Manda o Double de Campeão
CEFET-MG

Pedro Augusto

2 de abril de 2025

Índice

1 Array

1.1 Longest Increasing Subsequence

```
// Retorna a mauir subsequencia crescente dentro de um vetor
// O(n logn)
d7d vector<int> lis(vector<int>& arr) {
61d     vector<int> subseq;
ed5     for(int& x : arr) {
8a2         auto it = lower_bound(subseq.begin(), subseq.end(), x);
d3e         if (it == subseq.end()) subseq.push_back(x);
77c         else *it = x;
b53     }
cff     return subseq;
c0e }
```

2 DP

2.1 Exemplo Sapo

```
// There are N stones, numbered 1,2,...,N.
// For each i (1<=i<=N), the height of Stone i is hi.
// There is a frog who is initially on Stone 1.
// He will repeat the following action some number of times to reach
    Stone N:
// If the frog is currently on Stone i, jump to one of the following:
    Stone i+1,i+2,...,i+K. Here, a cost of|hi - hj| is incurred,
    wherej is the stone to land on.
// Find the minimum possible total cost incurred before the frog
    reaches Stone N.

dca int n, k;

// Top Down
4d3 int dp(int i) {
563     if(i == 0) return 0;
7f9     auto& ans = memo[i];
d64     if(~ans) return ans;

5f9     int ret = INF;
a7f     f(j, max(0ll,i-k), i)
f97         ret = min(ret, dp(j) + abs(h[j] - h[i]));

655     return ans = ret;
641 }

// Bootom Up
e63 int dp_2(int x) {

ecd     memo[0] = 0;
b85     f(i,1,x) {
90b         int best = INF;
203         f(j, max(0ll, i-k), i) {
```

```
428             best = min(best, memo[j] + abs(h[i] - h[j]));
8b8         }
bc2         memo[i] = best;
832     }

d56     return memo[x-1];
6f3 }

63d void solve() {
0a1     cin >> n >> k;
3f0     f(i,0,n) cin >> h[i];
8e4     cout << dp(n-1) << endl;
1d6 }
```

2.2 Is Subset Sum (Iterativo)

```
// Verifica se a soma de 0 <= i <= n elementos iguala a sum
// Temporal: O(sum * n)
// Espacial: O(sum * n)

c00 const int MAXN = 100;
bc4 const int MAXSUM = 5000;

759 bool isSubsetSum(vector<int>& v, int n, int sum) {
10a     f(i, 0, n + 1) { memo[i][0] = true; }
258     f(j, 1, sum + 1) { memo[0][j] = false; }

336     f(i, 1, n + 1) {
9e0         f(j, 1, sum + 1) {
a1d             if(j < v[i-1])
2b7                 memo[i][j] = memo[i-1][j];
295             else
c1f                 memo[i][j] = memo[i-1][j] || memo[i-1][j- v[i-1]];
66a         }
7f7     }
138     return memo[n][sum];
f54 }

c0b void solve(int n, int sum) {

70a     vector<int> v(n);
9b4     for(auto& x : n) cin >> x;

dbf     cout << (isSubsetSum(v, n, k) ? "S" : "N") << endl;
707 }
```

2.3 Knapsack tradicional

```
// O(n * cap)

b94 const int MAXN = 110;
689 const int MAXW = 1e5+10;

ba9 int n, memo[MAXN][MAXW];
310 int v[MAXN], w[MAXN];
74a int pego[MAXN] = {0};

// Retorna o lucro maximo
12c int dp(int id, int cap) {
1bb     if(cap < 0) return -LLINF;
ecb     if(id == n or cap == 0) return 0;
c1a     int &ans = memo[id][cap];
d64     if(~ans) return ans;
86f     return ans = max(dp(id+1, cap), dp(id+1, cap-w[id]) + v[id]);
d95 }

// Armazena em pego os itens pegos
7d0 void recuperar(int id, int cap) {
efa     if(id >= n) return;
fca     if(dp(id+1, cap-w[id]) + v[id] > dp(id+1, cap)) { // se pegar
    eh otimo
44c         pego[id] = true;
3fd         recuperar(id+1, cap-w[id]);
4ee     } else { // nao pegar eh otimo
884         pego[id] = false;
45d         recuperar(id+1, cap);
549     }
845 }


63d void solve() {

311     int cap; cin >> n >> cap;
457     memset(memo, -1, sizeof memo);

03b     f(i,0,n) { cin >> w[i] >> v[i]; }

304     int lucro_max = dp(0, cap);

ae7     recuperar(0, cap);

4cc     int lucro = 0, peso = 0;
418     f(i,0,n) {
```

```
ecd         if(pego[i]) {
73f             lucro += v[i];
20e             peso += w[i];
c3f         }
b7f     }

d13     assert(lucro_max == lucro and peso <= cap);
f6f }
```

3 Estruturas

3.1 BIT

```
// BIT de soma 0-based
//
// upper_bound(x) retorna o menor p tal que pref(p) > x
//
// Complexidades:
// build - O(n)
// update - O(log(n))
// query - O(log(n))
// upper_bound - O(log(n))

8eb struct Bit {
1a8     int n;
116     vector<int> bit;
e86     Bit(int _n=0) : n(_n), bit(n + 1) {}
70f     Bit(vector<int>& v) : n(v.size()), bit(n + 1) {
78a         for (int i = 1; i <= n; i++) {
671             bit[i] += v[i - 1];
edf             int j = i + (i & -i);
b8a             if (j <= n) bit[j] += bit[i];
806         }
e89     }
cf6     void update(int i, int x) { // soma x na posicao i
b64         for (i++; i <= n; i += i & -i) bit[i] += x;
850     }
cdb     int pref(int i) { // soma [0, i]
7c9         int ret = 0;
4d3         for (i++; i; i -= i & -i) ret += bit[i];
edf         return ret;
065     }
9e3     int query(int l, int r) {  // soma [l, r]
89b         return pref(r) - pref(l - 1);
e97     }
bdf     int upper_bound(int x) {
1ba         int p = 0;
0af         for (int i = __lg(n); i+1; i--)
6f5             if (p + (1<<i) <= n and bit[p + (1<<i)] <= x)
68e                 x -= bit[p += (1 << i)];
74e         return p;
be4     }
f75 };

63d void solve() {

70a     vector<int> v(n);

8b8     Bit bit (v);
edc }
```

3.2 DSU

```
// Une dois conjuntos e acha a qual conjunto um elemento pertence por
   seu id
//
// find e unite: O(a(n)) ~= O(1) amortizado

8d3 struct dsu {
825     vector<int> id, sz;

b33     dsu(int n) : id(n), sz(n, 1) { iota(id.begin(), id.end(), 0); }

0cf     int find(int a) { return a == id[a] ? a : id[a] = find(id[a]);
   }

440     void unite(int a, int b) {
605         a = find(a), b = find(b);
d54         if (a == b) return;
956         if (sz[a] < sz[b]) swap(a, b);
6d0         sz[a] += sz[b], id[b] = a;
ea7     }
8e1 };

// DSU de bipartido
//
// Une dois vertices e acha a qual componente um vertice pertence
// Informa se a componente de um vertice e bipartida
//
// find e unite: O(log(n))
```

```
8d3 struct dsu {
6f7     vector<int> id, sz, bip, c;

5b4     dsu(int n) : id(n), sz(n, 1), bip(n, 1), c(n) {
db8         iota(id.begin(), id.end(), 0);
f25     }

ef0     int find(int a) { return a == id[a] ? a : find(id[a]); }
f30     int color(int a) { return a == id[a] ? c[a] : c[a] ^
    color(id[a]); }

440     void unite(int a, int b) {
263         bool change = color(a) == color(b);
605         a = find(a), b = find(b);
a89         if (a == b) {
4ed             if (change) bip[a] = 0;
505             return;
32d         }

956         if (sz[a] < sz[b]) swap(a, b);
efe         if (change) c[b] = 1;
2cd         sz[a] += sz[b], id[b] = a, bip[a] &= bip[b];
22b     }
118 };


// DSU Persistente
//
// Persistencia parcial, ou seja, tem que ir
// incrementando o 't' no une
//
// find e unite: O(log(n))

8d3 struct dsu {
33c     vector<int> id, sz, ti;

733     dsu(int n) : id(n), sz(n, 1), ti(n, -INF) {
db8         iota(id.begin(), id.end(), 0);
aad     }

5e6     int find(int a, int t) {
6ba         if (id[a] == a or ti[a] > t) return a;
ea5         return find(id[a], t);
6cb     }

fa0     void unite(int a, int b, int t) {
84f         a = find(a, t), b = find(b, t);
d54         if (a == b) return;
956         if (sz[a] < sz[b]) swap(a, b);
35d         sz[a] += sz[b], id[b] = a, ti[b] = t;
513     }
6c6 };

// DSU com rollback
//
// checkpoint(): salva o estado atual de todas as variaveis
// rollback(): retorna para o valor das variaveis para
// o ultimo checkpoint
//
// Sempre que uma variavel muda de valor, adiciona na stack
//
// find e unite: O(log(n))
// checkpoint: O(1)
// rollback: O(m) em que m e o numero de vezes que alguma
// variavel mudou de valor desde o ultimo checkpoint

8d3 struct dsu {
825     vector<int> id, sz;
27c     stack<stack<pair<int&, int>>> st;

98d     dsu(int n) : id(n), sz(n, 1) {
1cc         iota(id.begin(), id.end(), 0), st.emplace();
8cd     }

bdf     void save(int &x) { st.top().emplace(x, x); }

30d     void checkpoint() { st.emplace(); }

5cf     void rollback() {
ba9         while(st.top().size()) {
6bf             auto [end, val] = st.top().top(); st.top().pop();
149             end = val;
f9a         }
25a         st.pop();
3c6     }

ef0     int find(int a) { return a == id[a] ? a : find(id[a]); }

440     void unite(int a, int b) {
605         a = find(a), b = find(b);
d54         if (a == b) return;
956         if (sz[a] < sz[b]) swap(a, b);
803         save(sz[a]), save(id[b]);
```

```
6d0         sz[a] += sz[b], id[b] = a;
1b9     }
c6e };
```

3.3 Fenwick Tree (BIT) Range

```
// Operacoes 0-based
// query(l, r) retorna a soma de v[l..r]
// update(l, r, x) soma x em v[l..r]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

796 namespace BitRange {
06d     int bit[2][MAX+2];
1a8     int n;

727     void build(int n2, vector<int>& v) {
1e3         n = n2;
535         for (int i = 1; i <= n; i++)
a6e             bit[1][min(n+1, i+(i&-i))] += bit[1][i] += v[i];
d31     }
1a7     int get(int x, int i) {
7c9         int ret = 0;
360         for (; i; i -= i&-i) ret += bit[x][i];
edf         return ret;
a4e     }
920     void add(int x, int i, int val) {
503         for (; i <= n; i += i&-i) bit[x][i] += val;
fae     }
3d9     int get2(int p) {
c7c         return get(0, p) * p + get(1, p);
33c     }
9e3     int query(int l, int r) { // zero-based
ff5         return get2(r+1) - get2(l);
25e     }
7ff     void update(int l, int r, int x) {
e5f         add(0, l+1, x), add(0, r+2, -x);
f58         add(1, l+1, -x*l), add(1, r+2, x*(r+1));
5ce     }
1b4 };

63d void solve() {
97a     vector<int> v {0,1,2,3,4,5}; // v[0] eh inutilizada
f98     BitRange::build(v.size(), v);

67f     int a = 0, b = 3;
3d5     BitRange::query(a, b); // v[a] + v[a+1] + ... + v[b] = 6  |
    1+2+3 = 6 | zero-based
a4b     BitRange::update(a, b, 2); // v[a...b] += 2 | zero-based
d65 }
```

3.4 SegTree

```
// Recursiva com Lazy Propagation
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
// Pode usar a seguinte funcao para indexar os nohs:
// f(l, r) = (l+r)|(l!=r), usando 2N de memoria
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

0d2 const int MAX = 1e5+10;

fb1 namespace SegTree {
098     int seg[4*MAX], lazy[4*MAX];
052     int n, *v;

b90     int op(int a, int b) { return a + b; }

2c4     int build(int p=1, int l=0, int r=n-1) {
3c7         lazy[p] = 0;
6cd         if (l == r) return seg[p] = v[l];
ee4         int m = (l+r)/2;
317         return seg[p] = op(build(2*p, l, m), build(2*p+1, m+1, r));
985     }

0d8     void build(int n2, int* v2) {
680         n = n2, v = v2;
6f2         build();
acb     }

ceb     void prop(int p, int l, int r) {
cdf         seg[p] += lazy[p]*(r-l+1);
2c9         if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
3c7         lazy[p] = 0;
```

```
c10     }

04a     int query(int a, int b, int p=1, int l=0, int r=n-1) {
6b9         prop(p, l, r);
527         if (a <= l and r <= b) return seg[p];
786         if (b < l or r < a) return 0;
ee4         int m = (l+r)/2;
19e         return op(query(a, b, 2*p, l, m), query(a, b, 2*p+1, m+1,
    r));
1c9     }

f33     int update(int a, int b, int x, int p=1, int l=0, int r=n-1) {
6b9         prop(p, l, r);
9a3         if (a <= l and r <= b) {
b94             lazy[p] += x;
6b9             prop(p, l, r);
534             return seg[p];
821         }
e9f         if (b < l or r < a) return seg[p];
ee4         int m = (l+r)/2;
a8f         return seg[p] = op(update(a, b, x, 2*p, l, m), update(a,
    b, x, 2*p+1, m+1, r));
08f     }

        // Se tiver uma seg de max, da pra descobrir em O(log(n))
        // o primeiro e ultimo elemento >= val numa range:

        // primeira posicao >= val em [a, b] (ou -1 se nao tem)
119     int get_left(int a, int b, int val, int p=1, int l=0, int
    r=n-1) {
6b9         prop(p, l, r);
f38         if (b < l or r < a or seg[p] < val) return -1;
205         if (r == l) return l;
ee4         int m = (l+r)/2;
753         int x = get_left(a, b, val, 2*p, l, m);
50e         if (x != -1) return x;
c3c         return get_left(a, b, val, 2*p+1, m+1, r);
68c     }

        // ultima posicao >= val em [a, b] (ou -1 se nao tem)
992     int get_right(int a, int b, int val, int p=1, int l=0, int
    r=n-1) {
6b9         prop(p, l, r);
f38         if (b < l or r < a or seg[p] < val) return -1;
205         if (r == l) return l;
ee4         int m = (l+r)/2;
1b1         int x = get_right(a, b, val, 2*p+1, m+1, r);
50e         if (x != -1) return x;
6a7         return get_right(a, b, val, 2*p, l, m);
1b7     }

        // Se tiver uma seg de soma sobre um array nao negativo v, da
            pra
        // descobrir em O(log(n)) o maior j tal que
            v[i]+v[i+1]+...+v[j-1] < val
89b     int lower_bound(int i, int& val, int p, int l, int r) {
6b9         prop(p, l, r);
6e8         if (r < i) return n;
b5d         if (i <= l and seg[p] < val) {
bff             val -= seg[p];
041             return n;
634         }
3ce         if (l == r) return l;
ee4         int m = (l+r)/2;
514         int x = lower_bound(i, val, 2*p, l, m);
ee0         if (x != n) return x;
8b9         return lower_bound(i, val, 2*p+1, m+1, r);
01d     }
a15 };

63d void solve() {
213     int n = 10;
89e     int v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
2d5     SegTree::build(n, v);

3af     cout << SegTree::query(0, 9) << endl; // seg[0] + seg[1] + ...
    + seg[9] = 55
310     SegTree::update(0, 9, 1); // seg[0,...,9] += 1
6d9 }
```

3.5 Sparse Table Disjunta

```
// Description: Sparse Table Disjunta para soma de intervalos
// Complexity Temporal: O(n log n) para construir e O(1) para consultar
// Complexidade Espacial: O(n log n)

125 const int MAX = 100010
d5a const int MAX2 = 20 // log(MAX)

82d namespace SparseTable {
9bf     int m[MAX2][2*MAX], n, v[2*MAX];
b90     int op(int a, int b) { return a + b; }
0d8     void build(int n2, int* v2) {
```

```
1e3         n = n2;
df4         for (int i = 0; i < n; i++) v[i] = v2[i];
a84         while (n&(n-1)) n++;
3d2         for (int j = 0; (1<<j) < n; j++) {
1c0             int len = 1<<j;
d9b             for (int c = len; c < n; c += 2*len) {
332                 m[j][c] = v[c], m[j][c-1] = v[c-1];
668                 for (int i = c+1; i <  c+len; i++) m[j][i] =
    op(m[j][i-1], v[i]);
432                 for (int i = c-2; i >= c-len; i--) m[j][i] =
    op(v[i], m[j][i+1]);
eda             }
f4d         }
ce3     }
9e3     int query(int l, int r) {
f13         if (l == r) return v[l];
e6d         int j = __builtin_clz(1) - __builtin_clz(l^r);
d67         return op(m[j][l], m[j][r]);
a7b     }
258 }

63d void solve() {
ce1     int n = 9;
1a3     int v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
3f7     SparseTable::build(n, v);
925     cout << SparseTable::query(0, n-1) << endl; // sparse[0] +
    sparse[1] + ... + sparse[n-1] = 45
241 }
```

3.6 Tabuleiro

```
// Description: Estrutura que simula um tabuleiro M x N, sem realmente
    criar uma matriz
// Permite atribuir valores a linhas e colunas, e consultar a posicao
    mais frequente
// Complexidade Atribuir: O(log(N))
// Complexidade Consulta: O(log(N))
// Complexidade verificar frequencia geral: O(N * log(N))
9a0 #define MAX_VAL 5 // maior valor que pode ser adicionado na matriz
    + 1

8ee class BinTree {
d9d     protected:
ef9         vector<int> mBin;
673     public:
d5e         explicit BinTree(int n) { mBin = vector(n + 1, 0); }
e44         void add(int p, const int val) {
dd1             for (auto size = mBin.size(); p < size; p += p & -p)
174                 mBin[p] += val;
b68         }

e6b         int query(int p) {
e1c             int sumToP {0};
b62             for (; p > 0; p -= p & -p)
ec1                 sumToP += mBin[p];
838             return sumToP;
793         }
a5f };

b6a class ReverseBinTree : public BinTree {
673     public:
83e         explicit ReverseBinTree(int n) : BinTree(n) {};

e44         void add(int p, const int val) {
850             BinTree::add(static_cast<int>(mBin.size()) - p, val);
705         }

e6b         int query(int p) {
164             return BinTree::query(static_cast<int>(mBin.size()) -
    p);
a21         }
6cf };

952 class Tabuleiro {
673     public:
177         explicit Tabuleiro(const int m, const int n, const int q)
    : mM(m), mN(n), mQ(q) {
958             mLinhas = vector<pair<int, int8_t>>(m, {0, 0});
d68             mColunas = vector<pair<int, int8_t>>(n, {0, 0});

66e             mAtribuicoesLinhas = vector(MAX_VAL,
    ReverseBinTree(mQ)); // aARvore[51]
9e5             mAtribuicoesColunas = vector(MAX_VAL,
    ReverseBinTree(mQ));
13b         }

bc2         void atribuirLinha(const int x, const int8_t r) {
e88             mAtribuirFileira(x, r, mLinhas, mAtribuicoesLinhas);
062         }

ca2         void atribuirColuna(const int x, const int8_t r) {
689             mAtribuirFileira(x, r, mColunas, mAtribuicoesColunas);
```

```
a40         }

d10         int maxPosLinha(const int x) {
f95             return mMaxPosFileira(x, mLinhas, mAtribuicoesColunas,
    mM);
8ba         }

ff7         int maxPosColuna(const int x) {
b95             return mMaxPosFileira(x, mColunas, mAtribuicoesLinhas,
    mN);
252         }

80e         vector<int> frequenciaElementos() {

a35             vector<int> frequenciaGlobal(MAX_VAL, 0);
45a             for(int i=0; i<mM; i++) {
ebd                 vector<int> curr = frequenciaElementos(i,
    mAtribuicoesColunas);
97f                 for(int j=0; j<MAX_VAL; j++)
ef3                     frequenciaGlobal[j] += curr[j];
094             }
01e             return frequenciaGlobal;
b7a         }

bf2     private:
69d         int mM, mN, mQ, mMoment {0};

0a6         vector<ReverseBinTree> mAtribuicoesLinhas,
    mAtribuicoesColunas;
f2d         vector<pair<int, int8_t>> mLinhas, mColunas;

e7a         void mAtribuirFileira(const int x, const int8_t r,
    vector<pair<int, int8_t>>& fileiras,
1d7                               vector<ReverseBinTree>& atribuicoes) {
224             if (auto& [oldQ, oldR] = fileiras[x]; oldQ)
bda                 atribuicoes[oldR].add(oldQ, -1);

914             const int currentMoment = ++mMoment;
b2c             fileiras[x].first = currentMoment;
80b             fileiras[x].second = r;
f65             atribuicoes[r].add(currentMoment, 1);
5de         }

2b8         int mMaxPosFileira(const int x, const vector<pair<int,
    int8_t>>& fileiras, vector<ReverseBinTree>&
    atribuicoesPerpendiculares, const int& currM) const {
1aa             auto [momentoAtribuicaoFileira, rFileira] =
    fileiras[x];

8d0             vector<int> fileiraFrequencia(MAX_VAL, 0);
729             fileiraFrequencia[rFileira] = currM;

85a             for (int8_t r {0}; r < MAX_VAL; ++r) {
8ca                 const int frequenciaR =
    atribuicoesPerpendiculares[r].query(momentoAtribuicaoFileira + 1);
04a                 fileiraFrequencia[rFileira] -= frequenciaR;
72e                 fileiraFrequencia[r] += frequenciaR;
6b0             }

b59             return MAX_VAL - 1 -
    (max_element(fileiraFrequencia.crbegin(),
    fileiraFrequencia.crend()) - fileiraFrequencia.crbegin());
372         }

7c4         vector<int> frequenciaElementos(int x,
    vector<ReverseBinTree>& atribuicoesPerpendiculares) const {

8d0             vector<int> fileiraFrequencia(MAX_VAL, 0);

583             auto [momentoAtribuicaoFileira, rFileira] = mLinhas[x];

083             fileiraFrequencia[rFileira] = mN;

85a             for (int8_t r {0}; r < MAX_VAL; ++r) {
8ca                 const int frequenciaR =
    atribuicoesPerpendiculares[r].query(momentoAtribuicaoFileira + 1);
04a                 fileiraFrequencia[rFileira] -= frequenciaR;
72e                 fileiraFrequencia[r] += frequenciaR;
6b0             }

2e6             return fileiraFrequencia;
15d         }

20c };

63d void solve() {

e29     int L, C, q; cin >> L >> C >> q;

56c     Tabuleiro tabuleiro(L, C, q);

a09     int linha = 0, coluna = 0, valor = 10; // linha e coluna sao 0
    based
b68     tabuleiro.atribuirLinha(linha, static_cast<int8_t>(valor)); //
```

```
    f(i,0,C) matriz[linha][i] = valor
34d     tabuleiro.atribuirColuna(coluna, static_cast<int8_t>(valor));
    // f(i,0,L) matriz[i][coluna] = valor

        // Freuencia de todos os elementos, de 0 a MAX_VAL-1
155     vector<int> frequenciaGeral = tabuleiro.frequenciaElementos();

176     int a = tabuleiro.maxPosLinha(linha); // retorna a posicao do
    elemento mais frequente na linha
981     int b = tabuleiro.maxPosColuna(coluna); // retorna a posicao
    do elemento mais frequente na coluna
9b5 }
```

3.7 Union-Find (Disjoint Set Union)

```
da5 const int MAXN = 1e3+10;

0cd struct UnionFind {
c55     int numSets;
320     int id[MAXN], sz[MAXN];

02d     UnionFind(int N) {
0bd         numSets = N;
faa         for (int i = 0; i < N; i++) {
963             id[i] = i;
02c             sz[i] = 1;
0e9         }
41f     }

aee     int find(int a) {
3da         return id[a] = (id[a] == a ? a : find(id[a]));
c7b     }

78b     void uni(int a, int b) {
605         a = find(a), b = find(b);
d54         if(a == b) return;
3c6         if(sz[a] > sz[b]) swap(a, b);
351         id[a] = b;
582         sz[b] += sz[a];
92a         --numSets;
650     }

3ae     int sizeOfSet(int a) { return sz[find(a)]; }
c59     int numDisjointSets() { return numSets; }
864 };
```

```
63d void solve() {

f98     int n, ed; cin >> n >> ed;
f4e     UnionFind uni(n);

31c     f(i,0,ed) {
602         int a, b; cin >> a >> b; a--, b--;
45e         uni.uni(a,b);
c0f     }

350     cout << uni.numDisjointSets() << endl;
01b }
```

4 Grafos

4.1 APSP - Floyd Warshall

```
// Calcula caminho minimo entre todos os pares de vertices
// O(V^3)

b94 const int MAXN = 110;

0eb int adj[MAXN][MAXN];

eac void printAnswer(int n) {
5bf     for (int u = 0; u < n; ++u)
a3f     for (int v = 0; v < n; ++v)
6ab     cout << "APSP("<<u<<", "<<v<<") = " << adj[u][v] << endl;
8c7 }

e4a void prepareParent() {
418     f(i,0,n) {
001         f(j,0,n) {
b5a             p[i][j] = i;
441         }
d89     }

b9b     for (int k = 0; k < n; ++k)
6cb     for (int i = 0; i < n; ++i)
a9e     for (int j = 0; j < n; ++j)
c1d         if (adj[i][k] + adj[k][j] < adj[i][j]) {
a1a             adj[i][j] = adj[i][k]+adj[k][j];
9b6             p[i][j] = p[k][j];
```

```
a04         }
c85 }

470 vector<int> restorePath(int u, int v) {

36d     if (adj[u][v] == INF) return {};
5b4     vector<int>  path;
81f     for (; v != u; v = p[u][v]) {
ff8         if (v == -1) return {};
80a         path.push_back(v);
3d1     }
960     path.push_back(u);
3a9     reverse(path.begin(), path.end());
535     return path;
16a }

230 void floyd_warshall(int n) {
e22     for (int k = 0; k < n; k++)
830     for (int i = 0; i < n; i++)
f90     for (int j = 0; j < n; j++)
ffd         adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]);
e78 }

e03 void solve(int n, int ed) {

418     f(i,0,n) {
001         f(j,0,n) {
59d             adj[i][j] = INF;
93d         }
774         adj[i][i] = 0;
c4e     }

c92     while(ed--) {
c48         int u, v, w; cin >> u >> v >> w; u--, v--;
7da         adj[u][v] = adj[v][u] = w;
2a0     }

803     floyd_warshall(n);

        // prepareParent();
        // auto path = restorePath(0, 3);
649 }

// Diametro do Grafo: maior valor de adj[u][v] != INF
```

4.2 BFS

```
// O(V + E)

2a7 const int MAXN = 1e4+10;

465 bool vis[MAXN];
be4 int d[MAXN], p[MAXN];
ea6 vector<int> adj [MAXN];

94c void bfs(int s) {

8b2     queue<int> q; q.push(s);
654     vis[s] = true, d[s] = 0, p[s] = -1;

14d     while (!q.empty()) {
0e6         int v = q.front(); q.pop();
c25         vis[v] = true;

f74         for (int u : adj[v]) {
1d6             if (!vis[u]) {
b9c                 vis[u] = true;
f73                 q.push(u);
                    // d[u] = d[v] + 1;
                    // p[u] = v;
3b2             }
3d2         }
cc2     }
75a }

63d void solve() {
98f     int n, ed;

19e     f(i,0,n) { d[i] = -1, p[i] = -1; }

c92     while(ed--) {
ed2         int u, v; cin >> u >> v; u--, v--;
cc9         adj[u].push_back(v);
1ea         adj[v].push_back(u);
3f1     }

19c     int s; bfs(s);
81a }

// Checar Fortemente Conexo: BFS adj e adj reverso, ver se todos os
    vertices foram visitados.
```

4.3 BFS - por niveis

```
//  Encontrar distancia entre S e outros pontos em que pontos estao
    agrupados (terminais)

ef8 const int MAXN = 510;
57d const int MAXEDG = 510; // maximo numero de terminais

9d3 int dist[MAXN];
a19 vector<int> niveisDoNode[MAXN], nodesDoNivel[MAXEDG];

94c void bfs(int s) {

735     queue<pair<int, int>> q; q.push({s, 0});

a93     dist[s] = 0;

14d     while (!q.empty()) {
2bc         auto [v, dis] = q.front(); q.pop();

400         for(auto nivel : niveisDoNode[v]) {
8fd             for(auto u : nodesDoNivel[nivel]) {
619                 if (dist[u] == 0) {
324                     q.push({u, dis+1});
554                     dist[u] = dis + 1;
12f                 }
46b             }
ffe         }
e19     }
e00 }

63d void solve() {

09d     int n, terminais, s, e;

6bf     f(i,0,terminais) {
509         int q; cin >> q;
1f4         while(q--) {
9aa             int v; v--;
e19             niveisDoNode[v].push_back(i);
b14             nodesDoNivel[i].push_back(v);
fd8         }
6ec     }

aff     bfs(s);

85a     cout << dist[e] << endl;
7b3 }
```

4.4 Emparelhamento Max Grafo Bipartido (Kuhn)

```
// Computa matching maximo em grafo bipartido
// 'n' e 'm' sao quantos vertices tem em cada particao
// chamar add(i, j) para add aresta entre o cara i
// da particao A, e o cara j da particao B
// (entao i < n, j < m)
// Para recuperar o matching, basta olhar 'ma' e 'mb'
// 'recover' recupera o min vertex cover como um par de
// {caras da particao A, caras da particao B}
//
// O(|V| * |E|)
// Na pratica, parece rodar tao rapido quanto o Dinitz


878 mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

6c6 struct kuhn {
14e     int n, m;
789     vector<vector<int>> g;
d3f     vector<int> vis, ma, mb;

40e     kuhn(int n_, int m_) : n(n_), m(m_), g(n),
8af         vis(n+m), ma(n, -1), mb(m, -1) {}

ba6     void add(int a, int b) { g[a].push_back(b); }

caf     bool dfs(int i) {
29a         vis[i] = 1;
29b         for (int j : g[i]) if (!vis[n+j]) {
8c9             vis[n+j] = 1;
2cf             if (mb[j] == -1 or dfs(mb[j])) {
bfe                 ma[i] = j, mb[j] = i;
8a6                 return true;
b17             }
82a         }
d1f         return false;
4ef     }
bf7     int matching() {
1ae         int ret = 0, aum = 1;
5a8         for (auto& i : g) shuffle(i.begin(), i.end(), rng);
392         while (aum) {
618             for (int j = 0; j < m; j++) vis[n+j] = 0;
```

```
c5d             aum = 0;
830             for (int i = 0; i < n; i++)
01f                 if (ma[i] == -1 and dfs(i)) ret++, aum = 1;
085         }
edf         return ret;
2ee     }
b0d };

63d void solve() {

be0     int n1; // Num vertices lado esquerdo grafo bipartido
e4c     int n2; // Num vertices lado direito grafo bipartido

761     kuhn K(n1, n2);

732     int edges;

6e0     while(edges--) {
b1f         int a, b; cin >> a >> b;
3dc         K.add(a,b); // a -> b
5b7     }

69b     int emparelhamentoMaximo = K.matching();
76b }
```

4.5 Fluxo - Dinitz (Max Flow)

```
// Encontra fluxo maximo de um grafo
// O(min(m * max_flow, n^2 m))
// Grafo com capacidades 1: O(min(m sqrt(m), m * n^(2/3)))
// Todo vertice tem grau de entrada ou saida 1: O(m sqrt(n))

472 struct dinitz {
61f     const bool scaling = false; // com scaling -> O(nm
    log(MAXCAP)),
206     int lim;                    // com constante alta
670     struct edge {
358         int to, cap, rev, flow;
7f9         bool res;
d36         edge(int to_, int cap_, int rev_, bool res_)
a94             : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
f70     };

002     vector<vector<edge>> g;
216     vector<int> lev, beg;
a71     ll F;
190     dinitz(int n) : g(n), F(0) {}

087     void add(int a, int b, int c) {
bae         g[a].emplace_back(b, c, g[b].size(), false);
4c6         g[b].emplace_back(a, 0, g[a].size()-1, true);
5c2     }
123     bool bfs(int s, int t) {
90f         lev = vector<int>(g.size(), -1); lev[s] = 0;
64c         beg = vector<int>(g.size(), 0);
8b2         queue<int> q; q.push(s);
402         while (q.size()) {
be1             int u = q.front(); q.pop();
bd9             for (auto& i : g[u]) {
dbc                 if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
b4f                 if (scaling and i.cap - i.flow < lim) continue;
185                 lev[i.to] = lev[u] + 1;
8ca                 q.push(i.to);
f97             }
e87         }
0de         return lev[t] != -1;
742     }
dfb     int dfs(int v, int s, int f = INF) {
50b         if (!f or v == s) return f;
88f         for (int& i = beg[v]; i < g[v].size(); i++) {
027             auto& e = g[v][i];
206             if (lev[e.to] != lev[v] + 1) continue;
ee0             int foi = dfs(e.to, s, min(f, e.cap - e.flow));
749             if (!foi) continue;
3c5             e.flow += foi, g[e.to][e.rev].flow -= foi;
45c             return foi;
618         }
bb3         return 0;
4b1     }
ff6     ll max_flow(int s, int t) {
a86         for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
9d1             while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
4ff         return F;
8b9     }
86f };

// Recupera as arestas do corte s-t
dbd vector<pair<int, int>> get_cut(dinitz& g, int s, int t) {
f07     g.max_flow(s, t);
68c     vector<pair<int, int>> cut;
1b0     vector<int> vis(g.g.size(), 0), st = {s};
321     vis[s] = 1;
3c6     while (st.size()) {
```

```
b17         int u = st.back(); st.pop_back();
322         for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
c17             vis[e.to] = 1, st.push_back(e.to);
d14     }
481     for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
9d2         if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i,
    e.to);
d1b     return cut;
1e8 }

63d void solve() {

1a8     int n; // numero de arestas
b06     dinitz g(n);

732     int edges;
6e0     while(edges--) {
1e1         int a, b, w; cin >> a >> b >> c;
f93         g.add(a,b,c); // a -> b com capacidade c
fa1     }

07a     int maxFlow = g.max_flow(SRC, SNK); // max flow de SRC -> SNK
a7b }
```

4.6 Fluxo - MinCostMaxFlow

```
// min_cost_flow(s, t, f) computa o par (fluxo, custo)
// com max(fluxo) <= f que tenha min(custo)
// min_cost_flow(s, t) -> Fluxo maximo de custo minimo de s pra t
// Se for um dag, da pra substituir o SPFA por uma DP pra nao
// pagar O(nm) no comeco
// Se nao tiver aresta com custo negativo, nao precisa do SPFA
//
// O(nm + f * m log n)

123 template<typename T> struct mcmf {
670     struct edge {
b75         int to, rev, flow, cap; // para, id da reversa, fluxo,
    capacidade
7f9         bool res; // se eh reversa
635         T cost; // custo da unidade de fluxo
892         edge() : to(0), rev(0), flow(0), cap(0), cost(0),
    res(false) {}
1d7         edge(int to_, int rev_, int flow_, int cap_, T cost_, bool
    res_)
f8d             : to(to_), rev(rev_), flow(flow_), cap(cap_),
    res(res_), cost(cost_) {}
723     };

002     vector<vector<edge>> g;
168     vector<int> par_idx, par;
f1e     T inf;
a03     vector<T> dist;

b22     mcmf(int n) : g(n), par_idx(n), par(n),
    inf(numeric_limits<T>::max()/3) {}

91c     void add(int u, int v, int w, T cost) { // de u pra v com cap
    w e custo cost
2fc         edge a = edge(v, g[v].size(), 0, w, cost, false);
234         edge b = edge(u, g[u].size(), 0, 0, -cost, true);

b24         g[u].push_back(a);
c12         g[v].push_back(b);
0ed     }

8bc     vector<T> spfa(int s) { // nao precisa se nao tiver custo
    negativo
871         deque<int> q;
3d1         vector<bool> is_inside(g.size(), 0);
577         dist = vector<T>(g.size(), inf);

a93         dist[s] = 0;
a30         q.push_back(s);
ecb         is_inside[s] = true;

14d         while (!q.empty()) {
b1e             int v = q.front();
ced             q.pop_front();
48d             is_inside[v] = false;

76e             for (int i = 0; i < g[v].size(); i++) {
9d4                 auto [to, rev, flow, cap, res, cost] = g[v][i];
e61                 if (flow < cap and dist[v] + cost < dist[to]) {
943                     dist[to] = dist[v] + cost;

ed6                     if (is_inside[to]) continue;
020                     if (!q.empty() and dist[to] > dist[q.front()])
    q.push_back(to);
b33                     else q.push_front(to);
b52                     is_inside[to] = true;
2d1                 }
```

```
8cd             }
f2c         }
8d7         return dist;
96c     }
2a2     bool dijkstra(int s, int t, vector<T>& pot) {
489         priority_queue<pair<T, int>, vector<pair<T, int>>,
    greater<>> q;
577         dist = vector<T>(g.size(), inf);
a93         dist[s] = 0;
115         q.emplace(0, s);
402         while (q.size()) {
91b             auto [d, v] = q.top();
833             q.pop();
68b             if (dist[v] < d) continue;
76e             for (int i = 0; i < g[v].size(); i++) {
9d4                 auto [to, rev, flow, cap, res, cost] = g[v][i];
e8c                 cost += pot[v] - pot[to];
e61                 if (flow < cap and dist[v] + cost < dist[to]) {
943                     dist[to] = dist[v] + cost;
441                     q.emplace(dist[to], to);
88b                     par_idx[to] = i, par[to] = v;
873                 }
de3             }
9d4         }
1d4         return dist[t] < inf;
c68     }

3d2     pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
3dd         vector<T> pot(g.size(), 0);
9e4         pot = spfa(s); // mudar algoritmo de caminho minimo aqui

d22         int f = 0;
ce8         T ret = 0;
4a0         while (f < flow and dijkstra(s, t, pot)) {
bda             for (int i = 0; i < g.size(); i++)
d2a                 if (dist[i] < inf) pot[i] += dist[i];

71b             int mn_flow = flow - f, u = t;
045             while (u != s){
90f                 mn_flow = min(mn_flow,
07d                     g[par[u]][par_idx[u]].cap -
    g[par[u]][par_idx[u]].flow);
3d1                 u = par[u];
935             }

1f2             ret += pot[t] * mn_flow;
476             u = t;
045             while (u != s) {
e09                 g[par[u]][par_idx[u]].flow += mn_flow;
d98                 g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
3d1                 u = par[u];
bcc             }

04d             f += mn_flow;
36d         }

15b         return make_pair(f, ret);
cc3     }

        // Opcional: retorna as arestas originais por onde passa flow
            = cap
182     vector<pair<int,int>> recover() {
24a         vector<pair<int,int>> used;
2a4         for (int i = 0; i < g.size(); i++) for (edge e : g[i])
587             if(e.flow == e.cap && !e.res) used.push_back({i,
    e.to});
f6b         return used;
390     }
697 };

63d void solve(){

1a8     int n; // numero de vertices
4c5     mcmf<int> mincost(n);

ab4     mincost.add(u, v, cap, cost); // unidirecional
983     mincost.add(v, u, cap, cost); // bidirecional

073     auto [flow, cost] = mincost.min_cost_flow(src, end/*,
    initialFlow*/);

da5 }
```

4.7 Fluxo - Problemas

```
// 1: Problema do Corte
7a9 - Entrada:
bc1     - N itens
388     - Curso Wi Inteiro
7c3     - M restricoes: se eu pegar Ai, eu preciso pegar Bi...
387 - Saida: valor maximo pegavel
```

```
ac2 - Solucao: corte maximo com Dinitz
019     - dinitz(n+m+1)
593     - f(i,0,n): i -> SNK com valor Ai
9eb     - f(i,0,m):
9e2         * SRC -> n+i com valor Wi
a9e         * ParaTodo dependente Bj: n+i -> Bj com peso INF
8a0     - ans = somatorio(Wi) - maxFlow(SRC,SNK);

/* ======================================================== */
```

4.8 IsBipartido

```
// Verifica se um grafo eh bipartido
// O(V+E)

b94 const int MAXN = 110;

ea6 vector<int> adj[MAXN];
496 int color[MAXN];

64d bool bipartido(int n) {

37f     int s = 0;
8b2     queue<int> q; q.push(s);
56d     color[s] = 0;
4fd     bool ans = true;

984     while (!q.empty() and ans) {
be1         int u = q.front(); q.pop();

cab         for (auto &v : adj[u]) {
f75             if (color[v] == INF) {
23d                 color[v] = 1 - color[u];
2a1                 q.push(v);
763             }
ec1             else if (color[v] == color[u]) {
7bd                 ans = false;
c2b                 break;
d89             }
d57         }
8fe     }

ba7     return ans;
61e }

5a4 void solve(int n) {
418     f(i,0,n) {
9b0         adj[i] = vector<int>();
f49         color[i] = INF;
417     }

        // preenche grafo ...

bcc     bool ans = bipartido(n);
b1b }
```

4.9 Kruskal

```
// Encontra a Arvore Geradora Minima (AGM) de um grafo
// O(E log V)

da5 const int MAXN = 1e3+10;

320 int id[MAXN], sz[MAXN];

aee int find(int a) {
3da     return id[a] = (id[a] == a ? a : find(id[a]));
c7b }

78b void uni(int a, int b) { // O(a(N)) amortizado
605     a = find(a), b = find(b);
d54     if(a == b) return;

3c6     if(sz[a] > sz[b]) swap(a,b);
6eb     id[a] = b, sz[b] += sz[a];
2de }

057 int kruskal(vector<tuple<int, int, int>>& edg) {

704     int cost = 0;
        // vector<tuple<int, int, int>> mst;
fea     for (auto [w,x,y] : edg) if (find(x) != find(y)) {
            // mst.emplace_back(w, x, y);
45f         cost += w;
cf2         uni(x,y);
815     }
12d     return cost;
798 }

e03 void solve(int n, int ed) {
```

```
f51     vector<tuple<int, int, int>> edg(n);

863     for(auto& [w,u,v] : edg) {
261         cin >> u >> v >> w; u--, v--;
afd     }

418     f(i,0,n) {
963         id[i] = i;
bc4         sz[i] = -1;
55b     }

c14     sort(all(edg));

772     int cost = kuskal(edg);
8c8 }

// VARIANTES

// Maximum Spanning Tree: sort(edg.rbegin(), edg.rend());

// 'Minimum' Spanning Subgraph:
//  - Algumas arestas ja foram adicionadas (maior prioridade - Questao
    das rodovias)
//  - Arestas que nao foram adicionadas (menor prioridade - ferrovias)
//  -> kruskal(rodovias); kruskal(ferrovias);

// Minimum Spanning Forest:
//  - Queremos uma floresta com k componentes
//  -> kruskal(edg); if(mst.sizer() == k) break;

//  MiniMax
//  - Encontrar menor caminho entre dous vertices com maior quantidade
    de arestas
//  -> kruskal(edg); dijsktra(mst);

// Second Best MST
//  - Encontrar a segunda melhor arvore geradora minima
//  -> kruskal(edg);
//  -> flag mst[i] = 1;
//  -> sort(cmp(edg.flag != -1)) => da prioridade para outras arestas
```

4.10 LCA com RMQ

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// dist(a, b) retorna a distancia entre a e b
//
// build - O(n)
// lca - O(1)
// dist - O(1)

67a template<typename T>
9f6 struct rmq {
517     vector<T> v;
1a8     int n;
bac     static const int b = 30;
70e     vector<int> mask, t;

6b4     int op(int x, int y) {
ffd         return v[x] < v[y] ? x : y;
18e     }
543     int msb(int x) {
391         return __builtin_clz(1) - __builtin_clz(x);
ee1     }
6ad     rmq() {}
43c     rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
2e5         for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
a61             at = (at << 1) & ((1 << b) - 1);
411             while (at and op(i, i - msb(at & -at)) == i)
282                 at ^= at & -at;
53c         }
9c0         for (int i = 0; i < n / b; i++)
e78             t[i] = b * i + b - 1 - msb(mask[b * i + b - 1]);
dce         for (int j = 1; (1 << j) <= n / b; j++)
122             for (int i = 0; i + (1 << j) <= n / b; i++)
ba5                 t[n / b * j + i] = op(t[n / b * (j - 1) + i], t[n
    / b * (j - 1) + i + (1 << (j - 1))]);
2d3     }
879     int small(int r, int sz = b) {
7e3         return r - msb(mask[r] & ((1 << sz) - 1));
c92     }
b7a     T query(int l, int r) {
cb5         if (r - l + 1 <= b)
e60             return small(r, r - l + 1);
7bf         int ans = op(small(l + b - 1), small(r));
e80         int x = l / b + 1, y = r / b - 1;
e25         if (x <= y) {
a4e             int j = msb(y - x + 1);
002             ans = op(ans, op(t[n / b * j + x], t[n / b * j + y -
    (1 << j) + 1]));
4b6         }
ba7         return ans;
6bf     }
```

```
b75 };

065 namespace lca {
282     vector<int> g[MAXN];
1d9     int v[2 * MAXN], pos[MAXN], dep[2 * MAXN];
8bd     int t;
2de     rmq<int> RMQ;

4cf     void dfs(int i, int d = 0, int p = -1) {
ae8         v[t] = i;
1f1         pos[i] = t;
949         dep[t] = d;
c82         t++;
45a         for (auto j : g[i])
f25             if(j != p) {
8ec                 dfs(j, d + 1, i);
ae8                 v[t] = i;
949                 dep[t] = d;
c82                 t++;
fcd             }
8de     }

789     void build(int n, int root) {
a34         t = 0;
14e         dfs(root);
659         vector<int> depVec(dep, dep + 2 * n - 1);
ac6         RMQ = rmq<int>(depVec);
631     }

7be     int lca(int a, int b) {
ab7         a = pos[a], b = pos[b];
d11         if(a > b) swap(a, b);
544         return v[RMQ.query(a, b)];
413     }

b5d     int dist(int a, int b) {
72b         int l = lca(a, b);
798         return dep[pos[a]] + dep[pos[b]] - 2 * dep[pos[l]];
2ed     }
767 }


5a4 void solve(int n) {

742     f(i,0,n-1){
bf7         int a, b; cin >> a >> b; a--; b--;
ec9         lca::g[a].push_back(b);
b2f         lca::g[b].push_back(a);
32a     }

a78     lca::build(n, 0); // raiz nesse caso eh 0

d00     cout << lca::dist(1,2) << endl // distancia entre vertices 1 e
    2 da arvore
83a }
```

4.11 Lowest Common Ancestor (LCA) com peso

```
// Encontra o LCA de uma arvore com peso, assim como a distancia
// entre 2 vertices.
//
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// dist(a, b) retorna a distancia entre a e b
//
// build - O(n)
// lca - O(1)
// dist - O(1)

47e const int MAXN = 1e5+10;

67a template<typename T>
9f6 struct rmq {
517     vector<T> v;
1a8     int n;
bac     static const int b = 30;
70e     vector<int> mask, t;

18e     int op(int x, int y) { return v[x] < v[y] ? x : y; }
ee1     int msb(int x) { return __builtin_clz(1) - __builtin_clz(x); }

6ad     rmq() {}
43c     rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
2e5         for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
a61             at = (at << 1) & ((1 << b) - 1);
411             while (at and op(i, i - msb(at & -at)) == i)
282                 at ^= at & -at;
53c         }
9c0         for (int i = 0; i < n / b; i++)
e78             t[i] = b * i + b - 1 - msb(mask[b * i + b - 1]);
dce         for (int j = 1; (1 << j) <= n / b; j++)
122             for (int i = 0; i + (1 << j) <= n / b; i++)
0ee                 t[n / b * j + i] = op(t[n / b * (j - 1) + i],
```

```
cc8                                     t[n / b * (j - 1) + i + (1
    << (j - 1))]);
2d3     }

879     int small(int r, int sz = b) {
7e3         return r - msb(mask[r] & ((1 << sz) - 1));
c92     }

b7a     T query(int l, int r) {
27b         if (r - l + 1 <= b) return small(r, r - l + 1);
7bf         int ans = op(small(l + b - 1), small(r));
e80         int x = l / b + 1, y = r / b - 1;
e25         if (x <= y) {
a4e             int j = msb(y - x + 1);
002             ans = op(ans, op(t[n / b * j + x], t[n / b * j + y -
    (1 << j) + 1]));
4b6         }
ba7         return ans;
6bf     }
b75 };

065 namespace lca {
2f8     vector<pair<int,int>> g[MAXN];

13c     int v[2 * MAXN];
e3b     int pos[MAXN];
9bb     int level[2 * MAXN];
8bd     int t;
2de     rmq<int> RMQ;

9d3     int dist[MAXN];

2f0     void dfs(int i, int l = 0, int p = -1, long long d = 0) {
ae8         v[t] = i;
1f1         pos[i] = t;
f68         level[t] = l;
0f9         dist[i] = d;
c82         t++;
eaf         for (auto edge : g[i]) {
149             int nxt = edge.first;
68a             int w = edge.second;
40e             if (nxt == p) continue;
749             dfs(nxt, l + 1, i, d + w);
ae8             v[t] = i;
f68             level[t] = l;
c82             t++;
001         }
165     }

cda     void build(int n, int root = 0) {
a34         t = 0;
14e         dfs(root);
c64         vector<int> levelVec(level, level + (2 * n - 1));
a0c         RMQ = rmq<int>(levelVec);
d91     }

7be     int lca(int a, int b) {
ab7         a = pos[a], b = pos[b];
d11         if (a > b) swap(a, b);
544         return v[RMQ.query(a, b)];
413     }

7a4     long long queryDist(int a, int b) {
851         int anc = lca(a, b);
88b         return dist[a] + dist[b] - 2LL * dist[anc];
731     }
c13 }

63d void solve() {

9ee     int n; cin >> n;

a45     f(i,0,n)
5af         lca::g[i].clear();

8b2     f(i,1,n) {
d25         int a, b, w; cin >> a >> b >> w;
cbc         lca::g[a].push_back({b, w});
b41         lca::g[b].push_back({a, w});
d6c     }

a78     lca::build(n, 0); // arvore com n vertices com raiz em 0

903     int lowestCommonAncertor = lca::lca(0,1); // LCA entre 0 e 1
258     int dist = lca::queryDist(0,1); // Distancia entre 0 e 1

df1 }
```

4.12 Pontos de Articulacao + Pontes

```
// Computa os pontos de articulacao (vertices criticos) de um grafo
//
```

```
// art[i] armazena o numero de novas componentes criadas ao deletar
    vertice i
// se art[i] >= 1, entao vertice i eh ponto de articulacao
//
// O(V + E)

aec const int MAXN = 410;

ea6 vector<int> adj[MAXN];
5d0 int id[MAXN], art[MAXN];
4ce stack<int> s;

3e1 int dfs_art(int i, int &t, int p = -1) {
cf0     int lo = id[i] = t++;
e07     int children = 0;
18e     s.push(i);
f78     for (int j : adj[i]) {
d09         if (j == p) continue;
9a3         if (id[j] == -1) {
c5f             children++;
206             int val = dfs_art(j, t, i);
0c3             lo = min(lo, val);
588             if (val >= id[i]) {
66a                 art[i]++;
bd9                 while (s.top() != j) s.pop();
2eb                 s.pop();
1f3             }
                // if (val > id[i]) aresta i-j eh ponte
682         }
4e6         else {
872             lo = min(lo, id[j]);
30c         }
798     }

924     if (p == -1) {
0d6         if (children > 1)
4f4             art[i] = children - 1;
295         else
2b9             art[i] = -1;
abc     }
253     return lo;
db5 }

4d9 void AP(int n) {

79e     s = stack<int>();
418     f(i,0,n) {
9c8         id[i] = -1;
2b9         art[i] = -1;
ec6     }
6bb     int t = 0;
418     f(i,0,n) {
766         if (id[i] == -1)
625             dfs_art(i, t, -1);
d39     }
f67 }

e03 void solve(int n, int ed) {

98f     int n, ed;
a45     f(i,0,n)
9b0         adj[i] = vector<int>();

c92     while(ed--) {
ba2         int a, b;
fab         adj[a].push_back(b);
c87         adj[b].push_back(a);
9a0     }

bd2     AP(n);

516     vector<int> pontos;
        // Para vertices nao-raiz, art[i] >= 0 indica que eh ponto de
            articulacao.
        // Para a raiz (i==0) ela so deve ser considerada se tiver 2
            ou mais filhos, ou seja, se art[0] > 0.
418     f(i,0,n) {
995         if (i == 0) {
0f8             if (art[i] > 0) pontos.push_back(i+1);
e74         } else {
72a             if (art[i] >= 0) pontos.push_back(i+1);
802         }
c23     }
fdb }
```

4.13 SSSP - Bellman Ford

```
// Aceita pesos negativos
//
// Conexo: O(VE)
// Desconexo: O(EV^2)
```

```
1a0 const int MAXEDG = 1e3+10;

203 tuple<int,int,int>> edg [MAXN];
989 int dist[MAXN];i

f6e int bellman_ford(int n, int src) {
11b     dist.assign(n+1, INT_MAX);

41e     f(i,0,n+2) {
d77         for(auto& [u, v, w] : edg) {
20a             if(dist[u] != INT_MAX and dist[v] > w + dist[u])
491                 dist[v] = dist[u] + w;
224         }
a1e     }

        // Possivel checar ciclos negativos (ciclo de peso total
           negativo)
d77     for(auto& [u, v, w] : edg) {
20a         if(dist[u] != INT_MAX and dist[v] > w + dist[u])
6a5             return 1;
40f     }

bb3     return 0;
0b4 }

e03 void solve(int n, int ed) {

040     f(i,0,n) { dist[i] = INF; }

31c     f(i,0,ed) {
2ef         int u, v, w; u--, v--;
732         edg[i] = {u,v,w};
5ce     }

447     bellman_ford(n, 0);
bd8 }
```

4.14 SSSP - Dijktra

```
// O(E log V)

2a7 const int MAXN = 1e4+10;

3ac vector<pair<int,int>> adj[MAXN];
9d3 int dist[MAXN];
// int parent[MAXN];

3f4 void dijkstra(int s) {
a93     dist[s] = 0; // se eventualmente puder voltar pra ca, tipo
    ciclo | salesman | remover essa linha
63c     priority_queue<pii, vector<pii>, greater<pii>> pq; pq.push({0,
    s});

502     while (!pq.empty()) {
5c1         auto [d, u] = pq.top(); pq.pop();
3e1         if (d > dist[u]) continue;
             // if(u == s and dist[u] < INF) break; | pra quando tiver
                 que fazer um ciclo

3c0         for (auto &[v, w] : adj[u]) {
c21             if (dist[u] + w >= dist[v]) continue;
491             dist[v] = dist[u]+w;
                // parent[v] = u;
bf3             pq.push({dist[v], v});
a42         }
6df     }
695 }

63d void solve() {

8ed     int v, ed; cin >> v >> ed;
98b     f(i,0,v) {  dist[i] = INF; }

c92     while(ed--) {
691         int a, b, w; cin >> a >> b >> w; a--, b--;
fbc         adj[a].emplace_back(b,w);
            // adj[b].emplace_back(a,w);
47c     }
458     int s; dijkstra(s);
c49 }
```

5 Matematica

5.1 Conversao de Bases

```
// Converte de 10 -> {2, 8, 10, 16} (log n)
// Converte de {2, 8, 10, 16} -> 10 (n)
9c7 char charForDigit(int digit) {
431     if (digit > 9) return digit + 87;
d4a     return digit + 48;
```

```
826 }

// 10 -> {2, 8, 10, 16}
0f3 string decimalToBase(int n, int base = 10) {
f40     if (not n) return "0";
461     stringstream ss;
fcb     for (int i = n; i > 0; i /= base) {
ac7         ss << charForDigit(i % base);
cd2     }
f1f     string s = ss.str();
01f     reverse(s.begin(), s.end());
047     return s;
01e }

9a2 int intForDigit(char digit) {
374     int intDigit = digit - 48;
545     if (intDigit > 9) return digit - 87;
a09     return intDigit;
acc }

// {2, 8, 10, 16} -> 10
e37 int baseToDecimal(const string& n, int base = 10) {
c18     int result = 0;
e09     int basePow =1;
000     for (auto it = n.rbegin(); it != n.rend(); ++it, basePow *=
    base)
445         result += intForDigit(*it) * basePow;
dc8     return result;
9f0 }
```

5.2 Divisores - Contar

```
// Conta o numero de divisores de um numero baseadp no Smallest Prime
    Factor

191 vector<int> spf; // Smallest Prime Factor

254 void computeSpf(int n) {
768     spf.resize(n + 1);
78a     for (int i = 1; i <= n; i++) {
cdc         spf[i] = i;
7a5     }
2ed     for (int i = 2; i * i <= n; i++) {
a58         if (spf[i] == i) {
985             for (int j = i * i; j <= n; j += i) {
d91                 if (spf[j] == j)
9a0                     spf[j] = i;
622             }
44b         }
1ee     }
0fe }

1e3 int getDivisorCount(int x) {
4e4     int cntDiv = 1;
40e     while (x > 1) {
80a         int p = spf[x];
ac9         int cnt = 0;
fa7         while (x % p == 0) {
f65             cnt++;
43f             x /= p;
cfd         }
2ba         cntDiv *= (cnt + 1);
646     }
a87     return cntDiv;
d96 }

63d void solve() {
1a8     int n; // maior dos numeros a ser computado a listagem
aee     computeSpf(n); // gera os spf para todos ate n
d9c     cout << getDivisorCount(n) << endl;
4f6 }
```

5.3 MDC e MMC

```
// O(log n)

// MDC entre 2 numeros
8c1 int mdc(int a, int b) {
ce2     for (int r = a % b; r; a = b, b = r, r = a % b);
73f     return b;
8f5 }

// MDC entre N numeros
460 int mdc_many(vector<int> arr) {
7b6     int result = arr[0];

aa6     for (int& num : arr) {
437         result = mdc(num, result);

c03         if(result == 1) return 1;
885     }
dc8     return result;
```

```
0c9 }

// MMC entre 2 numeros
3ec int mmc(int a, int b) {
6fe     return a / mdc(a, b) * b;
770 }

// MMC entre N numeros
1db int mmc_many(vector<int> arr) {
7b6     int result = arr[0];

05f     for (int &num : arr)
9c4         result = (num * result / mdc(num, result));
dc8     return result;
72c }
```

5.4 Numero de Digitos

```
// Calcula o numero de digitos de n
// 1234 = 4; 0 = 1

09c int numDigits(int n) {
209     if (n == 0) return 1;
662     n = std::abs(n);
146     return static_cast<int>(std::floor(std::log10(n))) + 1;
d2b }
```

5.5 Primos - Lowest Prime Factor

```
// Menor fator primo de n
// O(sqrt(n))

074 int lowestPrimeFactor(int n, int startPrime = 2) {
9d5     if (startPrime <= 3) {
fb4         if (not (n & 1)) return 2;
5a0         if (not (n % 3)) return 3;
72a         startPrime = 5;
43a     }

c94     for (int i = startPrime; i * i <= n; i += (i + 1) % 6 ? 4 : 2)
dcb         if (not (n % i))
d9a             return i;
041     return n;
6c5 }
```

5.6 Primos - Primo

```
// Verifica se um numero eh primo
// O(sqrt(n))
5f7 bool isPrime(int n) {
e32     return n > 1 and lowestPrimeFactor(n) == n;
822 }
```

5.7 Sieve

```
// Gera todos os primos do intervalo [1,lim]
// O(n log log n)

324 int _sieve_size;
467 bitset<10000010> bs;
632 vector<int> p;

5c4 void sieve(int lim) {
1ba     _sieve_size = lim+1;
0a3     bs.set();
e52     bs[0] = bs[1] = 0;
a0a     f(i,2,_sieve_size) {
a47         if (bs[i]) {
bfe             for (int j = i*i; j < _sieve_size; j += i) bs[j] = 0;
d8d             p.push_back(i);
70b         }
ab8     }
841 }
```

6 String

6.1 Longest Common Subsequence 1 (LCS)

```
// Retrorna a LCS entre as string S e T.
// Armazena em memo[i][j] o LCS_SZ de s[i...n] e t[j...m].
// Implementacao recursiva
//
// Temporal: O(n*m)
// Espacial: O(n*m)

da5 const int MAXN = 1e3+10;
```

```
dd0 int memo[MAXN][MAXN];

// Calcula tamanho do LCS recursivamente
28f int lcs_sz(string& s, string& t, int i, int j) {
45d     if(i == s.size() or j == t.size()) return 0;
e80     auto& ans = memo[i][j];
d64     if(~ans) return ans;
1a9     if(s[i] == t[j])
176         ans = 1 + lcs_sz(s,t,i+1, j+1);
295     else
3af         ans = max(
c19                 lcs_sz(s,t,i+1,j),
364                 lcs_sz(s,t,i,j+1)
616             );
ba7     return ans;
afa }

// Armazena em ans a LCS entre S e T
10e void lcs(string& ans, string& s, string& t, int i, int j) {
f86     if(i >= s.size() or j >= t.size()) return;
524     if(s[i] == t[j]) {
b80         ans.push_back(s[i]);
081         return lcs(ans, s, t, i+1, j+1);
a00     }
4cb     if(lcs_sz(s,t,i+1,j) > lcs_sz(s,t,i,j+1)) return lcs(ans, s,
    t, i+1, j);
4f2     return lcs(ans, s, t, i, j+1);
bc5 }

63d void solve() {

bfb     string s, t; cin >> s >> t;

457     memset(memo,-1, sizeof memo);

a4d     string ans; lcs(ans, s, t, 0,0);
886     cout << ans << endl;
030 }
```

6.2 Split de String

```
// O(|s| * |del|).
5a6 vector<string> split(string s, string del = " ") {
cd5     vector<string> retorno;
0f4     int start, end = -1*del.size();
016     do {
a3b         start = end + del.size();
257         end = s.find(del, start);
36c         retorno.push_back(s.substr(start, end - start));
3a7     } while (end != -1);
5fa     return retorno;
f80 }
```

7 Extra

7.1 fastIO.cpp

```
int read_int() {
    bool minus = false;
    int result = 0;
    char ch;
    ch = getchar();
    while (1) {
        if (ch == '-') break;
        if (ch >= '0' && ch <= '9') break;
        ch = getchar();
    }
    if (ch == '-') minus = true;
    else result = ch-'0';
    while (1) {
        ch = getchar();
        if (ch < '0' || ch > '9') break;
        result = result*10 + (ch - '0');
    }
    if (minus) return -result;
    else return result;
}
```

7.2 hash.sh

```
# Para usar (hash das linhas [l1, l2]):
# bash hash.sh arquivo.cpp l1 l2
sed -n $2','$3' p' $1 | sed '/^#w/d' | cpp -dD -P -fpreprocessed | tr
    -d '[:space:]' | md5sum | cut -c-6
```

7.3 stress.sh

```
P=a
make ${P} ${P}2 gen || exit 1
for ((i = 1; ; i++)) do
    ./gen $i > in
    ./${P} < in > out
    ./${P}2 < in > out2
    if (! cmp -s out out2) then
        echo "--> entrada:"
        cat in
        echo "--> saida1:"
        cat out
        echo "--> saida2:"
        cat out2
        break;
    fi
    echo $i
done
```

7.4 pragma.cpp

```
// Otimizacoes agressivas, pode deixar mais rapido ou mais devagar
#pragma GCC optimize("Ofast")
// Auto explicativo
#pragma GCC optimize("unroll-loops")
// Vetorizacao
#pragma GCC target("avx2")
// Para operacoes com bits
#pragma GCC target("bmi,bmi2,popcnt,lzcnt")
```

7.5 timer.cpp

```
// timer T; T() -> retorna o tempo em ms desde que declarou
using namespace chrono;
struct timer : high_resolution_clock {
    const time_point start;
    timer(): start(now()) {}
    int operator()() {
        return duration_cast<milliseconds>(now() - start).count();
    }
};
```

7.6 vimrc

```
189 "" {
d79 set ts=4 sw=4 mouse=a nu ai si undofile
7c9 function H(l)
496     return system("sed '/^#/d' | cpp -dD -P -fpreprocessed | tr -d
    '[:space:]' | md5sum", a:l)
0be endfunction
329 function P() range
dd8     for i in range(a:firstline, a:lastline)
```

```
ccc          let l = getline(i)
139          call cursor(i, len(l))
7c9          echo H(getline(search('{')[1], 'bc', i) ? searchpair('{',
    '', '}', 'bn') : i, i))[0:2] l
bf9      endfor
0be endfunction
90e vmap <C-H> :call P()<CR>
de2 "" }
```

7.7 debug.cpp

```
void debug_out(string s, int line) { cerr << endl; }
template<typename H, typename... T>
void debug_out(string s, int line, H h, T... t) {
    if (s[0] != ',') cerr << "Line(" << line << ") ";
    do { cerr << s[0]; s = s.substr(1);
    } while (s.size() and s[0] != ',');
    cerr << " = " << h;
    debug_out(s, line, t...);
}
#ifdef DEBUG
#define debug(...) debug_out(#__VA_ARGS__, __LINE__, __VA_ARGS__)
#else
#define debug(...) 42
#endif
```

7.8 makefile

```
CXX = g++
CXXFLAGS = -fsanitize=address,undefined -fno-omit-frame-pointer -g
    -Wall -Wshadow -std=c++17 -Wno-unused-result -Wno-sign-compare
    -Wno-char-subscripts #-fuse-ld=gold

clearexe:
    find . -maxdepth 1 -type f -executable -exec rm {} +
```

7.9 temp.cpp

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

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

#define all(a)        a.begin(), a.end()
#define int           long lo    ng int
#define double        long double
#define f(i,s,e)      for(int i=s;i<e;i++)
#define dbg(x) cout << #x << " = " << x << " ";
#define dbgl(x) cout << #x << " = " << x << endl;

#define vi            vector<int>
#define pii           pair<int,int>
#define endl          "\n"
#define print_v(a)    for(auto x : a)cout<<x<<" ";cout<<endl
#define print_vp(a)   for(auto x : a)cout<<x.first<<" "<<x.second<< endl
#define rf(i,e,s)     for(int i=e-1;i>=s;i--)
#define CEIL(a, b)    ((a) + (b - 1))/b
#define TRUNC(x, n)   floor(x * pow(10, n))/pow(10, n)
#define ROUND(x, n)   round(x * pow(10, n))/pow(10, n)

const int INF = 1e9;     // 2^31-1
const int LLINF = 4e18; // 2^63-1
const double EPS = 1e-9;
const int MAX = 1e6+10; // 10^6 + 10

void solve() {


}

int32_t main() { _

    int t = 1; // cin >> t;
    while (t--) {
        solve();
    }
    return 0;
}
```

7.10 rand.cpp

```
mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

int uniform(int l, int r){
    uniform_int_distribution<int> uid(l, r);
    return uid(rng);
}
```