Competitive programming Notebook 🎈

Meia noite eu te conto

Contents

1 General

1.1 Random

```cpp
random_device dev;
mt19937 rng(dev());

uniform_int_distribution<mt19937::result_type> dist(1, 6); // distribution in range [1, 6]

int val = dist(rng);
```

1.2 Split

```cpp
vector<string> split(string s, char key=' ') {
    vector<string> ans;
    string aux = "";

    for (int i = 0; i < (int)s.size(); i++) {
        if (s[i] == key) {
            if (aux.size() > 0) {
                ans.push_back(aux);
                aux = "";
            }
        } else {
            aux += s[i];
        }
    }

    if ((int)aux.size() > 0) {
        ans.push_back(aux);
    }

    return ans;
}
```

1.3 Base Converter

```cpp
const string digits = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

ll tobase10(string number, int base) {
    map<char, int> val;
    for (int i = 0; i < digits.size(); i++) {
        val[digits[i]] = i;
    }

    ll ans = 0, pot = 1;

    for (int i = number.size() - 1; i >= 0; i--) {
        ans += val[number[i]] * pot;
        pot *= base;
    }

    return ans;
}

string frombase10(ll number, int base) {
    if (number == 0) return "0";

    string ans = "";

    while (number > 0) {
        ans += digits[number % base];
        number /= base;
    }

    reverse(ans.begin(), ans.end());

    return ans;
}

// verifica se um nÃžmero estÃą na base especificada
bool verify_base(string num, int base) {
    map<char, int> val;
    for (int i = 0; i < digits.size(); i++) {
        val[digits[i]] = i;
    }

    for (auto digit : num) {
        if (val[digit] >= base) {
            return false;
        }
    }

    return true;
}
```

1.4 Template

```cpp
// MEIA NOITE EU TE CONTO
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

int main() { _
    return 0;
}
```

2 DS

2.1 Dsu

```cpp
/*
DSU - Disjoint Set Union (or Union Find)

find(x) -> find component that x is on
join(a, b) -> union of a set containing 'a' and set containing b

find / join with path compreension -> O(inv_Ackermann(n)) [O(1)]
find / join without path compreension -> O(logN)

https://judge.yosupo.jp/submission/126864
*/

struct DSU {

    int n = 0, components = 0;
    vector<int> parent;
    vector<int> size;

    DSU(int nn){
        n = nn;
        components = n;
        size.assign(n + 5, 1);
        parent.assign(n + 5, 0);
        iota(parent.begin(), parent.end(), 0);
    }

    int find(int x){
        if(x == parent[x]) {
            return x;
        }
```

```
        //path compression
        return parent[x] = find(parent[x]);
    }

    void join(int a, int b){
        a = find(a);
        b = find(b);
        if(a == b) {
            return;
        }
        if(size[a] < size[b]) {
            swap(a, b);
        }
        parent[b] = a;
        size[a] += size[b];
        components -= 1;
    }

    int sameSet(int a, int b) {
        a = find(a);
        b = find(b);
        return a == b;
    }

};
```

2.2 Ordered Set

```
// Ordered Set
//
// set roubado com mais operacoes
//
// para alterar para multiset
// trocar less para less_equal
//
// ordered_set<int> s
//
// order_of_key(k) // number of items strictly smaller than k -> int
// find_by_order(k) // k-th element in a set (counting from zero) -> iterator
//
// https://cses.fi/problemset/task/2169
//
// O(log N) para insert, erase (com iterator), order_of_key, find_by_order

using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T,null_type,less<T>, rb_tree_tag,tree_order_statistics_node_update>;
```

3 DP

3.1 Knapsack

3.2 Edit Distance

```
// Edit Distance / Levenshtein Distance
//
// numero minimo de operacoes
// para transformar
// uma string em outra
//
// tamanho da matriz da dp eh |a| x |b|
// edit_distance(a.size(), b.size(), a, b)
//
// https://cses.fi/problemset/task/1639
//
// O(n^2)

int tb[MAX][MAX];

int edit_distance(int i, int j, string &a, string &b) {
    if (i == 0) return j;
    if (j == 0) return i;

    int &ans = tb[i][j];

    if (ans != -1) return ans;

    ans = min({
        edit_distance(i-1, j, a, b) + 1,
        edit_distance(i, j-1, a, b) + 1,
        edit_distance(i-1, j-1, a, b) + (a[i-1] != b[j-1])
    });

    return ans;
}
```

4 Graph

4.1 Dfs

```
// DFS
//
// Percorre todos os vertices
// priorizando profundidade
//
// O(n+m)

vector<vector<int>> g;
vector<bool> vis;

void dfs(int s){
    if(vis[s]) return;
    vis[s] = true;
    for(auto v : g[s]){
        dfs(v);
    }
}
```