

Contents

| | | |
|-----|------------------------------|---|
| 1 | Template | 1 |
| 1.1 | Makefile | 1 |
| 2 | Graph | 1 |
| 2.1 | Dijkstra | 1 |
| 3 | Maths | 2 |
| 3.1 | Modular Arithmetic | 2 |
| 3.2 | Modnum | 2 |
| 3.3 | Sieve of Eratosthenes | 3 |
| 3.4 | Primality Test | 3 |
| 3.5 | Euclidean Algorithm | 4 |
| 3.6 | Extended Euclidean Algorithm | 4 |
| 4 | Geometry | 4 |
| 4.1 | Points | 4 |

1 Template

1.1 Makefile

```
1 default:
2     g++ -std=c++11 -Wall -Wextra -Wshadow -fsanitize=address -fsanitize=undefined
    ↪ -DLOCAL -D_GLIBCXX_DEBUG -g main.cc -o main
```

2 Graph

2.1 Dijkstra

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct Edge {
6     int u, v, w;
7     Edge(int u_=-1, int v_=-1, int w_=-1) : u(u_), v(v_), w(w_) {}
8 };
9
10 struct Node {
11     int u;
12     int64_t d;
13     Node(int u_, int64_t d_) : u(u_), d(d_) {}
14     bool operator<(const Node& o) const {
15         return d > o.d; // min-heap
16     }
17 };
18
```

```
19 struct Graph {
20     const int64_t inf = 1e18;
21     int n;
22     vector<vector<Edge>> adj;
23     vector<int64_t> dist;
24     vector<Edge> trace; // trace[u]: last edge to get to u from s
25
26     Graph(int n_) : n(n_), adj(n), dist(n, inf),
27         trace(n) {}
28
29     void addEdge(int u, int v, int w) {
30         adj[u].emplace_back(u, v, w);
31     }
32
33     int64_t dijkstra(int s, int t) {
34         priority_queue<Node> pq;
35         pq.emplace(s, 0);
36         dist[s] = 0;
37
38         while (!pq.empty()) {
39             Node cur = pq.top(); pq.pop();
40             int u = cur.u;
41             int64_t d = cur.d;
42
43             if (u == t) return dist[t];
44             if (d > dist[u]) continue;
45
46             for (const Edge& e : adj[u]) {
47                 int v = e.v;
48                 int w = e.w;
49                 if (dist[u] + w < dist[v]) {
50                     dist[v] = dist[u] + w;
51                     trace[v] = e;
52                     pq.emplace(v, dist[v]);
53                 }
54             }
55         }
56
57         return inf;
58     }
59
60     vector<Edge> getShortestPath(int s, int t) {
61         assert(dist[t] != inf);
62         vector<Edge> path;
63         int v = t;
64         while (v != s) {
65             Edge e = trace[v];
```

```

66         path.push_back(e);
67         v = e.u;
68     }
69     reverse(path.begin(), path.end());
70     return path;
71 }
72 };
73
74
75 int main() {
76     int n, m, s, t;
77     cin >> n >> m >> s >> t;
78
79     Graph g(n);
80
81     for (int i = 0; i < m; i++) {
82         int u, v, w;
83         cin >> u >> v >> w;
84         g.addEdge(u, v, w);
85     }
86
87     int64_t dist = g.dijkstra(s, t);
88
89     if (dist != g.inf) {
90         vector<Edge> path = g.getShortestPath(s, t);
91         cout << dist << ' ' << path.size() << '\n';
92         for (Edge e : path) cout << e.u << ' ' << e.v << '\n';
93     } else {
94         cout << "-1\n";
95     }
96
97     return 0;
98 }

```

3 Maths

3.1 Modular Arithmetic

```

1  // **Really important note**: inputs of the modAdd, modSub, and modMul
2  // functions must all be normalized (within the range [0..mod - 1]) before use
3
4  #pragma once
5
6  #include <bits/stdc++.h>
7
8  using namespace std;

```

```

9
10 int modAdd(int a, int b, int mod) {
11     a += b;
12     if (a >= mod) a -= mod;
13     return a;
14 }
15
16 int modSub(int a, int b, int mod) {
17     a -= b;
18     if (a < 0) a += mod;
19     return a;
20 }
21
22 int modMul(int a, int b, int mod) {
23     int64_t res = (int64_t) a * b;
24     return (int) (res % mod);
25 }
26
27 int64_t binPow(int64_t a, int64_t x) {
28     int64_t res = 1;
29     while (x) {
30         if (x & 1) res *= a;
31         a *= a;
32         x >>= 1;
33     }
34     return res;
35 }
36
37 int64_t modPow(int64_t a, int64_t x, int mod) {
38     int res = 1;
39     while (x) {
40         if (x & 1) res = modMul(res, a, mod);
41         a = modMul(a, a, mod);
42         x >>= 1;
43     }
44     return res;
45 }

```

3.2 Modnum

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4  #include "mod.hpp"
5
6  using namespace std;

```

```

7
8  template <typename T, int md>
9  struct Modnum {
10     using M = Modnum;
11     T v;
12     Modnum(T _v=0) : v(fix(_v)) {}
13
14     T fix(int64_t x) {
15         if (x < -md || x > 2 * md) x %= md;
16         if (x >= md) x -= md;
17         if (x < 0) x += md;
18         return x;
19     }
20
21     M operator+(M o) { return M(v + o.v); }
22     M operator-(M o) { return M(v - o.v); }
23     M operator*(M o) { return M(fix((int64_t) v * o.v)); }
24     M operator/(M o) {
25         return *this * modInv(o.v, md);
26     }
27     M pow(int64_t x) {
28         M a(v);
29         M res(1);
30         while (x) {
31             if (x & 1) res = res * a;
32             a = a * a;
33             x >>= 1;
34         }
35         return res;
36     }
37     friend istream& operator>>(istream& is, M& o) {
38         is >> o.v; o.v = o.fix(o.v); return is;
39     }
40     friend ostream& operator<<(ostream& os, const M& o) {
41         return os << o.v;
42     }
43 };

```

3.3 Sieve of Eratosthenes

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6

```

```

7  /// Sieve of Eratosthenes
8  /// Benchmark: 3314 ms/188.74 Mib for N = 5 * 1e8
9  /// Credit: KTH's notebook
10
11 namespace eratosthenes {
12     constexpr int MAX_N = (int) 5 * 1e8;
13     bitset<MAX_N + 1> is_prime;
14     vector<int> primes;
15
16     void sieve(int N) {
17         is_prime.set();
18         is_prime[0] = is_prime[1] = 0;
19
20         for (int i = 4; i <= N; i += 2) is_prime[i] = 0;
21
22         for (int i = 3; i * i <= N; i += 2) {
23             if (!is_prime[i]) continue;
24             for (int j = i * i; j <= N; j += i * 2) {
25                 is_prime[j] = 0;
26             }
27         }
28
29         for (int i = 2; i <= N; i++) {
30             if (is_prime[i]) primes.push_back(i);
31         }
32     }
33 }

```

3.4 Primality Test

```

1  // Simple primality test
2
3  #pragma once
4
5  #include <bits/stdc++.h>
6
7  template <typename T>
8  bool isPrime(T x) {
9      for (T d = 2; d * d <= x; d++) {
10         if (x % d == 0) return false;
11     }
12     return true;
13 }

```

3.5 Euclidean Algorithm

```
1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  template <typename T>
8  T gcd(T a, T b) {
9      if (a < b) swap(a, b);
10     while (b != 0) {
11         int r = a % b;
12         a = b;
13         b = r;
14     }
15     return a;
16 }
17
18 template <typename T>
19 int64_t lcm(T a, T b) {
20     return (int64_t) a / gcd(a, b) * b;
21 }
```

3.6 Extended Euclidean Algorithm

```
1  #pragma once
2
3  #include "mod.hpp"
4
5  // This solves the equation  $ax + by = \gcd(a, b)$ 
6  // Input:  $a, b$ 
7  // Output:  $g$  (returned),  $x, y$  (passed by ref)
8  int64_t extGcd(int64_t a, int64_t b, int64_t& x, int64_t& y) {
9      if (b == 0) {
10         x = 1;
11         y = 0;
12         return a;
13     }
14     int64_t x1, y1;
15     int64_t g = extGcd(b, a % b, x1, y1);
16     x = y1;
17     y = x1 - y1 * (a / b);
18     assert(g == 1);
19     return g;
20 }
```

4 Geometry

4.1 Points

```
1  #pragma once
2
3  #include <bits/stdc++.h>
4  #include "geoutil.hpp"
5
6  using namespace std;
7
8
9  template<typename T>
10 struct Point {
11     using P = Point;
12     T x, y;
13
14     Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
15
16     P operator+(const P &o) const { return P(x + o.x, y + o.y); }
17
18     P operator-(const P &o) const { return P(x - o.x, y - o.y); }
19
20     P operator*(T d) const { return P(x * d, y * d); }
21
22     P operator/(T d) const { return P(x / d, y / d); }
23
24     T dot(P o) const { return x * o.x + y * o.y; }
25
26     T cross(P o) const { return x * o.y - y * o.x; }
27
28     T abs2() const { return x * x + y * y; }
29
30     long double abs() const { return sqrt((long double) abs2()); }
31
32     double angle() const { return atan2(y, x); } //  $[-\pi, \pi]$ 
33     P unit() const { return *this / abs(); } // makes  $abs()=1$ 
34     P perp() const { return P(-y, x); } // rotates  $+\pi/2$ 
35
36     P rotate(double a) const { // ccw
37         return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
38     }
39
40     friend istream &operator>>(istream &is, P &p) {
```

```

41     return is >> p.x >> p.y;
42 }
43
44 friend ostream &operator<<(ostream &os, P &p) {
45     return os << "(" << p.x << ", " << p.y << ")";
46 }
47
48 // position of c relative to a->b
49 // > 0: c is on the left of a->b
50 friend T orient(P a, P b, P c) {
51     return (b - a).cross(c - a);
52 }
53
54 // Check if  $\vec{u}$  and  $\vec{v}$  are parallel
55 // ( $\vec{u} = c\vec{v}$ ) where  $c \in R$ 
56 friend bool parallel(P u, P v) {
57     return u.cross(v) == 0;
58 }
59
60 // Check if point p lies on the segment ab
61 friend bool onSegment(P a, P b, P p) {
62     return orient(a, b, p) == 0 &&
63         min(a.x, b.x) <= p.x &&
64         max(a.x, b.x) >= p.x &&
65         min(a.y, b.y) <= p.y &&
66         max(a.y, b.y) >= p.y;
67 }
68
69 friend bool boundingBox(P p1, P q1, P p2, P q2) {
70     if (max(p1.x, q1.x) < min(p2.x, q2.x)) return true;
71     if (max(p1.y, q1.y) < min(p2.y, q2.y)) return true;
72     if (max(p2.x, q2.x) < min(p1.x, q1.x)) return true;
73     if (max(p2.y, q2.y) < min(p1.y, q1.y)) return true;
74     return false;
75 }
76
77 friend bool intersect(P p1, P p2, P p3, P p4) {
78     // Check if two segments are parallel
79     if (parallel(p2 - p1, p4 - p3)) {
80         // Check if 4 ps are colinear
81         if (!parallel(p2 - p1, p3 - p1)) return false;
82         if (boundingBox(p1, p2, p3, p4)) return false;
83         return true;
84     }
85
86     // check if one line is completely on one side of the other
87     for (int i = 0; i < 2; i++) {

```

```

88         if (sgn((p2 - p1).cross(p3 - p1)) == sgn((p2 - p1).cross(p4 - p1))
89             && sgn((p2 - p1).cross(p3 - p1)) != 0) {
90             return false;
91         }
92         swap(p1, p3);
93         swap(p2, p4);
94     }
95     return true;
96 }
97
98 // Check if p is in  $\angle bac$  (including the rays)
99 friend bool inAngle(P a, P b, P c, P p) {
100     assert(orient(a, b, c) != 0);
101     if (orient(a, b, c) < 0) swap(b, c);
102     return orient(a, b, p) >= 0 && orient(a, c, p) <= 0;
103 }
104
105 // Angle  $\angle bac$  (+/-)
106 friend double directedAngle(P a, P b, P c) {
107     if (orient(a, b, c) >= 0) {
108         return (b - a).angle(c - a);
109     }
110     return 2 * PI - (b - a).angle(c - a);
111 }
112 };

```
