

# Competitive Programming Notebook

Raul Almeida

## Contents

<b>1</b>	<b>Graph</b>	<b>1</b>
1.1	Prim MST	1
1.2	Dijkstra SSSP	1
1.3	Graph Check	2
1.4	Articulations and Bridges	2
1.5	Euler Tour	2
1.6	Kahn's topological sort	2
1.7	Max Cardinality Bipartite Matching	2
1.8	Lowest Common Ancestor	2
1.9	Tarjan Strongly Connected Component	3
1.10	Bellman-Ford SSSP	3
1.11	Kruskal MST	3
1.12	Edmond Karp MaxFlow	3
1.13	Floyd Warshall APSP	4
<b>2</b>	<b>Math</b>	<b>4</b>
2.1	Sieve of Eratosthenes	4
2.2	Prime Factors w/ Optimized Trial Divisions	4
2.3	Extended Euclid for solving Linear Diophantine Equations	4
2.4	Floyd's algorithm cycle-finding	5
<b>3</b>	<b>Paradigm</b>	<b>5</b>
3.1	Coordinate Compression	5
3.2	128 Bit Integers	5
3.3	Binary Search (but beautiful)	5
<b>4</b>	<b>String</b>	<b>5</b>
4.1	Prefix Function (KMP)	5
<b>5</b>	<b>Structure</b>	<b>5</b>
5.1	Merge/Disjoint Union-Find	5
5.2	Bottom-Up Segment Tree	6
5.3	Segment Tree	6
<b>6</b>	<b>Extra</b>	<b>6</b>
6.1	Bashrc	6
6.2	Vim	7
6.3	Generator	7
6.4	C++ Template	7
6.5	Stress	7

## 1 Graph

### 1.1 Prim MST

```
1 // Status: tested (UVA10048)
2
3 vector<vector<pair<int, int>> adj(M), mst(M);
4 vector<bool> taken(M, false);
5 int cost = 0;
6 using iii = pair<int, pair<int, int>>;
7 priority_queue<iii, vector<iii>, greater<iii> pq;
8
9 void process(int v) {
10     taken[v] = true;
11     for (auto &[w, u]: adj[v])
12         if (!taken[u])
13             pq.push({w, {v, u}});
14 }
15
16 void run(int n) {
17     process(0);
18     while (!pq.empty()) {
19         int w = pq.top().first,
20             v = pq.top().second.first,
21             u = pq.top().second.second;
22         pq.pop();
23         if (!taken[u]) {
24             mst_cost += w;
25             mst[u].push_back({w, v});
26             mst[v].push_back({v, w});
27             process(u);
28         }
29     }
30     for (int v = 1; v < n; ++v)
31         if (!taken[v]) {
32             process(v);
33             run(n);
34         }
35 }
```

### 1.2 Dijkstra SSSP

```
1 // Status: tested (CF20C)
2
3 using ii = pair<int, int>;
4 const int inf = 0x3f3f3f3f;
5 vector<vector<ii>> adj(M);
6 vector<int> dist(M, inf), par(M, -1);
7
8 void dijkstra(int s) {
9     dist[s] = 0;
10    priority_queue<ii, vector<ii>,
11        greater<pair<int, int>>> pq;
12    pq.push(make_pair(0, s));
13    while (!pq.empty()) {
14        int w = pq.top().first;
15        int v = pq.top().second;
16        pq.pop();
17        if (w > dist[v]) continue;
18        for (auto &[d, u]: adj[v])
19            if (dist[v] != inf && dist[v]+d <
20                dist[u]) {
21                par[u] = v;
22            }
23    }
```

```

20         dist[u] = dist[v]+d;
21         pq.push(make_pair(dist[u], u));
22     }
23 }
24 }

```

## 1.3 Graph Check

```

1 // Usage: graphCheck(firstVertex, -1) (p stands
  for parent)
2
3 int UNVISITED = -1, EXPLORED = 0, VISITED = 1;
4 vector<vector<int>> adj(M);
5 vector<int> tin;
6
7 void graphCheck(int v, int p) { //vertex, parent
8     tin[v] = EXPLORED;
9     for (auto u: adj[v]) {
10         if (tin[u] == UNVISITED) { //tree edge
11             graphCheck(u, v);
12         } else if (tin[u] == EXPLORED) {
13             if (u == p)
14                 ; //two way edge u <-> v
15             else
16                 ; //back edge v -> u
17         } else if (tin[u] == VISITED) {
18             ; //forward/cross edge u-v
19         }
20     }
21     tin[v] = VISITED;
22 }

```

## 1.4 Articulations and Bridges

```

1 // Usage: dfs(source, -1)
2 // Status: not tested
3
4 int tk = 0;
5 vector<int> tin(M, -1);
6 vector<vector<int>> adj(M);
7
8 void dfs(int v, int p) {
9     tin[v] = low[v] = tk++;
10    int children = 0;
11    for (auto u: adj[v]) {
12        if (u == p) continue;
13        else if (tin[u] == -1) {
14            ++children;
15            dfs(u, v);
16            if (low[u] >= tin[v] && p != v)
17                ; //articulation point
18            if (low[u] > tin[v])
19                ; //bridge u-v
20            low[v] = min(low[v], low[u]);
21        } else {
22            low[v] = min(low[v], tin[u]);
23        }
24    }
25 }

```

## 1.5 Euler Tour

```

1 // Usage: tour(cyc.begin(), start\_vertex)
2 // Status: not tested
3 // Source: CP3 (pg. 205)
4
5 list<int> cyc;
6 vector<vector<int>> adj(M);
7 vector<vector<bool>> traversed(M, vector<bool>(M,
  false));
8
9 //euler tour (list for fast insertion)
10 void tour(list<int>::iterator i, int v) {
11     for (auto u: adj[v]) {
12         if (!traversed[v][u]) {
13             traversed[v][u] = true;

```

```

14         for (auto t: adj[u])
15             if (t == v && !traversed[u][t]) {
16                 traversed[u][t] = true;
17                 break;
18             }
19         tour(cyc.insert(i, v), u);
20     }
21 }
22 }

```

## 1.6 Kahn's topological sort

```

1 // Status: tested (UVA11060)
2
3 vector<vector<int>> adj(M);
4 vector<int> sorted;
5
6 void kahn(int n) {
7     vector<int> indeg(n, 0);
8     vector<bool> valid(n, true);
9     priority_queue<int> pq;
10
11     for (int v = 0; v < n; ++v)
12         for (auto u: adj[v])
13             indeg[u]++;
14     for (int v = 0; v < n; ++v)
15         if (!indeg[v]) pq.push(v);
16
17     while (!pq.empty()) {
18         int v = pq.top(); pq.pop();
19         sorted.push_back(v);
20         valid[v] = false;
21         for (auto u: adj[v])
22             if (valid[u] && (--indeg[u]) == 0)
23                 pq.push(u);
24     }
25 }

```

## 1.7 Max Cardinality Bipartite Matching

```

1 // Status: not tested
2 // Source: CP3 (pg. 209)
3
4 vector<vector<int>> adj(M);
5 vector<int> match(M, -1);
6 vector<bool> visited(M);
7
8 bool augment(int left) { //match one on the left
  with one on the right
9     if (visited[left]) return false;
10    visited[left] = true;
11    for (auto right: adj[left])
12        if (match[right] == -1 ||
13            augment(match[right])) {
14                match[right] = left;
15                return true;
16        }
17    return false;
18 }
19
20 //usage
21 //(mcbm = V iff there's at least one way to
  completely match both sides)
22 int mcbm = 0; //number of matched vertices
23 match.assign(M, -1);
24 for (int v = 0; v < ls; ++v) { //ls = size of the
  left set
25     visited.assign(ls, false);
26     mcbm += augment(v);
27 }

```

## 1.8 Lowest Common Ancestor

```

1 // Status: not tested
2
3 ///--- binary lifting
4 int n, l = ceil(log2(n));
5 vector<vector<int>> adj;
6 int tk = 0;
7 vector<int> tin(n), tout(n);
8 vector<vector<int>> up(n, vector<int>(l+1)); //
    ancestor
9
10 void dfs(int v, int p) { // run dfs(root, root) to
    initialize
11     tin[v] = ++tk;
12     up[v][0] = p;
13     for (int i = 1; i <= l; ++i)
14         up[v][i] = up[up[v][i-1]][i-1];
15     for (int u : adj[v])
16         if (u != p)
17             dfs(u, v);
18     tout[v] = ++tk;
19 }
20
21 bool ancestor(int v, int u) { // v is ancestor of u
22     return tin[v] <= tin[u] && tout[v] >= tout[u];
23 }
24
25 int lca(int v, int u) {
26     if (ancestor(v, u)) return v;
27     if (ancestor(u, v)) return u;
28     for (int i = l; i >= 0; --i)
29         if (!ancestor(up[v][i], u))
30             v = up[v][i];
31     return up[v][0];
32 }
33
34 ///--- euler path
35 using ii = pair<int, int>;
36 vector<ii> t;
37 vector<int> idx(n);
38 int tk = 1;
39
40 void dfs(int v, int d) { // call with dfs(root, 0);
41     for (auto u : adj[v]) {
42         st.update(tk, {d, v});
43         tk++;
44         dfs(u, d+1);
45     }
46     idx[v] = tk;
47     st.update(tk, {d, v});
48     tk++;
49 }
50
51 int lca(int v, int u) {
52     int l = idx[v], r = idx[u];
53     return st.minquery(l, r).second; // .first is
    depth
54 }

```

## 1.9 Tarjan Strongly Connected Component

```

1 // Usage: Tarjan(N, adj)
2 // Status: tested (UVA247, UVA11838)
3
4 vector<int> tin(M, -1), low(M, -1);
5 vector<vector<int>> adj(M);
6 stack<int> S;
7 int tk = 0;
8
9 void dfs(int v) {
10     low[v] = tin[v] = tk++;
11     S.push(v);
12     visited[v] = true;
13     for (auto u: adj[v]) {
14         if (tin[u] == -1)
15             dfs(u);
16         if (visited[u])

```

```

17         low[v] = min(low[v], low[u]);
18     }
19     if (low[v] == tin[v])
20         while (true) {
21             int u = S.top(); S.pop(); visited[u] =
                false;
22             if (u == v) break;
23         }
24 }

```

## 1.10 Bellman-Ford SSSP

```

1 // Status: tested (UVA1112, UVA10449)
2 const int inf = 0x3f3f3f3f;
3 vector<vector<pair<int, int>>> adj(M);
4 vector<int> dist(M, inf);
5
6 void bellmanFord(int n) {
7     for (int i = 0; i < n-1; ++i)
8         for (int v = 0; v < n; ++v)
9             for (auto &[u, w]: adj[v])
10                 if (dist[v] != inf)
11                     dist[u] = min(dist[u],
                        dist[v]+w);
12 }
13
14 //check if there are negative cycles
15 bool cycle(int n) {
16     bool ans = false;
17     for (int v = 0; v < n; ++v)
18         for (auto &[u, w]: adj[v])
19             ans |= dist[v] != inf && dist[u] >
                dist[v]+w;
20 }

```

## 1.11 Kruskal MST

```

1 // Usage: Kruskal(V, E, edges) (weighted edges)
2 // Status: tested (UVA1174)
3
4 using iii = pair<int, pair<int, int>>; //weight,
    two vertices
5 vector<iii> edges;
6 UnionFind muf;
7
8 int kruskal() {
9     int cost = 0;
10     sort(edges.begin(), edges.end());
11     for (auto a: edges) {
12         int w = a.first;
13         pair<int, int> e = a.second;
14         if (!muf.isSameSet(e.first, e.second)) {
15             cost += w;
16             muf.unionSet(e.first, e.second);
17         }
18     }
19     return cost;
20 }

```

## 1.12 Edmond Karp MaxFlow

```

1 // Status: tested (CSES1694, CSES1695)
2
3 vector<vector<int>> capacity(M, vector<int>(M,
    0)), adj(M);
4 vector<pair<int, int>> mc; //mincut edges
5
6 int bfs(int s, int t, vi &par) {
7     fill(all(par), -1);
8     par[s] = -2;
9     queue<pair<int, int>> q; q.push({s, inf});
10     while (!q.empty()) {
11         int v = q.front().first,
12             flow = q.front().second;
13         q.pop();
14         for (auto u: adj[v])

```

```

15         if (par[u] == -1 && capacity[v][u]) {
16             par[u] = v;
17             int new_flow = min(flow,
18                 capacity[v][u]);
19             if (u == t) return new_flow;
20             q.push({u, new_flow});
21         }
22     }
23     return 0;
24 }
25 int maxflow(int s, int t) {
26     int flow = 0;
27     vi par(M);
28     int new_flow;
29     while ((new_flow = bfs(s, t, par))) {
30         flow += new_flow;
31         int v = t;
32         while (v != s) {
33             int p = par[v];
34             capacity[p][v] -= new_flow;
35             capacity[v][p] += new_flow;
36             v = p;
37         }
38     }
39     return flow;
40 }
41
42 void mincut(int s, int t) {
43     maxflow(s, t);
44     stack<int> st;
45     vector<bool> visited(n, false);
46     vector<pair<int, int>> ans;
47     st.push(0);
48     while (!st.empty()) {
49         int v = st.top(); st.pop();
50         if (visited[v]) continue;
51         visited[v] = true;
52         for (auto u: adj[v])
53             if (capacity[v][u] > 0)
54                 st.push(u);
55         else
56             ans.push_back({v, u});
57     }
58     mc.clear();
59     for (auto &[v, u] : ans)
60         if (!visited[u])
61             mc.push_back({v, u});
62 }

```

## 1.13 Floyd Warshall APSP

```

1 // Usage: FloydWarshall(n, edges)
2 // Status: tested (UVA821, UVA1056)
3
4 struct edge { int v, u, w; };
5 const int inf = 0x3f3f3f3f;
6 vector<vector<int>> weight(M, vector<int>(M, inf));
7 vector<edge> edges;
8
9 void floydWarshall(int n) {
10     for (auto e: edges)
11         weight[e.v][e.u] = e.w;
12     for (int k = 0; k < n; ++k)
13         for (int i = 0; i < n; ++i)
14             for (int j = 0; j < n; ++j)
15                 if (max(weight[i][k],
16                     weight[k][j]) < inf)
17                     weight[i][j] =
18                         min(weight[i][j],
19                             weight[i][k]+weight[k][j]);
19 }

```

## 2 Math

### 2.1 Sieve of Eratosthenes

```

1 // Status: not tested
2
3 bitset<11234567> pr;
4 vector<int> factors(M, 0);
5 vector<int> primes;
6
7 void sieve(int n) {
8     pr.set();
9     for (int i = 2; i*i <= n; ++i)
10         if (pr[i]) { //factors[i] == 0
11             primes.push_back(i);
12             for (int p = i*i; p <= n; p += i) {
13                 pr[p] = false;
14                 factors[p]++;
15             }
16         }
17 }
18
19 // O(1) for small n, O(sieve_size) else
20 bool isPrime(int n) {
21     int sieve_size = 11234567;
22     if (n <= sieve_size) return pr[n];
23     for (auto p: primes) // only works if n <=
24         if (!(n%p)) return false;
25     return true;
26 }

```

### 2.2 Prime Factors w/ Optimized Trial Divisions

```

1 // Status: not tested
2 // Source: CP3 (pg. 238)
3
4 vector<int> primes;
5 vector<pair<int, int>> factors;
6
7 void pf(int n) {
8     for (auto p: primes) {
9         if (p*p > n) break;
10         int i = 0;
11         while (!(n%p)) {
12             n /= p;
13             i++;
14         }
15         factors.push_back({p, i});
16     }
17     if (n != 1) factors.push_back({n, 1});
18 }

```

### 2.3 Extended Euclid for solving Linear Diophantine Equations

```

1 // Status: not tested
2 // Source: CP3 (pg. 242)
3
4 int x, y, d;
5 void extendedEuclid(int a, int b) {
6     if (b == 0) { x = 1; y = 0; d = a; return; }
7     extendedEuclid(b, a%b);
8     int x1 = y;
9     int y1 = x - (a/b)*y;
10     x = x1;
11     y = y1;
12 }
13
14 void solve(int a, int b, int c, int i) { //i
15     solutions
16     extendedEuclid(a, b);

```

```

16     if (d%c) return;
17     x *= c/d;
18     y *= c/d;
19     do {
20         cout << x << ", " << y << '\n';
21         x += b/d;
22         y -= a/d;
23     } while (--i);
24 }

```

## 2.4 Floyd's algorithm cycle-finding

```

1 // Status: not tested
2 // Source: CPHB (p. 156)
3
4 int findCycle(int x) {
5     int a, b;
6     a = succ(x);
7     b = succ(succ(x));
8     while (a != b) {
9         a = succ(a);
10        b = succ(succ(b));
11    }
12    a = x;
13    while (a != b) {
14        a = succ(a);
15        b = succ(b);
16    }
17    int first = a; // first element in cycle
18    b = succ(a);
19    int length = 1;
20    while (a != b) {
21        b = succ(b);
22        length++;
23    }
24 }

```

## 3 Paradigm

### 3.1 Coordinate Compression

```

1 // Status: not tested
2 // Source: GEMA ICMC (YouTube)
3
4 vector<int> v, vals, cv; // all of the same size,
    cv = compressed v
5 vals = v;
6 sort(vals.begin(), vals.end());
7 vals.erase(unique(vals.begin(), vals.end()),
    vals.end());
8 for (int i = 0; i < n; ++i) {
9     int idx = lower_bound(vals.begin(),
        vals.end(), v[i]) - vals.begin();
10    cv[i] = idx;
11 }

```

### 3.2 128 Bit Integers

```

1 // Status: not tested
2 // Source: GEMA (YouTube)
3
4 // cout, cerr, etc; podedar over/underflow
5 ostream& operator<<(ostream& out, __int128 x) {
6     if (x == 0) return out << 0;
7     string s; bool sig = x < 0; x = x < 0 ? -x : x;
8     while(x > 0) s += x % 10 + '0', x /= 10;
9     if (sig) s += '-';
10    reverse(s.begin(), s.end());
11    return out << s;
12 }
13
14 // cin, etc; podedar over/underflow
15 istream& operator>>(istream& in, __int128& x) {
16     char c, neg = 0; while(isspace(c = in.get()));

```

```

17     if(!isdigit(c)) neg = (c == '-'), x = 0;
18     else x = c - '0';
19     while(isdigit(c = in.get())) x = (x << 3) + (x
        << 1) - '0' + c;
20     x = neg ? -x : x; return in;
21 }

```

## 3.3 Binary Search (but beautiful)

```

1 // Status: not tested
2 // Source: CPHB
3
4 // std
5 int l = 0, r = n-1;
6 while (l <= r) {
7     int m = l+(r-l)/2;
8     if (array[m] == x)
9         // found
10        if (array[m] > x) r = m-1;
11        else l = m+1;
12 }
13
14 // nice - binary steps
15 int k = 0;
16 for (int b = n/2; b > 0; b /= 2)
17     while (k+b < n && array[k+b] <= x)
18         k += b;
19 if (array[k] == x)
20     // found

```

## 4 String

### 4.1 Prefix Function (KMP)

```

1 // Status: not tested
2 // Source: CP-Algorithms
3
4 vector<int> prefix(string s) {
5     int n = s.length();
6     vector<int> pi(n, 0); // can be optimized if
        you know max prefix length
7     for (int i = 1; i < n; ++i) {
8         int j = pi[i-1];
9         while (j > 0 && s[i] != s[j])
10            j = pi[j-1];
11        if (s[i] == s[j])
12            j++;
13        pi[i] = j;
14    }
15    return pi;
16 }

```

## 5 Structure

### 5.1 Merge/Disjoint Union-Find

```

1 // Usage: UnionFind(N);
2 // Status: tested (UVA11503)
3
4 struct UnionFind {
5     int N;
6     vi par, rk, count;
7
8     UnionFind(int N) : N(N), par(N), rk(N, 0),
        count(N, 1) {
9         rep(i, 0, N) par[i] = i;
10    }
11
12    int findSet(int i) {
13        return par[i] == i ? i : (par[i] =
            findSet(par[i]));
14    }

```

```

15
16 int unionSet(int a, int b) {
17     int x = findSet(a), y = findSet(b);
18     if (x != y)
19         count[x] = count[y] =
20             (count[x]+count[y]);
21     if (rk[x] < rk[y])
22         par[x] = y;
23     else {
24         par[y] = x;
25         if (rk[x] == rk[y])
26             rk[x]++;
27     }
28     return count[x];
29 }
30
31 bool isSameSet(int i, int j) {
32     return findSet(i) == findSet(j);
33 }
34 };

```

## 5.2 Bottom-Up Segment Tree

```

1 // Usage: SegTree(N);
2 // Source: CP Handbook
3 // Status: not tested
4
5 struct SegTree {
6     unsigned int n;
7     vector<int> tree;
8
9     SegTree(vector<int> v) : n(v.size()), tree(2*n) {
10         for (int i = 0; i < n; ++i)
11             modify(i, v[i]);
12     }
13
14     int query(int a, int b) {
15         a += n, b += n;
16         int ans = 0;
17         while (a <= b) {
18             if (a%2 == 1) ans += tree[a++];
19             if (b%2 == 0) ans += tree[b--];
20             a >>= 1; b >>= 1;
21         }
22         return ans;
23     }
24
25     void modify(int k, int x) {
26         k += n;
27         tree[k] += x;
28         for (k /= 2; k >= 1; k /= 2)
29             tree[k] = tree[k<<1] + tree[(k<<1) + 1];
30     }
31 };

```

## 5.3 Segment Tree

```

1 // Usage: SegTree(N)
2
3 struct SegTree {
4     int N;
5     vi st, A;
6
7     SegTree(int N): N(N), st(4*n), A(n) {
8         init();
9     }
10
11     void init() { build(1, 0, n-1); }
12
13     int left(int i) { return i*2; }
14     int right(int i) { return i*2+1; }
15
16     //O(N)
17     void build(int v, int tl, int tr) {
18         if (tl == tr) st[v] = a[tl];
19         else {
20             int tm = (tl+tr)/2;

```

```

21             build(left(v), tl, tm);
22             build(right(v), tm+1, tr);
23             st[v] = max(st[left(v)], st[right(v)]);
24         }
25     }
26
27     //O(log n)
28     int maxquery(int v, int tl, int tr, int l, int
29         r) {
30         if (l > r) return -1;
31         if (l == tl && r == tr) return st[v];
32         int tm = (tl+tr)/2;
33         int q1 = maxquery(left(v), tl, tm, l,
34             min(r, tm));
35         int q2 = maxquery(right(v), tm+1, tr,
36             max(l, tm+1), r);
37         return max(q1, q2);
38     }
39
40     int maxquery(int l, int r) {
41         return maxquery(1, 0, n-1, l-1, r-1);
42     }
43
44     //O(log n)
45     void update(int v, int tl, int tr, int p, int
46         new_val) {
47         if (tl == tr) st[v] = new_val;
48         else {
49             int tm = (tl+tr)/2;
50             if (p <= tm)
51                 update(left(v), tl, tm, p,
52                     new_val);
53             else
54                 update(right(v), tm+1, tr, p,
55                     new_val);
56             st[v] = max(st[left(v)], st[right(v)]);
57         }
58     }
59
60     void update(int p, int new_val) {
61         update(1, 0, n-1, p-1, new_val);
62     }
63 };

```

## 6 Extra

### 6.1 Bashrc

```

1 xmodmap -e 'clear lock' -e 'keycode 66=Escape' #
2 caps -> esc
3
4 alias e=vim
5
6 BASE_CP="/home/raul/cp2022"
7
8 alias c='g++ -Wall -Wconversion -Wfatal-errors -g
9     -O2 -std=gnu++17 -fsanitize=undefined,address'
10 alias c14='g++ -Wall -Wconversion -Wfatal-errors
11     -g -O2 -std=gnu++14
12     -fsanitize=undefined,address'
13 alias p3='pypy3 -m py_compile'
14
15 tp () {
16     [ -f "$1.cpp" ] && echo "$1.cpp already
17     exists";
18     [ ! -f "$1.cpp" ] && tail -n +2
19     $BASE_CP/code/extra/template.cpp > $1.cpp
20     && vim $1.cpp;
21 }
22
23 clip () {
24     if [ -f "$1" ];
25     then
26         cat $1 | clip.exe;
27     else
28         echo "$1 not found"
29     fi

```

```
22 }
```

## 6.2 Vim

```
1 set et ts=2 sw=2 ai si cindent sta
2 set is tm=50 nu noeb sm "cul
3 sy on
```

## 6.3 Generator

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(int argc, char *argv[]) {
5     cin.tie(0); ios_base::sync_with_stdio(0);
6     if (argc < 2) {
7         cout << "usage: " << argv[0] << " <seed>\n";
8         exit(1);
9     }
10    srand(atoi(argv[1]));
11    // use rand() for random value
12 }
```

## 6.4 C++ Template

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 int main() {
6     ios_base::sync_with_stdio(0);
7     cin.tie(0);
8 }
```

## 6.5 Stress

```
1 for (( I=0; I < 5; I++ )); do
2     ./gen $I > a.in
3     ./brute < a.in > expected.txt
4     ./a.out < a.in > output.txt
5     if diff -u expected.txt output.txt; then : ; else
6         echo "--> input:"; cat a.in
7         echo "--> expected output:"; cat expected.txt
8         echo "--> received output:"; cat output.txt
9         break
10    fi
11    echo -n .
12 done
```