

.\*VSTU.\*

Team Reference Document

01/12/2018

CONTENTS

1. Code Templates	2
1.1. Basic Configuration	2
1.2. Vector	3
1.3. FFT	4
1.4. Matrix	5
1.5. SegmTree	5
1.6. Aho	6
1.7. Suffix Automaton	7
1.8. Stoer Wagner	8
1.9. Flow	8
1.10. Prefix function	9
1.11. BPWS	9
1.12. Bridge search	11
1.13. Lca	11
1.14. 2-SAT	12
1.15. Centroid	12
2. Misc	14
2.1. Debugging Tips	14
2.2. Solution Ideas	14
2.3. The Twelfefold Way	16
Practice Contest Checklist	17

## 1. CODE TEMPLATES

## 1.1. Basic Configuration.

1.1.1. *.vimrc*.

```
set cin nu ts=2 sw=2 sts=2 mouse=a
syn on
```

```
function! Compile()
    :!g++ -std=gnu++11 -g % -o %<.exe
endfunction
```

```
function! Run()
    :!time ./%<.exe
endfunction
```

```
map <F4> :call Compile(<cr>
map <F5> :call Run(<cr>
map <C-A> ggVG"+y
```

1.1.2. *stress and template*.

```
1 // g++ -std=c++11 main.cpp -o main -D"_DEBUG_TEMICH_"
2 // -Wall -Wextra -pedantic -std=c++11
3 // -O2 -Wshadow -Wformat=2 -Wfloat-equal
4 // -Wconversion -Wlogical-op -Wshift-overflow=2
5 // -Wduplicated-cond -Wcast-qual -Wcast-align
6 // -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
7 // -D_FORTIFY_SOURCE=2 -fsanitize=address
8 // -fsanitize=undefined -fno-sanitize-recover
9 // -fstack-protector
10 #pragma GCC optimize("O3")
11 #pragma GCC target(
12     "sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx")
13
14 #include <algorithm>
15 #include <cmath>
16 #include <functional>
17 #include <iostream>
18 #include <map>
19 #include <queue>
20 #include <set>
21 #include <sstream>
22 #include <string>
23 #include <vector>
24
25 using namespace std;
26
27 using LL = long long;
```

```
28 using pii = pair<int, int>;
29
30 #define X first
31 #define Y second
32
33 template<typename T>
34 ostream& operator<<(ostream& out, const vector<T>& v);
35
36 template<typename U, typename V>
37 ostream& operator<<(ostream& out, const map<U, V>& v);
38
39 template<typename U, typename V>
40 ostream& operator<<(ostream& out, const pair<U, V>& v);
41
42 template<typename U, typename V>
43 ostream& operator<<(ostream& out, const pair<U, V>& v) {
44     return out << "(" << v.first << ", " << v.second << ")";
45 }
46
47 template<typename U, typename V>
48 ostream& operator<<(ostream& out, const map<U, V>& v) {
49     out << "{";
50     bool f = false;
51     for (const auto& p : v) {
52         out << (!f ? "" : ", ") << p;
53         f = true;
54     }
55     return out << "}";
56 }
57
58 template<typename T>
59 ostream& operator<<(ostream& out, const vector<T>& v) {
60     out << "{";
61     for (int i = 0; i < int(v.size()); ++i)
62         out << (i == 0 ? "" : ", ") << v[i];
63     return out << "}";
64 }
65
66 void cerr_printer(bool start) {}
67 template<typename T, typename ... Args>
68 void cerr_printer(bool start, const T& x, const Args& ... args) {
69     if (!start) cerr << ", ";
70     cerr << x;
71     cerr_printer(false, args...);
72 }
73
74 template<typename ... Args>
```

```

75 void dbg(const char * name, int line, const Args& ... args) {
76     cerr << "[" << line << "]" (" << name << ") = (" ;
77     cerr_printer(true, args...);
78     cerr << ")" << endl;
79 }
80
81 #define DBG(...) { dbg(__VA_ARGS__, __LINE__, __VA_ARGS__); }
82
83 struct Solver {
84     void solve(istream& cin, ostream& cout) {
85         int a, b;
86         cin >> a >> b;
87         cout << a + b << endl;
88     }
89 };
90
91 struct Brute {
92     void solve(istream& cin, ostream& cout) {
93         int a, b;
94         cin >> a >> b;
95         while (b--) ++a;
96         cout << a << endl;
97     }
98 };
99
100 template <typename Solution>
101 struct SolutionStr {
102     string solve(string input) {
103         istringstream is(input);
104         ostringstream os;
105         Solution().solve(is, os);
106         return os.str();
107     }
108 };
109
110 string gen_input(int it) {
111     (void)it;
112     return "10 20";
113 }
114
115 void stress() {
116     for (int it = 0; it < 1000; ++it) {
117         auto input = gen_input(it);
118         auto brute_out = SolutionStr<Solver>().solve(input);
119         auto sol_out = SolutionStr<Brute>().solve(input);
120         if (sol_out != brute_out) {
121             cerr << "WA #" << it << endl;

```

```

122         cerr << "input: " << endl;
123         cerr << input << endl;
124         cerr << "expected: " << brute_out << endl;
125         cerr << "got: " << sol_out << endl;
126         exit(1);
127     }
128 }
129
130 cerr << "OK" << endl;
131 }
132
133 int main() {
134     #ifdef _DEBUG_TEMICH_
135     stress();
136     #endif
137     Solver().solve(cin, cout);
138 }

```

## 1.2. Vector.

```

1 struct Vec {
2     LL x, y;
3     explicit Vec(LL x = 0, LL y = 0) : x(x), y(y) {}
4     Vec operator+(const Vec& o) const {
5         return Vec(x + o.x, y + o.y); }
6     Vec operator-(const Vec& o) const {
7         return Vec(x - o.x, y - o.y); }
8     Vec operator*(const LL p) const {
9         return Vec(x * p, y * p); }
10    double len() const { return sqrt(x * x + y * y); }
11    LL cross(const Vec& o) const { return x * o.y - y * o.x; }
12    LL dot(const Vec& o) const { return x * o.x + y * o.y; }
13    static Vec read(istream& cin) {
14        LL x, y;
15        cin >> x >> y;
16        return Vec(x, y);
17    }
18 };
19
20 // CONVEX HULL: last point == first point
21 vector<Vec> convex_hull(vector<Vec> a) {
22     int n = a.size(), k = 0;
23     vector<Vec> p(n * 2);
24     sort(a.begin(), a.end());
25
26     for(int i = 0; i < n; p[k++] = a[i++])
27         while(k > 1 && (p[k - 1] - p[k - 2])
28             % (p[k - 1] - a[i]) >= 0) --k;

```

```

29   for(int i = n - 2, w = k; i >= 0; p[k++] = a[i--])
30       while(k > w && (p[k - 1] - p[k - 2])
31           % (p[k - 1] - a[i]) >= 0) --k;
32   p.resize(k);
33   return p;
34 }

```

### 1.3. FFT.

```

1  struct Complex {
2      long double re, im;
3      explicit Complex(long double re = 0,
4          long double im = 0) : re(re), im(im) {}
5      Complex operator+(const Complex& o) const {
6          return Complex(re + o.re, im + o.im); }
7      Complex operator-(const Complex& o) const {
8          return Complex(re - o.re, im - o.im); }
9      Complex operator*(const Complex& o) const {
10         return Complex(re * o.re - im * o.im, re * o.im + im * o.re); }
11 };
12
13 const int MAX_SHIFT = 22;
14 const int MAX_N      = 1 << MAX_SHIFT;
15
16 const double Pi = acos(-1);
17
18 Complex roots[MAX_N / 2];
19 int bit_reverse[MAX_N];
20
21 void prep() {
22     bit_reverse[0] = 0;
23     for (int i = 1; i < MAX_N; ++i)
24         bit_reverse[i] = (bit_reverse[i >> 1]
25             | ((i & 1) << MAX_SHIFT)) >> 1;
26
27     for (int i = 0; i + i < MAX_N; ++i) {
28         double angle = 2 * i * Pi / MAX_N;
29         roots[i] = Complex(cos(angle), sin(angle));
30     }
31 }
32
33 Complex arr[MAX_N];
34 void fft(int k) {
35     assert(k <= MAX_SHIFT);
36
37     const int n = 1 << k;
38     for (int i = 0; i < n; ++i) {
39         int rv = bit_reverse[i] >> (MAX_SHIFT - k);

```

```

40         if (rv < i) swap(arr[i], arr[rv]);
41     }
42
43     for (int bs = 2; bs <= n; bs *= 2) {
44         const int hbs = bs / 2;
45         const int factor = (MAX_N / 2) / hbs;
46         for (int i = 0; i < n; i += bs) {
47             for (int j = 0; j < hbs; ++j) {
48                 auto a = arr[i + j];
49                 auto b = arr[i + j + hbs] * roots[factor * j];
50                 arr[i + j] = a + b;
51                 arr[i + j + hbs] = a - b;
52             }
53         }
54     }
55 }
56
57 const int Base = 100;
58
59 void square(vector<int>& number) {
60     int sz = number.size() * 2;
61     int k = 1;
62     {
63         int rsz = 2;
64         while (rsz < sz) {
65             rsz *= 2;
66             ++k;
67         }
68
69         sz = rsz;
70     }
71
72     assert(sz <= MAX_N);
73
74     for (int i = 0; i < sz; ++i)
75         arr[i] = Complex(i < number.size() ? number[i] : 0);
76
77     fft(k);
78     for (int i = 0; i < sz; ++i)
79         arr[i] = arr[i] * arr[i];
80     fft(k);
81     reverse(arr + 1, arr + sz);
82
83     number.resize(sz);
84     int cr = 0;
85     for (int i = 0; i < sz; ++i) {
86         number[i] = cr + int(arr[i].re / sz + 0.5);

```

```

87     cr = number[i] / Base;
88     number[i] %= Base;
89 }
90
91 while (number.back() == 0) number.pop_back();
92 }

```

#### 1.4. Matrix.

```

1 struct Matrix {
2     ULL vals[N][N];
3     Matrix() {
4         for (int i = 0; i < N; ++i)
5             fill(vals[i], vals[i] + N, 0);
6     }
7
8     ULL* operator[](const int idx) {
9         return vals[idx];
10    }
11
12    const ULL* operator[](const int idx) const {
13        return vals[idx];
14    }
15
16    static Matrix Ident() {
17        Matrix res;
18        for (int i = 0; i < N; ++i)
19            res[i][i] = 1;
20
21        return res;
22    }
23
24    Matrix operator*(const Matrix& o) const {
25        Matrix res;
26
27        for (int i = 0; i < N; ++i) {
28            for (int j = 0; j < N; ++j) {
29                for (int k = 0; k < N; ++k) {
30                    res[i][j] += vals[i][k] * o[k][j];
31                    if (k == 7)
32                        res[i][j] %= MOD;
33                }
34                res[i][j] %= MOD;
35            }
36        }
37
38        return res;

```

```

39     }
40 };

```

#### 1.5. SegmTree.

```

1 class SegmTreeSum {
2     vector<int> tree;
3     int n;
4
5     int get(int v, int l, int r, int L, int R) const {
6         if (L > R) return 0;
7         if (l == L && r == R) return tree[v];
8
9         int mid = (l + r) / 2;
10
11        int a = get(2 * v + 1, l, mid, L, min(R, mid));
12        int b = get(2 * v + 2, mid + 1, r, max(L, mid + 1), R);
13
14        return a + b;
15    }
16
17    void set(int v, int l, int r, int pos, int val) {
18        if (l == r) {
19            tree[pos] = val;
20            return;
21        }
22
23        int mid = (l + r) / 2;
24
25        if (pos <= mid) set(2 * v + 1, l, mid, pos, val);
26        else set(2 * v + 2, mid + 1, r, pos, val);
27
28        tree[v] = tree[2 * v + 1] + tree[2 * v + 2];
29    }
30
31    public:
32    void init(int n_) {
33        n = n_;
34        tree.assign(4 * n, 0);
35    }
36
37    int get(int l, int r) const {
38        return get(0, 0, n - 1, l, r);
39    }
40
41    void set(int pos, int val) {
42        set(0, 0, n - 1, pos, val);
43    }

```

```

44 };
45
46 class SegmTreeMax {
47     vector<Pair> tree;
48     vector<int> psh;
49     int n;
50
51     void build(int v, int l, int r, const vector<int>& dp) {
52         if (l == r) {
53             tree[v] = Pair(dp[l], l);
54             return;
55         }
56
57         int mid = (l + r) / 2;
58
59         build(2 * v + 1, l, mid, dp);
60         build(2 * v + 2, mid + 1, r, dp);
61
62         tree[v] = max(tree[2 * v + 1], tree[2 * v + 2]);
63     }
64
65     void push(int v, int l, int r) {
66         if (l != r) {
67             psh[2 * v + 1] += psh[v];
68             psh[2 * v + 2] += psh[v];
69         }
70         tree[v].X += psh[v];
71         psh[v] = 0;
72     }
73
74     Pair getMax(int v, int l, int r, int L, int R) {
75         push(v, l, r);
76         if (L > R) return Pair(-INF, -INF);
77
78         if (l == L && r == R)
79             return tree[v];
80
81         int mid = (l + r) / 2;
82
83         Pair a = getMax(2 * v + 1, l, mid, L, min(R, mid));
84         Pair b = getMax(2 * v + 2, mid + 1, r, max(L, mid + 1), R);
85
86         return max(a, b);
87     }
88
89     void add(int v, int l, int r, int L, int R, int val) {
90         push(v, l, r);

```

```

91
92     if (L > R) return;
93     if (l == L && r == R) {
94         psh[v] += val;
95         push(v, l, r);
96         return;
97     }
98
99     int mid = (l + r) / 2;
100
101     add(2 * v + 1, l, mid, L, min(R, mid), val);
102     add(2 * v + 2, mid + 1, r, max(L, mid + 1), R, val);
103
104     tree[v] = max(tree[2 * v + 1], tree[2 * v + 2]);
105 }
106
107 public:
108     void init(const vector<int>& dp) {
109         n = dp.size();
110         tree.resize(4 * n);
111         psh.assign(4 * n, 0);
112
113         build(0, 0, n - 1, dp);
114     }
115
116     Pair getMax(int l, int r) {
117         return getMax(0, 0, n - 1, l, r);
118     }
119
120     void add(int l, int r, int val) {
121         add(0, 0, n - 1, l, r, val);
122     }
123 };

```

#### 1.6. Aho.

```

1 struct Matcher {
2     static const int LETTERS_COUNT = 'z' - 'a' + 1;
3     struct Next {
4         int nxt[LETTERS_COUNT];
5         Next() { fill(nxt, nxt + LETTERS_COUNT, -1); }
6         int& operator[](char c) { return nxt[c - 'a']; }
7     };
8
9     vector<Next> next;
10    vector<int> link;
11    vector<char> p_char;
12    vector<int> p;

```

```

13     vector<int> id;
14
15     void build(const set<string>& strings) {
16         int total_size = 0;
17         for (const auto& s : strings)
18             total_size += s.size();
19         next.reserve(total_size);
20         link.reserve(total_size);
21         p_char.reserve(total_size);
22         p.reserve(total_size);
23
24         push();
25
26         int _id = 0;
27         for (const auto& s : strings) {
28             add(s, _id);
29             ++_id;
30         }
31     }
32
33     void push() {
34         next.push_back(Next());
35         link.push_back(-1);
36         p_char.push_back('#');
37         p.push_back(-1);
38         id.push_back(-1);
39     }
40
41     void add(const string& s, int _id) {
42         int state = 0;
43
44         for (char c : s) {
45             int next_state = next[state][c];
46             if (next_state == -1) {
47                 push();
48                 p_char.back() = c;
49                 p.back() = state;
50                 next_state = p.size() - 1;
51                 next[state][c] = next_state;
52             }
53
54             state = next_state;
55         }
56
57         id[state] = _id;
58     }
59

```

```

60     int get_next(int state, char c) {
61         int x = _get_next(state, c);
62         // cerr << "get next " << state << " " << c << " = " << x << endl;
63         return x;
64     }
65
66     int _get_next(int state, char c) {
67         if (next[state][c] == -1 && state == 0)
68             return 0;
69         if (next[state][c] == -1)
70             next[state][c] = get_next(get_link(state), c);
71         return next[state][c];
72     }
73
74     int get_link(int state) { int x = _get_link(state);
75         // cerr << "get link " << state << " = " << x << endl;
76         return x;
77     }
78
79     int _get_link(int state) {
80         if (state == 0)
81             return 0;
82         if (p[state] == 0)
83             return 0;
84
85         int& l = link[state];
86         if (l == -1)
87             l = get_next(get_link(p[state]), p_char[state]);
88         return l;
89     }
90
91     int get_id(int state) { return id[state]; }
92 };

```

### 1.7. Suffix Automaton.

```

1     struct State {
2         map<char, int> nxt;
3         int link;
4         int len;
5         bool added;
6         int cnt;
7     };
8
9     State st[N];
10    int lst;
11    int sz;
12

```

```

13 void init() {
14     lst = 0;
15     sz = 1;
16     st[0].link = -1;
17     st[0].len = 1;
18 }
19
20 void ext(char c) {
21     // cerr << "ext : " << c << endl;
22     int cur = sz++;
23     st[cur].len = st[lst].len + 1;
24
25     int p;
26     for (p = lst; p != -1 && !st[p].nxt.count(c); p = st[p].link)
27         st[p].nxt[c] = cur;
28
29     if (p == -1) {
30         st[cur].link = 0;
31     } else {
32         int q = st[p].nxt[c];
33         if (st[p].len + 1 == st[q].len) {
34             st[cur].link = q;
35         } else {
36             int clone = sz++;
37             st[clone] = st[q];
38             st[clone].len = st[p].len + 1;
39             st[clone].cnt = st[st[clone].link].cnt;
40
41             st[q].link = st[cur].link = clone;
42
43             for (; p != -1 && st[p].nxt[c] == q; p = st[p].link)
44                 st[p].nxt[c] = clone;
45         }
46     }
47
48     lst = cur;
49     st[cur].cnt = st[st[cur].link].cnt;
50 }

```

#### 1.8. Stoer Wagner.

```

1 const int MAXN = 500;
2 int n, g[MAXN][MAXN];
3 int best_cost = 1000000000;
4 vector<int> best_cut;
5
6 void mincut() {
7     vector<int> v[MAXN];

```

```

8     for (int i=0; i<n; ++i)
9         v[i].assign(1, i);
10    int w[MAXN];
11    bool exist[MAXN], in_a[MAXN];
12    memset(exist, true, sizeof exist);
13    for (int ph=0; ph<n-1; ++ph) {
14        memset(in_a, false, sizeof in_a);
15        memset(w, 0, sizeof w);
16        for (int it=0, prev; it<n-ph; ++it) {
17            int sel = -1;
18            for (int i=0; i<n; ++i)
19                if (exist[i] && !in_a[i] && (sel == -1
20                    || w[i] > w[sel]))
21                    sel = i;
22            if (it == n-ph-1) {
23                if (w[sel] < best_cost)
24                    best_cost = w[sel], best_cut = v[sel];
25                v[prev].insert(v[prev].end(),
26                    v[sel].begin(), v[sel].end());
27                for (int i=0; i<n; ++i)
28                    g[prev][i] = g[i][prev] += g[sel][i];
29                exist[sel] = false;
30            }
31            else {
32                in_a[sel] = true;
33                for (int i=0; i<n; ++i)
34                    w[i] += g[sel][i];
35                prev = sel;
36            }
37        }
38    }
39 }

```

#### 1.9. Flow.

```

1 struct Edge {
2     int u, v, flow, cap;
3     Edge() : u(0), v(0), flow(0), cap(0) {}
4     Edge(int u, int v, int c) : u(u), v(v), flow(0), cap(c) {}
5 };
6
7 const int N = 666;
8 const int T = 1111;
9
10 const int MAXN = N + 2 * T + 100;
11
12
13 vector<int> g[500000];

```



```

14 vector<Edge> edges;
15
16 int flow, s, t;
17 int start[MAXN], used[MAXN], dist[MAXN];
18
19 bool bfs() {
20     memset(start, 0, sizeof(start));
21     memset(dist, -1, sizeof(dist));
22     dist[s] = 0;
23
24     queue<int> q;
25     q.push(s);
26
27     while (q.size()) {
28         int u = q.front();
29         q.pop();
30
31         for (int id : g[u]) {
32             Edge &e = edges[id];
33             int v = e.v;
34
35             if (dist[v] == -1 && e.flow < e.cap) {
36                 dist[v] = dist[u] + 1;
37                 q.push(v);
38             }
39         }
40     }
41     return dist[t] != -1;
42 }
43
44 int dfs(int u, int fl = -1) {
45     if (fl == -1) memset(used, false, sizeof(used));
46     used[u] = true;
47
48     if (u == t) return fl;
49
50     for (int &i = start[u]; i < g[u].size(); ++i) {
51         int id = g[u][i];
52         Edge &e = edges[id];
53         int v = e.v;
54         if (!used[v] && dist[v] == dist[u] + 1 && e.flow < e.cap) {
55             int can = e.cap - e.flow;
56             int df = dfs(v, fl == -1 ? can : min(fl, can));
57             if (df > 0) {
58                 edges[id ^ 0].flow += df;
59                 edges[id ^ 1].flow -= df;
60                 return df;

```

```

61     }
62 }
63 }
64 return 0;
65 }
66
67 void add_edge(int u, int v, int c) {
68     //cout << "add (" << u << " " << v << " " << c << ") " << endl;
69     g[u].push_back(edges.size());
70     edges.emplace_back(u, v, c);
71     g[v].push_back(edges.size());
72     edges.emplace_back(v, u, 0);
73 }
74
75 int calc(int ss, int tt) {
76     //cout << "calc (" << ss << ", " << tt << ")" << endl;
77     flow = 0, s = ss, t = tt;
78     while (bfs()) {
79         while (int add = dfs(ss)) {
80             flow += add;
81         }
82     }
83     return flow;
84 }

```

#### 1.10. Prefix function.

```

1 vector<int> prefix_function (string s) {
2     int n = (int) s.length();
3     vector<int> pi (n);
4     for (int i=1; i<n; ++i) {
5         int j = pi[i-1];
6         while (j > 0 && s[i] != s[j])
7             j = pi[j-1];
8         if (s[i] == s[j]) ++j;
9         pi[i] = j;
10    }
11    return pi;
12 }

```

#### 1.11. BPWS.

```

1 const int trivial_limit = 50;
2 int p[1000];
3
4 int gcd (int a, int b) {
5     return a ? gcd (b%a, a) : b;
6 }
7

```

```

8  int powmod (int a, int b, int m) {
9      int res = 1;
10     while (b)
11         if (b & 1)
12             res = (res * 1ll * a) % m, --b;
13         else
14             a = (a * 1ll * a) % m, b >>= 1;
15     return res;
16 }
17
18 bool miller_rabin (int n) {
19     int b = 2;
20     for (int g; (g = gcd (n, b)) != 1; ++b)
21         if (n > g)
22             return false;
23     int p=0, q=n-1;
24     while ((q & 1) == 0)
25         ++p, q >>= 1;
26     int rem = powmod (b, q, n);
27     if (rem == 1 || rem == n-1)
28         return true;
29     for (int i=1; i<p; ++i) {
30         rem = (rem * 1ll * rem) % n;
31         if (rem == n-1) return true;
32     }
33     return false;
34 }
35
36 int jacobi (int a, int b)
37 {
38     if (a == 0) return 0;
39     if (a == 1) return 1;
40     if (a < 0)
41         if ((b & 2) == 0)
42             return jacobi (-a, b);
43         else
44             return - jacobi (-a, b);
45     int a1=a, e=0;
46     while ((a1 & 1) == 0)
47         a1 >>= 1, ++e;
48     int s;
49     if ((e & 1) == 0 || (b & 7) == 1 || (b & 7) == 7)
50         s = 1;
51     else
52         s = -1;
53     if ((b & 3) == 3 && (a1 & 3) == 3)
54         s = -s;

```

```

55     if (a1 == 1)
56         return s;
57     return s * jacobi (b % a1, a1);
58 }
59
60 bool bpsw (int n) {
61     if ((int)sqrt(n+0.0) *
62         (int)sqrt(n+0.0) == n) return false;
63     int dd=5;
64     for (;;) {
65         int g = gcd (n, abs(dd));
66         if (1<g && g<n) return false;
67         if (jacobi (dd, n) == -1) break;
68         dd = dd<0 ? -dd+2 : -dd-2;
69     }
70     int p=1, q=(p*p-dd)/4;
71     int d=n+1, s=0;
72     while ((d & 1) == 0)
73         ++s, d>>=1;
74     long long u=1, v=p, u2m=1, v2m=p, qm=q, qm2=q*2, qkd=q;
75     for (int mask=2; mask<=d; mask<=1) {
76         u2m = (u2m * v2m) % n;
77         v2m = (v2m * v2m) % n;
78         while (v2m < qm2) v2m += n;
79         v2m -= qm2;
80         qm = (qm * qm) % n;
81         qm2 = qm * 2;
82         if (d & mask) {
83             long long t1 = (u2m * v) % n,
84                 t2 = (v2m * u) % n,
85                 t3 = (v2m * v) % n,
86                 t4 = ((u2m * u) % n) * dd % n;
87             u = t1 + t2;
88             if (u & 1) u += n;
89             u = (u >> 1) % n;
90             v = t3 + t4;
91             if (v & 1) v += n;
92             v = (v >> 1) % n;
93             qkd = (qkd * qm) % n;
94         }
95     }
96     if (u==0 || v==0) return true;
97     long long qkd2 = qkd*2;
98     for (int r=1; r<s; ++r) {
99         v = (v * v) % n - qkd2;
100         if (v < 0) v += n;
101         if (v < 0) v += n;

```

```

102     if (v >= n) v -= n;
103     if (v >= n) v -= n;
104     if (v == 0) return true;
105     if (r < s-1) {
106         qkd = (qkd * 1ll * qkd) % n;
107         qkd2 = qkd * 2;
108     }
109 }
110 return false;
111 }
112
113 bool prime (int n) {
114     // Call for prime check
115     for (int i=0; i<trivial_limit && p[i]<n; ++i)
116         if (n % p[i] == 0)
117             return false;
118     if (p[trivial_limit-1]*p[trivial_limit-1] >= n)
119         return true;
120     if (!miller_rabin (n))
121         return false;
122     return bpsw (n);
123 }
124
125 void prime_init() {
126     // Call before prime check
127     for (int i=2, j=0; j<trivial_limit; ++i) {
128         bool pr = true;
129         for (int k=2; k*k<=i; ++k)
130             if (i % k == 0)
131                 pr = false;
132         if (pr)
133             p[j++] = i;
134     }
135 }

```

#### 1.12. Bridge search.

```

1  const int MAXN = ...;
2  vector<int> g[MAXN];
3  bool used[MAXN];
4  int timer, tin[MAXN], fup[MAXN];
5
6  void dfs (int v, int p = -1) {
7      used[v] = true;
8      tin[v] = fup[v] = timer++;
9      for (size_t i=0; i<g[v].size(); ++i) {
10         int to = g[v][i];
11         if (to == p) continue;

```

```

12         if (used[to])
13             fup[v] = min (fup[v], tin[to]);
14         else {
15             dfs (to, v);
16             fup[v] = min (fup[v], fup[to]);
17             if (fup[to] > tin[v])
18                 IS_BRIDGE(v,to);
19         }
20     }
21 }
22
23 void find_bridges() {
24     timer = 0;
25     for (int i=0; i<n; ++i)
26         used[i] = false;
27     for (int i=0; i<n; ++i)
28         if (!used[i])
29             dfs (i);
30 }

```

#### 1.13. Lca.

```

1  int n, l;
2  vector < vector<int> > g;
3  vector<int> tin, tout;
4  int timer;
5  vector < vector<int> > up;
6
7  void dfs (int v, int p = 0) {
8      tin[v] = ++timer;
9      up[v][0] = p;
10     for (int i=1; i<=l; ++i)
11         up[v][i] = up[up[v][i-1]][i-1];
12     for (size_t i=0; i<g[v].size(); ++i) {
13         int to = g[v][i];
14         if (to != p)
15             dfs (to, v);
16     }
17     tout[v] = ++timer;
18 }
19
20 bool upper (int a, int b) {
21     return tin[a] <= tin[b] && tout[a] >= tout[b];
22 }
23
24 int lca (int a, int b) {
25     if (upper (a, b)) return a;
26     if (upper (b, a)) return b;

```

```

27     for (int i=l; i>=0; --i)
28         if (!upper (up[a][i], b))
29             a = up[a][i];
30     return up[a][0];
31 }
32
33 int main() {
34     // read
35     tin.resize (n), tout.resize (n), up.resize (n);
36     l = 1;
37     while ((l<<l) <= n) ++l;
38     for (int i=0; i<n; ++i) up[i].resize (l+1);
39     dfs (0);
40
41     for (;;) {
42         int a, b; // query
43         int res = lca (a, b); // answer
44     }
45
46 }

```

## 1.14. 2-SAT.

```

1  int n;
2  vector < vector<int> > g, gt;
3  vector<bool> used;
4  vector<int> order, comp;
5
6  void dfs1 (int v) {
7      used[v] = true;
8      for (size_t i=0; i<g[v].size(); ++i) {
9          int to = g[v][i];
10         if (!used[to])
11             dfs1 (to);
12     }
13     order.push_back (v);
14 }
15
16 void dfs2 (int v, int cl) {
17     comp[v] = cl;
18     for (size_t i=0; i<gt[v].size(); ++i) {
19         int to = gt[v][i];
20         if (comp[to] == -1)
21             dfs2 (to, cl);
22     }
23 }
24
25 int main() {

```

```

26     // read
27     used.assign (n, false);
28     for (int i=0; i<n; ++i)
29         if (!used[i])
30             dfs1 (i);
31
32     comp.assign (n, -1);
33     for (int i=0, j=0; i<n; ++i) {
34         int v = order[n-i-1];
35         if (comp[v] == -1)
36             dfs2 (v, j++);
37     }
38
39     for (int i=0; i<n; ++i)
40         if (comp[i] == comp[i^1]) {
41             puts ("NO SOLUTION");
42             return 0;
43         }
44     for (int i=0; i<n; ++i) {
45         int ans = comp[i] > comp[i^1] ? i : i^1;
46         printf ("%d ", ans);
47     }
48
49 }

```

## 1.15. Centroid.

```

1  struct decomposer_t
2  {
3      void process(const graph_t &g, int root)
4      {
5          // process tree `g` with root `root`
6
7      }
8
9      vector<int> cnt;
10
11     int dfs_cnt(const graph_t &g, int u, int pred = -1)
12     {
13         cnt[u] = 0;
14         for (size_t i = 0; i < g[u].size(); ++i)
15         {
16             int v = g[u][i].v;
17             if (v == pred)
18                 continue;
19             int cur = dfs_cnt(g, v, u);
20             cnt[u] += cur;
21         }

```

```
22     cnt[u]++;
23     return cnt[u];
24 }
25
26 int dfs_root(const graph_t &g, int u, int n, int pred = -1)
27 {
28     int mx = n - cnt[u];
29     for(size_t i = 0; i < g[u].size(); ++i)
30     {
31         int v = g[u][i].v;
32         if(v == pred)
33             continue;
34         mx = max(mx, cnt[v]);
35     }
36     if(mx <= n / 2)
37         return u;
38
39     for(size_t i = 0; i < g[u].size(); ++i)
40     {
41         int v = g[u][i].v;
42         if(v == pred)
43             continue;
44         int cur = dfs_root(g, v, n, u);
45         if(cur != -1)
46             return cur;
47     }
48     return -1;
49 }
50
51 int find_root(const graph_t &g, int root)
52 {
53     int n = dfs_cnt(g, root);
54     int u = dfs_root(g, root, n);
55     return u;
56 }
57
58 void delete_root(graph_t &g, int root)
59 {
60     for(size_t i = 0; i < g[root].size(); ++i)
61     {
62         int v = g[root][i].v;
63         for(size_t j = 0; j < g[v].size(); ++j)
64             if(g[v][j].v == root)
65             {
66                 swap(g[v][j], g[v].back());
67                 g[v].pop_back();
68                 break;
69             }
70     }
71 }
```

```
69     }
70 }
71 }
72
73 void process_tree(graph_t &g)
74 {
75     cnt.assign(g.size(), 0);
76     process_tree(g, 0);
77 }
78
79 void process_tree(graph_t &g, int root)
80 {
81     if(root == -1)
82         return;
83     root = find_root(g, root);
84     process(g, root);
85     delete_root(g, root);
86     for(size_t i = 0; i < g[root].size(); ++i)
87         process_tree(g, g[root][i].v);
88 }
89 };
```

2. MISC

2.1. Debugging Tips.

- Stack overflow? Recursive DFS on tree that is actually a long path?
- Floating-point numbers
  - Getting NaN? Make sure `acos` etc. are not getting values out of their range (perhaps `1+eps`).
  - Rounding negative numbers?
  - Outputting in scientific notation?
- Wrong Answer?
  - Read the problem statement again!
  - Are multiple test cases being handled correctly? Try repeating the same test case many times.
  - Integer overflow?
  - Think very carefully about boundaries of all input parameters
  - Try out possible edge cases:
    - \*  $n = 0, n = -1, n = 1, n = 2^{31} - 1$  or  $n = -2^{31}$
    - \* List is empty, or contains a single element
    - \*  $n$  is even,  $n$  is odd
    - \* Graph is empty, or contains a single vertex
    - \* Graph is a multigraph (loops or multiple edges)
    - \* Polygon is concave or non-simple
  - Is initial condition wrong for small cases?
  - Are you sure the algorithm is correct?
  - Explain your solution to someone.
  - Are you using any functions that you don't completely understand? Maybe STL functions?
  - Maybe you (or someone else) should rewrite the solution?
  - Can the input line be empty?
- Run-Time Error?
  - Is it actually Memory Limit Exceeded?

2.2. Solution Ideas.

- Dynamic Programming
  - Parsing CFGs: CYK Algorithm
  - Drop a parameter, recover from others
  - Swap answer and a parameter
  - When grouping: try splitting in two
  - $2^k$  trick
  - When optimizing
    - \* Convex hull optimization
      - $dp[i] = \min_{j < i} \{dp[j] + b[j] \times a[i]\}$
      - $b[j] \geq b[j + 1]$
      - optionally  $a[i] \leq a[i + 1]$
      - $O(n^2)$  to  $O(n)$
    - \* Divide and conquer optimization
      - $dp[i][j] = \min_{k < j} \{dp[i - 1][k] + C[k][j]\}$

- $A[i][j] \leq A[i][j + 1]$
- $O(kn^2)$  to  $O(kn \log n)$
- sufficient:  $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$ ,  $a \leq b \leq c \leq d$  (QI)
- \* Knuth optimization
  - $dp[i][j] = \min_{i < k < j} \{dp[i][k] + dp[k][j] + C[i][j]\}$
  - $A[i][j - 1] \leq A[i][j] \leq A[i + 1][j]$
  - $O(n^3)$  to  $O(n^2)$
  - sufficient: QI and  $C[b][c] \leq C[a][d]$ ,  $a \leq b \leq c \leq d$
- Greedy
- Randomized
- Optimizations
  - Use bitset (/64)
  - Switch order of loops (cache locality)
- Process queries offline
  - Mo's algorithm
- Square-root decomposition
- Precomputation
- Efficient simulation
  - Mo's algorithm
  - Sqrt decomposition
  - Store  $2^k$  jump pointers
- Data structure techniques
  - Sqrt buckets
  - Store  $2^k$  jump pointers
  - $2^k$  merging trick
- Counting
  - Inclusion-exclusion principle
  - Generating functions
- Graphs
  - Can we model the problem as a graph?
  - Can we use any properties of the graph?
  - Strongly connected components
  - Cycles (or odd cycles)
  - Bipartite (no odd cycles)
    - \* Bipartite matching
    - \* Hall's marriage theorem
    - \* Stable Marriage
  - Cut vertex/bridge
  - Biconnected components
  - Degrees of vertices (odd/even)
  - Trees
    - \* Heavy-light decomposition
    - \* Centroid decomposition
    - \* Least common ancestor
    - \* Centers of the tree
  - Eulerian path/circuit
  - Chinese postman problem

- Topological sort
  - (Min-Cost) Max Flow
  - Min Cut
    - \* Maximum Density Subgraph
  - Huffman Coding
  - Min-Cost Arborescence
  - Steiner Tree
  - Kirchhoff’s matrix tree theorem
  - Prüfer sequences
  - Lovász Toggle
  - Look at the DFS tree (which has no cross-edges)
  - Is the graph a DFA or NFA?
    - \* Is it the Synchronizing word problem?
- Mathematics
  - Is the function multiplicative?
  - Look for a pattern
  - Permutations
    - \* Consider the cycles of the permutation
  - Functions
    - \* Sum of piecewise-linear functions is a piecewise-linear function
    - \* Sum of convex (concave) functions is convex (concave)
  - Modular arithmetic
    - \* Chinese Remainder Theorem
    - \* Linear Congruence
  - Sieve
  - System of linear equations
  - Values too big to represent?
    - \* Compute using the logarithm
    - \* Divide everything by some large value
  - Linear programming
    - \* Is the dual problem easier to solve?
  - Can the problem be modeled as a different combinatorial problem? Does that simplify calculations?
- Logic
  - 2-SAT
  - XOR-SAT (Gauss elimination or Bipartite matching)
- Meet in the middle
- Only work with the smaller half ( $\log(n)$ )
- Strings
  - Trie (maybe over something weird, like bits)
  - Suffix array
  - Suffix automaton (+DP?)
  - Aho-Corasick
  - eerTree
  - Work with  $S + S$
- Hashing
- Euler tour, tree to array
- Segment trees
  - Lazy propagation
  - Persistent
  - Implicit
  - Segment tree of X
- Geometry
  - Minkowski sum (of convex sets)
  - Rotating calipers
  - Sweep line (horizontally or vertically?)
  - Sweep angle
  - Convex hull
- Fix a parameter (possibly the answer).
- Are there few distinct values?
- Binary search
- Sliding Window (+ Monotonic Queue)
- Computing a Convolution? Fast Fourier Transform
- Computing a 2D Convolution? FFT on each row, and then on each column
- Exact Cover (+ Algorithm X)
- Cycle-Finding
- What is the smallest set of values that identify the solution? The cycle structure of the permutation? The powers of primes in the factorization?
- Look at the complement problem
  - Minimize something instead of maximizing
- Immediately enforce necessary conditions. (All values greater than 0? Initialize them all to 1)
- Add large constant to negative numbers to make them positive
- Counting/Bucket sort

Catalan	$C_0 = 1, C_n = \frac{1}{n+1} \binom{2n}{n} = \sum_{i=0}^{n-1} C_i C_{n-i-1} = \frac{4n-2}{n+1} C_{n-1}$	
Stirling 1st kind	$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1, \begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ n \end{bmatrix} = 0, \begin{bmatrix} n \\ k \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$	#perms of $n$ objs with exactly $k$ cycles
Stirling 2nd kind	$\left\{ \begin{smallmatrix} n \\ 1 \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} n \\ n \end{smallmatrix} \right\} = 1, \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = k \left\{ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\} + \left\{ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\}$	#ways to partition $n$ objs into $k$ nonempty sets
Euler	$\left\langle \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right\rangle = \left\langle \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \right\rangle = 1, \left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle = (k+1) \left\langle \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\rangle + (n-k) \left\langle \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\rangle$	#perms of $n$ objs with exactly $k$ ascents
Euler 2nd Order	$\left\langle\!\!\left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle\!\!\right\rangle = (k+1) \left\langle\!\!\left\langle \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\rangle\!\!\right\rangle + (2n-k-1) \left\langle\!\!\left\langle \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\rangle\!\!\right\rangle$	#perms of $1, 1, 2, 2, \dots, n, n$ with exactly $k$ ascents
Bell	$B_1 = 1, B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} = \sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$	#partitions of $1..n$ (Stirling 2nd, no limit on k)

#labeled rooted trees	$n^{n-1}$
#labeled unrooted trees	$n^{n-2}$
#forests of $k$ rooted trees	$\frac{k}{n} \binom{n}{k} n^{n-k}$
$\sum_{i=1}^n i^2 = n(n+1)(2n+1)/6$	$\sum_{i=1}^n i^3 = n^2(n+1)^2/4$
$!n = n \times!(n-1) + (-1)^n$	$!n = (n-1)(!(n-1) + !(n-2))$
$\sum_{i=1}^n \binom{n}{i} F_i = F_{2n}$	$\sum_i \binom{n-i}{i} = F_{n+1}$
$\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}$	$x^k = \sum_{i=0}^k i! \left\{ \begin{smallmatrix} k \\ i \end{smallmatrix} \right\} \binom{x}{i} = \sum_{i=0}^k \left\langle \begin{smallmatrix} k \\ i \end{smallmatrix} \right\rangle \binom{x+i}{k}$
$a \equiv b \pmod{x, y} \Rightarrow a \equiv b \pmod{\text{lcm}(x, y)}$	$\sum_{d n} \phi(d) = n$
$ac \equiv bc \pmod{m} \Rightarrow a \equiv b \pmod{\frac{m}{\text{gcd}(c, m)}}$	$(\sum_{d n} \sigma_0(d))^2 = \sum_{d n} \sigma_0(d)^3$
$p$ prime $\Leftrightarrow (p-1)! \equiv -1 \pmod{p}$	$\text{gcd}(n^a - 1, n^b - 1) = n^{\text{gcd}(a, b)} - 1$
$\sigma_x(n) = \prod_{i=0}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$	$\sigma_0(n) = \prod_{i=0}^r (a_i + 1)$
$\sum_{k=0}^m (-1)^k \binom{n}{k} = (-1)^m \binom{n-1}{m}$	$\sum_{i=1}^n 2^{\omega(i)} = O(n \log n)$
$2^{\omega(n)} = O(\sqrt{n})$	$v_f^2 = v_i^2 + 2ad$
$d = v_i t + \frac{1}{2} a t^2$	$d = \frac{v_i + v_f}{2} t$
$v_f = v_i + at$	

2.3. The Twelfold Way. Putting  $n$  balls into  $k$  boxes.

Balls	same	distinct	same	distinct	
Boxes	same	same	distinct	distinct	Remarks
-	$p_k(n)$	$\sum_{i=0}^k \left\{ \begin{smallmatrix} n \\ i \end{smallmatrix} \right\}$	$\binom{n+k-1}{k-1}$	$k^n$	$p_k(n)$ : #partitions of $n$ into $\leq k$ positive parts
size $\geq 1$	$p(n, k)$	$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$	$\binom{n-1}{k-1}$	$k! \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$	$p(n, k)$ : #partitions of $n$ into $k$ positive parts
size $\leq 1$	$[n \leq k]$	$[n \leq k]$	$\binom{k}{n}$	$n! \binom{k}{n}$	$[cond]$ : 1 if $cond = true$ , else 0



PRACTICE CONTEST CHECKLIST

---

- How many operations per second? Compare to local machine.
- What is the stack size?
- How to use printf/scanf with long long/long double?
- Are `__int128` and `__float128` available?
- Does MLE give RTE or MLE as a verdict? What about stack overflow?
- What is `RAND_MAX`?
- How does the judge handle extra spaces (or missing newlines) in the output?
- Look at documentation for programming languages.
- Try different programming languages: C++, Java and Python.
- Try the submit script.
- Try local programs: `i?python[23]`, `factor`.
- Try submitting with `assert(false)` and `assert(true)`.
- Return-value from `main`.
- Look for directory with sample test cases.
- Make sure printing works.
- Remove this page from the notebook.