Contents

1 Algorithms

2 Data Structures

2.1 BIT.cpp

```
#include <bits/stdc++.h>
using namespace std;
struct BIT {
    int N;
    vector<int> val;
    BIT (int N) : N(N), val(N) {}

    void update (int idx, int v) {
        for (int x = idx; x < N; x += (x & -x))
            val[x] += v;
    }

    int query (int idx) {
        int ret = 0;
        for (int x = idx; x > 0; x -= (x & -x))
            ret += val[x];
        return ret;
    }
};
```

2.2 BIT_Range.cpp

```
#include <bits/stdc++.h>
using namespace std;
struct BIT_Range {
    int N;
    vector<int> val1, val2;
    BIT_Range (int N): N(N), val1(N), val2(N) {}

    void update (vector<int> &val, int idx, int v) {
        for (int x = idx; x < N; x += (x & -x))
            val[x] += v;
    }

    void update (int x1, int x2, int val) {
        update(val1, x1, val);
        update(val1, x2 + 1, -val);
        update(val2, x1, val * (x1 - 1));
        update(val2, x2 + 1, -val * x2);
    }

    int query (vector<int> &val, int idx) {
        int ret = 0;
        for (int x = idx; x > 0; x -= (x & -x))
            ret += val[x];
        return ret;
    }

    int query (int x) {
        return query(val1, x) * x - query(val2, x);
    }

    int query (int x1, int x2) {
        return query(x2) - query(x1 - 1);
    }
};
```

2.3 Treap.cpp

```
#include <bits/stdc++.h>
using namespace std;
int randomPriority () {
    return rand() * 65536 + rand();
}
struct Node {
    int val, p;
    Node *left, *right;
    Node (int val): val(val), p(randomPriority()) {
        left = nullptr;
        right = nullptr;
    }
};
struct Treap {
    Node* root;

    Treap () {
        root = nullptr;
    }

    // precondition: all values of u are smaller than all values of v
    Node* join (Node* u, Node* v) {
        if (u == nullptr)
            return v;
        if (v == nullptr)
            return u;
        if (u->p < v->p) {
            u->right = join(u->right, v);
            return u;
        }
        v->left = join(u, v->left);
        return v;
    }

    pair<Node*, Node*> split (Node* u, int k) {
        if (u == nullptr)
            return make_pair(nullptr, nullptr);
        if (u->val < k) {
            auto res = split(u->right, k);
            u->right = res.first;
            res.first = u;
            return res;
        } else if (u->val > k) {
```

```
            auto res = split(u->left, k);
            u->left = res.second;
            res.second = u;
            return res;
        } else {
            return make_pair(u->left, u->right);
        }
    }
    bool contains (int val) {
        return contains(root, val);
    }
    bool contains (Node* u, int val) {
        if (u == nullptr)
            return false;
        if (u->val < val)
            return contains(u->right, val);
        else if (u->val > val)
            return contains(u->left, val);
        return true;
    }
    void insert (int val) {
        if (contains(root, val))
            return;
        auto nodes = split(root, val);
        root = join(nodes.first, join(new Node(val), nodes.second));
    }
    void remove (int val) {
        if (root == nullptr)
            return;
        auto nodes = split(root, val);
        root = join(nodes.first, nodes.second);
    }
};
```

3 Geometry

3.1 ConvexHull.cpp

```
#include <bits/stdc++.h>
using namespace std;
struct Point {
    int x, y;
    Point (int x, int y): x(x), y(y) {}
    bool operator < (const Point& p) const {
        return make_pair(x, y) < make_pair(p.x, p.y);
    }
};

int ccw (Point p1, Point p2, Point p3) {
        return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
}

vector<Point> convexHull (vector<Point> pts) {
    vector<Point> u, l;
    sort(pts.begin(), pts.end());

    for (int i = 0; i < (int)pts.size(); i++) {
        int j = (int)l.size();
        while (j >= 2 && ccw(l[j - 2], l[j - 1], pts[i]) <= 0) {
            l.erase(l.end() - 1);
            j = (int)l.size();
        }
        l.push_back(pts[i]);
    }
    for (int i = (int)pts.size() - 1; i >= 0; i--) {
        int j = (int)u.size();
        while (j >= 2 && ccw(u[j - 2], u[j - 1], pts[i]) <= 0) {
            u.erase(u.end() - 1);
            j = (int)u.size();
        }
        u.push_back(pts[i]);
    }

    u.erase(u.end() - 1);
    l.erase(l.end() - 1);
    l.reserve(l.size() + u.size());
    l.insert(l.end(), u.begin(), u.end());

    return l;
}
```

4 Graph Theory

4.1 SCC.cpp

```
#include <bits/stdc++.h>
using namespace std;
struct SCC {
    int N, cnt, idCnt;
    vector<int> disc, lo, id;
    vector<bool> inStack;
    vector<vector<int>> adj;
    stack<int> s;

    SCC (int N): N(N), disc(N), lo(N), id(N), inStack(N), adj(N) {}

    void addEdge (int u, int v) {
        adj[u].push_back(v);
    }

    void dfs (int i) {
        disc[i] = lo[i] = ++cnt;
        inStack[i] = true;
        s.push(i);
        for (int j : adj[i]) {
            if (disc[j] == 0) {
                dfs(j);
                lo[i] = min(lo[i], lo[j]);
            } else if (inStack[j]) {
                lo[i] = min(lo[i], disc[j]);
            }
        }
        if (disc[i] == lo[i]) {
            while (s.top() != i) {
                inStack[s.top()] = false;
                id[s.top()] = idCnt;
                s.pop();
            }
            inStack[s.top()] = false;
            id[s.top()] = idCnt++;
            s.pop();
        }
    }

    void compute () {
        for (int i = 0; i < N; i++)
            if (disc[i] == 0)
                dfs(i);
    }
};
```

4.2 Biconnected_Components.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> edge;
struct BiconnectedComponents {
    int N, cnt = 0;
    vector<edge> bridges;
    vector<vector<edge>> components;
    vector<vector<int>> adj;
    stack<edge> s;
    vector<int> lo, disc;
    vector<bool> vis, cutVertex;
```

```
    BiconnectedComponents (int N): N(N), adj(N), lo(N), disc(N), vis(N),
        cutVertex(N) {}

    void addEdge (int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs (int u, int prev) {
        disc[u] = lo[u] = cnt++;
        vis[u] = true;
        int children = 0;
        for (int v : adj[u]) {
            if (!vis[v]) {
                children++;
                s.push({u, v});
                dfs(v, u);
                lo[u] = min(lo[u], lo[v]);
                if ((disc[u] == 0 && children > 1) || (disc[u] > 0 && lo[v]
                    >= disc[u])) {
                    cutVertex[u] = true;
                    components.push_back(vector<edge>());
                    while (s.top().first != u && s.top().second != v) {
                        components.back().push_back(edge(s.top().first, s.top
                            ().second));
                        s.pop();
                    }
                    components.back().push_back(edge(s.top().first, s.top().
                        second));
                    s.pop();
                }
                if (lo[v] > disc[u])
                    bridges.push_back(edge(s.top().first, s.top().second));
            } else if (v != prev && disc[v] < lo[u]) {
                lo[u] = disc[v];
                s.push({u, v});
            }
        }
    }

    void compute () {
        for (int i = 0; i < N; i++)
            if (!vis[i])
                dfs(i, -1);
    }
};
```

4.3 Dinic's.cpp

```
#include <bits/stdc++.h>
using namespace std;
struct Edge {
    int dest, cost, next;
    Edge (int dest, int cost, int next): dest(dest), cost(cost), next(next)
        {}
};

struct Network {
    int N, src, sink;
    vector<int> last, dist;
    vector<Edge> e;

    Network (int N, int src, int sink): N(N), src(src), sink(sink), last(N),
        dist(N) {
        fill(last.begin(), last.end(), -1);
    }

    void AddEdge (int x, int y, int xy, int yx) {
        e.push_back(Edge(y, xy, last[x]));
        last[x] = (int)e.size() - 1;
        e.push_back(Edge(x, yx, last[y]));
        last[y] = (int)e.size() - 1;
    }

    bool getPath () {
        fill(dist.begin(), dist.end(), -1);
        queue<int> q;
        q.push(src);
        dist[src] = 0;

        while (!q.empty()) {
            int curr = q.front(); q.pop();
            for (int i = last[curr]; i != -1; i = e[i].next) {
                if (e[i].cost > 0 && dist[e[i].dest] == -1) {
                    dist[e[i].dest] = dist[curr] + 1;
                    q.push(e[i].dest);
                }
            }
        }

        return dist[sink] != -1;
    }

    int dfs (int curr, int flow) {
        if (curr == sink)
            return flow;
        int ret = 0;
        for (int i = last[curr]; i != -1; i = e[i].next) {
            if (e[i].cost > 0 && dist[e[i].dest] == dist[curr] + 1) {
                int res = dfs(e[i].dest, min(flow, e[i].cost));
                ret += res;
                e[i].cost -= res;
                e[i ^ 1].cost += res;
                flow -= res;
                if (flow == 0)
                    break;
            }
        }
        return ret;
    }

    int getFlow () {
        int res = 0;
        while (getPath())
            res += dfs(src, 1 << 30);
        return res;
    }
};
```

4.4 LCA.cpp

```
#include <bits/stdc++.h>
using namespace std;
struct LCA {
    int N, LN;
    vector<int> depth;
    vector<vector<int>> pa;
    vector<vector<int>> adj;
    LCA (int N): N(N), LN(ceil(log(N) / log(2) + 1)), depth(N), pa(N, vector<
        int>(LN)), adj(N) {
        for (auto &x : pa)
            fill(x.begin(), x.end(), -1);
    }

    void addEdge (int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs (int u, int d, int prev) {
        depth[u] = d;
        pa[u][0] = prev;
        for (int v : adj[u])
            if (v != prev)
                dfs(v, d + 1, u);
    }
```

```
    void precompute () {
        for (int i = 1; i < LN; i++)
            for (int j = 0; j < N; j++)
                if (pa[j][i - 1] != -1)
                    pa[j][i] = pa[pa[j][i - 1]][i - 1];

    }

    int getLca (int u, int v) {
        if (depth[u] < depth[v])
            swap(u, v);
        for (int k = LN - 1; k >= 0; k--)
            if (pa[u][k] != -1 && depth[pa[u][k]] >= depth[v])
                u = pa[u][k];
        if (u == v)
            return u;
        for (int k = LN - 1; k >= 0; k--)
            if (pa[u][k] != -1 && pa[v][k] != -1 && pa[u][k] != pa[v][k])
                u = pa[u][k], v = pa[v][k];
        return pa[u][0];
    }
};
```

4.5 HLD.cpp

```
#include <bits/stdc++.h>
using namespace std;
struct HLD {
    int N, chainIndex;
    vector<vector<int>> adj;
    vector<int> sz, depth, chain, par, head;

    HLD (int N): N(N), adj(N), sz(N), depth(N), chain(N), par(N), head(N) {
        fill(head.begin(), head.end(), -1);
    }

    void addEdge (int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs (int u, int p, int d) {
        par[u] = p;
        depth[u] = d;
        sz[u] = 1;
        for (int v : adj[u]) {
            if (v != p) {
                dfs(v, u, d + 1);
                sz[u] += sz[v];
            }
        }
    }

    void build (int u, int p) {
        if (head[chainIndex] == -1)
            head[chainIndex] = u;
        chain[u] = chainIndex;

        int maxIndex = -1;
        for (int v : adj[u])
            if (v != p && (maxIndex == -1 || sz[v] > sz[maxIndex]))
                maxIndex = v;
        if (maxIndex != -1)
            build(maxIndex, u);
        for (int v : adj[u])
            if (v != p && v != maxIndex) {
                chainIndex++;
                build(v, u);
            }
    }

    void precompute () {
        dfs(0, -1, 0);
        build(0, -1);
    }

    int getLca (int u, int v) {
        while (chain[u] != chain[v]) {
            if (depth[head[chain[u]]] < depth[head[chain[v]]])
                v = par[head[chain[v]]];
            else
                u = par[head[chain[u]]];
        }
        return depth[u] < depth[v] ? u : v;
    }
};
```

5 Mathematics

5.1 Euclid.cpp

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

int mod (int a, int b) {
    return ((a % b) + b) % b;
}

int gcd (int a, int b) {
    return b == 0 ? a : (gcd(b, a % b));
}

int lcm (int a, int b) {
    return a / gcd(a, b) * b;
}

// returns (d, x, y) such that d = gcd(a, b) and d = ax * by
vector<int> euclid (int a, int b) {
    int x = 1, y = 0, x1 = 0, y1 = 1, t;
    while (b != 0) {
        int q = a / b;
        t = x;
        x = x1;
        x1 = t - q * x1;
        t = y;
        y = y1;
        y1 = t - q * y1;
        t = b;
        b = a - q * b;
        a = t;
    }
    vector<int> ret = {a, x, y};
    if (a <= 0) ret = {-a, -x, -y};
    return ret;
}

// finds all solutions to ax = b mod n
vector<int> linearEquationSolver (int a, int b, int n) {
    vector<int> ret;
    vector<int> res = euclid(a, b);
    int d = res[0], x = res[1];

    if (b % d == 0) {
        x = mod(x * (b / d), n);
        for (int i = 0; i < d; i++)
            ret.push_back(mod(x + i * (n / d), n));
    }

    return ret;
}

// computes x and y such that ax + by = c; on failure, x = y = -1 << 30
void linearDiophantine (int a, int b, int c, int &x, int &y) {
    int d = gcd(a, b);

    if (c % d != 0) {
        x = y = -1 << 30;
    } else {
```

```
        a /= d;
        b /= d;
        c /= d;
        vector<int> ret = euclid(a, b);
        x = ret[1] * c;
        y = ret[2] * c;
    }
}

// precondition: m > 0 && gcd(a, m) = 1
int modInverse (int a, int m) {
    a = mod(a, m);
    return a == 0 ? 0 : mod((1 - modInverse(m % a, a) * m) / a, m);
}

// precondition: p is prime
vector<int> generateInverse (int p) {
    vector<int> res(p);
    res[1] = 1;
    for (int i = 2; i < p; ++i)
        res[i] = (p - (p / i) * res[p % i] % p) % p;
    return res;
}

// solve x = a[i] (mod p[i]), where gcd(p[i], p[j]) == 1
int simpleRestore (vector<int> a, vector<int> p) {
    int res = a[0];
    int m = 1;
    for (int i = 1; i < (int)a.size(); i++) {
        m *= p[i - 1];
        while (res % p[i] != a[i])
            res += m;
    }
    return res;
}

int garnerRestore (vector<int> a, vector<int> p) {
    vector<int> x(a.size());
    for (int i = 0; i < (int)x.size(); ++i) {
        x[i] = a[i];
        for (int j = 0; j < i; ++j) {
            x[i] = (int) modInverse(p[j], p[i]) * (x[i] - x[j]);
            x[i] = (x[i] % p[i] + p[i]) % p[i];
        }
    }
    int res = x[0];
    int m = 1;
    for (int i = 1; i < (int)a.size(); i++) {
        m *= p[i - 1];
        res += x[i] * m;
    }
    return res;
}
```

5.2 Combinatorics.cpp

```
#include <bits/stdc++.h>
typedef long long ll;

ll modpow (ll base, ll pow, ll mod) {
    if (pow == 0)
        return 1L;
    if (pow == 1)
        return base;
    if (pow % 2)
        return base * modpow(base * base % mod, pow / 2, mod) % mod;
    return modpow(base * base % mod, pow / 2, mod);
}

ll factorial (ll n, ll m) {
    ll ret = 1;
    for (int i = 2; i <= n; i++)
        ret = (ret * i) % m;
    return ret;
}

// precondition: p is prime
ll divMod (ll i, ll j, ll p) {
    return i * modpow(j, p - 2, p) % p;
}

// precondition: p is prime; O(log P) if you precompute factorials
ll fastChoose (ll n, ll k, ll p) {
    return divMod(divMod(factorial(n, p), factorial(k, p), p), factorial(n -
        k, p), p);
}

// number of partitions of n
ll partitions (ll n, ll m) {
    ll dp[n + 1];
    memset(dp, 0, sizeof dp);
    dp[0] = 1;
    for (int i = 1; i <= n; i++)
        for (int j = i; j <= n; j++)
            dp[j] = (dp[j] + dp[j - 1]) % m;
    return dp[n] % m;
}

ll stirling1 (int n, int k, long m) {
    ll dp[n + 1][k + 1];
    memset(dp, 0, sizeof dp);
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= k; j++) {
            dp[i][j] = ((i - 1) * dp[i - 1][j]) % m;
            dp[i][j] = (dp[i][j] + dp[i - 1][j - 1]) % m;
        }
    return dp[n][k];
}

ll stirling2 (int n, int k, ll m) {
    ll dp[n + 1][k + 1];
    memset(dp, 0, sizeof dp);
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= k; j++) {
            dp[i][j] = (j * dp[i - 1][j]) % m;
            dp[i][j] = (dp[i][j] + dp[i - 1][j - 1]) % m;
        }
    return dp[n][k];
}

ll eulerian1 (int n, int k, ll m) {
    if (k > n - 1 - k)
        k = n - 1 - k;
    ll dp[n + 1][k + 1];
    memset(dp, 0, sizeof dp);
    for (int j = 1; j <= k; j++)
        dp[0][j] = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= k; j++) {
            dp[i][j] = ((i - j) * dp[i - 1][j - 1]) % m;
            dp[i][j] = (dp[i][j] + ((j + 1) * dp[i - 1][j]) % m) % m;
        }
    return dp[n][k] % m;
}

ll eulerian2 (int n, int k, ll m) {
    ll dp[n + 1][k + 1];
    memset(dp, 0, sizeof dp);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= k; j++) {
            if (i == j) {
                dp[i][j] = 0;
            } else {
                dp[i][j] = ((j + 1) % dp[i - 1][j]) % m;
                dp[i][j] = (((2 * i - 1 - j) * dp[i - 1][j - 1]) % m + dp[i][
```

```
                    j]) % m;
            }
        }
    return dp[n][k] % m;
}

// precondition: p is prime
ll catalan (int n, ll p) {
    return fastChoose(2 * n, n, p) * modpow(n + 1, p - 2, p) % p;
}
```

5.3 GaussJordon.cpp

```
/*
 * 1) Solving system of linear equations (AX=B), stored in B
 * 2) Inverting matrices (AX=I), stored in A
 * 3) Computing determinants of square matrices, returned as T
 */

#include <bits/stdc++.h>
#define EPS 1e-10
using namespace std;
typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
T GaussJordan(VVT &a, VVT &b) {
    const int n = a.size();
    const int m = b[0].size();
    VI irow(n), icol(n), ipiv(n);
    T det = 1;

    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk
                    = k; }
        if (fabs(a[pj][pk]) < EPS)
            return 0;
        ipiv[pk]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        if (pj != pk) det *= -1;
        irow[i] = pj;
        icol[i] = pk;

        T c = 1.0 / a[pk][pk];
        det *= a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
        }
    }

    for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
        for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
    }
    return det;
}
```