

# Competitive programming Notebook

Meia noite eu te conto

## Contents

<b>1</b>	<b>General</b>	<b>2</b>
1.1	Template . . . . .	2
1.2	Base Converter . . . . .	2
1.3	Split . . . . .	2
1.4	Random . . . . .	2
<b>2</b>	<b>String</b>	<b>2</b>
2.1	Triexor . . . . .	2
<b>3</b>	<b>Math</b>	<b>3</b>
3.1	2sat . . . . .	3
<b>4</b>	<b>DS</b>	<b>4</b>
4.1	Dsu . . . . .	4
4.2	Ordered Set . . . . .	4
<b>5</b>	<b>Graph</b>	<b>5</b>
5.1	Dinic . . . . .	5
5.2	Dfs . . . . .	6
5.3	Ford Fulkerson . . . . .	6
<b>6</b>	<b>Geometry</b>	<b>6</b>
6.1	Convex Hull . . . . .	6
<b>7</b>	<b>DP</b>	<b>7</b>
7.1	Edit Distance . . . . .	7
7.2	Lcs . . . . .	7
7.3	Knapsack . . . . .	8



# 1 General

## 1.1 Template

```

1 // MEIA NOITE EU TE CONTO
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 #define _ ios_base::sync_with_stdio(0);cin.tie(0);
7
8 typedef long long ll;
9
10 const int INF = 0x3f3f3f3f;
11 const ll LINF = 0x3f3f3f3f3f3f3f3fll;
12
13 int main() { _
14     return 0;
15 }

```

## 1.2 Base Converter

```

1 const string digits = "0123456789
  ABCDEFGHIJKLMNOPQRSTUVWXYZ";
2
3 ll tobase10(string number, int base) {
4     map<char, int> val;
5     for (int i = 0; i < digits.size(); i++) {
6         val[digits[i]] = i;
7     }
8
9     ll ans = 0, pot = 1;
10
11     for (int i = number.size() - 1; i >= 0; i--) {
12         ans += val[number[i]] * pot;
13         pot *= base;
14     }
15
16     return ans;
17 }
18
19 string frombase10(ll number, int base) {
20     if (number == 0) return "0";
21
22     string ans = "";
23
24     while (number > 0) {
25         ans += digits[number % base];
26         number /= base;
27     }
28
29     reverse(ans.begin(), ans.end());
30
31     return ans;
32 }
33
34 // verifica se um número está na base especificada
35 bool verify_base(string num, int base) {
36     map<char, int> val;
37     for (int i = 0; i < digits.size(); i++) {
38         val[digits[i]] = i;
39     }
40
41     for (auto digit : num) {
42         if (val[digit] >= base) {
43             return false;
44         }
45     }
46
47     return true;
48 }

```

## 1.3 Split

```

1 vector<string> split(string s, char key=' ') {
2     vector<string> ans;
3     string aux = "";
4
5     for (int i = 0; i < (int)s.size(); i++) {
6         if (s[i] == key) {
7             if (aux.size() > 0) {
8                 ans.push_back(aux);
9                 aux = "";
10            }
11        } else {
12            aux += s[i];
13        }
14    }
15
16    if ((int)aux.size() > 0) {
17        ans.push_back(aux);
18    }
19
20    return ans;
21 }

```

## 1.4 Random

```

1 random_device dev;
2 mt19937 rng(dev());
3
4 uniform_int_distribution<mt19937::result_type> dist
  (1, 6); // distribution in range [1, 6]
5
6 int val = dist(rng);

```

# 2 String

## 2.1 Triexor

```

1 // TrieXOR
2 //
3 // adiciona, remove e verifica se existe strings
  binarias
4 // max_xor(x) = maximiza o xor de x com algum valor
  da trie
5 //
6 // raiz = 0
7 //
8 // https://codeforces.com/problemset/problem/706/D
9 //
10 // 0(|s|) adicionar, remover e buscar
11
12 struct TrieXOR {
13     int n, alph_sz, nxt;
14     vector<vector<int>> trie;
15     vector<int> finish, paths;
16
17     TrieXOR() {}
18
19     TrieXOR(int n, int alph_sz = 2) : n(n), alph_sz(
  alph_sz) {
20         nxt = 1;
21         trie.assign(n, vector<int>(alph_sz));
22         finish.assign(n * alph_sz, 0);
23         paths.assign(n * alph_sz, 0);
24     }
25
26     void add(int x) {
27         int curr = 0;
28
29         for (int i = 31; i >= 0; i--) {
30             int b = ((x >> i) > 0);

```

```

31         if (trie[curr][b] == 0)
32             trie[curr][b] = nxt++;
33
34         paths[curr]++;
35         curr = trie[curr][b];
36     }
37
38     paths[curr]++;
39     finish[curr]++;
40 }
41
42 void rem(int x) {
43     int curr = 0;
44
45     for (int i = 31; i >= 0; i--) {
46         int b = ((x & (1 << i)) > 0);
47
48         paths[curr]--;
49         curr = trie[curr][b];
50     }
51
52     paths[curr]--;
53     finish[curr]--;
54 }
55
56 int search(int x) {
57     int curr = 0;
58
59     for (int i = 31; i >= 0; i--) {
60         int b = ((x & (1 << i)) > 0);
61
62         if (trie[curr][b] == 0) return false;
63
64         curr = trie[curr][b];
65     }
66
67     return (finish[curr] > 0);
68 }
69
70 int max_xor(int x) { // maximum xor with x and
71     any number of trie
72     int curr = 0, ans = 0;
73
74     for (int i = 31; i >= 0; i--) {
75         int b = ((x & (1 << i)) > 0);
76         int want = b^1;
77
78         if (trie[curr][want] == 0 || paths[trie[
79 curr][want]] == 0) want ^= 1;
80         if (trie[curr][want] == 0 || paths[trie[
81 curr][want]] == 0) break;
82         if (want != b) ans |= (1 << i);
83
84         curr = trie[curr][want];
85     }
86
87     return ans;
88 }
89
90 // n(a) = 2*x e n(~a) = 2*x+1
91 // a = 2 ; n(a) = 4 ; n(~a) = 5 ; n(a)^1 = 5 ; n(~a)
92 // ^1 = 4
93
94 // https://cses.fi/problemset/task/1684/
95 // https://codeforces.com/gym/104120/problem/E
96 // (add_eq, add_true, add_false e at_most_one nÃo
97 // foram testadas)
98
99 // O(n + m)
100
101 struct sat {
102     int n, tot;
103     vector<vector<int>> adj, adjt; // grafo original,
104     grafo transposto
105     vector<int> vis, comp, ans;
106     stack<int> topo; // ordem topolÃgica
107
108     sat() {}
109     sat(int n_) : n(n_), tot(n), adj(2*n), adjt(2*n)
110     {}
111
112     void dfs(int x) {
113         vis[x] = true;
114
115         for (auto e : adj[x]) {
116             if (!vis[e]) dfs(e);
117         }
118
119         topo.push(x);
120     }
121
122     void dfst(int x, int& id) {
123         vis[x] = true;
124         comp[x] = id;
125
126         for (auto e : adjt[x]) {
127             if (!vis[e]) dfst(e, id);
128         }
129     }
130
131     void add_impl(int a, int b) { // a -> b = (!a or
132     b)
133         a = (a >= 0 ? 2*a : -2*a-1);
134         b = (b >= 0 ? 2*b : -2*b-1);
135
136         adj[a].push_back(b);
137         adj[b^1].push_back(a^1);
138
139         adjt[b].push_back(a);
140         adjt[a^1].push_back(b^1);
141     }
142
143     void add_or(int a, int b) { // a or b
144         add_impl(~a, b);
145     }
146
147     void add_nor(int a, int b) { // a nor b = !(a or
148     b)
149         add_or(~a, b), add_or(a, ~b), add_or(~a, ~b);
150     }
151
152     void add_and(int a, int b) { // a and b
153         add_or(a, b), add_or(~a, b), add_or(a, ~b);
154     }
155
156     void add_nand(int a, int b) { // a nand b = !(a
157     and b)
158         add_or(~a, ~b);
159     }
160
161     void add_xor(int a, int b) { // a xor b = (a != b
162     )
163
164 }

```

## 3 Math

### 3.1 2sat

```

1 // 2SAT
2 //
3 // verifica se existe e encontra soluÃo
4 // para fÃrmulas booleanas da forma
5 // (a or b) and (!a or c) and (...)
6 //
7 // indexado em 0

```

```

73     add_or(a, b), add_or(~a, ~b);
74 }
75
76 void add_xnor(int a, int b) { // a xnor b = !(a
xor b) = (a == b)
77     add_xor(~a, b);
78 }
79
80 void add_true(int a) { // a = T
81     add_or(a, ~a);
82 }
83
84 void add_false(int a) { // a = F
85     add_and(a, ~a);
86 }
87
88 // magia - brunomaletta
89 void add_true_old(int a) { // a = T (n sei se
funciona)
90     add_impl(~a, a);
91 }
92
93 void at_most_one(vector<int> v) { // no max um
verdadeiro
94     adj.resize(2*(tot+v.size()));
95     for (int i = 0; i < v.size(); i++) {
96         add_impl(tot+i, ~v[i]);
97         if (i) {
98             add_impl(tot+i, tot+i-1);
99             add_impl(v[i], tot+i-1);
100         }
101     }
102     tot += v.size();
103 }
104
105 pair<bool, vector<int>> solve() {
106     ans.assign(n, -1);
107     comp.assign(2*tot, -1);
108     vis.assign(2*tot, 0);
109     int id = 1;
110
111     for (int i = 0; i < 2*tot; i++) if (!vis[i])
dfs(i);
112
113     vis.assign(2*tot, 0);
114     while (topo.size()) {
115         auto x = topo.top();
116         topo.pop();
117
118         if (!vis[x]) {
119             dfst(x, id);
120             id++;
121         }
122     }
123
124     for (int i = 0; i < tot; i++) {
125         if (comp[2*i] == comp[2*i+1]) return {
false, {} };
126         ans[i] = (comp[2*i] > comp[2*i+1]);
127     }
128
129     return {true, ans};
130 }
131 };

```

## 4 DS

### 4.1 Dsu

```

1 /*
2 DSU - Disjoint Set Union (or Union Find)
3

```

```

4 find(x) -> find component that x is on
5 join(a, b) -> union of a set containing 'a' and set
containing b
6
7 find / join with path comprehension -> O(inv_Ackermann
(n)) [O(1)]
8 find / join without path comprehension -> O(logN)
9
10 https://judge.yosupo.jp/submission/126864
11 */
12
13 struct DSU {
14
15     int n = 0, components = 0;
16     vector<int> parent;
17     vector<int> size;
18
19     DSU(int nn){
20         n = nn;
21         components = n;
22         size.assign(n + 5, 1);
23         parent.assign(n + 5, 0);
24         iota(parent.begin(), parent.end(), 0);
25     }
26
27     int find(int x){
28         if(x == parent[x]) {
29             return x;
30         }
31         //path compression
32         return parent[x] = find(parent[x]);
33     }
34
35     void join(int a, int b){
36         a = find(a);
37         b = find(b);
38         if(a == b) {
39             return;
40         }
41         if(size[a] < size[b]) {
42             swap(a, b);
43         }
44         parent[b] = a;
45         size[a] += size[b];
46         components -= 1;
47     }
48
49     int sameSet(int a, int b) {
50         a = find(a);
51         b = find(b);
52         return a == b;
53     }
54 }
55 };

```

### 4.2 Ordered Set

```

1 // Ordered Set
2 //
3 // set roubado com mais operacoes
4 //
5 // para alterar para multiset
6 // trocar less para less_equal
7 //
8 // ordered_set<int> s
9 //
10 // order_of_key(k) // number of items strictly
smaller than k -> int
11 // find_by_order(k) // k-th element in a set (
counting from zero) -> iterator
12 //
13 // https://cses.fi/problemset/task/2169
14 //

```

```

15 // O(log N) para insert, erase (com iterator),
    order_of_key, find_by_order
16
17 using namespace __gnu_pbds;
18 template <typename T>
19 using ordered_set = tree<T,null_type,less<T>,
    rb_tree_tag,tree_order_statistics_node_update>;

```

## 5 Graph

### 5.1 Dinic

```

1 // Dinic / Dinitz
2 //
3 // max-flow / min-cut
4 //
5 // https://cses.fi/problemset/task/1694/
6 //
7 // O(E * V^2)
8
9 using ll = long long;
10 const ll FLOW_INF = 1e18 + 7;
11
12 struct Edge {
13     int from, to;
14     ll cap, flow;
15     Edge* residual; // a inversa da minha aresta
16
17     Edge() {};
```

```

18
19     Edge(int from, int to, ll cap) : from(from), to(to), cap(cap), flow(0) {};
```

```

20
21     ll remaining_cap() {
22         return cap - flow;
23     }
24
25     void augment(ll bottle_neck) {
26         flow += bottle_neck;
27         residual->flow -= bottle_neck;
28     }
29
30     bool is_residual() {
31         return cap == 0;
32     }
33 };
34
35 struct Dinic {
36     int n;
37     vector<vector<Edge*>> adj;
38     vector<int> level, next;
39
40     Dinic(int n): n(n) {
41         adj.assign(n+1, vector<Edge*>());
42         level.assign(n+1, -1);
43         next.assign(n+1, 0);
44     }
45
46     void add_edge(int from, int to, ll cap) {
47         auto e1 = new Edge(from, to, cap);
48         auto e2 = new Edge(to, from, 0);
49
50         e1->residual = e2;
51         e2->residual = e1;
52
53         adj[from].push_back(e1);
54         adj[to].push_back(e2);
55     }
56
57     bool bfs(int s, int t) {
58         fill(level.begin(), level.end(), -1);
59         queue<int> q;

```

```

60
61         q.push(s);
62         level[s] = 1;
63
64         while (q.size()) {
65             int curr = q.front();
66             q.pop();
67
68             for (auto edge : adj[curr]) {
69                 if (edge->remaining_cap() > 0 &&
50 level[edge->to] == -1) {
70                     level[edge->to] = level[curr] +
51 1;
71
72                     q.push(edge->to);
73                 }
74             }
75
76             return level[t] != -1;
77         }
78
79         ll dfs(int x, int t, ll flow) {
80             if (x == t) return flow;
81
82             for (int& cid = next[x]; cid < (int)adj[x].
size(); cid++) {
83                 auto& edge = adj[x][cid];
84                 ll cap = edge->remaining_cap();
85
86                 if (cap > 0 && level[edge->to] == level[x
] + 1) {
87                     ll sent = dfs(edge->to, t, min(flow,
cap)); // bottle neck
88                     if (sent > 0) {
89                         edge->augment(sent);
90                         return sent;
91                     }
92                 }
93             }
94
95             return 0;
96         }
97
98         ll solve(int s, int t) {
99             ll max_flow = 0;
100
101             while (bfs(s, t)) {
102                 fill(next.begin(), next.end(), 0);
103
104                 while (ll sent = dfs(s, t, FLOW_INF)) {
105                     max_flow += sent;
106                 }
107             }
108
109             return max_flow;
110         }
111
112         // path recover
113         vector<bool> vis;
114         vector<int> curr;
115
116         bool dfs2(int x, int& t) {
117             vis[x] = true;
118             bool arrived = false;
119
120             if (x == t) {
121                 curr.push_back(x);
122                 return true;
123             }
124
125             for (auto e : adj[x]) {
126                 if (e->flow > 0 && !vis[e->to]) { // !e->
is_residual() &&

```

```

127         bool aux = dfs2(e->to, t);
128
129         if (aux) {
130             arrived = true;
131             e->flow--;
132         }
133     }
134 }
135
136 if (arrived) curr.push_back(x);
137
138 return arrived;
139 }
140
141 vector<vector<int>> get_paths(int s, int t) {
142     vector<vector<int>> ans;
143
144     while (true) {
145         curr.clear();
146         vis.assign(n+1, false);
147
148         if (!dfs2(s, t)) break;
149
150         reverse(curr.begin(), curr.end());
151         ans.push_back(curr);
152     }
153
154     return ans;
155 }
156 };

```

## 5.2 Dfs

```

1 // DFS
2 //
3 // Percorre todos os vertices
4 // priorizando profundidade
5 //
6 // O(n+m)
7
8 vector<vector<int>> g;
9 vector<bool> vis;
10
11 void dfs(int s){
12     if(vis[s]) return;
13     vis[s] = true;
14     for(auto v : g[s]){
15         dfs(v);
16     }
17 }

```

## 5.3 Ford Fulkerson

```

1 // Ford-Fulkerson
2 //
3 // max-flow / min-cut
4 //
5 // MAX nÃ§s
6 //
7 // https://cses.fi/problemset/task/1694/
8 //
9 // O(m * max_flow)
10
11 using ll = long long;
12 const int MAX = 510;
13
14 struct Flow {
15     int n;
16     ll adj[MAX][MAX];
17     bool used[MAX];
18
19     Flow(int n) : n(n) {};

```

```

20
21 void add_edge(int u, int v, ll c) {
22     adj[u][v] += c;
23     adj[v][u] = 0; // cuidado com isso
24 }
25
26 ll dfs(int x, int t, ll amount) {
27     used[x] = true;
28
29     if (x == t) return amount;
30
31     for (int i = 1; i <= n; i++) {
32         if (adj[x][i] > 0 && !used[i]) {
33             ll sent = dfs(i, t, min(amount, adj[x
34 ][i]));
35
36             if (sent > 0) {
37                 adj[x][i] -= sent;
38                 adj[i][x] += sent;
39
40                 return sent;
41             }
42         }
43     }
44
45     return 0;
46 }
47
48 ll max_flow(int s, int t) { // source and sink
49     ll total = 0;
50     ll sent = -1;
51
52     while (sent != 0) {
53         memset(used, 0, sizeof(used));
54         sent = dfs(s, t, INT_MAX);
55         total += sent;
56     }
57
58     return total;
59 };

```

## 6 Geometry

### 6.1 Convex Hull

```

1 // Convex Hull - Monotone Chain
2 //
3 // Convex Hull is the subset of points that forms the
4 // smallest convex polygon
5 // which encloses all points in the set.
6 // https://cses.fi/problemset/task/2195/
7 // https://open.kattis.com/problems/convexhull (
8 // counterclockwise)
9 //
10 // O(n log(n))
11
12 typedef long long ftype;
13
14 struct Point {
15     ftype x, y;
16
17     Point() {};
18     Point(ftype x, ftype y) : x(x), y(y) {};
19
20     bool operator<(Point o) {
21         if (x == o.x) return y < o.y;
22         return x < o.x;
23     }
24
25     bool operator==(Point o) {

```



```

25     return x == o.x && y == o.y;
26 }
27 };
28
29 ftype cross(Point a, Point b, Point c) {
30     // v: a -> c
31     // w: a -> b
32
33     // v: c.x - a.x, c.y - a.y
34     // w: b.x - a.x, b.y - a.y
35
36     return (c.x - a.x) * (b.y - a.y) - (c.y - a.y) *
37         (b.x - a.x);
38 }
39
40 ftype dir(Point a, Point b, Point c) {
41     // 0 -> colineares
42     // -1 -> esquerda
43     // 1 -> direita
44
45     ftype cp = cross(a, b, c);
46
47     if (cp == 0) return 0;
48     else if (cp < 0) return -1;
49     else return 1;
50 }
51
52 vector<Point> convex_hull(vector<Point> points) {
53     sort(points.begin(), points.end());
54     points.erase( unique(points.begin(), points.end())
55         , points.end()); // somente pontos distintos
56     int n = points.size();
57
58     if (n == 1) return { points[0] };
59
60     vector<Point> upper_hull = {points[0], points
61         [1]};
62     for (int i = 2; i < n; i++) {
63         upper_hull.push_back(points[i]);
64
65         int sz = upper_hull.size();
66
67         while (sz >= 3 && dir(upper_hull[sz-3],
68             upper_hull[sz-2], upper_hull[sz-1]) == -1) {
69             upper_hull.pop_back();
70             upper_hull.pop_back();
71             upper_hull.push_back(points[i]);
72             sz--;
73         }
74     }
75
76     vector<Point> lower_hull = {points[n-1], points[n
77         -2]};
78     for (int i = n-3; i >= 0; i--) {
79         lower_hull.push_back(points[i]);
80
81         int sz = lower_hull.size();
82
83         while (sz >= 3 && dir(lower_hull[sz-3],
84             lower_hull[sz-2], lower_hull[sz-1]) == -1) {
85             lower_hull.pop_back();
86             lower_hull.pop_back();
87             lower_hull.push_back(points[i]);
88             sz--;
89         }
90     }
91
92     // reverse(lower_hull.begin(), lower_hull.end());
93     // counterclockwise
94
95     for (int i = (int)lower_hull.size() - 2; i > 0; i
96         --) {
97         upper_hull.push_back(lower_hull[i]);
98     }
99 }

```

```

90 }
91
92 return upper_hull;
93 }

```

## 7 DP

### 7.1 Edit Distance

```

1 // Edit Distance / Levenshtein Distance
2 //
3 // numero minimo de operacoes
4 // para transformar
5 // uma string em outra
6 //
7 // tamanho da matriz da dp eh |a| x |b|
8 // edit_distance(a.size(), b.size(), a, b)
9 //
10 // https://cses.fi/problemset/task/1639
11 //
12 // O(n^2)
13
14 int tb[MAX][MAX];
15
16 int edit_distance(int i, int j, string &a, string &b)
17 {
18     if (i == 0) return j;
19     if (j == 0) return i;
20
21     int &ans = tb[i][j];
22
23     if (ans != -1) return ans;
24
25     ans = min({
26         edit_distance(i-1, j, a, b) + 1,
27         edit_distance(i, j-1, a, b) + 1,
28         edit_distance(i-1, j-1, a, b) + (a[i-1] != b[
29             j-1])
30     });
31
32     return ans;
33 }

```

### 7.2 Lcs

```

1 // LCS (Longest Common Subsequence)
2 //
3 // maior subsequencia comum entre duas strings
4 //
5 // tamanho da matriz da dp eh |a| x |b|
6 // lcs(a, b) = string da melhor resposta
7 // dp[a.size()][b.size()] = tamanho da melhor
8 // resposta
9 //
10 // https://atcoder.jp/contests/dp/tasks/dp_f
11 //
12 // O(n^2)
13
14 string lcs(string a, string b) {
15     int n = a.size();
16     int m = b.size();
17
18     int dp[n+1][m+1];
19     pair<int, int> p[n+1][m+1];
20
21     memset(dp, 0, sizeof(dp));
22     memset(p, -1, sizeof(p));
23
24     for (int i = 1; i <= n; i++) {
25         for (int j = 1; j <= m; j++) {
26             if (a[i-1] == b[j-1]) {

```

```
26         dp[i][j] = dp[i-1][j-1] + 1;
27         p[i][j] = {i-1, j-1};
28     } else {
29         if (dp[i-1][j] > dp[i][j-1]) {
30             dp[i][j] = dp[i-1][j];
31             p[i][j] = {i-1, j};
32         } else {
33             dp[i][j] = dp[i][j-1];
34             p[i][j] = {i, j-1};
35         }
36     }
37 }
38 }
39
40 // recuperar resposta
41
42 string ans = "";
43 pair<int, int> curr = {n, m};
44
```

```
45 while (curr.first != 0 && curr.second != 0) {
46     auto [i, j] = curr;
47
48     if (a[i-1] == b[j-1]) {
49         ans += a[i-1];
50     }
51
52     curr = p[i][j];
53 }
54
55 reverse(ans.begin(), ans.end());
56
57 return ans;
58 }
```

### 7.3 Knapsack