astratto.

Possiamo usare il numero di operazioni “rilevanti”

Ricerca del minimo

```
int minimo = vet[0];
for(int i=0;i<n;i++){
        if(vet[i]<minimo)
                minimo = vet[i];
}
```

Ricerca di un elemento

```
int pos = -1;
for(int i=0;i<n;i++){
        if(vet[i]==cercato)
                pos = i;
}
```

NB: le operazioni che consideriamo devono essere “semplici”

Notazione O(n)

n = dimensione del nostro input.

Nel caso della ricerca di un elemento, n = numero di elementi dell'array.

O(n) significa che vengono eseguite al massimo n operazioni.

Se scorriamo tutto l'array una volta -> O(n)

Se facciamo un doppio ciclo -> O(n^2)

E così via…

A cosa serve a noi tutto questo?

Solitamente

1 secondo = 1 milione di operazioni.

Quindi:

- con input molto piccoli: possiamo fare quello che vogliamo
- con input fino a n=1000 -> possiamo fare un algoritmo in O(n^2)
- con input più grandi dobbiamo fare algoritmi migliori, per esempio in O(n)

Selection sort

```
for(int i=0;i<n;i++) {
        //trovo il minimo
        int minimo = vet[i];
        int pos = i;
        for(int j=i;j<n;j++) {
            if(vet[j]<minimo) {
                minimo = vet[j];
                pos = j;
            }
        }
        //scambio minimo
        int temp = vet[pos];
        vet[pos] = vet[i];
        vet[i] = temp;
}
```

il ciclo esterno viene ripetuto n volte

il ciclo interno viene ripetuto al massimo n volte (la prima volta i va da 0 a n)

-> O(n^2)!!!

Bubble sort

```
bool notOrdinato = true;

while(notOrdinato) {
        notOrdinato = false;
        for(int i=0;i<numElem-1;i++) {
                if(vet[i]>vet[i+1]) {
                        notOrdinato = true;
                        int temp = vet[i];
                        vet[i] = vet[i+1];
                        vet[i+1] = temp;
                }
        }
}
```

quante volte viene ripetuto il while?

al massimo n volte

il ciclo interno viene ripetuto n volte

→ O(n^2)!!!

Ma quindi selectionSort e bubbleSort hanno la stessa complessità?

Cosa succede quando l’array è già ordinato?

Ordinamento (2)

L’algoritmo di sort che utilizza il c++ è molto ottimizzato e ordina i dati in

O(n log(n))

Usatelo in modo furbo.

Esercizi di oggi

Facili:

- candies
- quadrati

Un po’ più challenging:

- 3x2
- gasoline
- quadri
- disuguaglianze
- turni