Competitive Programming Notebook

Raul Almeida

Contents

1 Graph

1.1 Prim MST

```cpp
// take smallest edge that leads to vertex v
//
// Time: O(E log V)
// Space: O(V + E)
// Status: tested (UVA10048)

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
// Time: O((V+E) log V)
// Space: O(V^2)
//
// Status: tested (CF20C)

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])
            if (dist[v] != inf && dist[v]+d <
                dist[u]) {
                par[u] = v;
                dist[u] = dist[v]+d;
                pq.push(make_pair(dist[u], u));
            }
    }
}
```

1.3 Graph Check

```
// Check for tree edges, back edges and forward
    edges
//
// Usage: graphCheck(firstVertex, -1) (p stands
    for parent)
// Time: O(V + E)
// Space: O(V + E)

int UNVISITED = -1, EXPLORED = 0, VISITED = 1;
vector<vector<int>> adj(M);
vector<int> tin;

void graphCheck(int v, int p) { //vertex, parent
    tin[v] = EXPLORED;
    for (auto u: adj[v]) {
        if (tin[u] == UNVISITED) { //tree edge
            graphCheck(u, v);
        } else if (tin[u] == EXPLORED) {
            if (u == p)
                ; //two way edge u <-> v
            else
                ; //back edge v -> u
        } else if (tin[u] == VISITED) {
            ; //forward/cross edge u-v
        }
    }
    tin[v] = VISITED;
}
```

1.4 Articulations and Bridges

```
// Articulation point v: if you remove vertex v,
    then the connected component to which it
    belongs becomes disconnected
// Bridge u,v: if you remove the edge u->v, then
    you can't reach v from u
//
// Usage: dfs(source, -1)
//
// Time: O(V + E)
// Space: O(V + E)
// Status: not tested

int tk = 0;
vector<int> tin(M, -1);
vector<vector<int>> adj(M);

void dfs(int v, int p) {
    tin[v] = low[v] = tk++;
    int children = 0;
    for (auto u: adj[v]) {
        if (u == p) continue;
        else if (tin[u] == -1) {
            ++children;
            dfs(u, v);
            if (low[u] >= tin[v] && p != v)
                ; //articulation point
            if (low[u] > tin[v])
                ; //bridge u-v
            low[v] = min(low[v], low[u]);
        } else {
            low[v] = min(low[v], tin[u]);
        }
    }
}
```

1.5 Euler Tour

```
// Find the closest neighbor that has a path back
    to the current vertex to build an euler tour
//
// Euler path: visits each edge once
// Tour/cycle/circuit: euler path that starts and
    ends at same node
//
// Undirected and has path: every vertex has even
    degree or two have odd degree
// Undirected and has circuit: every vertex has
    even degree
// Directed and has path: indeg[i]-outdeg[i] == 1
    for at most one i, -1 for at most one i, 0 for
    all other i
// Directed and has circuit: indeg[i] == outdeg[i]
    for every i
//
// Usage: tour(cyc.begin(), start\_vertex)
// Time: O(E^2)
// Status: not tested
// Source: CP3 (pg. 205)

list<int> cyc;
vector<vector<int>> adj(M);
vector<vector<bool>> traversed(M, vector<bool>(M,
    false));

//euler tour (list for fast insertion)
void tour(list<int>::iterator i, int v) {
    for (auto u: adj[v]) {
        if (!traversed[v][u]) {
            traversed[v][u] = true;
            for (auto t: adj[u])
                if (t == v && !traversed[u][t]) {
                    traversed[u][t] = true;
                    break;
                }
            tour(cyc.insert(i, v), u);
        }
    }
}
```

1.6 Kahn's topological sort

```
// particular order (alphabetical)
// Time: O(VE)
// Space: O(V+E)
// Status: tested (UVA11060)

vector<vector<int>> adj(M);
vector<int> sorted;

void kahn(int n) {
    vector<int> indeg(n, 0);
    vector<bool> valid(n, true);
    priority_queue<int> pq;

    for (int v = 0; v < n; ++v)
        for (auto u: adj[v])
            indeg[u]++;
    for (int v = 0; v < n; ++v)
        if (!indeg[v]) pq.push(v);

    while (!pq.empty()) {
        int v = pq.top(); pq.pop();
        sorted.push_back(v);
        valid[v] = false;
        for (auto u: adj[v])
            if (valid[u] && !(--indeg[u]))
                pq.push(u);
    }
```

```
}
```

1.7 Max Cardinality Bipartite Matching

```
// Jump from free to matched edges until you've used them all
//
// Time: O(VE)
// Status: not tested
// Source: CP3 (pg. 209)

vector<vector<int>> adj(M);
vector<int> match(M, -1);
vector<bool> visited(M);

bool augment(int left) { //match one on the left with one on the right
    if (visited[left]) return false;
    visited[left] = true;
    for (auto right: adj[left])
        if (match[right] == -1 || augment(match[right])) {
            match[right] = left;
            return true;
        }
    return false;
}

//usage
//(mcbm = V iff there's at least one way to completely match both sides)
int mcbm = 0; //number of matched vertices
match.assign(M, -1);
for (int v = 0; v < ls; ++v) {//ls = size of the left set
    visited.assign(ls, false);
    mcbm += augment(v);
}
```

1.8 Lowest Common Ancestor

```
// Binary lift to binary search the LCA or euler path
//
// Time: O(N log N)
// Space: O(N log N)
// Status: not tested

//--- binary lifting
int n, l = ceil(log2(n));
vector<vector<int>> adj;
int tk = 0;
vector<int> tin(n), tout(n);
vector<vector<int>> up(n, vector<int>(l+1)); //ancestor

void dfs(int v, int p) { // run dfs(root, root) to initialize
    tin[v] = ++tk;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];
    for (int u : adj[v])
        if (u != p)
            dfs(u, v);
    tout[v] = ++tk;
}

bool ancestor(int v, int u) { // v is ancestor of u
    return tin[v] <= tin[u] && tout[v] >= tout[u];
}

int lca(int v, int u) {
    if (ancestor(v, u)) return v;
    if (ancestor(u, v)) return u;
    for (int i = l; i >= 0; --i)
        if (!ancestor(up[v][i], u))
            v = up[v][i];
    return up[v][0];
}

//--- euler path
using ii = pair<int, int>;
vector<ii> t;
vector<int> idx(n);
int tk = 1;

void dfs(int v, int d) { // call with dfs(root, 0);
    for (auto u : adj[v]) {
        st.update(tk, {d, v});
        tk++;
        dfs(u, d+1);
    }
    idx[v] = tk;
    st.update(tk, {d, v});
    tk++;
}

int lca(int v, int u) {
    int l = idx[v], r = idx[u];
    return st.minquery(l, r).second; // .first is depth
}
```

1.9 Tarjan Strongly Connected Component

```
// A node can reach any other node in its own SCC (DFS+stack)
//
// Usage: Tarjan(N, adj)
// Time: O(V + E)
// Space: O(V + E)
// Status: tested (UVA247, UVA11838)

vector<int> tin(M, -1), low(M, -1);
vector<vector<int> adj(M);
stack<int> S;
int tk = 0;

void dfs(int v) {
    low[v] = tin[v] = tk++;
    S.push(v);
    visited[v] = true;
    for (auto u: adj[v]) {
        if (tin[u] == -1)
            dfs(u);
        if (visited[u])
            low[v] = min(low[v], low[u]);
    }
    if (low[v] == tin[v])
        while (true) {
            int u = S.top(); S.pop(); visited[u] = false;
            if (u == v) break;
        }
}
```

1.10 Bellman-Ford SSSP

```
// Time: O(VE)
// Space: O(V + E)
// Status: tested (UVA1112, UVA10449)

const int inf = 0x3f3f3f3f;
vector<vector<pair<int, int>>> adj(M);
vector<int> dist(M, inf);

void bellmanFord(int n) {
    for (int i = 0; i < n-1; ++i)
        for (int v = 0; v < n; ++v)
```

```
            for (auto &[u, w]: adj[v])
                if (dist[v] != inf)
                    dist[u] = min(dist[u],
                        dist[v]+w);
}

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

1.11 Kruskal MST

```
// order edges by increasing weight, then use a
    MUF to know if each edge is useful (if it
    connects two previously disconnected vertices)
//
// Min Span Subgraph: previously process fixed
    edges
//
// Min Span Forest: count number of sets on the MUF
//
// 2nd Best MST: run kruskal; for each chosen
    edge, flag it as unavailable and run it
    without using that edge ($O(VE)$)
//
// Minimax: max edge weight on the MST (maximin:
    min)
//
// Usage: Kruskal(V, E, edges) (weighted edges)
//
// Time: O(E log V)
// Space: O(V + E)
// Status: tested (UVA1174)

using iii = pair<int, pair<int, int>>; //weight,
    two vertices
vector<iii> edges;
UnionFind muf;

int kruskal() {
    int cost = 0;
    sort(edges.begin(), edges.end());
    for (auto a: edges) {
        int w = a.first;
        pair<int, int> e = a.second;
        if (!muf.isSameSet(e.first, e.second)) {
            cost += w;
            muf.unionSet(e.first, e.second);
        }
    }
    return cost;
}
```

1.12 Edmond Karp MaxFlow

```
// Ford-Fulkerson's method with BFS = O(VE) BFS
    calls, O(E) per BFS
//
// Vertex weights: if vertex V has a weight,
    create Vin (receives all in-edges of V and has
    an edge to Vout) and Vout(receives an edge
    from Vin and has all out-edges of V); Vin-Vout
    has the weight from V
//
// MinCut: run EdmondKarp, S-T sets are: all V
    that you can reach from the source with edges
    of positive residual capacity and all other V
//
// MultiSource/MultiSink: create a super source
    with infinite capacity pointing to all
    sources, analogous for sinks
//
// Max Cardinality Bipartite Matching: use
    capacity 1 on all edges and apply the
    multi-source and multi-sink strategies
//
// Time: O(VE^2)
// Space: O(V + E)
// Status: tested (CSES1694, CSES1695)

vector<vector<int>> capacity(M, vector<int>(M,
    0)), adj(M);
vector<pair<int, int>> mc; //mincut edges

int bfs(int s, int t, vi &par) {
    fill(all(par), -1);
    par[s] = -2;
    queue<pair<int, int>> q; q.push({s, inf});
    while (!q.empty()) {
        int v = q.front().first,
            flow = q.front().second;
        q.pop();
        for (auto u: adj[v])
            if (par[u] == -1 && capacity[v][u]) {
                par[u] = v;
                int new_flow = min(flow,
                    capacity[v][u]);
                if (u == t) return new_flow;
                q.push({u, new_flow});
            }
    }
    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vi par(M);
    int new_flow;
    while ((new_flow = bfs(s, t, par))) {
        flow += new_flow;
        int v = t;
        while (v != s) {
            int p = par[v];
            capacity[p][v] -= new_flow;
            capacity[v][p] += new_flow;
            v = p;
        }
    }
    return flow;
}

void mincut(int s, int t) {
    maxflow(s, t);
    stack<int> st;
    vector<bool> visited(n, false);
    vector<pair<int, int>> ans;
    st.push(0);
    while (!st.empty()) {
        int v = st.top(); st.pop();
        if (visited[v]) continue;
        visited[v] = true;
        for (auto u: adj[v])
            if (capacity[v][u] > 0)
                st.push(u);
            else
                ans.push_back({v, u});
    }
    mc.clear();
    for (auto &[v, u] : ans)
        if (!visited[u])
            mc.push_back({v, u});
}
```

1.13 Floyd Warshall APSP

```
// Also works for SSSP (V <= 400)
//
// Printing path: p[i][j] set to i (last node that
    appears before j on the path), then p[i][j] =
    p[k][j] on update.
```

```
//
// Transitive Closure: weight is boolean (init as 1 if there's an edge), update with bitwise OR
//
// Minimax/Maximin: w[i][j] = min(w[i][j], max(w[i][k], w[k][j]))
//
// Finding negative/cheapest cycle: init w[i][i] = inf; run(); any w[i][i] != inf is a cycle and the smallest is the cheapest; any w[i][i] < 0 is negative
//
// This can also be used for finding SCCs (check with transitive closure)
//
// Usage: FloydWarshall(n, edges)
// Time: O(V^3+E)
// Space: O(V^2+E)
// Status: tested (UVA821, UVA1056)

struct edge { int v, u, w; };
const int inf = 0x3f3f3f3f;
vector<vector<int>> weight(M, vector<int>(M, inf));
vector<edge> edges;

void floydWarshall(int n) {
    for (auto e: edges)
        weight[e.v][e.u] = e.w;
    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                if (max(weight[i][k], weight[k][j]) < inf)
                weight[i][j] = min(weight[i][j], weight[i][k]+weight[k][j]);
}
```

2 Math

2.1 Sieve of Eratosthenes

```
// Time: O(n log log n)
// Space: O(n)
// Status: not tested

bitset<11234567> pr;
vector<int> factors(M, 0);
vector<int> primes;

void sieve(int n) {
    pr.set();
    for (int i = 2; i*i <= n; ++i)
        if (pr[i]) { //factors[i] == 0
            primes.push_back(i);
            for (int p = i*i; p <= n; p += i) {
                pr[p] = false;
                factors[p]++;
            }
        }
}

// O(1) for small n, O(sieve_size) else
bool isPrime(int n) {
    int sieve_size = 11234567;
    if (n <= sieve_size) return pr[n];
    for (auto p: primes) // only works if n <= primes.back()^2
        if (!(n%p)) return false;
    return true;
}
```

2.2 Prime Factors w/ Optimized Trial Divisions

```
// Time: O(Pi(sqrt(n)))
// Space: O(n)
// Status: not tested
// Source: CP3 (pg. 238)

vector<int> primes;
vector<pair<int, int>> factors;

void pf(int n) {
    for (auto p: primes) {
        if (p*p > n) break;
        int i = 0;
        while (!(n%p)) {
            n /= p;
            i++;
        }
        factors.push_back({p, i});
    }
    if (n != 1) factors.push_back({n, 1});
}
```

2.3 Extended Euclid for solving Linear Diophantine Equations

```
// Time: O(log min(a, b))
// Status: not tested
// Source: CP3 (pg. 242)

int x, y, d;
void extendedEuclid(int a, int b) {
    if (b == 0) { x = 1; y = 0; d = a; return; }
    extendedEuclid(b, a%b);
    int x1 = y;
    int y1 = x - (a/b)*y;
    x = x1;
    y = x1;
}

void solve(int a, int b, int c, int i) { //i solutions
    extendedEuclid(a, b);
    if (d%c) return;
    x *= c/d;
    y *= c/d;
    do {
        cout << x << ", " << y << '\n';
        x += b/d;
        y -= a/d;
    } while (--i);
}
```

2.4 Floyd's algorithm cycle-finding

```
// Slow and fast (tortoise and hare)
//
// Time: O(V)
// Space: O(1)
// Status: not tested
// Source: CPHB (p. 156)

int findCycle(int x) {
    int a, b;
    a = succ(x);
    b = succ(succ(x));
    while (a != b) {
        a = succ(a);
        b = succ(succ(b));
    }
    a = x;
    while (a != b) {
        a = succ(a);
        b = succ(b);
    }
    int first = a; // first element in cycle
    b = succ(a);
    int length = 1;
```

```
    while (a != b) {
        b = succ(b);
        length++;
    }
}
```

3 Paradigm

3.1 Coordinate Compression

```
// normalize vector access; can also be done with map/set but high constant
//
// Time: O(N log N)
// Status: not tested
// Source: GEMA ICMC (YouTube)

vector<int> v, vals, cv; // all of the same size, cv = compressed v
vals = v;
sort(vals.begin(), vals.end());
vals.erase(unique(vals.begin(), vals.end()), vals.end());
for (int i = 0; i < n; ++i) {
    int idx = lower_bound(vals.begin(), vals.end(), v[i]) - vals.begin();
    cv[i] = idx;
}
```

3.2 128 Bit Integers

```
// GCC extension; 2^127 -1 ~ 10^38
//
// Status: not tested
// Source: GEMA (YouTube)

// cout, cerr, etc; pode dar over/underflow
ostream& operator<<(ostream& out, __int128 x) {
    if (x == 0) return out << 0;
    string s; bool sig = x < 0; x = x < 0 ? -x : x;
    while(x > 0) s += x % 10 + '0', x /= 10;
    if (sig) s += '-';
    reverse(s.begin(), s.end());
    return out << s;
}

// cin, etc; pode dar over/underflow
istream& operator>>(istream& in, __int128& x) {
    char c, neg = 0; while(isspace(c = in.get()));
    if(!isdigit(c)) neg = (c == '-'), x = 0;
    else x = c - '0';
    while(isdigit(c = in.get())) x = (x << 3) + (x << 1) - '0' + c;
    x = neg ? -x : x; return in;
}
```

3.3 Binary Search (but beautiful)

```
// binary steps
//
// Time: O(log N)
// Status: not tested
// Source: CPHB

// std
int l = 0, r = n-1;
while (l <= r) {
    int m = l+(r-l)/2;
    if (array[m] == x)
        // found
    if (array[m] > x) r = m-1;
    else l = m+1;
}

// nice
int k = 0;
for (int b = n/2; b > 0; b /= 2)
    while (k+b < n && array[k+b] <= x)
        k += b;
if (array[k] == x)
    // found
```

4 String

4.1 Prefix Function (KMP)

```
//
// to find ocurrences of s in t, use the string s+'%'+t, then look for pi[i] = s.length() on the "t side"
// Time: O(N)
// Status: not tested
// Source: CP-Algorithms

vector<int> prefix(string s) {
    int n = s.length();
    vector<int> pi(n, 0); // can be optimized if you know max prefix length
    for (int i = 1; i < n; ++i) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

5 Structure

5.1 Merge/Disjoint Union-Find

```
//
// Usage: UnionFind(N);
// Time: O(AM), ackerman * num_operations
// Space: O(N), elements
// Status: tested (UVA11503)

struct UnionFind {
    int N;
    vi par, rk, count;

    UnionFind(int N) : N(N), par(N), rk(N, 0), count(N, 1) {
        rep(i, 0, N) par[i] = i;
    }

    int findSet(int i) {
        return par[i] == i ? i : (par[i] = findSet(par[i]));
    }

    int unionSet(int a, int b) {
        int x = findSet(a), y = findSet(b);
        if (x != y)
            count[x] = count[y] = (count[x]+count[y]);
        if (rk[x] < rk[y])
            par[x] = y;
        else {
            par[y] = x;
            if (rk[x] == rk[y])
                rk[x]++;
        }
        return count[x];
    }
```

```
    bool isSameSet(int i, int j) {
        return findSet(i) == findSet(j);
    }
};
```

5.2 Bottom-Up Segment Tree

```
//
// Usage: SegTree(N);
//
// Source: CP Handbook
// Time: O(log N)
// Space: O(N), elements
// Status: not tested

struct SegTree {
  unsigned int n;
  vector<int> tree;

  SegTree(vector<int> v) : n(v.size()), tree(2*n) {
    for (int i = 0; i < n; ++i)
      modify(i, v[i]);
  }

  int query(int a, int b) {
    a += n, b += n;
    int ans = 0;
    while (a <= b) {
      if (a%2 == 1) ans += tree[a++];
      if (b%2 == 0) ans += tree[b--];
      a >>= 1; b >>= 1;
    }
    return ans;
  }

  void modify(int k, int x) {
    k += n;
    tree[k] += x;
    for (k /= 2; k >= 1; k /= 2)
      tree[k] = tree[k<<1] + tree[(k<<1) + 1];
  }
};
```

5.3 Segment Tree

```
//
// Usage: SegTree(N)

struct SegTree {
    int N;
    vi st, A;

    SegTree(int N): N(N), st(4*n), A(n) {
        init();
    }

    void init() { build(1, 0, n-1); }

    int left(int i) { return i*2; }
    int right(int i) { return i*2+1; }

    //O(N)
    void build(int v, int tl, int tr) {
        if (tl == tr) st[v] = a[tl];
        else {
            int tm = (tl+tr)/2;
            build(left(v), tl, tm);
            build(right(v), tm+1, tr);
            st[v] = max(st[left(v)], st[right(v)]);
        }
    }

    //O(log n)
    int maxquery(int v, int tl, int tr, int l, int
        r) {
        if (l > r) return -1;
        if (l == tl && r == tr) return st[v];
        int tm = (tl+tr)/2;
        int q1 = maxquery(left(v), tl, tm, l,
            min(r, tm));
        int q2 = maxquery(right(v), tm+1, tr,
            max(l, tm+1), r);
        return max(q1, q2);
    }

    int maxquery(int l, int r) {
        return maxquery(1, 0, n-1, l-1, r-1);
    }

    //O(log n)
    void update(int v, int tl, int tr, int p, int
        new_val) {
        if (tl == tr) st[v] = new_val;
        else {
            int tm = (tl+tr)/2;
            if (p <= tm)
                update(left(v), tl, tm, p,
                    new_val);
            else
                update(right(v), tm+1, tr, p,
                    new_val);
            st[v] = max(st[left(v)], st[right(v)]);
        }
    }

    void update(int p, int new_val) {
        update(1, 0, n-1, p-1, new_val);
    }
};
```

6 Extra

6.1 Bashrc

```
xmodmap -e 'clear lock' -e 'keycode 66=Escape' #
    caps -> esc
alias e=vim

BASE_CP="/home/raul/cp2022"

alias c='g++ -Wall -Wconversion -Wfatal-errors -g
    -O2 -std=gnu++17 -fsanitize=undefined,address'
alias c14='g++ -Wall -Wconversion -Wfatal-errors
    -g -O2 -std=gnu++14
    -fsanitize=undefined,address'
alias p3='pypy3 -m py_compile'

tp () {
    [ -f "$1.cpp" ] && echo "$1.cpp already
        exists";
    [ ! -f "$1.cpp" ] && tail -n +2
        $BASE_CP/code/extra/template.cpp > $1.cpp
        && vim $1.cpp;
}

clip () {
    if [ -f "$1" ];
    then
        cat $1 | clip.exe;
    else
        echo "$1 not found"
    fi
}
```

6.2 Vim

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

6.3 Generator

```cpp
#include <bits/stdc++.h>
using namepsace std;

int main(int argc, char *argv[]) {
  cin.tie(0); ios_base::sync_with_stdio(0);
  if (argc < 2) {
    cout << "usage: " << argv[0] << " <seed>\n";
    exit(1);
  }
  srand(atoi(argv[1]));
  // use rand() for random value
}
```

6.4 C++ Template

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
}
```

6.5 Stress

```sh
for (( I=0; I < 5; I++ )); do
  ./gen $I > a.in
  ./brute < a.in > expected.txt
  ./a.out < a.in > output.txt
  if diff -u expected.txt output.txt; then : ; else
    echo "--> input:"; cat a.in
    echo "--> expected output:"; cat expected.txt
    echo "--> received output:"; cat output.txt
    break
  fi
  echo -n .
done
```