

Manda o Double de Campeão  
CEFET-MG

Pedro Augusto

9 de fevereiro de 2025

Índice

<b>1 Estruturas</b>	<b>1</b>
1.1 DSU	1
1.2 Fenwick Tree (BIT)	3
1.3 SegTree	3
1.4 Sparse Table Disjunta	5
1.5 Tabuleiro	5
1.6 Union-Find (Disjoint Set Union)	7
<b>2 Grafos</b>	<b>7</b>
2.1 Emparelhamento Max Grafo Bipartido (Kuhn)	7
2.2 Fluxo - Dinitz (Max Flow)	8
2.3 Fluxo - MinCostMaxFlow	9
2.4 Fluxo - Problemas	11
<b>3 Extra</b>	<b>11</b>
3.1 fastIO.cpp	11
3.2 hash.sh	11
3.3 template.cpp	11

3.4 stress.sh	12
3.5 pragma.cpp	12
3.6 timer.cpp	12
3.7 vimrc	12
3.8 debug.cpp	13
3.9 makefile	13
3.10 rand.cpp	13
<b>1 Estruturas</b>	<b>7</b>
<b>1.1 DSU</b>	<b>7</b>
// Une dois conjuntos e acha a qual conjunto um elemento pertence por seu id	
//	
// find e unite: $O(a(n)) \sim O(1)$ amortizado	
8d3	<b>struct</b> 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 };

```

## 1.2 Fenwick Tree (BIT)

```

// 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))

e04 namespace bit {
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*1), add(1, r+2, x*(r+1));
5ce     }
17a };

63d void solve() {

97a     vector<int> v {0,1,2,3,4,5}; // v[0] eh inutilizada
c7b     bit::build(v.size(), v);

67f     int a = 0, b = 3;
9b0     bit::query(a, b); // v[a] + v[a+1] + ... + v[b] = 6 | 1+2+3 =
6 | zero-based
b3d     bit::update(a, b, 2); // v[a...b] += 2 | zero-based
7b4 }

```

## 1.3 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 }

```

## 1.4 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)

2b7 #include <bits/stdc++.h>
ca4 using namespace std;

005 #define MAX 100010
352 #define 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 }
```

## 1.5 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.cbegin(),
fileiraFrequencia.crend()) - fileiraFrequencia.cbegin());
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 }

```

## 1.6 Union-Find (Disjoint Set Union)

```

f3b const int MAX = 5e4+10;

074 int p[MAX], ranking[MAX], setSize[MAX];

0cd struct UnionFind {
c55     int numSets;

02d     UnionFind(int N) {
680         iota(p,p+N+1,0);
340         memset(ranking, 0, sizeof ranking);
f0a         memset(setSize, 1, sizeof setSize);
0bd         numSets = N;
142     }

```

```

c59     int numDisjointSets() { return numSets; }
a5b     int sizeOfSet(int i) { return setSize[find(i)]; }

8ee     int find(int i) { return (p[i] == i) ? i : (p[i] =
find(p[i])); }
da3     bool same(int i, int j) { return find(i) == find(j); }
92e     void uni(int i, int j) {
ea5         if (same(i, j))
505             return;
c56         int x = find(i), y = find(j);
e4f         if (ranking[x] > ranking[y])
9dd             swap(x, y);
ae9         p[x] = y;
6e9         if (ranking[x] == ranking[y])
3cf             ++ranking[y];
223         setSize[y] += setSize[x];
92a         --numSets;
e3f     }
b6b };

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 }

```

## 2 Grafos

### 2.1 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 }

```

## 2.2 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 }

```

## 2.3 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 }

```

## 2.4 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);

/* ===== */

```

## 3 Extra

### 3.1 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;
}

```

### 3.2 hash.sh

```

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

```

### 3.3 template.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 long
#define double long double
#define f(i,s,e) for(int i=s;i<e;i++)
#define dbg(x) cout << #x << " = " << x << " ";

```

```

#define dbg1(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;
}

```

### 3.4 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
    fi
done

```

```

        break;
    fi
    echo $i
done

```

### 3.5 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")

```

### 3.6 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();
    }
};

```

### 3.7 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 "" }

```

### 3.8 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

```

### 3.9 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 {} +

```

### 3.10 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);
}

```