Notebook - Maratona de Programação

Py tá O(N)

Contents

1 Algoritmos

1.1 Busca Binaria

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

bool check(int valor, int x) {
    return valor <= x;
}

int bb(int a, int b, int x){
    int l = a;
    int r = b;
    while (l < r) {
        int mid = (l + r) / 2;
        if (check(mid, x)) r = mid;
        else l = mid + 1;
    }
    return l;
}

bool check(int valor) {
    return valor <= 10;
}

int bb_menor(int a, int b){
    int l = a;
    int r = b;
    while (l < r) {
        int mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }

    return l;
}


int bb_maior(int a, int b){
    int l = a;
    int r = b;
    while (l < r) {
        int mid = (l + r) / 2;
        if (!check(mid)) r = mid;
        else l = mid + 1;
    }

}
```

1.2 Busca Binaria Double

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

typedef long long ll;
const int MAX = 1e5+1;
const double EPS = 0.0000001;

vector<int> v(100001);
int n;
ll check(double x){
    ll sum = 0;
    for(int i=0; i<n; i++){
        sum += (v[i]/x);
    }
    return sum;
}

int main(){

    int k;
    cin>>n>>k;

    for(int i=0; i<n; i++)cin>>v[i];

    double l=0.0000000, r=10000000.0000000;
    double mid;
    while(r-l>EPS){
        mid = (double)((l + r)/2);
        if (check(mid)>=k){
            l=mid;
        }
        else{
            r = mid;
        }
    }

    cout<<fixed<<setprecision(7)<<mid<<endl;

    return 0;
}
```

1.3 Busca Binaria Resposta

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define loop(i,a,n) for(int i=a; i < n; i++)

ll upperbound(ll maior, ll k, vector<ll> tabuas){
    ll mid = 0, l = 0, r = maior, count = 300;
    ll aux;
    while((l < r) && (count--)){
        aux = 0;
        mid = (l+r)/2;
        loop(i,0,tabuas.size()){
            if(mid > 0){aux += (tabuas[i]/mid);}
        }
        if(aux >= k){l = mid;}
        else{r = mid;}
    }

    ll aux_2 = 0;
    loop(i,0,tabuas.size()){
        aux_2 += (tabuas[i]/(mid+1));
    }
    if(aux_2 >= k){return mid+1;}

    if(aux < k){
        int aux_2 = 0;
        loop(i,0,tabuas.size()){
            if(mid - 1 > 0){aux_2 += (tabuas[i]/(mid
-1));}
        }
        if(aux_2 >= k){return mid-1;}
    }

    return mid;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie( NULL);
    cout.tie(NULL);
    int n; cin >> n;
    ll k; cin >> k;
    vector<ll> tabuas(n);
    ll maior = 0;
    loop(i,0,n){
        cin >> tabuas[i];
        maior = max(maior,tabuas[i]);
    }
    cout << upperbound(maior,k,tabuas);
}
```

1.4 Delta

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

int main(){
    int n, q;
    cin >> n >> q;
    vector<int> v(n,0);
    vector<int> delta(n+2, 0);

    while(q--){
        int l, r, x;
        cin >> l >> r >> x;
        delta[l] += x;
        delta[r+1] -= x;
    }

    int atual = 0;
    for(int i=0; i < n; i++){
        atual += delta[i];
        v[i] = atual;
    }

    for(int i=0; i< n; i++){
        cout << v[i] << " ";
    }
    cout << endl;

    return 0;
}
```

1.5 Fast Exponentiaton

```cpp
int fast_exp(int base, int e){
    if(e == 0) return 1;
    if(e % 2) return base * fast_exp(base * base,e/2)
    ;
    else return fast_exp(base * base, e/2);
}
```

1.6 Psum

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

#define input(x) for (auto &it : x) cin >> it
typedef long long ll;
vector<ll> psum(1e5);

int solve(int l, int r){
    if(l==0) return psum[r];
    else return psum[r] - psum[l-1];
}

int main(){

    int n, q;
    cin>>n>>q;

    vector<int> v(n);
    input(v);
    for(int i=0; i<n; i++){
        if(i==0)psum[i] = v[i];
        else psum[i] = psum[i-1] + v[i];
    }
    while(q--){
        int l, r;
        cin>>l>>r;

        cout<<(solve(l,r))<<endl;
    }

    return 0;
}
```

2 DP

2.1 Dp

```cpp
// DP - Dynamic Programming

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

typedef long long ll;
const int MAX = 110;

int n;
int tab[MAX];
vector<int> v;

ll dp(int i){
    if(i>=n) return 0;
    if(tab[i] != -1) return tab[i];

    int pega = v[i] + dp(i+2);
    int npega = dp(i+1);

    tab[i] = max(pega, npega);
    return tab[i];
}

int main(){
    memset(tab, -1, sizeof(tab));
    cin>>n;

    v.assign(n, 0);

    cout<<dp(0)<<endl;

    return 0;
}
```

2.2 Knapsack

```cpp
/* A Naive recursive implementation of
 0-1 Knapsack problem */
#include <bits/stdc++.h>
using namespace std;

// Returns the maximum value that
// can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{

    // Base Case
    if (n == 0 || W == 0)
        return 0;

    // If weight of the nth item is more
    // than Knapsack capacity W, then
    // this item cannot be included
    // in the optimal solution
    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);

    // Return the maximum of two cases:
    // (1) nth item included
    // (2) not included
    else
        return max(val[n - 1] + knapSack(W - wt[n -
    1], wt, val, n - 1), knapSack(W, wt, val, n - 1))
    ;
}

// Driver code
int main()
{
    int val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);
    cout << knapSack(W, wt, val, n);
    return 0;
}

// This code is contributed by rathbhupendra
```

2.3 Mochila Iterativa

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

const int maxn = 110, maxp = 1e5+10;
const long long inf = 0x3f3f3f3f3f3f3f3f; // ~= 10^18

int v[maxn], p[maxn];
long long dp[maxn][maxp];

int main() {
    int n, C; scanf("%d %d", &n, &C);
    for(int i = 1; i <= n; i++)
        scanf("%d %d", &p[i], &v[i]);

    long long ans = 0;
    // inicializando o vetor
    for(int i = 1; i <= n; i++)
        for(int P = p[i]; P <= C; P++)
            dp[i][P] = -inf;
    // definindo o caso base
    dp[0][0] = 0;

    for(int i = 1; i <= n; i++) {
        for(int P = 0; P <= C; P++) {
            dp[i][P] = dp[i-1][P];
            if(P >= p[i])
                dp[i][P] = max(dp[i][P], dp[i-1][P-p[i]] + v[i]);
            ans = max(ans, dp[i][P]);
        }
    }

    printf("%lld\n", ans);
}
```

2.4 Mochila Recursiva

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

const int maxn = 110, maxp = 1e5+10;

int v[maxn], p[maxn], n;
long long dp[maxn][maxp];
bool vis[maxn][maxp];

long long solve(int i, int P) {
    if(i == n+1) return 0; // caso base, nao ha mais itens para se considerar
    if(vis[i][P]) return dp[i][P];
    vis[i][P] = 1;

    // primeira possibilidade, ãno adicionar o elemento
    dp[i][P] = solve(i+1, P);

    // segunda possibilidade, adicionar o elemento.
    // Lembrar de tirar o maximo com o valor ja calculado da primeira possibilidade
    if(P >= p[i])
        dp[i][P] = max(dp[i][P], solve(i+1, P - p[i]) + v[i]);

    return dp[i][P];
}

int main() {
    int C; scanf("%d %d", &n, &C);
    for(int i = 1; i <= n; i++)
        scanf("%d %d", &p[i], &v[i]);
    printf("%lld\n", solve(1, C));
}
```

3 ED

3.1 Bit

```
// Bitwise Operations

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


// Verificar se o bit esta ligado
bool isSet(int bitPosition, int number) {
    bool ret = ((number & (1 << bitPosition)) != 0);
    return ret;
}

// Ligar o bit
bool setBit(int bitPosition, int number) {
    return (number | (1 << bitPosition) );
}

// Gerando todos os subconjuntos de um conjunto em binario
void possibleSubsets(char S[], int N) {
    for(int i = 0;i < (1 << N); ++i) {  // i = [0, 2^N - 1]
        for(int j = 0;j < N;++j)
            if(i & (1 << j))  // se o j-esimo bit de i esta setado, printamos S[j]
                cout << S[j] << " ";
        cout << endl;
    }
}
```

3.2 Merge Sort

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

#define INF 1000000000

void merge_sort(vector<int> &v){
    if(v.size()==1)return;

    vector<int> v1, v2;

    for(int i=0; i<v.size()/2; i++) v1.push_back(v[i]);
    for(int i=v.size()/2; i<v.size(); i++) v2.push_back(v[i]);

    merge_sort(v1);
    merge_sort(v2);
```

```cpp

    v1.push_back(INF);
    v2.push_back(INF);

    int ini1=0, ini2=0;

    for(int i=0; i<v.size(); i++){
        if(v1[ini1]<v2[ini2]){
            v[i] = v1[ini1];
            ini1++;
        }else{
            v[i] = v2[ini2];
            ini2++;
        }
    }
    return;
}
```

3.3 Segtree 1

```cpp
#include <bits/stdc++.h>
using namespace std;
class SegTree{
    vector<int> st;
    int size;

    int el_neutro = -(1e9 + 7);

    int f(int a, int b){
        return max(a,b);
    }

    int query(int sti, int stl, int str, int l, int r){
    //O nodo esta fora do intervalo que estamos interessados, retorne o elemento neutro que nao afeta a consulta
        if(str < l || r < stl)
            return el_neutro;

    // O nodo esta completamente incluido no intervalos que estamos interessados, retorne a informacao contida naquele nodo.
        if(stl >= l and str <= r)
            return st[sti];

    // Se chegarmos aqui, eh porque esse Nodo esta parcialmente contido no intervalo que estamos interessados. Entao, continuamos procurando nos filhos.
        int mid = (str+stl)/2;

        return f(query(sti*2+1,stl,mid,l,r),query(sti*2+2,mid+1,str,l,r));
    }

    void update(int sti, int stl, int str, int i, int amm){
        // Chegamos no indice que queremos, vamos atualizar o valor
        if(stl == i and str == i){
            st[sti] = amm;
            return;
        }
        // O intervalo que estamos nao contem o indice que queremos atualizar, retorne
        if(stl > i or str < i)
            return;

        // O intervalo contem o indice, mas temos que chegar no nodo especifico, recurse para os filhos.
        int mid = (stl + str)/2;

        update(sti*2+1,stl,mid,i,amm);
        update(sti*2+2,mid+1,str,i,amm);
        // Apos os filhos mais em baixo, precisamos atualizar o valor desse nodo
        st[sti] = f(st[sti*2+1],st[sti*2+2]);
    }
    public:
        SegTree(int n):  st(4*n,0){size = n;}
        int query(int l, int  r){return query(0,0,size-1,l,r);}
        void update(int i, int amm){update(0,0,size-1,i,amm);}
};

int main(){
    vector<int> v;
    SegTree st(v.size());
    for(int i = 0; i< v.size();i++){
        st.update(i,v[i]);
    }
}
```

3.4 Segtree 2

```cpp

#include <bits/stdc++.h>
#define ff first
#define ss second
#define ll long long
#define pb push_back
#define sws ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);

using namespace std;

const int MAX = 1e5; // tamanho maximo do vetor
const ll LLINF = 0x3f3f3f3f3f3f3f3f;

// End Template //

ll tree[4*MAX], vet[MAX];

void build(int l, int r, int no){
    if(l==r){
        tree[no] = vet[l];
        return;
    }
    int mid = (l+r)/2;
    build(l, mid, 2*no);
    build(mid+1, r, 2*no+1);

    tree[no] = tree[2*no] + tree[2*no+1];
}

void update(int id, int x, int l, int r, int no){
    if(l==r){
        tree[no] = x;
        return;
    }

    int mid = (l+r)/2;
    if(id<=mid)
        update(id, x, l, mid, 2*no); // esquerda
    else
        update(id, x, mid+1, r, 2*no+1);

    tree[no] = tree[2*no] + tree[2*no+1];
}

ll query(int A, int B, int l, int r, int no){
    // caso 1
    if(B<l or r<A) return 0;
    // caso 2
    if(A<=l and r<=B) return tree[no];
    // caso 3
    int mid = (l+r)/2;
    ll sumLeft = query(A, B, l, mid, 2*no);
    ll sumRight = query(A, B, mid+1, r, 2*no+1);

    return sumLeft + sumRight;
}


int32_t main()
{sws;

    int n, m, opt, id, v, l, r;
    cin >> n >> m;
    for(int i=0;i<n;i++)
        cin >> vet[i];

    build(0, n-1, 1);

    for(int i=0;i<m;i++){
        cin >> opt;
        if(opt==1){ // update
            cin >> id >> v;
            update(id, v, 0, n-1, 1);
        }else{ // query
            cin >> l >> r;
            cout << query(l, r-1, 0, n-1, 1) << endl;
        }
    }



    return 0;
}
```

3.5 Segtree Lazy Propagation

```cpp
#include <bits/stdc++.h>
#define ll long long

using namespace std;

const int MAX = 1e5; // tamanho maximo do vetor
const ll LLINF = 0x3f3f3f3f3f3f3f3f;

// End Template //

vector<ll> lazy(4*MAX, -1);
ll tree[4*MAX], vet[MAX];
int N;

ll merge(ll a, ll b){
    return a + b;
}

void build(int l=0, int r=N-1, int no=1){
    if(l==r){
        tree[no] = vet[l];
        return;
    }
    int mid = (l+r)/2;
    build(l, mid, 2*no);
    build(mid+1, r, 2*no+1);

    tree[no] = merge(tree[2*no], tree[2*no+1]);
}

void prop(int l, int r, int no){
    if(lazy[no]!=-1){
        tree[no] = (r-l+1)*lazy[no];
        if(l!=r){
            lazy[2*no] = lazy[2*no+1] = lazy[no];
        }
        lazy[no] = -1;
    }
}

void update(int A, int B, int x, int l=0, int r=N-1,
    int no=1){
    prop(l, r, no);
    // caso 1
    if(B<l or r<A) return;
    // caso 2
    if(A<=l and r<=B){
        lazy[no] = x;
        prop(l, r, no);
        return;
    }
    // caso 3
    int mid = (l+r)/2;

    update(A, B, x, l, mid, 2*no);
    update(A, B, x, mid+1, r, 2*no+1);

    tree[no] = merge(tree[2*no], tree[2*no+1]);
}

ll query(int A, int B, int l=0, int r=N-1, int no=1){
    prop(l, r, no);
    // caso 1
    if(B<l or r<A) return 0;
    // caso 2
    if(A<=l and r<=B) return tree[no];
    // caso 3
    int mid = (l+r)/2;

    return merge(query(A, B, l, mid, 2*no),
                 query(A, B, mid+1, r, 2*no+1));
}


int32_t main()
{

    int Q, opt, a, b, l, r, k;
    cin >> N >> Q;
    for(int i=0;i<N;i++)
        cin >> vet[i];

    build();

    for(int i=0;i<Q;i++){
        cin >> opt;
        if(opt==1){ // update
            cin >> a >> b >> k;
            a--;b--;
            update(a, b, k);
        }else{ // query
            cin >> l >> r;
            l--;r--; // indice indexado em 0
            cout << query(l, r) << endl;
        }
    }

    return 0;
}
```

4 Grafos

4.1 Bellman Ford

```cpp
/*
```

```
Algoritmo de busca de caminho minimo em um digrafo (
    grafo orientado ou dirigido) ponderado, ou seja,
    cujas arestas tem peso, inclusive negativo.
*/

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

// pode usar uma tuple
struct Edge {
    // [de onde vem, pra onde vai, peso]
    int from, to, custo;

    Edge(int a=0, int b=0,int c=0 ){
        from = a;
        to=b;
        custo = c;
    }

};

int main(){

    int n, m;
    cin>>n>>m;
    vector<Edge> arestas(m);

    for(int i=0; i<m; i++){
        int a, b, c;
        cin>>a>>b>>c;
        arestas[i] = Edge(a, b, c);
    }

    vector<int> distancia(n + 1, 100000000);
    distancia[1]=0;
    for(int i=0; i<n-1; i++){
        for(auto aresta : arestas){
            if (distancia[aresta.from] + aresta.custo
 < distancia[aresta.to]){
                distancia[aresta.to] = distancia[
    aresta.from] + aresta.custo;
            }
        }
    }

    for(int i=1; i<=n; i++){
        cout<<"Distancia ate o vertice "<<i<<" "<<
    distancia[i]<<endl;
    }

    return 0;
}
```

4.2 Bfs

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

//------------------------
#define MAXN 50050

int n, m;
bool visited[MAXN];
vector<int> lista[MAXN];
//------------------------

void bfs(int x){

    queue<int> q;
    q.push(x);
    while(!q.empty()){
        int v = q.front();
        q.pop();
        visited[v] = true;
        for(auto i : lista[v]){
            if(!visited[i]){
                q.push(i);
            }
        }
    }
}
```

4.3 Dfs

```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

//------------------------
#define MAXN 50050

int n, m;
bool visited[MAXN];
vector<int> lista[MAXN];
//------------------------

void dfs(int x){
    visited[x] = true;
    for(auto i : lista[x]){
        if(!visited[x]){
            dfs(i);
        }
    }
}

void dfsStack(int x){
    stack<int> s;
    s.push(x);
    while(!s.empty()){
        int v = s.top();
        s.pop();
        visited[v] = true;
        for(auto i : lista[v]){
            if(!visited[i]){
                s.push(i);
            }
        }
    }
}
```

4.4 Diametro Arvore Bfs

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

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> pii;
const int MAX = 1e5+10;
const ll MOD = 1e9+5;

vector<int> adj(MAX);

pair<int, int> bfs(int s, int N){

    vi dist(N + 1, MAX); dist[s] = 0;
    queue<int> q; q.push(s);
    int last = s;

    while(!q.empty()){
        auto u = q.front();q.pop();
        last = u;

        for(auto v: adj[u]){
            if(dist[v]==MAX){
                dist[v]=dist[u]+1;
                q.push(v);
            }
        }
    }

    return {last, dist[last]};
}

int diameter(int N){
    auto [v, _] = bfs(1, N);
    auto [w, D] = bfs(v, N);

    return D;
}
```

4.5 Diametro Arvore Dfs

```
// DIAMETRO ARVORE - DFS

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

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> pii;
const int MAX = 1e5+10;
const ll MOD = 1e9+5;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;

int to_leaf[MAX];
int max_length[MAX];
int dist[MAX];

vector<int> adj(MAX);
/*
void dfs(int u, int p, vector<int> &dist){
    for(auto [v, w] : adj[u]){
        if(v!=p){
            dist[v] = dist[u] + w;
            dfs(v, u, dist);
        }
    }
}

int solve(int n){
    vector<int> dist(n+1, 0);

    dfs(0, -1, dist);

    auto v = (int)(max_element(dist.begin(), dist.end()) - dist.begin());

    dist[v] = 0;
    dfs(v, -1, dist);

    return *max_element(dist.begin(), dist.end());
}*/

void dfs(int u, int p){
    vi ds;

    for(auto v: adj[u]){
        if(v==p)continue;

        dfs(v, u);
        ds.pb(to_leaf[v]);
    }
    sort(ds.begin(), ds.end());

    to_leaf[u] = ds.empty() ? 0 : ds.back() + 1;

    auto N = ds.size();

    switch(N){
        case 0:
            max_length[u]=0;
            break;
        case 1:
            max_length[u] = ds.back() + 1;
            break;
        default:
            max_length[u] = ds[N-1] + ds[N-2] + 2;
    }
}

int diameter(int root, int N){
    dfs(root, 0);

    int d=0;

    for(int u=1; u<=N; u++){
        d= max(d, max_length[u]);
    }
}
```

4.6 Dijkstra

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long

const int N = 100005;
const ll oo = 1e18;

ll d[N]; // vetor onde guardamos as distancias

int n; // numeros de vertices

// lista de adjacencias guarda
// pair <vertice para onde a aresta vai, peso da aresta>
vector<pair<int, ll>> g[N];

void dijkstra(int start){

    // inicialmente a distancia do vertice
    // start para todo os outros eh infinita
    for(int u = 1; u <= n; u++)
        d[u] = oo;

    // fila de prioridade de pair<ll, int>, mas que o
    // menor pair fica no topo da fila
    // guardamos um pair <distancia ate o vertice, vertice>
    // assim o topo da fila sempre eh o vertice com menor distancia
    priority_queue<pair<ll, int>, vector<pair<ll, int>>,
    greater<pair<ll, int>> > pq;

    d[start] = 0;
    pq.emplace(d[start], start);

    ll dt, w;
    int u, v;
    while(!pq.empty()){
        tie(dt, u) = pq.top(); pq.pop();
        if(dt > d[u]) continue;
        for(auto edge : g[u]){
            tie(v, w) = edge;

            // se a distancia ate o u somado com o peso
            // da aresta eh menor do que a distancia ate o v que
            // tinhamos antes, melhoramos a distancia ate o v
            if(d[v] > d[u] + w){
                d[v] = d[u] + w;
                pq.emplace(d[v], v);
            }
        }
    }
}

int main(){

    // le o input, qnt de vertices, arestas
    // e vertice inicial(start)
    int start = 0; // inicial
    dijkstra(start);

    for(int u = 1; u <= n; u++){
        printf("Distancia de %d para %d: %lld\n", start, u, d[u]);
    }

}
```

4.7 Dsu

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

const int MAX = 1e5+10;

int parent[MAX];

void make(int v){
    parent[v] = v;
}

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

void _union(int a, int b){
    a = find(a);
    b = find(b);
    if (a != b)
        parent[b] = a;
}

int main(){

    return 0;
}
```

4.8 Floyd Warshall

```
/*
Algoritmo de caminho mais curto com todos os pares, o que significa que calcula o caminho mais curto entre todos os pares de ós.
*/

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

const int oo = 100000000; // infinito

int main(){

    int n, m;
    cin>>n>>m;

    vector<vector<int>> dist(n+1, vector<int> (n+1));

    for(int i=0; i<n+1; i++){
        for(int j=0; j<n+1; j++){
            dist[i][j] = oo;
        }
    }

    for(int i=0; i<n +1; i++){
        dist[i][i]=0;
    }

    for(int i=0; i<m; i++){
        int comeca, termina, custo;
        cin>>comeca>>termina>>custo;

            // grafo direcionado
        dist[comeca][termina] = custo;
    }

    for(int k=1; k<=n; k++){ // intermediario
        for(int i=1; i<=n; i++){
            for(int j=1; j<=n; j++){
                //(i,k,j) = ir de i pra j passando por k;

                // relaxar distancia de i pra j
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
        return 0;
}
```

4.9 Kruskall

```
/*
Busca uma árvore geradora ímnima para um grafo conexo com pesos.
*/

#include <iostream>
#include <algorithm>

using namespace std;

struct t_aresta{
    int dis;
    int x, y;
};

bool comp(t_aresta a, t_aresta b){ return a.dis < b.dis; }

//--------------------
#define MAXN 50500
#define MAXM 200200

int n, m; // únmero de évrtices e arestas
t_aresta aresta[MAXM];

// para o union find
int pai[MAXN];
int peso[MAXN];

// a árvore
t_aresta mst[MAXM];
//--------------------

```

```cpp
// çõfunes do union find
int find(int x){
    if(pai[x] == x) return x;
    return pai[x] = find(pai[x]);
}

void join(int a, int b){

    a = find(a);
    b = find(b);

    if(peso[a] < peso[b]) pai[a] = b;
    else if(peso[b] < peso[a]) pai[b] = a;
    else{
        pai[a] = b;
        peso[b]++;
    }

}


int main(){

    // ler a entrada
    cin >> n >> m;

    for(int i = 1;i <= m;i++)
        cin >> aresta[i].x >> aresta[i].y >> aresta[i
    ].dis;


    // inicializar os pais para o union-find
    for(int i = 1;i <= n;i++) pai[i] = i;

    // ordenar as arestas
    sort(aresta+1, aresta+m+1, comp);

    int size = 0;
    for(int i = 1;i <= m;i++){

        if( find(aresta[i].x) != find(aresta[i].y) ){
     // se estiverem em componentes distintas
            join(aresta[i].x, aresta[i].y);

            mst[++size] = aresta[i];
        }

    }

    // imprimir a MST
    for(int i = 1;i < n;i++) cout << mst[i].x << " "
    << mst[i].y << " " << mst[i].dis << "\n";
    return 0;
}
```

4.10 Lca

```cpp
/*
Lowest Common ancestor (LCA) - é o nome ítpico dado
    para o seguinte problema: dado uma Árvore cuja
    raiz é um évrtice áarbitrrio e dois évrtices u,v
    que a pertencem, diga qual é o ón mais baixo(
    relativo a raiz) que é ancestral de u,v.
*/

#include <bits/stdc++.h>
using namespace std;
const int SIZE = 1e5;
int depth[SIZE];
vector<int> graph[SIZE];

void pre_process_depth(int u, int d) {
    depth[u] = d;
    for(auto adj : graph[u]) {
        pre_process_depth(adj, d + 1);
    }
}

int p2k[SIZE][log2(SIZE)+1];
int lca(int u, int v) {
    if(depth[u] < depth[v]) swap(u,v);
    for (int i = 20; i >= 0; --i) {
        if(depth[p2k[u][i]] >= depth[v])
            u = p2k[u][i];
    }
    if(u == v) return u;
    for (int i = 20; i >= 0; --i) {
        if(p2k[v][i] != p2k[u][i]) {
            v = p2k[v][i];
            u = p2k[u][i];
        }
    }
    return pai[v];
}

int climb(int node, int k){
    for(int i = 20; i >= 0; i--) {
        if(k >= (1 << i)) {
            node = p2k(node,i);
            k -= (1 << i);
        }
    }
    return node;
}

int dist(int u, int v){
    return depth[u] + depth[v] -2*depth[lca(u,v)];
}

int main() {
    // codigo
    // le os pais e monta o grafo
    pai[raiz] = raiz;
    pre_proccess_depth(raiz); // tipicamente qual
    vertice é a raiz nao importa
    for(int node = 0; node < SIZE; node++){
        p2k[node][0] = pai[node];
    }
    for(int node = 0; node < SIZE; node++) {
        for(int k = 1; k <= log2(SIZE); k++) {
            p2k[node][k] = p2k[p2k[node][k-1]][k-1];
        }
    }
    // resolve problema
}
```

5 Math

5.1 Combinatoria

```cpp
// quantidade de çõcombinaes ípossveis sem çãrepetio
    de 2 numeros
int comb(int k){
    if(k==1)return 1;
    else if(k==0)return 0;
    return (k*(k-1))/2;
}
```

5.2 Divisores

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

vector<long long> get_divisors(long long n){
    vector<long long> divs;
    for(long long i = 1; i*i <=n; i++){
        if(n%i == 0){
            divs.push_back(i);
            long long j = n/i;
            if(j != i)
                divs.push_back(j);
        }
    }
    return divs;
}
```

5.3 Fatora

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

map<int,int> fatora(int n){
    vector<int> lp;
    map<int,int> exp;
    int count=0;
    while(n>1){
        exp[lp[n]]++;
        n/=lp[n];
    }
    return exp;
}
```

5.4 Mdc

```
// Greatest common divisor / MDC

long long gcd(long long a, long long b){
    return b ? gcd(b, a % b) : a;
}

// or just use __gcd(a,b)
```

5.5 Mmc

```
// Least Common Multiple - MMC
#include <bits/stdc++.h>
using namespace std;

long long lcm(long long a, long long b){
    return (a/__gcd(a,b)*b);
}
```

5.6 Pa

```
// óSomatrio de 1 a K
int pa(int k){
    return (k*(k+1))/2;
}
```

5.7 Primos

```
// PRIMALIDADE

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

const int MAX = 1e5+7;

void crivo(){
    vector<int> crivo(MAX, 1);
    for(int i=2; i*i<=MAX; i++){
        if(crivo[i]==1){
            for(int j=i+i; j<MAX; j+=i){
                crivo[j]=0;
            }
        }
    }
}

bool is_prime(int num){
    for(int i = 2; i*i<= num; i++) {
       if(num % i == 0) {
          return false;
       }
    }
    return true;
}
```

6 Template

6.1 Template

```
#include <bits/stdc++.h>
using namespace std;
//g++ -std=c++17 -O2 -Wall run.cpp -o run

#define endl "\n"
#define sws std::ios::sync_with_stdio(false); cin.tie
    (NULL); cout.tie(NULL);
#define all(x) x.begin(), x.end()
#define input(x) for (auto &it : x) cin >> it
#define print(x) for (auto &it : x) cout << it << ' '
    ; cout<<endl;
#define dbg(msg, x) cout << msg << " = " << x << endl
#define pb push_back
#define mp make_pair
#define ff first
#define ss second
#define TETO(a, b) ((a) + (b-1))/(b)
#define loop(i,a,n) for(int i=a; i < n; i++)
typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> pii;
const ll MOD = 1e9+7;
const int MAX = 1e4+5;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;
//---------------------------------//
//         éF que o AC vem          //
//---------------------------------//

int main(){ sws;


    return 0;
}
```