Competitive Programming Notebook

Raul Almeida

Contents

1 Graph

1.1 Prim MST

```cpp
// Status: tested (UVA10048)
// O(E log V) time, O(V+E) space

vector<vector<pair<int, int>> adj(M), mst(M);
vector<bool> taken(M, false);
int cost = 0;
using iii = pair<int, pair<int, int>>;
priority_queue<iii, vector<iii>, greater<iii> pq;

void process(int v) {
    taken[v] = true;
    for (auto &[w, u]: adj[v])
        if (!taken[u])
            pq.push({w, {v, u}});
}

void run(int n) {
    process(0);
    while (!pq.empty()) {
        int w = pq.top().first,
            v = pq.top().second.first,
            u = pq.top().second.second;
        pq.pop();
        if (!taken[u]) {
            mst_cost += w;
            mst[u].push_back({w, v});
            mst[v].push_back({v, w});
            process(u);
        }
    }
    for (int v = 1; v < n; ++v)
        if (!taken[v]) {
            process(v);
            run(n);
        }
}
```

1.2 Dijkstra SSSP

```cpp
// Status: tested (CF20C)
// O((V+E) log V) time, O(V^2) space

using ii = pair<int, int>;
const int inf = 0x3f3f3f3f;
vector<vector<ii>> adj(M);
vector<int> dist(M, inf), par(M, -1);

void dijkstra(int s) {
    dist[s] = 0;
    priority_queue<ii, vector<ii>,
            greater<pair<int, int>>> pq;
    pq.push(make_pair(0, s));
    while (!pq.empty()) {
        int w = pq.top().first;
        int v = pq.top().second;
        pq.pop();
        if (w > dist[v]) continue;
        for (auto &[d, u]: adj[v])
```

```
19        if (dist[v] != inf && dist[v]+d <
              dist[u]) {
20            par[u] = v;
21            dist[u] = dist[v]+d;
22            pq.push(make_pair(dist[u], u));
23        }
24    }
25 }
```

1.3   Graph Check

```
1 // Usage: graphCheck(firstVertex, -1) (p stands
     for parent)
2 // O(V+E) time & space
3
4 int UNVISITED = -1, EXPLORED = 0, VISITED = 1;
5 vector<vector<int>> adj(M);
6 vector<int> tin;
7
8 void graphCheck(int v, int p) { //vertex, parent
9    tin[v] = EXPLORED;
10   for (auto u: adj[v]) {
11       if (tin[u] == UNVISITED) { //tree edge
12           graphCheck(u, v);
13       } else if (tin[u] == EXPLORED) {
14           if (u == p)
15               ; //two way edge u <-> v
16           else
17               ; //back edge v -> u
18       } else if (tin[u] == VISITED) {
19           ; //forward/cross edge u-v
20       }
21   }
22   tin[v] = VISITED;
23 }
```

1.4   Articulations and Bridges

```
1 // Usage: dfs(source, -1)
2 // Status: not tested
3 // O(V+E) time & space
4
5 int tk = 0;
6 vector<int> tin(M, -1);
7 vector<vector<int>> adj(M);
8
9 void dfs(int v, int p) {
10   tin[v] = low[v] = tk++;
11   int children = 0;
12   for (auto u: adj[v]) {
13       if (u == p) continue;
14       else if (tin[u] == -1) {
15           ++children;
16           dfs(u, v);
17           if (low[u] >= tin[v] && p != v)
18               ; //articulation point
19           if (low[u] > tin[v])
20               ; //bridge u-v
21           low[v] = min(low[v], low[u]);
22       } else {
23           low[v] = min(low[v], tin[u]);
24       }
25   }
26 }
```

1.5   Euler Tour

```
1 // Usage: tour(cyc.begin(), start\_vertex)
2 // Status: not tested
3 // Source: CP3 (pg. 205)
4 // O(E^2) time
5
6 list<int> cyc;
7 vector<vector<int>> adj(M);
8 vector<vector<bool>> traversed(M, vector<bool>(M,
     false));
```

```
9
10 //euler tour (list for fast insertion)
11 void tour(list<int>::iterator i, int v) {
12    for (auto u: adj[v]) {
13        if (!traversed[v][u]) {
14            traversed[v][u] = true;
15            for (auto t: adj[u])
16                if (t == v && !traversed[u][t]) {
17                    traversed[u][t] = true;
18                    break;
19                }
20            tour(cyc.insert(i, v), u);
21        }
22    }
23 }
```

1.6   Kahn's topological sort

```
1 // Status: tested (UVA11060)
2 // O(VE) time, O(V+E) space
3
4 vector<vector<int>> adj(M);
5 vector<int> sorted;
6
7 void kahn(int n) {
8    vector<int> indeg(n, 0);
9    vector<bool> valid(n, true);
10   priority_queue<int> pq;
11
12   for (int v = 0; v < n; ++v)
13       for (auto u: adj[v])
14           indeg[u]++;
15   for (int v = 0; v < n; ++v)
16       if (!indeg[v]) pq.push(v);
17
18   while (!pq.empty()) {
19       int v = pq.top(); pq.pop();
20       sorted.push_back(v);
21       valid[v] = false;
22       for (auto u: adj[v])
23           if (valid[u] && !(--indeg[u]))
24               pq.push(u);
25   }
26 }
```

1.7   Max Cardinality Bipartite Matching

```
1 // Status: not tested
2 // Source: CP3 (pg. 209)
3 // O(VE) time
4
5 vector<vector<int>> adj(M);
6 vector<int> match(M, -1);
7 vector<bool> visited(M);
8
9 bool augment(int left) { //match one on the left
     with one on the right
10   if (visited[left]) return false;
11   visited[left] = true;
12   for (auto right: adj[left])
13       if (match[right] == -1 ||
              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  // O(N log N) time, O(N log N) space
3
4  //--- binary lifting
5  int n, l = ceil(log2(n));
6  vector<vector<int>> adj;
7  int tk = 0;
8  vector<int> tin(n), tout(n);
9  vector<vector<int>> up(n, vector<int>(l+1)); //
       ancestor
10
11 void dfs(int v, int p) { // run dfs(root, root) to
       initialize
12     tin[v] = ++tk;
13     up[v][0] = p;
14     for (int i = 1; i <= l; ++i)
15         up[v][i] = up[up[v][i-1]][i-1];
16     for (int u : adj[v])
17         if (u != p)
18             dfs(u, v);
19     tout[v] = ++tk;
20 }
21
22 bool ancestor(int v, int u) { // v is ancestor of u
23     return tin[v] <= tin[u] && tout[v] >= tout[u];
24 }
25
26 int lca(int v, int u) {
27     if (ancestor(v, u)) return v;
28     if (ancestor(u, v)) return u;
29     for (int i = l; i >= 0; --i)
30         if (!ancestor(up[v][i], u))
31             v = up[v][i];
32     return up[v][0];
33 }
34
35 //--- euler path
36 using ii = pair<int, int>;
37 vector<ii> t;
38 vector<int> idx(n);
39 int tk = 1;
40
41 void dfs(int v, int d) { // call with dfs(root, 0);
42     for (auto u : adj[v]) {
43         st.update(tk, {d, v});
44         tk++;
45         dfs(u, d+1);
46     }
47     idx[v] = tk;
48     st.update(tk, {d, v});
49     tk++;
50 }
51
52 int lca(int v, int u) {
53     int l = idx[v], r = idx[u];
54     return st.minquery(l, r).second; // .first is
          depth
55 }
```

1.9 Tarjan Strongly Connected Component

```
1  // Usage: Tarjan(N, adj)
2  // Status: tested (UVA247, UVA11838)
3  // O(V+E) time & space
4
5  vector<int> tin(M, -1), low(M, -1);
6  vector<vector<int>> adj(M);
7  stack<int> S;
8  int tk = 0;
```

```
9
10 void dfs(int v) {
11     low[v] = tin[v] = tk++;
12     S.push(v);
13     visited[v] = true;
14     for (auto u: adj[v]) {
15         if (tin[u] == -1)
16             dfs(u);
17         if (visited[u])
18             low[v] = min(low[v], low[u]);
19     }
20     if (low[v] == tin[v])
21         while (true) {
22             int u = S.top(); S.pop(); visited[u] =
                  false;
23             if (u == v) break;
24         }
25 }
```

1.10 Kosaraju SCC

```
1  // Usage: Tarjan(N, adj)
2  // not tested
3  // O(V+E) time & space (2 dfs calls)
4
5  int n; // number of vertices
6  vector<vector<int>> adj(n), adj_rev(n);
7  vector<bool> used(n);
8  vector<int> order, component;
9
10 void dfs1(int v) {
11   used[v] = true;
12   for (auto u: adj[v])
13     if (!used[u])
14       dfs1(u);
15   order.push_back(v);
16 }
17
18 void dfs2(int v) {
19   used[v] = true;
20   component.push_back(v);
21   for (auto u: adj_rev[v])
22     if (!used[u])
23       dfs2(u);
24 }
25
26 void kosaraju() {
27   for (int i = 0; i < n; ++i)
28     if (!used[i]) dfs1(i);
29
30   used.assign(n, false);
31   reverse(order.begin(), order.end());
32
33   for (auto v: order)
34     if (!used[v]) {
35       dfs2(v);
36       // ...process vertices in component
37       component.clear();
38     }
39 }
```

1.11 Bellman-Ford SSSP

```
1  // Status: tested (UVA1112, UVA10449)
2  // O(VE) time, O(V+E) space
3  const int inf = 0x3f3f3f3f;
4  vector<vector<pair<int, int>>> adj(M);
5  vector<int> dist(M, inf);
6
7  void bellmanFord(int n) {
8      for (int i = 0; i < n-1; ++i)
9          for (int v = 0; v < n; ++v)
10             for (auto &[u, w]: adj[v])
11                 if (dist[v] != inf)
12                     dist[u] = min(dist[u],
                          dist[v]+w);
13 }
```

```
14
15 //check if there are negative cycles
16 bool cycle(int n) {
17     bool ans = false;
18     for (int v = 0; v < n; ++v)
19         for (auto &[u, w]: v)
20             ans |= dist[v] != inf && dist[u] >
                    dist[v]+w;
21 }
```

1.12 Kruskal MST

```
1 // Usage: Kruskal(V, E, edges) (weighted edges)
2 // Status: tested (UVA1174)
3 // O(E log V) time, O(V+E) space
4
5 using iii = pair<int, pair<int, int>>; //weight,
        two vertices
6 vector<iii> edges;
7 UnionFind muf;
8
9 int kruskal() {
10     int cost = 0;
11     sort(edges.begin(), edges.end());
12     for (auto a: edges) {
13         int w = a.first;
14         pair<int, int> e = a.second;
15         if (!muf.isSameSet(e.first, e.second)) {
16             cost += w;
17             muf.unionSet(e.first, e.second);
18         }
19     }
20     return cost;
21 }
```

1.13 Edmond Karp MaxFlow

```
1 // Status: tested (CSES1694, CSES1695)
2 // O(VE^2) time, O(V+E) space
3
4 vector<vector<int>> capacity(M, vector<int>(M,
        0)), adj(M);
5 vector<pair<int, int>> mc; //mincut edges
6
7 int bfs(int s, int t, vi &par) {
8     fill(all(par), -1);
9     par[s] = -2;
10     queue<pair<int, int>> q; q.push({s, inf});
11     while (!q.empty()) {
12         int v = q.front().first,
13             flow = q.front().second;
14         q.pop();
15         for (auto u: adj[v])
16             if (par[u] == -1 && capacity[v][u]) {
17                 par[u] = v;
18                 int new_flow = min(flow,
                        capacity[v][u]);
19                 if (u == t) return new_flow;
20                 q.push({u, new_flow});
21             }
22     }
23     return 0;
24 }
25
26 int maxflow(int s, int t) {
27     int flow = 0;
28     vi par(M);
29     int new_flow;
30     while ((new_flow = bfs(s, t, par))) {
31         flow += new_flow;
32         int v = t;
33         while (v != s) {
34             int p = par[v];
35             capacity[p][v] -= new_flow;
36             capacity[v][p] += new_flow;
37             v = p;
38         }
```

```
39     }
40     return flow;
41 }
42
43 void mincut(int s, int t) {
44     maxflow(s, t);
45     stack<int> st;
46     vector<bool> visited(n, false);
47     vector<pair<int, int>> ans;
48     st.push(s); // changed from 0 to s
49     while (!st.empty()) {
50         int v = st.top(); st.pop();
51         if (visited[v]) continue;
52         visited[v] = true;
53         for (auto u: adj[v])
54             if (capacity[v][u] > 0)
55                 st.push(u);
56             else
57                 ans.push_back({v, u});
58     }
59     mc.clear();
60     for (auto &[v, u] : ans)
61         if (!visited[u])
62             mc.push_back({v, u});
63 }
```

1.14 Floyd Warshall APSP

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

2 Math

2.1 Sieve of Eratosthenes

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

```
22    int sieve_size = 11234567;
23    if (n <= sieve_size) return pr[n];
24    for (auto p: primes) // only works if n <=
         primes.back()^2
25        if (!(n%p)) return false;
26    return true;
27 }
```

2.2   Prime Factors w/ Optimized Trial Divisions

```
1  // Status: not tested
2  // Source: CP3 (pg. 238)
3  // O(pi(sqrt(n))) time, O(n) space
4
5  vector<int> primes;
6  vector<pair<int, int>> factors;
7
8  void pf(int n) {
9      for (auto p: primes) {
10         if (p*p > n) break;
11         int i = 0;
12         while (!(n%p)) {
13             n /= p;
14             i++;
15         }
16         factors.push_back({p, i});
17     }
18     if (n != 1) factors.push_back({n, 1});
19 }
```

2.3   Extended Euclid for solving Linear Diophantine Equations

```
1  // Status: not tested
2  // Source: CP3 (pg. 242)
3  // O(log min(a, b)) time
4
5  int x, y, d;
6  void extendedEuclid(int a, int b) {
7      if (b == 0) { x = 1; y = 0; d = a; return; }
8      extendedEuclid(b, a%b);
9      int x1 = y;
10     int y1 = x - (a/b)*y;
11     x = x1;
12     y = x1;
13 }
14
15 void solve(int a, int b, int c, int i) { //i
       solutions
16     extendedEuclid(a, b);
17     if (d%c) return;
18     x *= c/d;
19     y *= c/d;
20     do {
21         cout << x << ", " << y << '\n';
22         x += b/d;
23         y -= a/d;
24     } while (--i);
25 }
```

2.4   Floyd's algorithm cycle-finding

```
1  // Status: not tested
2  // Source: CPHB (p. 156)
3  // O(V) time
4
5  int findCycle(int x) {
6      int a, b;
7      a = succ(x);
8      b = succ(succ(x));
9      while (a != b) {
10         a = succ(a);
11         b = succ(succ(b));
```

```
12     }
13     a = x;
14     while (a != b) {
15         a = succ(a);
16         b = succ(b);
17     }
18     int first = a; // first element in cycle
19     b = succ(a);
20     int length = 1;
21     while (a != b) {
22         b = succ(b);
23         length++;
24     }
25 }
```

3   Paradigm

3.1   Coordinate Compression

```
1  // Status: not tested
2  // Source: GEMA ICMC (YouTube)
3  // O(N log N) time
4
5  vector<int> v, vals, cv; // all of the same size,
       cv = compressed v
6  vals = v;
7  sort(vals.begin(), vals.end());
8  vals.erase(unique(vals.begin(), vals.end()),
       vals.end());
9  for (int i = 0; i < n; ++i) {
10     int idx = lower_bound(vals.begin(),
           vals.end(), v[i]) - vals.begin();
11     cv[i] = idx;
12 }
```

3.2   128 Bit Integers

```
1  // Status: not tested
2  // Source: GEMA (YouTube)
3
4  // cout, cerr, etc; pode dar 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; pode dar 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  // O(log N) time
4
5  // std
6  int l = 0, r = n-1;
7  while (l <= r) {
8      int m = l+(r-l)/2;
9      if (array[m] == x)
10         // found
11     if (array[m] > x) r = m-1;
12     else l = m+1;
```

```
13 }
14
15 // nice - binary steps
16 int k = 0;
17 for (int b = n/2; b > 0; b /= 2)
18     while (k+b < n && array[k+b] <= x)
19         k += b;
20 if (array[k] == x)
21     // found
```

4   String

4.1   Prefix Function (KMP)

```
1 // Status: not tested
2 // Source: CP-Algorithms
3 // O(N) time
4
5 vector<int> prefix(string s) {
6     int n = s.length();
7     vector<int> pi(n, 0); // can be optimized if
         you know max prefix length
8     for (int i = 1; i < n; ++i) {
9         int j = pi[i-1];
10        while (j > 0 && s[i] != s[j])
11            j = pi[j-1];
12        if (s[i] == s[j])
13            j++;
14        pi[i] = j;
15    }
16    return pi;
17 }
```

5   Structure

5.1   Merge/Disjoint Union-Find

```
1 // Usage: UnionFind(N);
2 // Status: tested (UVA11503)
3 // O(Ackermann * N) time, O(N) space
4
5 struct UnionFind {
6     int N;
7     vi par, rk, count;
8
9     UnionFind(int N) : N(N), par(N), rk(N, 0),
         count(N, 1) {
10        rep(i, 0, N) par[i] = i;
11    }
12
13    int findSet(int i) {
14        return par[i] == i ? i : (par[i] =
             findSet(par[i]));
15    }
16
17    int unionSet(int a, int b) {
18        int x = findSet(a), y = findSet(b);
19        if (x != y)
20            count[x] = count[y] =
                 (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 // Complexity:
5 //   build: O(n)
6 //   query: O(log n)
7 //   modify: O(log n)
8 //   + uses less space than top-down 4n segtree (2n
        here)
9
10 struct SegTree {
11     unsigned int n;
12     vector<int> tree;
13
14     SegTree(vector<int> v) : n(v.size()), tree(2*n) {
15         for (int i = 0; i < n; ++i)
16             modify(i, v[i]);
17     }
18
19     int query(int a, int b) {
20         a += n, b += n;
21         int ans = 0;
22         while (a <= b) {
23             if (a%2 == 1) ans += tree[a++];
24             if (b%2 == 0) ans += tree[b--];
25             a >>= 1; b >>= 1;
26         }
27         return ans;
28     }
29
30     void modify(int k, int x) {
31         k += n;
32         tree[k] += x;
33         for (k /= 2; k >= 1; k /= 2)
34             tree[k] = tree[k<<1] + tree[(k<<1) + 1];
35     }
36 };
```

5.3   Segment Tree

```
1 // Usage: SegTree(N);
2 // Complexity:
3 //   build: O(n)
4 //   query: O(n)
5 //   modify: O(n)
6
7 struct SegTree {
8     int N;
9     vi st, A;
10
11    SegTree(int N): N(N), st(4*n), A(n) {
12        init();
13    }
14
15    void init() { build(1, 0, n-1); }
16
17    int left(int i) { return i*2; }
18    int right(int i) { return i*2+1; }
19
20    void build(int v, int tl, int tr) {
21        if (tl == tr) st[v] = a[tl];
22        else {
23            int tm = (tl+tr)/2;
24            build(left(v), tl, tm);
25            build(right(v), tm+1, tr);
26            st[v] = max(st[left(v)], st[right(v)]);
27        }
28    }
29
30    int maxquery(int v, int tl, int tr, int l, int
        r) {
31        if (l > r) return -1;
32        if (l == tl && r == tr) return st[v];
33        int tm = (tl+tr)/2;
34        int q1 = maxquery(left(v), tl, tm, l,
             min(r, tm));
```

```cpp
35          int q2 = maxquery(right(v), tm+1, tr,
                max(l, tm+1), r);
36          return max(q1, q2);
37      }
38
39      int maxquery(int l, int r) {
40          return maxquery(1, 0, n-1, l-1, r-1);
41      }
42
43      void update(int v, int tl, int tr, int p, int
           new_val) {
44          if (tl == tr) st[v] = new_val;
45          else {
46              int tm = (tl+tr)/2;
47              if (p <= tm)
48                  update(left(v), tl, tm, p,
                        new_val);
49              else
50                  update(right(v), tm+1, tr, p,
                        new_val);
51              st[v] = max(st[left(v)], st[right(v)]);
52          }
53      }
54
55      void update(int p, int new_val) {
56          update(1, 0, n-1, p-1, new_val);
57      }
58 };
```

6 Extra

6.1 Bashrc

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

```cpp
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

```cpp
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
```