

## Contents

<b>1</b>	<b>Template</b>	<b>1</b>
1.1	Makefile	1
1.2	vimrc	1
<b>2</b>	<b>Graph</b>	<b>1</b>
2.1	Dijkstra	1
2.2	Strongly Connected Components	2
<b>3</b>	<b>Maths</b>	<b>2</b>
3.1	Modular Arithmetic	2
3.2	Modnum	3
3.3	Sieve of Eratosthenes	3
3.4	Primality Test	4
3.5	Euclidean Algorithm	4
3.6	Extended Euclidean Algorithm	4
3.7	Euler's Totient Function	4
3.8	Matrix	4
<b>4</b>	<b>Strings</b>	<b>5</b>
4.1	Trie	5
4.2	Z function	6
4.3	Suffix Array	6
<b>5</b>	<b>Flows</b>	<b>6</b>
5.1	Dinic Max Flow	6
<b>6</b>	<b>Matching</b>	<b>7</b>
6.1	Hopcroft-Karp Bipartite Matching	7
<b>7</b>	<b>Geometry</b>	<b>8</b>
7.1	Utility	8
7.2	Point	8
7.3	Polygon	9
<b>8</b>	<b>C++ STL</b>	<b>10</b>
8.1	vector	10
8.2	set	10
8.3	map	10
8.4	unordered_set and unordered_map	10
8.5	pair	10
8.6	string	11
8.7	Other useful utilities	11

## 1 Template

### 1.1 Makefile

```
1 BASIC := -std=c++11 -Wall -Wextra -Wshadow -g -DLOCAL
2 VERBOSE := -fsanitize=address -fsanitize=undefined
   ↳ -D_GLIBCXX_DEBUG
3
4 main: main.cc
5     g++ $(BASIC) $(VERBOSE) $< -o $@
```

### 1.2 vimrc

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

## 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 }

```

---

## 2.2 Strongly Connected Components

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  // https://judge.yosupo.jp/problem/scc
6  // Properties:
7  // - component graph is a DAG
8  // - traversed graph has the same sccs
9  // In this implementation, each component is sorted in
   ↳ topological order
10 struct Graph {
11     int n;
12     vector<vector<int>> adj;
13     vector<vector<int>> adj_t;
14     vector<int> mark;
15     vector<int> order;
16     vector<int> leader;
17     vector<vector<int>> components;
18
19     Graph(int n_) : n(n_), adj(n), adj_t(n),
20         mark(n), leader(n) {}

```

```

21
22     void addEdge(int u, int v) {
23         adj[u].push_back(v);
24         adj_t[v].push_back(u);
25     }
26
27     void dfsForward(int u) {
28         assert(mark[u] == 0);
29         mark[u] = 1;
30
31         for (int v : adj[u]) {
32             if (mark[v] == 0) {
33                 dfsForward(v);
34             }
35         }
36
37         order.push_back(u);
38     }
39
40     void dfsBackward(int u, int p) {
41         assert(mark[u] == 1);
42         mark[u] = 2;
43         leader[u] = p;
44
45         for (int v : adj_t[u]) {
46             if (mark[v] == 1) {
47                 dfsBackward(v, p);
48             }
49         }
50
51         components.back().push_back(u);
52     }
53
54     vector<vector<int>> scc() { // Kosaraju's algorithm
55         fill(mark.begin(), mark.end(), 0);
56         for (int u = 0; u < n; u++) {
57             if (mark[u] == 0) {
58                 dfsForward(u);
59             }
60         }
61
62         reverse(order.begin(), order.end());
63
64         for (int u : order) {
65             if (mark[u] == 1) {
66                 components.emplace_back();
67                 dfsBackward(u, u);

```

```

68         }
69     }
70
71     return components;
72 }
73 };
74
75 int main() {
76     int n, m;
77     cin >> n >> m;
78
79     Graph g(n);
80
81     for (int i = 0; i < m; i++) {
82         int u, v;
83         cin >> u >> v;
84         g.addEdge(u, v);
85     }
86
87     vector<vector<int>> components = g.scc();
88
89     cout << components.size() << '\n';
90
91     for (vector<int>& comp : components) {
92         cout << comp.size() << ' ';
93         for (int u : comp) {
94             cout << u << ' ';
95         }
96         cout << '\n';
97     }
98
99     return 0;
100 }

```

---

## 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  #include "mod_inverse.hpp"
6
7  using namespace std;
8
9  template <typename T, int md>
10 struct Modnum {
11     using M = Modnum;
12     T v;
13     Modnum(int64_t v_=0) : v(fix(v_)) {}
14
15     T fix(int64_t x) {
16         if (x < -md || x > 2 * md) x %= md;
17         if (x >= md) x -= md;
18         if (x < 0) x += md;
19         return x;
20     }
21
22     M operator-() { return M(-v); };
23     M operator+(M o) { return M(v + o.v); }
24     M operator-(M o) { return M(v - o.v); }
25     M operator*(M o) { return M(fix((int64_t) v * o.v));
↵ }
26     M operator/(M o) { return *this * modInv(o.v, md); }
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
38     friend istream& operator>>(istream& is, M& o) {
39         is >> o.v; o.v = o.fix(o.v); return is;
40     }
41
42     friend ostream& operator<<(ostream& os, const M& o) {
43         return os << o.v;
44     }
45
46     friend T abs(const M& m) { if (m.v < 0) return -m.v;
↵ return m.v; }
47 };

```

## 3.3 Sieve of Eratosthenes

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  /// Sieve of Eratosthenes
6  /// Benchmark: 3314 ms/188.74 Mib for N = 5 * 1e8
7  /// Credit: KTH's notebook
8  constexpr int MAX_N = (int) 5 * 1e8;
9  bitset<MAX_N + 1> is_prime;
10 vector<int> primes;
11
12 void sieve(int N) {
13     is_prime.set();
14     is_prime[0] = is_prime[1] = 0;
15
16     for (int i = 4; i <= N; i += 2) is_prime[i] = 0;
17
18     for (int i = 3; i * i <= N; i += 2) {
19         if (!is_prime[i]) continue;
20         for (int j = i * i; j <= N; j += i * 2) {
21             is_prime[j] = 0;
22         }
23     }
24
25     for (int i = 2; i <= N; i++) {
26         if (is_prime[i]) primes.push_back(i);
27     }
28 }
29
30 // https://judge.yosupo.jp/problem/enumerate_primes
31 int main() {
32     int N, a, b;
33     cin >> N >> a >> b;
34     sieve(N);
35     int num_primes = primes.size();
36     vector<int> res;
37
38     for (int j = 0; a * j + b < num_primes; j++) {
39         res.push_back(primes[a * j + b]);
40     }
41
42     cout << num_primes << ' ' << res.size() << '\n';
43
44     for (int p : res) {

```

```
45     cout << p << ' ' ;
46 }
47     cout << '\n';
48 }
```

### 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 }
```

### 3.7 Euler's Totient Function

```
1 #pragma once
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 // Euler's totient function
8 //  $\phi(i)$  = number of coprime numbers of n in the range
    ↪ [1..n]
9 // Multiplicative property:  $\phi(a * b) = \phi(a) * \phi(b)$ 
10 // Complexity:  $O(\sqrt{n})$ 
11 int eulerPhi(int n) {
12     int res = n;
13     for (int i = 2; i * i <= n; i++) {
14         if (n % i == 0) {
15             while (n % i == 0) {
16                 n /= i;
17             }
18             res -= res / i;
19         }
20     }
```

```
19     }
20 }
21 if (n > 1) {
22     res -= res / n;
23 }
24 return res;
25 }
26
27 // Complexity:  $O(n \log \log(n))$ 
28 vector<int> eulerPhiN(int n) {
29     vector<int> phi(n + 1);
30     phi[0] = 0;
31     phi[1] = 1;
32
33     for (int i = 2; i <= n; i++) phi[i] = i;
34
35     for (int i = 2; i <= n; i++) {
36         if (phi[i] == i) {
37             for (int j = i; j <= n; j += i) {
38                 phi[j] -= phi[j] / i;
39             }
40         }
41     }
42
43     return phi;
44 }
```

### 3.8 Matrix

```
1 #pragma once
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 template <typename T>
8 struct vec2d : public vector<vector<T>> {
9     vec2d(int n=0, int m=0, T t=T())
10         : vector<vector<T>>(n, vector<T>(m, t)) {}
11 };
12
13 template <typename T>
14 struct Matrix : vec2d<T> {
15     int n;
```

```

16 Matrix(int n_, T t=T()) : vec2d<T>(n_, n_, t), n(n_)
17 ↪ {}
18
19 Matrix operator+(const Matrix& o) const {
20     assert(n == o.n);
21     const Matrix& a = *this;
22     Matrix res(n);
23
24     for (int i = 0; i < n; i++) {
25         for (int j = 0; j < n; j++) {
26             res[i][j] = a[i][j] + o[i][j];
27         }
28     }
29
30     return res;
31 }
32
33 Matrix operator-(const Matrix& o) const {
34     assert(n == o.n);
35     const Matrix& a = *this;
36     Matrix res(n);
37
38     for (int i = 0; i < n; i++) {
39         for (int j = 0; j < n; j++) {
40             res[i][j] = a[i][j] - o[i][j];
41         }
42     }
43
44     return res;
45 }
46
47 Matrix operator*(const Matrix& o) const {
48     assert(n == o.n);
49     const Matrix& a = *this;
50     Matrix res(n, 0);
51
52     for (int i = 0; i < n; i++) {
53         for (int j = 0; j < n; j++) {
54             for (int k = 0; k < n; k++) {
55                 res[i][j] = res[i][j] + a[i][k] *
56                 ↪ o[k][j];
57             }
58         }
59     }
60     return res;

```

```

61
62 void identity() {
63     Matrix& a = *this;
64     for (int i = 0; i < n; i++) {
65         for (int j = 0; j < n; j++) {
66             if (i == j) a[i][j] = 1;
67             else a[i][j] = 0;
68         }
69     }
70 }
71
72 // Gauss method. Complexity:  $O(n^3)$ 
73 friend T determinant(const Matrix& mat) {
74     int n = mat.n;
75     Matrix a(n);
76
77     for (int i = 0; i < n; i++) {
78         for (int j = 0; j < n; j++) {
79             a[i][j] = mat[i][j];
80         }
81     }
82
83     const double EPS = 1E-9;
84     T det = 1;
85
86     for (int i = 0; i < n; ++i) {
87         int k = i;
88
89         for (int j = i + 1; j < n; j++) {
90             if (abs(a[j][i]) > abs(a[k][i])) {
91                 k = j;
92             }
93         }
94
95         if (abs(a[k][i]) < EPS) {
96             det = 0;
97             break;
98         }
99
100         swap(a[i], a[k]);
101
102         if (i != k) det = -det;
103
104         det = det * a[i][i];
105
106         for (int j = i + 1; j < n; j++) {
107             a[i][j] = a[i][j] / a[i][i];

```

```

108     }
109
110     for (int j = 0; j < n; j++) {
111         if (j != i && abs(a[j][i]) > EPS) {
112             for (int k = i + 1; k < n; k++) {
113                 a[j][k] = a[j][k] - a[i][k] *
114                 ↪ a[j][i];
115             }
116         }
117     }
118
119     return det;
120 }
121 };

```

## 4 Strings

### 4.1 Trie

```

1 #pragma once
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6
7 struct Trie {
8     const int ALPHA = 26;
9     vector<vector<int>>> trie;
10    vector<int> eow;
11
12    int ord(char c) { return c - 'a'; }
13
14    Trie() {
15        trie.emplace_back(ALPHA, -1);
16        eow.push_back(0);
17    }
18
19    void add(const string& word) {
20        int node = 0;
21
22        for (char c : word) {
23            int x = ord(c);
24
25            if (trie[node][x] == -1) {
26                trie[node][x] = trie.size();

```

```

27         trie.emplace_back(ALPHA, -1);
28         eow.push_back(0);
29     }
30
31     node = trie[node][x];
32     eow[node]++;
33 }
34 }
35 };

```

## 4.2 Z function

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  // z[i]: length of the longest common prefix between s
8  ↪ and
9  // its substring starting at i
10 vector<int> zFunction(const string& s) {
11     int n = s.length();
12     vector<int> z(n);
13     z[0] = n;
14     int l = 0;
15     int r = 0;
16
17     for (int i = 1; i < n; i++) {
18         if (i <= r) {
19             z[i] = min(z[i - l], r - i + 1);
20         }
21         while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
22             z[i]++;
23         }
24         if (i + z[i] - 1 > r) {
25             l = i;
26             r = i + z[i] - 1;
27         }
28     }
29     return z;
30 }

```

## 4.3 Suffix Array

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  // sa[i] = the starting index of the ith suffix (starting
6  ↪ at 0)
7  // sorted in lexicographic order
8  vector<int> suffix_array(const string& s_, int alpha=256)
9  ↪ {
10     string s = s_ + '\0';
11     int n = s.size();
12     vector<int> p(n);
13     vector<int> cnt(max(alpha, n), 0);
14
15     for (int i = 0; i < n; i++) cnt[s[i]]++;
16     for (int i = 1; i < alpha; i++) cnt[i] += cnt[i - 1];
17     for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
18
19     vector<int> g(n);
20     g[p[0]] = 0;
21
22     for (int i = 1; i < n; i++) {
23         g[p[i]] = g[p[i - 1]] + (s[p[i]] != s[p[i - 1]]);
24     }
25
26     vector<int> pn(n);
27     vector<int> gn(n);
28
29     for (int len = 1; len < n; len <= 1) {
30         for (int i = 0; i < n; i++) {
31             pn[i] = p[i] - len; // transfer the pos from
32             ↪ second to pair
33             if (pn[i] < 0) pn[i] += n; // cyclic
34         }
35
36         int num_groups = g[p[n - 1]] + 1;
37         fill(cnt.begin(), cnt.begin() + num_groups, 0);
38
39         // Radix sort
40         for (int i = 0; i < n; i++) cnt[g[pn[i]]]++;
41         for (int i = 1; i < num_groups; i++) cnt[i] +=
42             cnt[i - 1];
43         for (int i = n - 1; i >= 0; i--)
44             p[--cnt[g[pn[i]]]] = pn[i];
45         gn[p[0]] = 0;

```

```

41
42     for (int i = 1; i < n; i++) {
43         pair<int, int> prev, cur;
44         prev.first = g[p[i - 1]];
45         cur.first = g[p[i]];
46         prev.second = g[p[i - 1]] + len - (p[i - 1] +
47             ↪ len >= n ? n : 0);
48         cur.second = g[p[i]] + len - (p[i] + len >= n
49             ↪ ? n : 0);
50         gn[p[i]] = gn[p[i - 1]] + (cur != prev);
51     }
52     g.swap(gn);
53     p.erase(p.begin());
54     return p;
55 }

```

## 5 Flows

### 5.1 Dinic Max Flow

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  /// Dinic algorithm for max flow
8  /// This versionshould work on flow graph with float
9  ↪ capacities
10 /// Time complexity: O(|V|^2|E|)
11
12 template <typename T>
13 struct FlowEdge {
14     int u, v;
15     T c, f;
16
17     FlowEdge(int _u, int _v, T _c, T _f) :
18         u(_u), v(_v), c(_c), f(_f) {}
19 };
20
21 template <typename T>
22 struct Dinic {
23     static constexpr T inf = numeric_limits<T>::max();
24     static constexpr T eps = (T) 1e-9;
25     int n;

```

```

25 int s, t;
26 vector<vector<int>> adj; // stores indices of edges
27 vector<int> level;      // shortest distance from
    ↪ source
28 vector<int> ptr;        // points to the next edge
    ↪ which can be used
29 vector<FlowEdge<T>> edges;
30
31 Dinic(int _n, int _s, int _t)
32     : n(_n), s(_s), t(_t), adj(_n), level(_n),
    ↪ ptr(_n) {}
33
34 void addEdge(int u, int v, int c, int rc=0) {
35     int eid = (int) edges.size();
36     adj[u].push_back(eid);
37     adj[v].push_back(eid + 1);
38     edges.emplace_back(u, v, c, 0);
39     edges.emplace_back(v, u, rc, 0);
40 }
41
42 bool bfs() {
43     fill(level.begin(), level.end(), -1);
44     level[s] = 0;
45     queue<int> q;
46     q.push(s);
47
48     while (!q.empty()) {
49         int u = q.front();
50         q.pop();
51
52         for (int eid : adj[u]) {
53             const auto& e = edges[eid];
54             if (e.c - e.f <= eps || level[e.v] != -1)
    ↪ continue;
55             level[e.v] = level[u] + 1;
56             q.push(e.v);
57         }
58     }
59
60     return level[t] != -1;
61 }
62
63 T dfs(int u, T flow) {
64     if (u == t) return flow;
65
66     for (int& j = ptr[u]; j < (int) adj[u].size();
    ↪ j++) {

```

```

67         int eid = adj[u][j];
68         const auto& e = edges[eid];
69         if (e.c - e.f > eps && level[e.v] == level[u]
    ↪ + 1) {
70             T df = dfs(e.v, min(e.c - e.f, flow));
71             if (df > eps) {
72                 edges[eid].f += df;
73                 edges[eid ^ 1].f -= df;
74                 return df;
75             }
76         }
77     }
78
79     return 0;
80 }
81
82 T maxFlow() {
83     T f = 0;
84
85     while (bfs()) {
86         fill(ptr.begin(), ptr.end(), 0);
87         T total_df = 0;
88         while (true) {
89             T df = dfs(s, inf);
90             if (df <= eps) break;
91             total_df += df;
92         }
93         if (total_df <= eps) break;
94         f += total_df;
95     }
96
97     return f;
98 }
99 };

```

## 6 Matching

### 6.1 Hopcroft-Karp Bipartite Matching

```

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

```

```

8
9  // Bipartite matching. Vertices from both halves start
    ↪ from 0
10 // Time complexity:  $O(\sqrt{|V|}|E|)$ 
11 struct HopcroftKarp {
12     const int INF = (int) 1e9;
13     int nu;
14     int nv;
15     vector<vector<int>> adj;
16     vector<int> layer;
17     vector<int> u_mate;
18     vector<int> v_mate;
19
20     HopcroftKarp(int nu, int nv) : nu(nu), nv(nv) {
21         adj.resize(nu);
22         layer.resize(nu);
23         u_mate.resize(nu, -1);
24         v_mate.resize(nv, -1);
25     }
26
27     void addEdge(int u, int v) {
28         adj[u].push_back(v);
29     }
30
31     bool bfs() {
32         // Find all possible augmenting paths
33         queue<int> q;
34
35         for (int u = 0; u < nu; u++) {
36             // Consider only unmatched edges
37             if (u_mate[u] == -1) {
38                 layer[u] = 0;
39                 q.push(u);
40             } else {
41                 layer[u] = INF;
42             }
43         }
44
45         bool has_path = false;
46
47         while (!q.empty()) {
48             int u = q.front();
49             q.pop();
50
51             for (int &v : adj[u]) {
52                 if (v_mate[v] == -1) {
53                     has_path = true;

```

```

54         } else if (layer[v_mate[v]] == INF) {
55             layer[v_mate[v]] = layer[u] + 1;
56             q.push(v_mate[v]);
57         }
58     }
59 }
60
61 return has_path;
62 }
63
64 bool dfs(int u) {
65     if (layer[u] == INF) return false;
66
67     for (int v : adj[u]) {
68         if ((v_mate[v] == -1) ||
69             (layer[v_mate[v]] == layer[u] + 1 &&
70              ↪ dfs(v_mate[v]))) {
71             v_mate[v] = u;
72             u_mate[u] = v;
73             return true;
74         }
75     }
76
77     return false;
78 }
79
80 vector<pair<int, int>> maxMatching() {
81     int matching = 0;
82
83     while (bfs()) { // there is at least 1 augmenting
84         ↪ path
85         for (int u = 0; u < nu; u++) {
86             if (u_mate[u] == -1 && dfs(u)) {
87                 ++matching;
88             }
89         }
90     }
91
92     vector<pair<int, int>> res;
93
94     for (int u = 0; u < nu; u++) {
95         if (u_mate[u] == -1) continue;
96         res.emplace_back(u, u_mate[u]);
97     }
98     assert(res.size() == matching);
99     return res;
100 }

```

```

99 };
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

## 7 Geometry

### 7.1 Utility

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  const double PI = acos(-1);
8
9  template <typename T>
10 int sgn(T x) {
11     if (x > 0) return 1;
12     if (x < 0) return -1;
13     return 0;
14 }
15
16 int inc(int i, int n, int by=1) {
17     i += by;
18     if (i >= n) i -= n;
19     return i;
20 }
21
22 double degToRad(double d) {
23     return d * PI / 180.0;
24 }
25
26 double radToDeg(double r) {
27     return r * 180.0 / PI;
28 }

```

### 7.2 Point

```

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     P operator+(const P &o) const { return P(x + o.x, y +
16         ↪ o.y); }
17     P operator-(const P &o) const { return P(x - o.x, y -
18         ↪ o.y); }
19     P operator*(T d) const { return P(x * d, y * d); }
20     P operator/(T d) const { return P(x / d, y / d); }
21     T dot(P o) const { return x * o.x + y * o.y; }
22     T cross(P o) const { return x * o.y - y * o.x; }
23     T abs2() const { return x * x + y * y; }
24     long double abs() const { return sqrt((long double)
25         ↪ abs2()); }
26     double angle() const { return atan2(y, x); } //
27         ↪  $[-\pi, \pi]$ 
28     P unit() const { return *this / abs(); } // makes
29         ↪  $abs()=1$ 
30     P perp() const { return P(-y, x); } // rotates  $+\pi/2$ 
31
32     P rotate(double a) const { // ccw
33         ↪ return P(x * cos(a) - y * sin(a), x * sin(a) + y
34         ↪ * cos(a));
35     }
36
37     friend istream &operator>>(istream &is, P &p) {
38         return is >> p.x >> p.y;
39     }
40
41     friend ostream &operator<<(ostream &os, P &p) {
42         return os << "(" << p.x << ", " << p.y << ")";
43     }
44
45     // position of c relative to a->b
46     // > 0: c is on the left of a->b
47     friend T orient(P a, P b, P c) {
48         return (b - a).cross(c - a);
49     }
50
51     // Check if  $\vec{u}$  and  $\vec{v}$  are parallel
52     // ( $\vec{u} = c\vec{v}$ ) where  $c \in \mathbb{R}$ )
53     friend bool parallel(P u, P v) {
54         return u.cross(v) == 0;
55     }
56 }

```



```

49     }
50
51     // Check if point p lies on the segment ab
52     friend bool onSegment(P a, P b, P p) {
53         return orient(a, b, p) == 0 &&
54             min(a.x, b.x) <= p.x &&
55             max(a.x, b.x) >= p.x &&
56             min(a.y, b.y) <= p.y &&
57             max(a.y, b.y) >= p.y;
58     }
59
60     friend bool boundingBox(P p1, P q1, P p2, P q2) {
61         if (max(p1.x, q1.x) < min(p2.x, q2.x)) return
62             ↪ true;
63         if (max(p1.y, q1.y) < min(p2.y, q2.y)) return
64             ↪ true;
65         if (max(p2.x, q2.x) < min(p1.x, q1.x)) return
66             ↪ true;
67         if (max(p2.y, q2.y) < min(p1.y, q1.y)) return
68             ↪ true;
69         return false;
70     }
71
72     friend bool intersect(P p1, P p2, P p3, P p4) {
73         // Check if two segments are parallel
74         if (parallel(p2 - p1, p4 - p3)) {
75             // Check if 4 ps are colinear
76             if (!parallel(p2 - p1, p3 - p1)) return
77                 ↪ false;
78             if (boundingBox(p1, p2, p3, p4)) return
79                 ↪ false;
80             return true;
81         }
82
83         // check if one line is completely on one side of
84         ↪ the other
85         for (int i = 0; i < 2; i++) {
86             if (sgn(orient(p1, p2, p3)) == sgn(orient(p1,
87                 ↪ p2, p4))
88                 && sgn(orient(p1, p2, p3)) != 0) {
89                 return false;
90             }
91             swap(p1, p3);
92             swap(p2, p4);
93         }
94         return true;
95     }
96 }

```

```

88
89     // Check if p is in Lbac (including the rays)
90     friend bool inAngle(P a, P b, P c, P p) {
91         assert(orient(a, b, c) != 0);
92         if (orient(a, b, c) < 0) swap(b, c);
93         return orient(a, b, p) >= 0 && orient(a, c, p) <=
94             ↪ 0;
95     }
96
97     // Angle Lbac (+/-)
98     friend double directedAngle(P a, P b, P c) {
99         if (orient(a, b, c) >= 0) {
100             return (b - a).angle(c - a);
101         }
102         return 2 * PI - (b - a).angle(c - a);
103     }

```

## 7.3 Polygon

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4  #include "point.hpp"
5  #include "geoutil.hpp"
6  #include "../maths/euclidean.hpp"
7
8  using namespace std;
9
10 template <typename T>
11 struct Polygon {
12     using P = Point<T>;
13
14     int n = 0;
15     vector<P> ps;
16     Polygon() : n(0) {}
17     Polygon(vector<P>& ps) : n(ps.size()), ps(ps) {}
18
19     void add(P p) {
20         ps.push_back(p);
21         n++;
22     }
23
24     int64_t twiceArea() {
25         int64_t area = 0;
26         for (int i = 0; i < n; i++) {

```

```

27             P p1 = ps[i];
28             P p2 = ps[inc(i, n)];
29             area += p1.cross(p2);
30         }
31         return abs(area);
32     }
33
34     double area() {
35         return twiceArea() / 2.0;
36     }
37
38     int64_t boundaryLattice() {
39         int64_t res = 0;
40         for (int i = 0; i < n; i++) {
41             int j = i + 1; if (j == n) j = 0;
42             P p1 = ps[i];
43             P p2 = ps[j];
44             P v = p2 - p1;
45             res += gcd(abs(v.x), abs(v.y));
46         }
47         return res;
48     }
49
50     int64_t interiorLattice() {
51         return (twiceArea() - boundaryLattice()) / 2 + 1;
52     }
53
54     bool isConvex() {
55         int pos = 0;
56         int neg = 0;
57
58         for (int i = 0; i < n; i++) {
59             P p1 = ps[i];
60             P p2 = ps[inc(i, n, 1)];
61             P p3 = ps[inc(i, n, 2)];
62             int o = orient(p1, p2, p3);
63             if (o > 0) pos = 1;
64             if (o < 1) neg = 1;
65         }
66
67         return pos ^ neg;
68     }
69
70     // -1: outside; 1: inside; 0: on boundary
71     int vsPoint(P r) {
72         int crossing = 0;

```

```
73     for (int i = 0; i < n; i++) {
74         P p1 = ps[i];
75         P p2 = ps[inc(i, n)];
76         if (onSegment(p1, p2, r)) {
77             return 0;
78         }
79         if (((p2.y >= r.y) - (p1.y >= r.y)) *
80             ↪ orient(r, p1, p2) > 0) {
81             crossing++;
82         }
83         if (crossing & 1) return 1;
84         return -1;
85     }
86 };
87
88 template <typename T>
89 Polygon<T> convexHull(vector<Point<T>> points) {
90     using P = Point<T>;
91
92     sort(points.begin(), points.end(),
93          [](const P& p1, const P& p2) {
94             if (p1.x == p2.x) return p1.y < p2.y;
95             return p1.x < p2.x;
96         });
97
98     vector<P> hull;
99
100    for (int step = 0; step < 2; step++) {
101        int s = hull.size();
102        for (const P& c : points) {
103            while ((int) hull.size() - s >= 2) {
104                P a = hull.end()[-2];
105                P b = hull.end()[-1];
106                // <= if points on the edges are
107                ↪ accepted, < otherwise
108                if (orient(a, b, c) <= 0) break;
109                hull.pop_back();
110            }
111            hull.push_back(c);
112        }
113        hull.pop_back();
114        reverse(points.begin(), points.end());
115    }
116
117    return Polygon<T>(hull);
118 }
```

8 C++ STL

8.1 vector

Underlying implementation: dynamic array

Method	Complexity
size_t size()	$O(1)$
void push_back(T v)	$O(1)$
void emplace_back(Args args...)	$O(1)$
void pop_back()	$O(1)$
T back()	$O(1)$
void erase(iterator position)	$O(n)$

- Resize (values in vector stay unchanged): v.resize(n)
- Resize and fill: v.assign(n, val)
- Fill: fill(v.begin(), v.end(), val)
- Reverse: reverse(v.begin(), v.end())
- Pythonic get element backwards:
  - v.end()[-1]: last element
  - v.end()[-2]: second-last element
- Sort():

```
1 // by default: non-decreasing, v must be of
2 ↪ comparator type
3 sort(v.begin(), v.end());
4 // custom comparator
5 sort(v.begin(), v.end(), [](const Obj& o1, const
6 ↪ Obj& o2) {
7     return o1.x < o2.x;
8 });
```

8.2 set

Condition: must be of a comparable type (define the < operator).  
Underlying implementation: self-balancing BST

Method	Complexity
size_t size()	$O(1)$
void insert(T v)	$O(1)$
void emplace(Args args...)	$O(1)$
iterator find(T v)	$O(\log(n))$
void erase(iterator position)	$O(\log(n))$

- Check if an element v is in set s: if (s.find(v) != s.end())

- Get minimum element: \*(m.begin())
- Get maximum element: \*(m.rbegin())

8.3 map

Condition: key must be of a comparable type (define the < operator).

Underlying implementation: self-balancing BST

Method	Complexity
size_t size()	$O(1)$
void insert(pair<K, V> keyvalpair)	$O(1)$
void emplace(K key, V value)	$O(1)$
iterator find(T v)	$O(\log(n))$
void erase(iterator position)	$O(\log(n))$

- Check if a key k is in map m: if (m.find(k) != m.end())
- Get value of key k in map m: m[k] or m.find(k)→second
- Get minimum key-value pair: \*(m.begin())
- Get key of minimum pair: m.begin()→first
- Get value of minimum pair: m.begin()→second
- Get maximum key-value pair: \*(m.rbegin())
- Get key of maximum pair: m.rbegin()→first
- Get value of maximum pair: m.rbegin()→second

8.4 unordered\_set and unordered\_map

Underlying implementation: hash table

Note: stay always from these unless you know what you are doing. There are scenarios where you think these can be faster than set and map, but either:

- The speed-up it will be negligible
- It will actually be unexpectedly slower

Operations: pretty much share the same interface with set and map, except for things that require order.

8.5 pair

Lexicographically comparable

## 8.6 string

- Mutable: `s[0] = 'a'` is OK.
- Concatenation:
  - `s += 'a'` takes  $O(1)!$
  - `s += t` takes  $O(\text{length}(t))$
- Substring:
  - `s.substr(i)` returns suffix starting from  $i$
  - `s.substr(i, 3)` returns suffix starting from  $i$  of maximum length 3 (can be shorter if reaches end)

## 8.7 Other useful utilities

`min(x, y)`, `max(x, y)`, `swap(x, y)`