Competitive Programming

Riccardo Maso Davide Guidobene Florian Sabani Davide Cazzin

Novembre 2021

1 Problemi

Con la Competitive Programming definiamo l'attività' per la quale andiamo a risolvere un problema strutturato in modo tale da avere :

1. Descrizione Testuale del problema
2. Definizione del File Input
3. Input Constraint
4. Esempio di Input/Expected Output
5. Time & Space Constraint (TL & SL)

Lo scopo del programmatore competitivo e' quello di leggere il testo del problema, analizzarne la complessità e svilupparne una soluzione in grado di dato l'input, fornire sempre l'output atteso, rientrando nei limiti delle risorse tempo e spazio che l'algoritmo può usare.

Es :`https://open.kattis.com/problems/hello` `https://open.kattis.com/problems/10kindsofpeople`

2 Classi di complessità

[parlare della BigO Notation]
Per poter risolvere i problemi rientrando nel TL dobbiamo fare leva sull' Input Constraint datoci dalla descrizione del problema.In particolare avendo N la dimensione dei dati in input, generalmente possiamo scegliere un algoritmo risolutivo seguendo quanto dice la seguente tabella :

n	Worst AC Algorithm	Comment
$\leq [10..11]$	$O(n!), O(n^6)$	e.g. Enumerating permutations (Section 3.2)
$\leq [15..18]$	$O(2^n \times n^2)$	e.g. DP TSP (Section 3.5.2)
$\leq [18..22]$	$O(2^n \times n)$	e.g. DP with bitmask technique (Section 8.3.1)
≤ 100	$O(n^4)$	e.g. DP with 3 dimensions + $O(n)$ loop, ${}_nC_{k=4}$
≤ 400	$O(n^3)$	e.g. Floyd Warshall's (Section 4.5)
$\leq 2K$	$O(n^2 \log_2 n)$	e.g. 2-nested loops + a tree-related DS (Section 2.3)
$\leq 10K$	$O(n^2)$	e.g. Bubble/Selection/Insertion Sort (Section 2.2)
$\leq 1M$	$O(n \log_2 n)$	e.g. Merge Sort, building Segment Tree (Section 2.3)
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	Most contest problem has $n \leq 1M$ (I/O bottleneck)

3 Quale linguaggio?

C/C++ Python Java

4 Read/Write

Sempre attenzione nel rendere operazioni I/O efficienti.

```
// Read something
int value;
cin >> value;

// Print something
cout << value << "\n";
cout << value << endl; // with flush
```

5 Template

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned int uint;
#define REP(x,l,u) for(ll x = l; x < u; x++)
#define RREP(x,l,u) for(ll x = l; x >= u; x--)
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
#define mst(x,v) memset(x, v, sizeof(x))
#define sz(x) (ll)x.size()
void in() {}
template <typename A> void in(A & x) { cin >> x; }
template <typename A, typename B> void in(pair<A,B> & x) { in(x.first); in(x.second); }
template <typename A> void in(vector<A> & x) { REP(i,0,(ll)x.size()) in(x[i]); }
template <typename Head, typename... Tail> void in(Head & H, Tail & ... T) {in(H); in(T...); }

void solve(){
	// INSERT YOUR SOLUTION HERE
}

int main() {
	ios::sync_with_stdio(0); // fast io
	cin.tie(0); // fast io

	int t;
	cin >> t;
	while(t--){
		solve();
	}

	return 0;
}
```

6 Vector

7 Sort

In competitive programming molto difficilmente dovremmo scrivere un algoritmo di sorting, la maggioranza delle volte useremo il sort che ci viene fornito dalla standard library.

```
void sort (RandomAccessIterator first, RandomAccessIterator last);

void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);
```

Il sort utilizzato da questa funzione è chiamato IntroSort e ha una complessità temporale $O(nlog(n))$ e spaziale $O(log(n))$.

Il comparator di default che viene utilizzato dalla funzione sort, nel caso non ne venga fornito un altro, è $less < int >$ e ordina gli elementi presenti all'interno della struttura dati in ordine non decrescente. Per ordinare il vettore con ordine non crescente possiamo usare il comparator $greater < int >$, come nel seguente esempio:

```
vector<int> v = {1,4,5,6,3,2};
sort(v.begin(), v.end(), greater<int>);
```

Nel caso volessimo ordinare il nostro vettore secondo un criterio diverso da quello del comparator $less < int >$ e $greater < int >$ allora dobbiamo definire un comparator. Supponiamo ad esempio di aver un vettore di $pair < string, int >$ e vogliamo che il vettore sia ordinato in base all'intero con ordine non decrescente e nel caso di interi uguali devono essere ordinati in ordine alfabetico.

8 Struct