Competitive Programming Notebook

Contents

1 Range queries **2**
1.1 Segment Tree Comprimida . . . 2
1.2 Segment Tree Base . . . 2
1.3 Prefix Sum 2d . . . 3

2 String **3**
2.1 Suffix Array . . . 3
2.2 Trie . . . 4
2.3 3 Hash . . . 4
2.4 Hash Sem Ordem . . . 4

3 Geometria **5**
3.1 Convex Hull . . . 5
3.2 Ponto . . . 6

4 Buscas e stl **6**
4.1 Subset Sum . . . 6
4.2 Busca Binaria . . . 7
4.3 Kadane . . . 7
4.4 Stack Monotonica . . . 7

5 Matematica **8**
5.1 Matrizes . . . 8
5.2 Fexp E Comuns . . . 8
5.3 Divisores . . . 8
5.4 Fatoracao Prima E Spf . . . 8
5.5 Crivo . . . 9
5.6 Combinacao . . . 9
5.7 Modulo Struct . . . 9

6 Dp **10**
6.1 Operacoes-bitwise . . . 10
6.2 Digit Dp . . . 10
6.3 Knapsack 2d . . . 10
6.4 Knapsack 1d . . . 11
6.5 Lcis . . . 11
6.6 Moedas . . . 11
6.7 Lcs . . . 11
6.8 Tree Matching . . . 12

7 Grafos **12**
7.1 Bicolorabilidade . . . 12
7.2 Prim . . . 12
7.3 Dfs . . . 13
7.4 Bfs . . . 13
7.5 Achar Ciclos . . . 14
7.6 Toposort . . . 14
7.7 Euler Tour E Lca . . . 14
7.8 Djikstra . . . 15
7.9 Dinic . . . 15
7.10 Dsu . . . 16
7.11 Pontes . . . 17
7.12 Floyd Warshall . . . 17
7.13 Scc . . . 17

1 Range queries

1.1 Segment Tree Comprimida

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

void update(int no, int l, int r, int pos, int val, vector<int> &segtree){
    if((pos < l) or (r < pos)){
        return;
    }
    if(l == r){
        segtree[no] += val;
        return;
    }

    int mid = (l + r)/2;
    update(2* no, l, mid, pos, val, segtree);
    update((2* no) + 1, mid + 1, r, pos, val, segtree);
    segtree[no] = segtree[2 * no] + segtree[(2 * no)+ 1];

}

int query(int no, int l, int r, int lq, int rq, vector<int> &segtree){
    if((rq < l) or (r < lq)){
        return 0;
    }
    if((lq <= l) and (r <= rq)){
        return segtree[no];
    }

    int mid = (l + r)/2;
    int ans = query(2 * no, l, mid, lq, rq, segtree);
    ans += query((2 * no) + 1, mid + 1, r, lq, rq, segtree);

    return ans;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, q, sal, k, j;
    char c;
    vector<int> all;
    vector<int> orig;
    vector<tuple<char, int, int>> queries;
    sort(all.begin(), all.end());
    all.erase(unique(all.begin(), all.end()), all.end());
    int range;
    vector<int> segtree(4 * (all.size()), 0);
    for(int i = 0; i < orig.size(); i++){
        range = lower_bound(all.begin(), all.end(), orig[i]) - all.begin();
        update(1, 1, all.size(), range, 1, segtree);
    }

    return 0;
}
```

1.2 Segment Tree Base

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

void build(int no, int l, int r, vector<long long int> &segtree, vector<long long int> &orig){
    if(l == r){
        segtree[no] = orig[l];
        return;
    }

    int mid = (l + r)/2;
    build(2 * no, l, mid, segtree, orig);
    build((2 * no) + 1, mid + 1, r, segtree, orig);
    segtree[no] = segtree[2 * no] + segtree[(2 * no) + 1];
}

void update(int no, int l, int r, int pos, long long int val, vector<long long int> &segtree){
    if((pos < l) or (r < pos)){
        return;
    }
    if(l == r){
        segtree[no] = val;
        return;
    }

    int mid = (l + r)/2;
    update(2* no, l, mid, pos, val, segtree);
    update((2* no) + 1, mid + 1, r, pos, val, segtree);
    segtree[no] = segtree[2 * no] + segtree[(2 * no)+ 1];

}

long long int query(int no, int l, int r, int lq, int rq, vector<long long int> &segtree){
    if((rq < l) or (r < lq)){
        return 0;
    }
    if((lq <= l) and (r <= rq)){
        return segtree[no];
    }

    int mid = (l + r)/2;
    long long int ans = query(2 * no, l, mid, lq, rq, segtree);
    ans += query((2 * no) + 1, mid + 1, r, lq, rq, segtree);

    return ans;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int n;
    vector<long long int> segtree(4 * n);
    //ADICIONAR X EM L E -X EM R+1 EH IGUAL A ADICIONAR X EM [L, R]
    //SEGTREE PARA ACHAR MENORES:
    // for(int i = 0; i < n; i++){
    //     range = upper_bound(mansort.begin(), mansort.end(), man[i]) - mansort.begin();
    //     dir[i] = query(1, 1, n, range, n, segtree);
    //     update(1, 1, n, range, 1, segtree);
    // }

    //SegEuler -> usar updates ao inves do build (ou mapear vetor a[euler_in[i]] - > orig[i]
    // query em euler_in[i], euler_out[i]
    // update em euler_in[i]

    return 0;
}
```

1.3 Prefix Sum 2d

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int xi, yi, yf, xf, n, q;
    vector<vector<int>> prefsum(n + 1, vector<int>(n + 1, 0));
    vector<vector<int>> floresta;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            prefsum[i][j] = floresta[i - 1][j - 1] + prefsum[i - 1][j] + prefsum[i][j - 1] - prefsum[i - 1][j - 1];
        }
    }
    for(int i = 0; i < q; i++){
        cin >> yi;
        cin >> xi;
        cin >> yf;
        cin >> xf;


        cout << (prefsum[yf][xf] - prefsum[yf][xi - 1] - prefsum[yi - 1][xf] + prefsum[yi - 1][xi - 1]) << endl;
    }

    return 0;
}
```

2 String

2.1 Suffix Array

```cpp
#include <bits/stdc++.h>
#define endl '\n'
#define ll long long int
using namespace std;

struct SA{
    int sz;
    vector<ll> suffarr, endrank, lcp;
    SA(int s){
        sz = s;
        suffarr = vector<ll>(sz+1); //espaÃ§o p/ sentinela vai ser apagado
        endrank = vector<ll>(sz);
        lcp = vector<ll>(sz);
    }
    void suffarray(string &s){
        //ajuste a sentinela se necessario
        s.push_back(char(0));
        const ll n = s.size();
        vector<ll> rank(n), temp(n);
        vector<pair<ll, ll>> etc(n);
        for(int i = 0; i < n; i++){
            etc[i] = {s[i], i};
        }
        sort(etc.begin(), etc.end());
        for(int i = 0; i < n; i++){
            suffarr[i] = etc[i].second;
        }
        rank[suffarr[0]] = 0;
        for(int i = 1; i < n; i++){
            rank[suffarr[i]] = rank[suffarr[i-1]] + (etc[i].first != etc[i-1].first);
        }

        for(int k = 1; k < n; k *= 2){
            int classes = rank[suffarr[n-1]] + 1;
            vector<ll> cnt(classes, 0);
            for(int i = 0; i < n; i++) cnt[rank[i]]++;

            vector<ll> pos(classes, 0);
            for(int i = 1; i < classes; i++) pos[i] = pos[i-1] + cnt[i-1];

            for(int i = 0; i < n; i++){
                int j = suffarr[i] - k;
                if(j < 0) j+=n;
                temp[pos[rank[j]]++] = j;
            }
            suffarr = temp;

            temp[suffarr[0]] = 0;
            for(int i = 1; i < n; i++){
                pair<ll, ll> prev = {rank[suffarr[i-1]], rank[(suffarr[i-1] + k) % n]};
                pair<ll, ll> curr = {rank[suffarr[i]], rank[(suffarr[i] + k) % n]};

                temp[suffarr[i]] = temp[suffarr[i-1]] + (curr != prev);
            }
            rank = temp;
        }
        s.pop_back();
        suffarr.erase(suffarr.begin());
        return;
    }

    //lcp[i] = lcp de i-1 e i; lcp[0] deve ser ignorado
    void lcpv(string &s){
        s.push_back(char(0));
        int n = suffarr.size();

        for(int i = 0; i < n; i++){
            endrank[suffarr[i]] = i;
        }
        ll k = 0;

        for(int i = 0; i < n; i++){
            if(endrank[i] == 0) lcp[0] = 0;
            else{
                ll j = suffarr[endrank[i] - 1];
                while(s[i+k] == s[j+k]) k++;
                lcp[endrank[i]] = k;
                if(k > 0) k--;
            }
        }
        s.pop_back();
        return;
    }

    //retorna tamanho do lcp entre sufixos da posicao i e j
    //so pra mostrar que eh uma query de minimo no lcp
    //ao inves disso use seg
    ll get_lcp(int i, int j){
        int l = endrank[i];
        int r = endrank[j];
        if(l > r) swap(l, r);
        if(i==j) return (sz - suffarr[l]);

        ll ans = lcp[l+1];
        for(int k = l+2; k <= r; k++){
            ans = min(lcp[k], ans);
        }
        return ans;
    }
};
```

2.2 Trie

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

struct Node{
    int next[26];
    int subtree = 0;
};

void add(string s, vector<Node> &trie){
    int curr = 0;

    for(auto c : s){
        if(trie[curr].next[c - 'a'] == 0){
            trie[curr].next[c - 'a'] = trie.size();
            trie.push_back(Node());
        }

        trie[curr].subtree += 1;
        curr = trie[curr].next[c - 'a'];
    }
    trie[curr].subtree += 1;
}

int query(string s, vector<Node> &trie){
    int curr = 0;

    for(auto c : s){
        if(trie[curr].next[c - 'a'] == 0){
            return 0;
        }
        curr = trie[curr].next[c - 'a'];
    }
    return trie[curr].subtree;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    vector<Node> trie(1);
    //trie pode ser modificada com DFS para propagar
    mudancas
    //TRIE DE XOR -> max(busca diferentes) e min(
    busca igual)

    return 0;
}
```

2.3 3 Hash

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);


    const long long int mod = 1000000009;
    const long long int mod2 = 1000000007;
    const long long int mod3 = 999999937;
    string s;
    cin >> s;
    const long long int k = 277;
    const long long int l = 149;
    const long long int p = 37;
    vector<long long int> pot(s.size() + 1);
    vector<long long int> pot2(s.size() + 1);
    pot[0] = 1;
    pot2[0] = 1;
    for(int p = 1; p <= s.size(); p++){
        pot[p] = (pot[p - 1] * k) % mod;
        pot2[p] = (pot2[p - 1] * l) % mod2;
    }
    vector<long long int> hashupto1(s.size());
    vector<long long int> hashupto2(s.size());
    hashupto1[0] = s[0];
    hashupto2[0] = s[0];
    for(int i = 1; i < s.size(); i++){
        hashupto1[i] = ((hashupto1[i - 1] * k) % mod)
     + s[i];
        hashupto1[i] = hashupto1[i] % mod;
        hashupto2[i] = ((hashupto2[i - 1] * l) % mod2
    ) + s[i];
        hashupto2[i] = hashupto2[i] % mod2;
    }
    //hash(l..r) = pref(r) - (pref(l - 1) * (k^(r-l
    +1))) % MOD
    //aa = hashupto1[i + (pref.size() - 1)] - ((
    hashupto1[i - 1] * pot[pref.size()]) % mod);
    //aa = (((aa % mod) + mod) % mod);
    //bb = hashupto2[i + (pref.size() - 1)] - ((
    hashupto2[i - 1] * pot2[pref.size()]) % mod2);
    //bb = (((bb % mod2) + mod2) % mod2);
    //achar periodos facil: hash(0..(n-p)) == hash(p
    ...n) -> so se n for parcial


    return 0;
}
```

2.4 Hash Sem Ordem

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;



int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    const long long int k = 277;
    const long long int l = 149;
    const long long int p = 37;

    const long long int mod = 1000000009;
    const long long int mod2 = 1000000007;
    const long long int mod3 = 999999937;

    int n, q;
    cin >> n;
    cin >> q;
    vector<long long int> num(n);
    vector<long long int> bnum(n);
    vector<long long int> pot(n+1);
    vector<long long int> pot2(n+1);
    vector<long long int> pot3(n+1);
    pot[0] = 1;
    pot[1] = 1;
```

```cpp
    pot[2] = 1;
    for(int po = 1; po <= n; po++){
        pot[po] = (pot[po - 1] * k) % mod;
        pot2[p] = (pot2[po - 1] * l) % mod2;
        pot3[po] = (pot3[po - 1] * p) % mod3;
    }
    for(int i = 0; i < n; i++){
        cin >> num[i];
    }
    for(int i = 0; i < n; i++){
        cin >> bnum[i];
    }
    vector<long long int> hashsuml(n+1);
    vector<long long int> hashsumk(n+1);
    vector<long long int> hashsump(n+1);
    hashsumk[0] = 0;
    hashsuml[0] = 0;
    hashsump[0] = 0;
    hashsumk[1] = (num[0] * pot[num[0]]) % mod;
    hashsuml[1] = (num[0] * pot2[num[0]]) % mod2;
    hashsump[1] = (num[0] * pot3[num[0]]) % mod3;

    vector<long long int> bhashsuml(n+1);
    vector<long long int> bhashsumk(n+1);
    vector<long long int> bhashsump(n+1);
    bhashsumk[0] = 0;
    bhashsuml[0] = 0;
    bhashsump[0] = 0;
    bhashsumk[1] = (bnum[0] * pot[bnum[0]]) % mod;
    bhashsuml[1] = (bnum[0] * pot2[bnum[0]]) % mod2;
    bhashsump[1] = (bnum[0] * pot3[bnum[0]]) % mod3;
    for(int i = 1; i < n; i++){
        hashsumk[i+1] = ((hashsumk[i] + ((num[i] * pot[num[i]]) % mod)) % mod);
        hashsuml[i+1] = ((hashsuml[i] + ((num[i] * pot2[num[i]]) % mod2)) % mod2);
        hashsump[i+1] = (hashsump[i] + ((num[i] * pot3[num[i]]) % mod3)) % mod3;
    }
    for(int i = 1; i < n; i++){
        bhashsumk[i+1] = ((bhashsumk[i] + ((bnum[i] * pot[bnum[i]]) % mod)) % mod);
        bhashsuml[i+1] = ((bhashsuml[i] + ((bnum[i] * pot2[bnum[i]]) % mod2)) % mod2);
        bhashsump[i+1] = (bhashsump[i] + ((bnum[i] * pot3[bnum[i]]) % mod3)) % mod3;
    }

    return 0;
}
```

3 Geometria

3.1 Convex Hull

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

struct Point{
    //alterar tipos se preciso
    long long int x, y;
    Point(long long int x, long long int y){
        this->x = x;
        this->y = y;
    }
    Point operator+(Point o){ return Point(x + o.x, y + o.y); }
    Point operator-(Point o){ return Point(x - o.x, y - o.y); }
    Point operator*(long long int k){ return Point(k*x, k*y); }
    double len(){ return hypot(x, y); }
    long long int cross(Point o){ return ((x * o.y) - (y*o.x)); }
    bool operator<(Point o){ return(tie(x, y) < tie(o.x, o.y)); }
    bool operator==(Point o){ return (tie(x, y) == tie(o.x, o.y)); }
};

int orientation(Point a, Point b, Point c){
    Point ab = b - a;
    Point bc = c - b;
    long long int v = ab.cross(bc);

    if(v < 0){
        return -1; //horario
    }
    if(v > 0){
        return 1; //antihorario
    }
    return 0;
}

bool cw(Point a, Point b, Point c, bool include_collinear){
    int o = orientation(a, b, c);
    return ((o < 0) || (include_collinear && (o == 0)));
}

bool ccw(Point a, Point b, Point c, bool include_collinear){
    int o = orientation(a, b, c);
    return ((o > 0) || (include_collinear && (o == 0)));
}

void convex_hull(vector<Point> &a, bool include_collinear=false){
    if(a.size() == 1){
        return;
    }
    sort(a.begin(), a.end());
    Point p1 = a[0];
    Point p2 = a.back();
    vector<Point> up, down;
    up.push_back(p1);
    down.push_back(p1);

    for(int i = 1; i < (int)a.size(); i++){
        if((i == a.size() - 1) || (cw(p1, a[i], p2, include_collinear))){
            while((up.size() >= 2) && !(cw(up[up.size() - 2], up[up.size() - 1], a[i], include_collinear))){
                up.pop_back();
            }
            up.push_back(a[i]);
        }
        if((i == a.size() - 1) || (ccw(p1, a[i], p2, include_collinear))){
            while((down.size() >= 2) && !(ccw(down[down.size() - 2], down[down.size() - 1], a[i], include_collinear))){
                down.pop_back();
            }
            down.push_back(a[i]);
        }
    }
    if(include_collinear && (up.size() == a.size())){
        reverse(a.begin(), a.end());
        return;
    }
    a.clear();
    for(int i = 0; i < (int)up.size(); i++){
        a.push_back(up[i]);
    }
    for(int i = down.size() - 2; i > 0; i--){
        a.push_back(down[i]);
    }
}

bool insidetriangle(Point a, Point b, Point c, Point point) {
    long long int s1 = abs((b-a).cross(c-b));
    long long int area1 = abs((point - a).cross(point - b));
    long long int area2 = abs((point - b).cross(point - c));
    long long int area3 = abs((point - c).cross(point - a));
    long long int s2 = area1 + area2 + area3;
    return s1 == s2;
}

bool isinside(vector<Point> &hull, Point p){
    int n = hull.size();
    if(n == 1){
        return (hull.front() == p);
    }
    int l = 1;
    int r = n - 1;
    int mid;
    while(abs(r - l) > 1){
        mid = (r+l)/2;
        Point tomid = hull[mid] - hull[0];
        Point topoint = p - hull[0];
        if(topoint.cross(tomid) < 0){
            //a esquerda
            r = mid;
        } else{
            l = mid;
        }
    }
    //Point vec = hull[r] - hull[l];
    //Point tovec = p - hull[l];
    //return (tovec.cross(vec) > 0);
    return insidetriangle(hull[0], hull[l], hull[r], p);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

3.2 Ponto

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

//trocar os long longs por outro tipo desejado, ou:
//template<typename T>
struct Point{
    long long int x, y;
    Point(long long int x, long long int y){
        this->x = x;
        this->y = y;
    }
    Point operator+(Point o){ return Point(x + o.x, y + o.y); }
    Point operator-(Point o){ return Point(x - o.x, y - o.y); }
    Point operator*(long long int k){ return Point(k*x, k*y); }
    double len(){ return hypot(x, y); }
    long long int cross(Point o){ return ((x * o.y) - (y*o.x)); }
    bool operator<(Point o){ return(tie(x, y) < tie(o.x, o.y)); }
    bool operator==(Point o){ return (tie(x, y) == tie(o.x, o.y)); }
};

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t;
    long long int x1, x2, x3, y1, y2, y3;
    for(int k = 0; k < t; k++){
        Point p1 = Point(x2 - x1, y2 - y1);
        Point p2 = Point(x3 - x1, y3 - y1);
        //produto vetorial
        long long int check = p2.cross(p1);
        if(check == 0){
            cout << "TOUCH" << endl;
        } else if(check > 0){
            cout << "RIGHT" << endl;
        } else{
            cout << "LEFT" << endl;
        }
    }

    //numerador = abs(((y2 - y) * cx) - ((x2 - x) * cy) + (x2 * y) - (y2 * x));
    //denominador = sqrt(((y2 - y) * (y2 - y)) + ((x2 - x) * (x2 - x)));
    //dist = numerador/denominador;
    //distancia entre reta formada pelos pontos (x, y) e (x2, y2) ate o ponto (cx, cy)

    return 0;
}
```

4 Buscas e stl

4.1 Subset Sum

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

void subsetsum(int n, vector<int> &all, vector<long long int> &atual){
    if(n == all.size()){
        //verificacao
        return;
    }

    subsetsum(n + 1, all, atual);
    atual.push_back(all[n]);
    subsetsum(n + 1, all, atual);
    atual.pop_back();
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    //subset sum com bitmask
    int n, arr[100];
    long long int sum;
    vector<long long int> vec;
    for (int i = 0; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            if (i & (1 << j)) {
                sum += arr[j];
            }
        }
        vec.push_back(sum);
        sum = 0;

    }


    return 0;
}
```

4.2 Busca Binaria

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, p;
    int l = 0;
    int r = n - 1;
    int mid;
    bool pos;
    int maximpossivel = 0;
    vector<int> sortado;
    while(r >= l){
        mid = l + (r - l)/2;
        pos = true;

        //CHECAGEM

        if(!pos){
            l = mid + 1;
            maximpossivel = max(maximpossivel, mid);
        } else{
            r = mid - 1;
        }
    }
    upper_bound(sortado.begin(), sortado.end(), p);
    //PRIMEIRO ELEMENTO >= P
    lower_bound(sortado.begin(), sortado.end(), p);
    //PRIMEIRO ELEMENTO > P
    //subtrair .begin() retorna indice
    //subtrair upper do lower retorna quantidade

    return 0;
}
```

4.3 Kadane

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

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int len, elemento;
    long long maxsum, prevsum;
    int arr[len];
    prevsum = arr[0];
    maxsum = arr[0];
    //maior soma em subarray
    for (int j = 1; j < len; j++){
        if((prevsum + arr[j]) < arr[j]){
            prevsum = arr[j];
        } else{
            prevsum += arr[j];
        }

        if (prevsum > maxsum){
            maxsum = prevsum;
        }
    }
    cout << maxsum << endl;

    return 0;
}
```

4.4 Stack Monotonica

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    vector<long long int> tabuas(n);
    stack<long long int> monotesq;
    stack<long long int> monotdir;
    vector<long long int> mindir(n);
    vector<long long int> minesq(n);

    for(int i = 0; i < n; i++){
        while((monotesq.size() > 0) and (tabuas[i] <
    tabuas[monotesq.top()])){
            monotesq.pop();
        }

        if(monotesq.size() > 0){
            minesq[i] = monotesq.top();
        } else{
            minesq[i] = -1;
        }
        monotesq.push(i);
    }
    //GUARDA MENOR MAIS PROXIMO A ESQUERDA

    for(int i = 1; i <= n; i++){
        while((monotdir.size() > 0) and (tabuas[n - i
    ] <= tabuas[monotdir.top()])){
            monotdir.pop();
        }

        if(monotdir.size() > 0){
            mindir[n - i] = monotdir.top();
        } else{
            mindir[n - i] = n;
        }
        monotdir.push((n - i));
    }
    //VERSAO INVERTIDA

    long long int ar;
    long long int maxarea = 0;
    for(int i = 0; i < n; i++){
        ar = (mindir[i] - minesq[i] - 1) * tabuas[i];
        maxarea = max(ar, maxarea);
    }
    cout << maxarea << endl;


    return 0;
}
```

5 Matematica

5.1 Matrizes

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

vector<vector<long long int>> mult(vector<vector<long long int>> &a, vector<vector<long long int>> &b, long long int MOD){
    vector<vector<long long int>> res(a.size(), vector<long long int>(b[0].size()));

    for(int i = 0; i < a.size(); i++){
        for(int j = 0; j < b[0].size(); j++){
            res[i][j] = 0;
            for(int k = 0; k < a[0].size(); k++){
                res[i][j] += (a[i][k] * b[k][j]) % MOD;
                res[i][j] = res[i][j] % MOD;
            }
        }
    }

    return res;
}

vector<vector<long long int>> fexp(vector<vector<long long int>> &a, long long int e, long long int MOD){
    vector<vector<long long int>> ans(4, vector<long long int>(4, 0));
    ans[0][0] = 1;
    ans[1][1] = 1;
    ans[2][2] = 1;
    ans[3][3] = 1;

    while(e){
        if(e & 1){
            ans = mult(a, ans, MOD);
        }
        a = mult(a, a, MOD);
        e >>= 1;
    }

    return ans;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    //recorrencia em matrizes
    //matrix T * matrix (f0) = matrix (f2)
    //                  (f1)          (f1)

    return 0;
}
```

5.2 Fexp E Comuns

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

const long long int mod = 1000000007;

long long int fexp(long long int a, long long int b){
    long long int ans = 1;
    while(b != 0){
        if(b & 1){
            ans = (ans * a) % mod;
        }
        a = (a * a) % mod;
        b >>= 1;
    }
    return ans;
}

long long int gcd(long long int a, long long int b){
    if(!b){
        return a;
    } else{
        return gcd(b, a % b);
    }
    //ja implementado em __gcd()
}

long long int lcm(long long int a, long long int b){
    return ((a*b)/gcd(a,b);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;

}
```

5.3 Divisores

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    vector<int> divs;
    for(int i = 1; (i * i)<=n; i++){
        if(n % i == 0){
            divs.push_back(i);
            if(i != n/i){
                divs.push_back(n/i);
            }
        }
    }

    return 0;
}
```

5.4 Fatoracao Prima E Spf

```
#include <bits/stdc++.h>
#define endl '\n'
#define ll long long int
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    ll maxn;
    //SHORTEST PRIME FACTOR
    vector<ll> spf(maxn+1, 0);
    ll curr;
    for(ll i = 2; i <= maxn; i++){
        if(spf[i] == 0){
            spf[i] = i;
            curr = i*i;
            while(curr <= maxn){
                if(spf[curr] == 0){
                    spf[curr] = i;
                }
                curr += i;
            }
        }
    }

    //FATORACAO MELHORADA COM SPF -> nlogn
    ll v;
    vector<map<ll, int>> fat(maxn+1);
    for(int i = 2; i <= maxn; i++){
        v = i;
        while(v > 1){
            porfavor[i][spf[v]]++;
            v = v/spf[v];
        }
    }

    //FATORACAO PRIMA PADRAO -> n * sqrt(n)
    int n;
    vector<int> primos;
    map<int, int> freq;
    for(int i = 2; i*i <= n; i++){
        int cnt = 0;
        while(n % i == 0){
            n /= i;
            cnt++;
        }
        if(cnt > 0){
            freq[i] += cnt;
            primos.push_back(i);
        }
    }
    if(n > 1){
        primos.push_back(n);
        freq[n]++;
    }

    return 0;
}
```

5.5 Crivo

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int lim;
    vector<bool> isprime;
    isprime[0] = false;
    isprime[1] = false;
    vector<int> primes;
    //crivo base, acha primos
    for(int i = 2; i < lim; i++){
        if(isprime[i]){
            primes.push_back(i);
            for(int j = i*2; j < lim; j+=i){
                isprime[j] = false;
            }
        }
    }

    //crivo da soma dos divisores no intervalo
    vector<int> sumdivisor;
    for(int i = 1; i < lim; i++){
        for(int j = i; j < lim; j+= i){
            sumdivisor[j] += i;
        }
    }

    //crivo da quantidade de divisore dos nÃžmeros no
     intervalo
    vector<int> numdivisors;
    for(int i = 1; i < lim; i++){
        for(int j = i; j < lim; j+= i){
            numdivisors[j]++;
        }
    }

    return 0;
}
```

5.6 Combinacao

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

const long long int mod = 1000000007;

long long int comb(long long int n, long long int i){
    long long int denom = 1;
    long long int num = 1;

    for(int j = 0; j < i; j++){
        num *= (n - j);
        num /= (j + 1);
    }
    //COMBINACAO ITERATIVA
    return num;
}

long long int fexp(long long int a, long long int b);

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    long long int n, m;
    vector<long long int> fatn(2000100);
    fatn[0] = 1;
    for(long long int i = 1; i < fatn.size(); i++){
        fatn[i] = (fatn[i - 1] * i) % mod;
    }
    //combinacao = (n!/(i!*(n-i)))
    //combinacao com repeticao C(n, i) = C(n + i - 1,
     i);
    long long int aa = ((fatn[m] * fatn[n - 1])) %
    mod;
    long long int bb = fexp(aa, mod - 2);
    long long int combrep = (fatn[n + m - 1] * bb) %
    mod;

    long long int comb = ((fatn[n])/(fatn[n-m] * fatn
    [m]));
    //para operacoes com modulo eh preciso ao inves
    de dividir, multiplicar
    //pelo inverso modular (n ^ mod-2)

    return 0;
}
```

5.7 Modulo Struct

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

```

```cpp
const ll mod = 1e9 + 7;

struct mint{
    ll val;
    mint(){
        this->val = 0;
    }
    mint(ll val){
        this->val = ((val % mod) + mod) % mod;
    }

    mint operator+(mint v){
        return ((((val + v.val) % mod) + mod) % mod);
    }
    mint operator-(mint v){
        return ((((val - v.val) % mod) + mod) % mod);
    }
    mint operator*(mint v){
        return ((((val * v.val) % mod) + mod) % mod);
    }
    friend ostream& operator<<(ostream& os, const mint& m) {
        return os << m.val;
    }

    friend istream& operator>>(istream& is, mint& m) {
        ll x;
        is >> x;
        m = mint(x);
        return is;
    }
};
```

6 Dp

6.1 Operacoes-bitwise

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;


int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    //2^n = (1<<n)
    int n, i, mask;
    for(int mask = 0; mask <(1<<n); mask++);
    //iterar pela mask n

    if(mask&(1<<i)); //se bit i for 1
    mask = mask|(1<<i); ///ligar bit i
    mask = mask^(1<<i); //flipar bit i

    return 0;
}
```

6.2 Digit Dp

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

//dp para numeros sem 2 digitos adjacentes iguais
long long int rec(int pos, int lastdig, bool start, bool smaller, vector<int> &num, vector<vector<vector<vector<long long int>>>> &dp){
    if(pos >= num.size()){
        return 1;
    }

    if(dp[pos][lastdig][start][smaller] != -1){
        return dp[pos][lastdig][start][smaller];
    }

    long long int ans = 0;
    if(smaller){
        for(int i = 0; i <= 9; i++){
            if(start){
                if(i > 0){
                    ans += rec(pos + 1, i, false, smaller, num, dp);
                } else{
                    ans += rec(pos + 1, i, true, smaller, num, dp);
                }
            } else{
                if(i != lastdig){
                    ans += rec(pos + 1, i, start, smaller, num, dp);
                }
            }
        }
    } else{
        for(int i = 0; i <= num[pos]; i++){
            if(start){
                if(i > 0){
                    ans += rec(pos + 1, i, false, !(i == num[pos]), num, dp);
                } else{
                    ans += rec(pos + 1, i, true, !(i == num[pos]), num, dp);
                }
            } else{
                if(i != lastdig){
                    ans += rec(pos + 1, i, start, !(i == num[pos]), num, dp);
                }
            }
        }
    }
    dp[pos][lastdig][start][smaller] = ans;
    return dp[pos][lastdig][start][smaller];

}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    //digit dp: iterar pelos digitos
    //lembrar da ideia base: quando chegar na posiÃğÃčo
    //nÃčo pode mudar o valor!!

    return 0;
}
```

6.3 Knapsack 2d

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

long long int knapsack(vector<tuple<long long int, long long int>> &itens, vector<vector<long long int>> &dp, long long int w, int i){
    if(w == 0){
        return 0;
    }
    if(i >= itens.size()){
        return 0;
    }
    if(dp[w][i] != -1){
        return dp[w][i];
    }

    long long int ans = knapsack(itens, dp, w, i + 1);
    if(get<0>(itens[i]) <= w){
        ans = max(ans, (get<1>(itens[i]) + knapsack(itens, dp, (w - get<0>(itens[i])), i + 1)));
    }
    dp[w][i] = ans;
    return ans;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

6.4 Knapsack 1d

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

long long int knap(map<int, int> &custo, vector<long long int> &dp, int w){
    if(w == 0){
        return 0;
    }

    if(dp[w] != -1){
        return dp[w];
    }

    long long int ans = 0;
    for(auto i : custo){
        if((w - i.first) >= 0){
            ans = max(ans, (knap(custo, dp, (w - i.first))) + i.second);
        }
    }
    dp[w] = ans;
    return dp[w];
}


int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    //mesmos principios da digit, lembre-se do kongey donk
    //estados, mudanÃğas, dp[i][j] -> dp[i- 1][j], dp[i - 1][j + 1], dp[i - 1][j - 1]

    return 0;
}
```

6.5 Lcis

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, pos;
    cin >> n;
    vector<int> v(n);
    for(int i = 0; i < n; i++){
        cin >> v[i];
    }
    vector<int> aux;
    aux.push_back(v[0]);
    for(int i = 1; i < n; i++){
        pos = upper_bound(aux.begin(), aux.end(), v[i]) - aux.begin();
        if(pos >= aux.size()){
            if(v[i] != aux[pos - 1]){
                aux.push_back(v[i]);
            }
        } else{
            if(v[i] != aux[pos - 1]){
                aux[pos] = v[i];
            }
        }
    }
    cout << aux.size() << endl;

    return 0;
}
```

6.6 Moedas

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

//dp das moedas
long long int coinsum(vector<long long int> &dp, vector<long long int> &moedas, long long int w){
    if(w == 0){
        return 0;
    }
    if(dp[w] != -1){
        return dp[w];
    }

    long long int ans = INT_MAX;
    for(int i = 0; i < moedas.size(); i++){
        if(w - moedas[i] >= 0){
            ans = min(ans, coinsum(dp, moedas, w - moedas[i]) + 1);
        }
    }
    dp[w] = ans;
    return dp[w];
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

6.7 Lcs

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

const int MAXN = 5*1e3 + 5;
int memo[MAXN][MAXN];

string s, t;

inline int LCS(int i, int j){
    if(i == s.size() || j == t.size()) return 0;
    if(memo[i][j] != -1) return memo[i][j];

    if(s[i] == t[j]) return memo[i][j] = 1 + LCS(i+1, j+1);

    return memo[i][j] = max(LCS(i+1, j), LCS(i, j+1));
}

int LCS_It(){
    for(int i=s.size()-1; i>=0; i--)
        for(int j=t.size()-1; j>=0; j--)
            if(s[i] == t[j])
                memo[i][j] = 1 + memo[i+1][j+1];
            else
                memo[i][j] = max( memo[i+1][j], memo[i][j+1] );

    return memo[0][0];
}

string RecoverLCS(int i, int j){
    if(i == s.size() || j == t.size()) return "";

    if(s[i] == t[j]) return s[i] + RecoverLCS(i+1, j+1);

    if(memo[i+1][j] > memo[i][j+1]) return RecoverLCS(i+1, j);

    return RecoverLCS(i, j+1);
}
//creditos para SamuellH12
/*****************************
LCS - Longest Common Subsequence

Complexity: O(N^2)

* Recursive:
memset(memo, -1, sizeof memo);
LCS(0, 0);

* Iterative:
LCS_It();

* RecoverLCS
  Complexity: O(N)
  Recover one of all the possible LCS
******************************/
```

6.8 Tree Matching

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

void tmaux(int n, int p, vector<vector<long long int>> &dp, vector<vector<int>> &adj){
    long long int sum1 = 0;
    long long int sum2 = 0;

    for(auto v : adj[n]){
        if(v != p){
            tmaux(v, n, dp, adj);
            sum1 += max(dp[0][v], dp[1][v]);
        }
    }
    dp[0][n] = sum1;

    for(auto v : adj[n]){
        if(v != p){
            sum2 = max(sum2, (1 + dp[0][v] + dp[0][n] - max(dp[0][v], dp[1][v])));
        }
    }
    dp[1][n] = sum2;
}

long long int tmatch(int n, vector<vector<long long int>> &dp, vector<vector<int>> &adj){
    tmaux(n, -1, dp, adj);
    return max(dp[0][1], dp[1][1]);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

7 Grafos

7.1 Bicolorabilidade

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

bool dfs(int v, vector<vector<int>> &adj, vector<bool> &visitado, vector<bool> &cor){
    visitado[v] = true;

    for(auto u : adj[v]){
        if(!visitado[u]){
            cor[u] = !cor[v];
            if(!dfs(u, adj, visitado, cor)){
                return false;
            }
        } else if(cor[u] == cor[v]){
            return false;
        }
    }
    return true;
}

void dfs2(int v, long long int dist, vector<bool> &cor, vector<bool> &visitado, vector<vector<tuple<int, long long int>>> &adj){
    visitado[v] = true;
    for(auto u : adj[v]){
        if(!visitado[get< 0>(u)]){
            if((dist + get<1>(u)) % 2 == 0){
                cor[get<0>(u)] = cor[1];
            } else{
                cor[get<0>(u)] = !cor[1];
            }
            dfs2(get<0>(u), dist + get<1>(u), cor, visitado, adj);
        }
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

7.2 Prim

```cpp
#include <bits/stdc++.h>
#define endl '\n'
```

```cpp
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m, a, b;
    long long int c;
    vector<vector<tuple<long long int, int>>> adj(n + 1);
    vector<int> na_arvore;
    vector<bool> visitado(n + 1, false);
    priority_queue<tuple<long long int, int>> minheap;
    for(int i = 0; i < m; i++){
        adj[a].push_back(tuple(-(c), b));
        adj[b].push_back(tuple(-(c), a));
    }
    minheap.push(tuple(0, 1));
    long long int custo, d;
    int o;
    custo = 0;
    while(minheap.size() > 0){
        d = get<0>(minheap.top());
        o = get<1>(minheap.top());
        minheap.pop();

        if(!visitado[o]){
            visitado[o] = true;
            na_arvore.push_back(o);
            custo += -(d);

            for(auto v : adj[o]){
                if(!visitado[get<1>(v)]){
                    minheap.push(v);
                }
            }
        }
    }

    //OBS: KRUSKALL
    // sort arestas
    // for custo u, v em arestas:
    //  if(find(u) != find(v)):
    //      join(u, v)
    //      total = total + custo
    //prim expande uma arvore, kruskall cria e vai juntando

    return 0;
}
```

7.3 Dfs

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

void dfs(int n, vector<vector<int>> &adj, vector<bool> &vis){
    vis[n] = true;
    for(auto i : adj[n]){
        if(!vis[i]){
            dfs(i, adj, vis);
        }
    }

    return;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

7.4 Bfs

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m;
    string s;
    tuple<int, int, int> start;
    char c;
    vector<vector<char>> mapa(n);
    queue<tuple<int, int, int>> fila;
    vector<vector<bool>> visitado(n, vector<bool>(m, false));

    visitado[get<0>(start)][get<1>(start)] = true;
    fila.push(start);
    tuple<int, int> end;
    int x, y, camada, dist;
    dist = 0;
    while(fila.size() > 0){
        y = get<0>(fila.front());
        x = get<1>(fila.front());
        camada = get<2>(fila.front());
        fila.pop();

        if(mapa[y][x] == 'B'){
            dist =  camada;
            end = tuple(y, x);
            break;
        }

        if((y + 1 < n)){
            if(!visitado[y + 1][x]){
                visitado[y + 1][x] = true;
                fila.push(tuple(y + 1, x, camada + 1));
            }
        }

        if((y - 1 >= 0)){
            if(!visitado[y - 1][x]){
                visitado[y - 1][x] = true;
                fila.push(tuple(y - 1, x, camada + 1));
            }
        }

        if((x + 1 < m)){
            if(!visitado[y][x + 1]){
                visitado[y][x + 1] = true;
                fila.push(tuple(y, x + 1, camada + 1));
            }
        }

        if((x - 1 >= 0)){
            if(!visitado[y][x - 1]){
                visitado[y][x - 1] = true;
                fila.push(tuple(y, x - 1, camada + 1));
            }
        }

    }
    //DIAMETRO DA ARVORE:
    // ACHAR PONTO U MAIS DISTANTE DE INICIAL
    // ACHAR PONTO V MAIS DISTANTE DE U
    // DIAMETRO SERA U, V

    //LEMBRAR DE MULTISOURCE

    return 0;
}
```

7.5 Achar Ciclos

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int dfs(bool &cicblock, int &cicstart, int cidade,
    int anterior, vector<int> &ciclo, vector<bool> &
    visitado, vector<vector<int>> &adj){
    if(visitado[cidade]){
        if(cicstart == 0){
            ciclo.push_back(cidade);
            cicstart = cidade;
        }
        return cidade;
    } else{
        int fim = 0;
        visitado[cidade] = true;

        for(auto i : adj[cidade]){
            if(i != anterior){
                fim = dfs(cicblock, cicstart, i,
    cidade, ciclo, visitado, adj);

                if(cidade == cicstart){
                    //ciclo.push_back(cidade);
                    cicblock = true;
                }

                if(fim != -1){
                    if(!cicblock){
                        ciclo.push_back(cidade);
                        return cidade;
                    }
                }
            }
        }

        return -1;
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    //def dfs(atual, anterior):
    //  if(visitado[atual]): return
    //  visitado[atual] = true
    //  for nxt in adj[atual]:
    //      if(nxt != anterior):
    //          fim = dfs(nxt, atual)
    //          if(fim != -1): ciclo.adiciona(atual)
    //          if(fim == atual OU  fim ==
    JA_TERMINOU): retorne JA_TERMINOU
    //
    //  retorne -1

    return 0;
}
```

7.6 Toposort

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m, a, b;
    vector<int> grau_entrada(n + 1, 0);
    vector<vector<int>> requisitos(n + 1);
    for(int i = 0; i < m; i++){
        cin >> a;
        cin >> b;
        requisitos[a].push_back(b);
        grau_entrada[b] += 1;
    }

    queue<int> fila;
    for(int i = 1; i <= n; i++){
        if(grau_entrada[i] == 0){
            fila.push(i);
        }
    }

    vector<int> toposort;
    int u;
    while(fila.size() > 0){
        u = fila.front();
        fila.pop();

        toposort.push_back(u);
        for(auto v : requisitos[u]){
            grau_entrada[v]--;
            if(grau_entrada[v] == 0){
                fila.push(v);
            }
        }
    }

    if(toposort.size() == n){
        for(int i = 0; i < n; i++){
            cout << toposort[i] << " ";
        }
        cout << endl;
    } else{
        cout << "IMPOSSIBLE" << endl;
    }


    return 0;
}
```

7.7 Euler Tour E Lca

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int tempo = 0;

void euler(int v, vector<vector<long long int>> &adj,
     vector<bool> &vis, vector<long long int> &
    euler_in, vector<long long int> &euler_out){
    vis[v] = true;
    tempo++;
    euler_in[v] = tempo;
    //euler tour
    for(auto u : adj[v]){
        if(!vis[u]){
            euler(u, adj, vis, euler_in, euler_out);
        }
    }
    //tempo++; -> desnecessario
    euler_out[v] = tempo;
}

int lca(int a, int b, vector<long long int> &euler_in, vector<long long int> &euler_out, vector<vector<long long int>> &pai){
    if((euler_in[a] <= euler_in[b]) and (euler_out[a] >= euler_out[b])){
        return a;
    } else if((euler_in[b] <= euler_in[a]) and (euler_out[b] >= euler_out[a])){
        return b;
    } else{
        int cursor = a;
        for(int i = 20; i >= 0; i--){
            if(!((euler_in[pai[cursor][i]] <= euler_in[b]) and (euler_out[pai[cursor][i]] >= euler_out[b]))){
                cursor = pai[cursor][i];
            }
        }
        return pai[cursor][0];
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, q, p, a, b;
    //AJUSTAR VALOR DE ITERAÃĞÃČO DE ACORDO COM LOGN
    cin >> n >> q;
    vector<vector<long long int>> pai(n+1, vector<long long int>(21, -1));
    vector<long long int> euler_in(n+1);
    vector<long long int> euler_out(n+1);
    vector<vector<long long int>> adj(n+1);
    vector<bool> vis(n+1, false);
    pai[1][0] = 1;
    for(int i = 2; i <= n; i++){
        cin >> p;
        pai[i][0] = p;
        adj[p].push_back(i);
        adj[i].push_back(p);
    }
    //precalcular pais
    for(int j = 1; j <= 20; j++){
        for(int i = 1; i <= n; i++){
            if(pai[i][j - 1] != -1){
                pai[i][j] = pai[pai[i][j-1]][j-1];
            }
        }
    }

    return 0;
}
```

7.8 Djikstra

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m;
    vector<long long int> distancias(n + 1, LLONG_MAX);
    vector<vector<tuple<long long int, int>>> adj(n + 1);
    distancias[1] = 0;
    priority_queue<tuple<long long int, int>> minheap;
    minheap.push(tuple(0, 1));
    long long int d, c;
    int a, b, o;
    for(int i = 0; i < m; i++){
        cin >> a;
        cin >> b;
        cin >> c;

        adj[a].push_back(tuple(c, b));
    }
    while(minheap.size() > 0){
        d = -(get<0>(minheap.top()));
        o = get<1>(minheap.top());
        minheap.pop();

        if(d <= distancias[o]){
            for(auto v : adj[o]){
                if(distancias[get<1>(v)] > distancias[o] + get<0>(v)){
                    distancias[get<1>(v)] = distancias[o] + get<0>(v);
                    minheap.push(tuple(-(distancias[get<1>(v)]), get<1>(v)));
                }
            }
        }
    }
    // LEMBRE DE GRAPH MODELLING
    // ADICIONAR ESTADOS COMO EM DP

    return 0;
}
```

7.9 Dinic

```cpp
#include <bits/stdc++.h>
#define endl '\n'
#define ll long long int
using namespace std;

struct aresta{
    ll u, v, c;
    aresta(){
        u = 0;
        v = 0;
        c = 0;
    }
    aresta(ll u2, ll v2, ll c2){
        u = u2;
        v = v2;
        c = c2;
    }
};

struct Dinic{
    ll n, src, sink;
    //vetor all permite achar real e inversa facilmente
    vector<aresta> all;
    vector<vector<int>> adj;
    vector<ll> level;
    //proxima aresta disponivel
    vector<int> nxt;

    Dinic(ll n2, ll src2, ll sink2){
        n = n2;
        src = src2;
        sink = sink2;
        adj = vector<vector<int>>(n+1);
    }

    void add(ll u, ll v, ll c, bool dir){
        //no dinic sao necessarios arestas opostas de
     capacidade 0
        all.push_back(aresta(u, v, c));
        adj[u].push_back(all.size()-1);
        //se for nao direcionado trocar 0 por c
        ll invcap = 0;
        if(dir){
            invcap = c;
        }
        all.push_back(aresta(v, u, invcap));
        adj[v].push_back(all.size()-1);
    }

    //bfs para gerar o vetor lvl por meio de arestas
    nao saturadas
    //deve sinalizar se chegou na pia (caso contrario
     o algoritmo terminou)
    bool bfs(){
        level = vector<ll>(n+1, -1);
        queue<tuple<int, ll>> fila;
        level[src] = 0;
        fila.push(tuple<int, ll>(src, 0));

        while(!fila.empty()){
            int curr = get<0>(fila.front());
            ll d = get<1>(fila.front());
            fila.pop();

            for(auto e : adj[curr]){
                if((all[e].c > 0) && (level[all[e].v]
     == -1)){
                    level[all[e].v] = d+1;
                    fila.push(tuple<int, ll>(all[e].v
    , d+1));
                }
            }
        }

        return (level[sink] != -1);
    }


    ll dfs(int curr, ll bottleneck){
        if((curr == sink) or (bottleneck == 0)){
            return bottleneck;
        }

        //atualiza prox aresta, lembrar de resetar
        for(int &curredge = nxt[curr]; curredge < (
    adj[curr].size()); curredge++){
            int ar = adj[curr][curredge];
            int nv = all[ar].v;
            //apenas arestas que progridem
            if((level[nv] == (level[curr] + 1))){
                ll check = dfs(nv, min(bottleneck,
    all[ar].c));
                if(check > 0){
                    all[ar].c -= check;
                    //flipar o ultimo bit checa
    inverso
                    all[ar ^ 1].c += check;
                    return check;
                }
            }
        }
        //blocking flow
        return 0;
    }

    ll maxflow(){
        ll flow = 0;
        while(bfs()){
            nxt = vector<int>(n+1, 0);
            while(true){
                ll check = dfs(src, 1e12);
                if(check > 0){
                    flow += check;
                } else{
                    break;
                }
            }
        }
        return flow;
    }

};

/*
mincut = maxflow
apos ultima bfs, level[i] == -1 se esta do lado da
    fonte

emparelhamento max:
criar src e sink, direcionar arestas src->lado 1,
    lado 2-> sink
todas arestas tem peso 1
emp = maxflow

cobertura minima do bipartido = emparelhamento
grids sao bipartidos

ind - max independent set
cob(n) = n - ind(n)

em um grafo com arestas unitarias:
maxflow = quantidade de caminhos aresta-disjuntos s-t
cap(corte minimo) = maximo de arestas removidos que
    quebra os caminhos de s-t

caminhos vertice-disjuntos = maxflow onde vertices
    sao separados em v1 e v2 com
aresta unitaria de v1-v2, v1 onde entra e v2 onde sai

fechamento maximo: reduzir para min cut
lado da fonte: o que adquiriu
lado do ralo: o que ignorou

se valor real i >= 0 : fonte -> i (valor)
else i -> ralo (-valor)

dependencias: se precisa de A pra ter B, aresta: B ->
     A (infinito)
*/
```

7.10 Dsu

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int find(int n, vector<int> &rep){
    if(n == rep[n]){
        return rep[n];
    } else{
        rep[n] = find(rep[n], rep);
        return rep[n];
    }
}

void join(int n, int v, vector<int> &rep, map<int,
    int> &size){
    n = find(n, rep);
    v = find(v, rep);

    if(n == v){
        return;
    }

    if(size[n] < size[v]){
        swap(v, n);
    }

    rep[v] = n;
    size[n] += size[v];
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m;
    map<int, int> size;
    vector<int> rep(n + 1);
    for(int i = 1; i <= n; i++){
        rep[i] = i;
        size[i]++;
    }

    return 0;
}
```

7.11 Pontes

```
#include <bits/stdc++.h>
#define endl '\n'
#define ll long long int
using namespace std;

int t = 0;
vector<ll> pontes;

void dfs(int v, int p, vector<vector<tuple<ll, int>>> &adj, vector<bool> &vis, vector<int> &in, vector<int> &low){
    vis[v] = true;
    t++;
    in[v] = t;
    low[v] = t;

    bool skip = false;
    for(auto [id, u] : adj[v]){
        if(u==p && !skip){
            skip = true;
        } else{
            if(vis[u]){
                low[v] = min(low[v], in[u]);
            } else{
                dfs(u, v, adj, vis, in, low);
                low[v] = min(low[v], low[u]);
                if(low[u] > in[v]){
                    pontes.push_back(id);
                }
            }
        }
    }
}
```

7.12 Floyd Warshall

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m, a, b;
    long long int c;
    vector<vector<long long int>> distancias(n + 1, vector<long long int>(n + 1, LLONG_MAX));
    for(int i = 0; i < m; i++){
        cin >> a;
        cin >> b;
        cin >> c;
        // lembrar arestas duplas
        distancias[a][b] = min(c, distancias[a][b]);
        distancias[b][a] = min(c, distancias[b][a]);
    }

    for(int k = 1; k <= n; k++){
        for(int u = 1; u <= n; u++){
            for(int v = 1; v <= n; v++){
                if((distancias[u][k] != LLONG_MAX) and (distancias[k][v] != LLONG_MAX)){
                    distancias[u][v] = min(distancias[u][v], (distancias[u][k] + distancias[k][v]));
                }
            }
        }
    }
    // analisar quantidade, serve pra poucos vÃľrtices e saber distÃćncia entre todas



    return 0;
}
```

7.13 Scc

```
#include <bits/stdc++.h>
#define endl '\n'
#define ll long long int
using namespace std;

void dfs(int v, vector<vector<int>> &adj, vector<bool> &vis, vector<int> &order){
    vis[v] = true;
    for(auto u : adj[v]){
        if(!vis[u]){
            dfs(u, adj, vis, order);
        }
    }
    order.push_back(v);

    return;
}

void kosaraju(vector<vector<int>> &adj, vector<int> &c, vector<vector<int>> &comps, vector<vector<int>> &dag){
    int n = adj.size();
    vector<int> order;
    vector<bool> vis(n, false);

    for(int i = 1; i < n; i++){
        if(!vis[i]){
            dfs(i, adj, vis, order);
        }
    }

    vector<vector<int>> adj_transp(n);
    for(int v = 1; v < n; v++){
        for(int u : adj[v]){
            adj_transp[u].push_back(v);
        }
    }
    vis = vector<bool>(n, false);
```

```cpp
    reverse(order.begin(), order.end());

    for(auto v : order){
        if(!vis[v]){
            vector<int> comp;
            dfs(v, adj_transp, vis, comp);
            comps.push_back(comp);
        }
    }

    vector<bool> mark(comps.size(), false);
    for(int i = 0; i < comps.size(); i++){
        for(auto v : comps[i]){
            c[v] = i;
        }
    }

    for(int i = 1; i < n; i++){
        for(auto v : adj[i]){
            if(c[i]==c[v] || mark[c[v]]) continue;

            mark[c[v]] = true;
            dag[c[i]].push_back(c[v]);
        }
        for(auto v : adj[i]) mark[c[v]] = false;
    }

}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    //2SAT
    //i = x
    //i^1 = ~x
    if(!pos){
        cout << "IMPOSSIVEL" << endl;
    } else{
        vector<bool> val(m+1);
        for(int i = 1; i <= 2*m + 1; i+=2){
            val[i/2] = co[i] < co[i^1];
        }
        for(int i = 1; i <= m; i++){
            if(val[i]){
                cout << "S ";
            } else{
                cout << "N ";
            }
        }
        cout << endl;
    }

    /*
    void add_or(int u, int v){
        E[eval(~u)].push_back(eval(v));
        E[eval(~v)].push_back(eval(u));
    }
    void add_nand(int u, int v) {
        E[eval(u)].push_back(eval(~v));
        E[eval(v)].push_back(eval(~u));
    }
    void set_true (int u){ E[eval(~u)].push_back(eval(u)); }
    void set_false(int u){ set_true(~u); }
    void add_imply(int u, int v){ E[eval(u)].push_back(eval(v)); }
    void add_and  (int u, int v){ set_true(u); set_true(v);    }
    void add_nor  (int u, int v){ add_and(~u, ~v); }
    void add_xor  (int u, int v){ add_or(u, v); add_nand(u, v); }
    void add_xnor (int u, int v){ add_xor(u, ~v); }
    */


    return 0;
}
```