Contents

1 Template

1.1 Makefile

```
BASIC := -std=c++11 -Wall -Wextra -Wshadow -g -DLOCAL
VERBOSE := -fsanitize=address -fsanitize=undefined -D_GLIBCXX_DEBUG

main: main.cc
	g++ $(BASIC) $(VERBOSE) $< -o $@
```

1.2 vimrc

```
filetype plugin indent on
set nu rnu
set ai ts=4 shiftwidth=4 sts=4 et
set spr sb
set clipboard=unnamed,unnamedplus
```

2 Graph

2.1 Dijkstra

```
#include <bits/stdc++.h>

using namespace std;

struct Edge {
    int u, v, w;
    Edge(int u_=-1, int v_=-1, int w_=-1) : u(u_), v(v_), w(w_) {}
};

struct Node {
    int u;
    int64_t d;
    Node(int u_, int64_t d_) : u(u_), d(d_) {}
    bool operator<(const Node& o) const {
        return d > o.d; // min-heap
    }
};

struct Graph {
    const int64_t inf = 1e18;
    int n;
    vector<vector<Edge>> adj;
    vector<int64_t> dist;
    vector<Edge> trace; // trace[u]: last edge to get to u from s

    Graph(int n_) : n(n_), adj(n), dist(n, inf),
        trace(n) {}

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

    int64_t dijkstra(int s, int t) {
        priority_queue<Node> pq;
        pq.emplace(s, 0);
        dist[s] = 0;

        while (!pq.empty()) {
            Node cur = pq.top(); pq.pop();
            int u = cur.u;
            int64_t d = cur.d;

            if (u == t) return dist[t];
            if (d > dist[u]) continue;

            for (const Edge& e : adj[u]) {
                int v = e.v;
                int w = e.w;
                if (dist[u] + w < dist[v]) {
```

```
                    dist[v] = dist[u] + w;
                    trace[v] = e;
                    pq.emplace(v, dist[v]);
                }
            }
        }

        return inf;
    }

    vector<Edge> getShortestPath(int s, int t) {
        assert(dist[t] != inf);
        vector<Edge> path;
        int v = t;
        while (v != s) {
            Edge e = trace[v];
            path.push_back(e);
            v = e.u;
        }
        reverse(path.begin(), path.end());
        return path;
    }
};


int main() {
    int n, m, s, t;
    cin >> n >> m >> s >> t;

    Graph g(n);

    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        g.addEdge(u, v, w);
    }

    int64_t dist = g.dijkstra(s, t);

    if (dist != g.inf) {
        vector<Edge> path = g.getShortestPath(s, t);
        cout << dist << ' ' << path.size() << '\n';
        for (Edge e : path) cout << e.u << ' ' << e.v << '\n';
    } else {
        cout << "-1\n";
    }

    return 0;
}
```

3 Maths

3.1 Modular Arithmetic

```
// **Really important note**: inputs of the modAdd, modSub, and modMul
// functions must all be normalized (within the range [0..mod - 1]) before use

#pragma once

#include <bits/stdc++.h>

using namespace std;

int modAdd(int a, int b, int mod) {
    a += b;
    if (a >= mod) a -= mod;
    return a;
}

int modSub(int a, int b, int mod) {
    a -= b;
    if (a < 0) a += mod;
    return a;
}

int modMul(int a, int b, int mod) {
    int64_t res = (int64_t) a * b;
    return (int) (res % mod);
}

int64_t binPow(int64_t a, int64_t x) {
    int64_t res = 1;
    while (x) {
        if (x & 1) res *= a;
        a *= a;
        x >>= 1;
    }
    return res;
}

int64_t modPow(int64_t a, int64_t x, int mod) {
    int res = 1;
    while (x) {
```

```
        if (x & 1) res = modMul(res, a, mod);
        a = modMul(a, a, mod);
        x >>= 1;
    }
    return res;
}
```

3.2 Modnum

```
#pragma once

#include <bits/stdc++.h>
#include "mod.hpp"

using namespace std;

template <typename T, int md>
struct Modnum {
    using M = Modnum;
    T v;
    Modnum(T _v=0) : v(fix(_v)) {}

    T fix(int64_t x) {
        if (x < -md || x > 2 * md) x %= md;
        if (x >= md) x -= md;
        if (x < 0) x += md;
        return x;
    }

    M operator+(M o) { return M(v + o.v); }
    M operator-(M o) { return M(v - o.v); }
    M operator*(M o) { return M(fix((int64_t) v * o.v)); }
    M operator/(M o) {
        return *this * modInv(o.v, md);
    }
    M pow(int64_t x) {
        M a(v);
        M res(1);
        while (x) {
            if (x & 1) res = res * a;
            a = a * a;
            x >>= 1;
        }
        return res;
    }
    friend istream& operator>>(istream& is, M& o) {
        is >> o.v; o.v = o.fix(o.v); return is;
    }
    friend ostream& operator<<(ostream& os, const M& o) {
        return os << o.v;
    }
};
```

3.3 Sieve of Eratosthenes

```
#include <bits/stdc++.h>

using namespace std;

/// Sieve of Eratosthenes
/// Benchmark: 3314 ms/188.74 Mib for N = 5 * 1e8
/// Credit: KTH's notebook
constexpr int MAX_N = (int) 5 * 1e8;
bitset<MAX_N + 1> is_prime;
vector<int> primes;

void sieve(int N) {
    is_prime.set();
    is_prime[0] = is_prime[1] = 0;

    for (int i = 4; i <= N; i += 2) is_prime[i] = 0;

    for (int i = 3; i * i <= N; i += 2) {
        if (!is_prime[i]) continue;
        for (int j = i * i; j <= N; j += i * 2) {
            is_prime[j] = 0;
        }
    }

    for (int i = 2; i <= N; i++) {
        if (is_prime[i]) primes.push_back(i);
    }
}

// https://judge.yosupo.jp/problem/enumerate_primes
int main() {
    int N, a, b;
    cin >> N >> a >> b;
    sieve(N);
    int num_primes = primes.size();
    vector<int> res;

```

```
    for (int j = 0; a * j + b < num_primes; j++) {
        res.push_back(primes[a * j + b]);
    }

    cout << num_primes << ' ' << res.size() << '\n';

    for (int p : res) {
        cout << p << ' ';
    }
    cout << '\n';
}
```

3.4 Primality Test

```
// Simple primality test

#pragma once

#include <bits/stdc++.h>

template <typename T>
bool isPrime(T x) {
    for (T d = 2; d * d <= x; d++) {
        if (x % d == 0) return false;
    }
    return true;
}
```

3.5 Euclidean Algorithm

```
#pragma once

#include <bits/stdc++.h>

using namespace std;

template <typename T>
T gcd(T a, T b) {
    if (a < b) swap(a, b);
    while (b != 0) {
        int r = a % b;
        a = b;
        b = r;
    }
    return a;
}

template <typename T>
int64_t lcm(T a, T b) {
    return (int64_t) a / gcd(a, b) * b;
}
```

3.6 Extended Euclidean Algorithm

```
#pragma once

#include "mod.hpp"

// This solves the equation ax + by = gcd(a, b)
// Input: a, b
// Output: g (returned), x, y (passed by ref)
int64_t extGcd(int64_t a, int64_t b, int64_t& x, int64_t& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int64_t x1, y1;
    int64_t g = extGcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    assert(g == 1);
    return g;
}
```

4 Strings

4.1 Trie

```
#pragma once

#include <bits/stdc++.h>

using namespace std;

struct Trie {
    const int ALPHA = 26;
    vector<vector<int>> trie;
    vector<int> eow;
```

```
    int ord(char c) { return c - 'a'; }

    Trie() {
        trie.emplace_back(ALPHA, -1);
        eow.push_back(0);
    }

    void add(const string& word) {
        int node = 0;

        for (char c : word) {
            int x = ord(c);

            if (trie[node][x] == -1) {
                trie[node][x] = trie.size();
                trie.emplace_back(ALPHA, -1);
                eow.push_back(0);
            }

            node = trie[node][x];
            eow[node]++;
        }
    }
};
```

5 Geometry

5.1 Utility

```
#pragma once

#include <bits/stdc++.h>

using namespace std;

const double PI = acos(-1);

template <typename T>
int sgn(T x) {
    if (x > 0) return 1;
    if (x < 0) return -1;
    return 0;
}

int inc(int i, int n, int by=1) {
    i += by;
    if (i >= n) i -= n;
    return i;
}

double degToRad(double d) {
    return d * PI / 180.0;
}

double radToDeg(double r) {
    return r * 180.0 / PI;
}
```

5.2 Point

```
#pragma once

#include <bits/stdc++.h>
#include "geoutil.hpp"

using namespace std;


template<typename T>
struct Point {
    using P = Point;
    T x, y;

    Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
    P operator+(const P &o) const { return P(x + o.x, y + o.y); }
    P operator-(const P &o) const { return P(x - o.x, y - o.y); }
    P operator*(T d) const { return P(x * d, y * d); }
    P operator/(T d) const { return P(x / d, y / d); }
    T dot(P o) const { return x * o.x + y * o.y; }
    T cross(P o) const { return x * o.y - y * o.x; }
    T abs2() const { return x * x + y * y; }
    long double abs() const { return sqrt((long double) abs2()); }
    double angle() const { return atan2(y, x); } // [-π, π]
    P unit() const { return *this / abs(); } // makes abs()=1
    P perp() const { return P(-y, x); } // rotates +π/2

    P rotate(double a) const { // ccw
        return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
    }

    friend istream &operator>>(istream &is, P &p) {
```

```
        return is >> p.x >> p.y;
    }

    friend ostream &operator<<(ostream &os, P &p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }

    // position of c relative to a->b
    // > 0: c is on the left of a->b
    friend T orient(P a, P b, P c) {
        return (b - a).cross(c - a);
    }

    // Check if u and v are parallel
    // (u = cv) where c ∈ R)
    friend bool parallel(P u, P v) {
        return u.cross(v) == 0;
    }

    // Check if point p lies on the segment ab
    friend bool onSegment(P a, P b, P p) {
        return orient(a, b, p) == 0 &&
               min(a.x, b.x) <= p.x &&
               max(a.x, b.x) >= p.x &&
               min(a.y, b.y) <= p.y &&
               max(a.y, b.y) >= p.y;
    }

    friend bool boundingBox(P p1, P q1, P p2, P q2) {
        if (max(p1.x, q1.x) < min(p2.x, q2.x)) return true;
        if (max(p1.y, q1.y) < min(p2.y, q2.y)) return true;
        if (max(p2.x, q2.x) < min(p1.x, q1.x)) return true;
        if (max(p2.x, q2.x) < min(p1.x, q1.x)) return true;
        return false;
    }

    friend bool intersect(P p1, P p2, P p3, P p4) {
        // Check if two segments are parallel
        if (parallel(p2 - p1, p4 - p3)) {
            // Check if 4 ps are colinear
            if (!parallel(p2 - p1, p3 - p1)) return false;
            if (boundingBox(p1, p2, p3, p4)) return false;
            return true;
        }

        // check if one line is completely on one side of the other
        for (int i = 0; i < 2; i++) {
            if (sgn(orient(p1, p2, p3)) == sgn(orient(p1, p2, p4))
                && sgn(orient(p1, p2, p3)) != 0) {
                return false;
            }
            swap(p1, p3);
            swap(p2, p4);
        }
        return true;
    }

    // Check if p is in ∠bac (including the rays)
    friend bool inAngle(P a, P b, P c, P p) {
        assert(orient(a, b, c) != 0);
        if (orient(a, b, c) < 0) swap(b, c);
        return orient(a, b, p) >= 0 && orient(a, c, p) <= 0;
    }

    // Angle ∠bac (+/-)
    friend double directedAngle(P a, P b, P c) {
        if (orient(a, b, c) >= 0) {
            return (b - a).angle(c - a);
        }
        return 2 * PI - (b - a).angle(c - a);
    }
};
```

5.3 Polygon

```
#pragma once

#include <bits/stdc++.h>
#include "point.hpp"
#include "geoutil.hpp"
#include "../maths/euclidean.hpp"

using namespace std;

template <typename T>
struct Polygon {
    using P = Point<T>;

    int n = 0;
    vector<P> ps;
    Polygon() : n(0) {}
    Polygon(vector<P>& ps) : n(ps.size()), ps(ps) {}

```

```
    void add(P p) {
        ps.push_back(p);
        n++;
    }

    int64_t twiceArea() {
        int64_t area = 0;
        for (int i = 0; i < n; i++) {
            P p1 = ps[i];
            P p2 = ps[inc(i, n)];
            area += p1.cross(p2);
        }
        return abs(area);
    }

    double area() {
        return twiceArea() / 2.0;
    }

    int64_t boundaryLattice() {
        int64_t res = 0;
        for (int i = 0; i < n; i++) {
            int j = i + 1; if (j == n) j = 0;
            P p1 = ps[i];
            P p2 = ps[j];
            P v = p2 - p1;
            res += gcd(abs(v.x), abs(v.y));
        }
        return res;
    }

    int64_t interiorLattice() {
        return (twiceArea() - boundaryLattice()) / 2 + 1;
    }

    bool isConvex() {
        int pos = 0;
        int neg = 0;

        for (int i = 0; i < n; i++) {
            P p1 = ps[i];
            P p2 = ps[inc(i, n, 1)];
            P p3 = ps[inc(i, n, 2)];
            int o = orient(p1, p2, p3);
            if (o > 0) pos = 1;
            if (o < 1) neg = 1;
        }

        return pos ^ neg;
    }

    // -1: outside; 1: inside; 0: on boundary
    int vsPoint(P r) {
        int crossing = 0;
        for (int i = 0; i < n; i++) {
            P p1 = ps[i];
            P p2 = ps[inc(i, n)];
            if (onSegment(p1, p2, r)) {
                return 0;
            }
            if (((p2.y >= r.y) - (p1.y >= r.y)) * orient(r, p1, p2) > 0) {
                crossing++;
            }
        }
        if (crossing & 1) return 1;
        return -1;
    }
};

template <typename T>
Polygon<T> convexHull(vector<Point<T>> points) {
    using P = Point<T>;

    sort(points.begin(), points.end(),
         [](const P& p1, const P& p2) {
             if (p1.x == p2.x) return p1.y < p2.y;
             return p1.x < p2.x;
         });

    vector<P> hull;

    for (int step = 0; step < 2; step++) {
        int s = hull.size();
        for (const P& c : points) {
            while ((int) hull.size() - s >= 2) {
                P a = hull.end()[-2];
                P b = hull.end()[-1];
                // <= if points on the edges are accepted, < otherwise
                if (orient(a, b, c) <= 0) break;
                hull.pop_back();
            }
            hull.push_back(c);
        }
        hull.pop_back();
```

```
        reverse(points.begin(), points.end());
    }

    return Polygon<T>(hull);
}
```