

Contents

| | | |
|----------|-------------------------------|-----------|
| 1 | Base | 3 |
| 1.1 | vimrc | 3 |
| 1.2 | Template | 3 |
| 2 | Graph | 3 |
| 2.1 | LCA | 3 |
| 2.2 | SCC | 4 |
| 2.3 | Matching | 4 |
| 2.4 | Max Flow BFS | 5 |
| 2.5 | Max Flow Dinic | 5 |
| 2.6 | Min Cost Max Flow | 6 |
| 2.7 | Cut Vertex | 7 |
| 2.8 | Cut Edge | 8 |
| 2.9 | Bellman Ford | 8 |
| 2.10 | Dijkstra | 8 |
| 2.11 | Prim | 9 |
| 2.12 | DSU | 9 |
| 2.13 | Eulerian Tour | 9 |
| 2.14 | Topological Sort | 9 |
| 3 | Geometry | 10 |
| 3.1 | Geometry | 10 |
| 3.2 | Convex Hull | 13 |
| 4 | Data Structures | 14 |
| 4.1 | Fenwick1 | 14 |
| 4.2 | Fenwick2 | 14 |
| 4.3 | Segment Tree | 14 |
| 4.4 | Segment Tree Lazy Propagation | 15 |
| 4.5 | RMQ | 16 |
| 4.6 | Trie | 16 |
| 5 | String | 17 |
| 5.1 | Hash | 17 |
| 5.2 | KMP | 17 |
| 5.3 | Suffix Array | 17 |

| | | |
|----------|-----------------------------------|-----------|
| 6 | Number Theory | 18 |
| 6.1 | Phi | 18 |
| 6.2 | 370 SGU | 18 |
| 6.3 | Euclid | 18 |
| 6.4 | C(n, r) | 19 |
| 6.5 | FFT | 20 |
| 6.6 | Gauss Jordan | 20 |
| 7 | Other | 21 |
| 7.1 | Read Input | 21 |
| 7.2 | LIS | 22 |
| 7.3 | Divide and Conquer Tree | 22 |

1 Base

1.1 vimrc

```

1 set sw=4
2 set ts=4
3
4 set si          " super indentation
5 set number      " line numbers
6 syntax on       " syntax highlighting
7 set cursorline  " highlight current line
8
9 set bs=2
10
11 set mouse=a      " mouse works normally
12 set gdefault    " global replacement
13
14 set fdm=indent   " folding method
15 set foldlevelstart=99 " at first all folds are open

```

1.2 Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define pb push_back
5 #define mp make_pair
6 #define SQR(a) ((a) * (a))
7 #define SZ(x) ((int) (x).size())
8 #define ALL(x) (x).begin(), (x).end()
9 #define CLR(x, a) memset(x, a, sizeof x)
10 #define VAL(x) #x << " = " << (x) << " "
11 #define FOREACH(i, x) for(__typeof((x).begin()) i = (x).begin();
12     i != (x).end(); i++)
13 #define FOR(i, n) for (int i = 0; i < (n); i++)
14 #define X first
15 #define Y second
16
17 typedef long long ll;
18 typedef pair<ll, ll> pll;
19 typedef pair<int, int> pii;

```

```

20 const int MAXN = 1 * 1000 + 10;
21
22 int main () {
23     ios::sync_with_stdio(false);
24     return 0;
25 }

```

2 Graph

2.1 LCA

```

1 vector<int> adj[MAXN];
2 int par[MAXN][MAXL], h[MAXN];
3 bool mark[MAXN];
4
5 void dfs(int x) {
6     mark[x] = true;
7     for (int i = 0; i < SZ(adj[x]); i++) {
8         int v = adj[x][i];
9         if (!mark[v]) {
10             par[v][0] = x, h[v] = h[x] + 1;
11             dfs(v);
12         }
13     }
14 }
15
16 int get_parent(int x, int k) {
17     for (int i = 0; i < MAXL; i++)
18         if ((1 << i) & k) x = par[x][i];
19     return x;
20 }
21
22 int lca(int x, int y) {
23     if (h[y] > h[x]) swap(x, y);
24     x = get_parent(x, h[x] - h[y]);
25
26     if (x == y) return x;
27
28     for (int i = MAXL - 1; i >= 0; i--)
29         if (par[x][i] != par[y][i])
30             x = par[x][i], y = par[y][i];

```

```

31     return par[x][0];
32 }
33
34 int main () {
35     par[0][0] = -1;
36     dfs(0);
37     for (int i = 1; i < MAXL; i ++)
38         for (int j = 0; j < n; j ++)
39             par[j][i] = par[par[j][i - 1]][i - 1];
40 }

```

2.2 SCC

```

1 vector <int> adj[N];
2 stack <int> S, P;
3 int mrk[N], ind, col[N], CL;
4
5 void dfs(int v) {
6     mrk[v] = ++ind;
7     S.push(v);
8     P.push(v);
9     for(int i = 0; i < Size(adj[v]); ++i) {
10         int u = adj[v][i];
11         if(!mrk[u])
12             dfs(u);
13         else
14             while(mrk[u] < mrk[S.top()])
15                 S.pop();
16     }
17     if(S.top() == v) {
18         mrk[v] = INF;
19         col[v] = ++CL;
20         while(P.top() != v) {
21             col[P.top()] = CL;
22             mrk[P.top()] = INF;
23             P.pop();
24         }
25         P.pop();
26         S.pop();
27     }
28 }
29

```

```

30 //main: for(int i = 1; i <= n; ++i)
31 //     if(!mrk[i]) dfs(i);

```

2.3 Matching

```

1 int match[3][MAXN]; // 0 for first part, 1 for second part
2 bool mark[MAXN];
3 vector<int> adj[MAXN]; // adjacent list for first part nodes
4 int n, m, p;
5 // n: number of nodes in first part
6 // m: number of nodes in second part
7 // p: number of edges
8
9 bool dfs(int x) {
10     if (mark[x]) return false;
11
12     mark[x] = true;
13     for (int i = 0; i < SZ(adj[x]); i++) {
14         int v = adj[x][i];
15         if (match[1][v] == -1 || dfs(match[1][v])) {
16             match[0][x] = v;
17             match[1][v] = x;
18             return true;
19         }
20     }
21     return false;
22 }
23
24 void bi_match() {
25     CLR(match, -1);
26     for (int i = 0; i < n; i++) {
27         CLR(mark, 0);
28         bool check = false;
29         for (int j = 0; j < m; j++)
30             if (!mark[j] && match[0][j] == -1)
31                 check |= dfs(j);
32         if (!check) break;
33     }
34 }
35
36 int main () {
37     cin >> n >> m >> p;

```

```

38 for (int i = 0; i < p; i++) {
39     int x, y; cin >> x >> y; x--, y--;
40     // x: a node in first part [0, n)
41     // y: a node in second part [0, m)
42
43     adj[x].pb(y);
44 }
45
46 bi_match();
47 int ans = 0;
48 FOR(i, n) ans += (match[0][i] != -1);
49 cout << ans << endl;
50 return 0;
51 }

```

2.4 Max Flow BFS

```

1 #include <queue>
2 #include <cstring>
3
4 const int N = 100;
5 int mat[N][N];
6
7 int viz[N], network[N][N], parent[N];
8 bool anotherPath(int start, int end) {
9     memset(viz, 0, sizeof viz);
10    memset(parent, -1, sizeof parent);
11    viz[start] = true;
12    queue<int> q;
13    q.push(start);
14    while (!q.empty()) {
15        int z = q.front(); q.pop();
16        viz[z] = true;
17        for (int i=0; i<N; i++) {
18            if (network[z][i] <= 0 || viz[i]) continue;
19            viz[i] = true;
20            parent[i] = z;
21            if (i == end) return true;
22            q.push(i);
23        }
24    }
25    return false;

```

```

26 }
27 int maxflow(int start, int end) {
28     memcpy(network, mat, sizeof(mat));
29     int total = 0;
30     while (anotherPath(start, end)) {
31         int flow = network[parent[end]][end];
32         int curr = end;
33         while (parent[curr] >= 0) {
34             flow = min(flow, network[parent[curr]][curr]);
35             curr = parent[curr];
36         }
37         curr = end;
38         while (parent[curr] >= 0) {
39             network[parent[curr]][curr] -= flow;
40             network[curr][parent[curr]] += flow;
41             curr = parent[curr];
42         }
43         total += flow;
44     }
45     return total;
46 }

```

2.5 Max Flow Dinic

```

1 #include <iostream>
2 #include <queue>
3
4 using namespace std;
5
6 #define REP(i,n) for((i)=0;(i)<(int)(n);(i)++)
7 typedef int F;
8 #define F_INF (1<<29)
9 #define MAXV 10000
10 #define MAXE 1000000 // E*2!
11
12 F cap[MAXE], flow[MAXE];
13 int to[MAXE], _prev[MAXE], last[MAXV], used[MAXV], level[MAXV];
14
15 struct MaxFlow {
16     int V, E;
17
18     MaxFlow(int n) {

```

```

19     int i;
20     V = n; E = 0;
21     REP(i,V) last[i] = -1;
22 }
23
24 void add_edge(int x, int y, F f) { //directed edge
25     cap[E] = f; flow[E] = 0; to[E] = y;
26     _prev[E] = last[x]; last[x] = E; E++;
27
28     cap[E] = 0; flow[E] = 0; to[E] = x;
29     _prev[E] = last[y]; last[y] = E; E++;
30 }
31
32 bool bfs(int s, int t){
33     int i;
34     REP(i,V) level[i] = -1;
35     queue<int> q;
36     q.push(s); level[s] = 0;
37     while(!q.empty()){
38         int x = q.front(); q.pop();
39         for(i=last[x]; i>=0; i=_prev[i])
40             if(level[to[i]] == -1 && cap[i] > flow[i]) {
41                 q.push(to[i]);
42                 level[to[i]] = level[x] + 1;
43             }
44     }
45     return (level[t] != -1);
46 }
47
48 F dfs(int v, int t, F f){
49     int i;
50     if(v == t) return f;
51     for(i=used[v]; i>=0; used[v]= i = _prev[i])
52         if(level[to[i]] > level[v] && cap[i] > flow[i]) {
53             F tmp = dfs(to[i], t, min(f, cap[i]-flow[i]));
54             if(tmp > 0) {
55                 flow[i] += tmp;
56                 flow[i^1] -= tmp;
57                 return tmp;
58             }
59 }

```

```

60     return 0;
61 }
62
63 F maxflow(int s, int t) {
64     int i;
65     while(bfs(s,t)) {
66         REP(i,V) used[i] = last[i];
67         while(dfs(s,t,F_INF) != 0);
68     }
69     F ans = 0;
70     for(i=last[s]; i>=0; i=_prev[i])
71         ans += flow[i];
72     return ans;
73 }
74
75 };

```

2.6 Min Cost Max Flow

```

1 #include <iostream>
2 #include <queue>
3
4 using namespace std;
5
6 #define REP(i,n) for((i)=0;(i)<(int)(n);(i)++)
7
8 //XXX change these lines!
9 typedef int F;
10 typedef long long C;
11 #define F_INF (1<<29)
12 #define C_INF (1LL<<60)
13 #define MAXV 3000
14 #define MAXE 10000 // E*2! [or E*4 for bidirected graphs]
15
16 //no need to initialize these variables!
17 int V,E;
18 F cap[MAXE];
19 C cost[MAXE],dist[MAXV],pot[MAXV];
20 int to[MAXE],prv[MAXE],last[MAXV],path[MAXV];
21 bool used[MAXV];
22 priority_queue<pair<C, int>> q;
23

```

```

24 //output
25 F flow[MAXE];
26
27 class MinCostFlow {
28 public:
29     MinCostFlow(int n);
30     int add_edge(int x, int y, F w, C c); // zero based &&
31     pair <F, C> mincostflow(int s, int t);
32 private:
33     pair <F, C> search(int s, int t);
34     void bellman(int s);
35 };
36
37 //////////////////////////////////BACKEND////////////////////////////////////
38 MinCostFlow::MinCostFlow(int n){
39     V = n; E = 0;
40     int i; REP(i,V) last[i] = -1;
41 }
42 int MinCostFlow::add_edge(int x, int y, F w, C c){
43     cap[E] = w; flow[E] = 0; cost[E] = c; to[E] = y; prv[E] =
44     last[x]; last[x] = E; E++;
45     cap[E] = 0; flow[E] = 0; cost[E] = -c; to[E] = x; prv[E] =
46     last[y]; last[y] = E; E++;
47     return E-2;
48 }
49 void MinCostFlow::bellman(int s){
50     int i,x,e;
51     REP(i,V) pot[i] = C_INF;
52     pot[s] = 0;
53     REP(i,V+10) REP(x,V) for(e=last[x];e>=0;e=prv[e]) if(cap[e] >
54     0) pot[to[e]] = min(pot[to[e]], pot[x] + cost[e]);
55 }
56 pair <F, C> MinCostFlow::search(int s, int t){
57     F ansf=0; C ansc=0;
58     int i;
59     REP(i,V) used[i] = false;
60     REP(i,V) dist[i] = C_INF;
61     dist[s] = 0; path[s] = -1; q.push(make_pair(0,s));
62     while(!q.empty()){
63         int x = q.top().second; q.pop();

```

```

61         if(used[x]) continue; used[x] = true;
62         for(int e=last[x];e>=0;e=prv[e]) if(cap[e] > 0){
63             C tmp = dist[x] + cost[e] + pot[x] - pot[to[e]];
64             if(tmp < dist[to[e]] && !used[to[e]]){
65                 dist[to[e]] = tmp;
66                 path[to[e]] = e;
67                 q.push(make_pair(-dist[to[e]],to[e]));
68             }
69         }
70     }
71     REP(i,V) pot[i] += dist[i];
72     if(used[t]){
73         ansf = F_INF;
74         for(int e=path[t];e>=0;e=path[to[e^1]]) ansf = min(ansf,
75         cap[e]);
76         for(int e=path[t];e>=0;e=path[to[e^1]]) {ansc += cost[e]
77         * ansf; cap[e] -= ansf; cap[e^1] += ansf; flow[e] += ansf;
78         flow[e^1] -= ansf;}
79     }
80     return make_pair(ansf,ansc);
81 }
82 pair <F, C> MinCostFlow::mincostflow(int s, int t){
83     F ansf=0; C ansc=0;
84     int i;
85     bellman(s);
86     while(1){
87         pair <F, C> p = search(s,t);
88         if(!used[t]) break;
89         ansf += p.first; ansc += p.second;
90     }
91     return make_pair(ansf,ansc);
92 }
93 //////////////////////////////////
94
95 int main() {
96     return 0;
97 }

```

2.7 Cut Vertex

```

1 bool mark[MAXN], ans[MAXN];
2 int edge[MAXN], h[MAXN];

```

```

3 vector<int> adj[MAXN];
4
5 void dfs(int x, int par, int dep) {
6     mark[x] = true; h[x] = dep;
7     edge[x] = 1e9;
8
9     bool check = false;
10    int cnt = 0;
11    for (int i = 0; i < SZ(adj[x]); i++) {
12        int v = adj[x][i]; if (v == par) continue;
13        if (mark[v]) edge[x] = min(edge[x], h[v]);
14        else {
15            cnt++;
16            dfs(v, x, dep + 1);
17            if (edge[v] >= dep) check = true;
18            edge[x] = min(edge[x], edge[v]);
19        }
20    }
21
22    ans[x] = check;
23    if (par == -1 && cnt < 2) ans[x] = false;
24 }

```

2.8 Cut Edge

```

1 int backEdge[MAXN], h[MAXN], mark[MAXN], cut[MAXN];
2 vector<pii> adj[MAXN];
3
4 void dfs(int x, int par, int len) {
5     mark[x] = true, backEdge[x] = 1e9, h[x] = len;
6
7     for (int i = 0; i < SZ(adj[x]); i++) {
8         int v = adj[x][i].X, idx = adj[x][i].Y;
9         if (mark[v] && v != par) backEdge[x] = min(backEdge[x], h[v]);
10        ;
11        else if (!mark[v]) {
12            dfs(v, x, len + 1);
13            int tmp = backEdge[v];
14            if (tmp > h[x]) cut[idx] = true;
15            backEdge[x] = min(backEdge[x], tmp);
16        }
17    }
18 }

```

```

17 }

```

2.9 Bellman Ford

```

1 int n, m;
2 int ex[MAXN], ey[MAXN], ew[MAXN], d[MAXN];
3
4 bool bellman(int start) {
5     FOR(i, n) d[i] = INF;
6     d[start] = 0;
7
8     FOR(i, n - 1) FOR(j, m) {
9         int x = ex[j], y = ey[j]; double w = tw[j];
10        d[y] = min(d[y], d[x] + w);
11    }
12
13    // check if graph has a negative cycle
14    FOR(i, m) {
15        int x = ex[i], y = ey[i]; double w = tw[i];
16        if (d[y] > d[x] + w) return false;
17    }
18
19    return true;
20 }

```

2.10 Dijkstra

```

1 const int MAXN = 10 * 1000 + 10;
2 const ll INF = 1e9;
3
4 ll dis[MAXN];
5 set<pii> s;
6 bool mark[MAXN];
7 vector<pii> adj[MAXN];
8
9 void dij(int start) {
10    for (int i = 0; i < MAXN; i++) dis[i] = INF;
11
12    CLR(mark, 0); s.clear();
13    mark[start] = true;
14    dis[start] = 0;
15    s.insert(mp(0, start));

```



```

16 while (SZ(s)) {
17     int x = s.begin()->Y; s.erase(s.begin());
18
19     for (int i = 0; i < SZ(adj[x]); i++) {
20         int v = adj[x][i].X, w = adj[x][i].Y;
21         if (dis[v] > dis[x] + w) {
22             if (mark[v]) s.erase(mp(dis[v], v));
23             else mark[v] = true;
24
25             dis[v] = dis[x] + w;
26             s.insert(mp(dis[v], v));
27         }
28     }
29 }
30 }
31 }

```

2.11 Prim

```

1 const int N = 1000 * 100 + 5;
2 vector<pii> adj[N];
3 int ans, mrk[N];
4
5 void prim(int v) {
6     int w;
7     set<pii> st;
8     st.insert(mp(0, v));
9     while(!st.empty()) {
10         v = st.begin()-> Y;
11         w = st.begin()-> X;
12         st.erase(st.begin());
13         if(mrk[v]++) continue;
14         ans += w;
15
16         for(int i = 0; i < Size(adj[v]); ++i)
17             if(!mrk[adj[v][i].Y])
18                 st.insert(adj[v][i]);
19     }
20 }

```

2.12 DSU

```

1 int par[MAXN];
2 pair<int, pii> e[MAXN];
3
4 int father(int x) {
5     return par[x] == -1 ? x : par[x] = father(par[x]);
6 }
7
8 bool merge(int x, int y) {
9     x = father(x);
10    y = father(y);
11    if (x != y) par[y] = x;
12    return x != y;
13 }
14
15 fill(par, par + n, -1);

```

2.13 Eulerian Tour

```

1 void euler(int x) {
2     for (int i = 0; i < SZ(graph[x]); i++) {
3         int v = graph[x][i];
4         if (!vis[x][v]) {
5             vis[x][v] = vis[v][x] = true;
6             euler(v);
7         }
8     }
9     tour.pb(x);
10 }

```

2.14 Topological Sort

```

1 // This function uses performs a non-recursive topological sort.
2 //
3 // Running time:  $O(|V|^2)$ . If you use adjacency lists (vector<
4 // map<int> >),
5 //
6 // the running time is reduced to  $O(|E|)$ .
7 //
8 // INPUT: w[i][j] = 1 if i should come before j, 0 otherwise
9 // OUTPUT: a permutation of 0,...,n-1 (stored in a vector)
10 // which represents an ordering of the nodes which
11 // is consistent with w
12 //

```

```

11 // If no ordering is possible, false is returned.
12
13 #include <iostream>
14 #include <queue>
15 #include <cmath>
16 #include <vector>
17
18 using namespace std;
19
20 typedef double T;
21 typedef vector<T> VT;
22 typedef vector<VT> VVT;
23
24 typedef vector<int> VI;
25 typedef vector<VI> VVI;
26
27 bool TopologicalSort (const VVI &w, VI &order){
28     int n = w.size();
29     VI parents (n);
30     queue<int> q;
31     order.clear();
32
33     for (int i = 0; i < n; i++){
34         for (int j = 0; j < n; j++){
35             if (w[j][i]) parents[i]++;
36             if (parents[i] == 0) q.push (i);
37         }
38
39         while (q.size() > 0){
40             int i = q.front();
41             q.pop();
42             order.push_back (i);
43             for (int j = 0; j < n; j++) if (w[i][j]){
44                 parents[j]--;
45                 if (parents[j] == 0) q.push (j);
46             }
47         }
48
49         return (order.size() == n);
50     }

```

3 Geometry

3.1 Geometry

```

1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <cassert>
5
6 using namespace std;
7
8 double INF = 1e100;
9 double EPS = 1e-12;
10
11 struct PT {
12     double x, y;
13     PT() {}
14     PT(double x, double y) : x(x), y(y) {}
15     PT(const PT &p) : x(p.x), y(p.y) {}
16     PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
17     PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
18     PT operator * (double c) const { return PT(x*c, y*c); }
19     PT operator / (double c) const { return PT(x/c, y/c); }
20 };
21
22 double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
23 double dist2(PT p, PT q) { return dot(p-q,p-q); }
24 double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
25 ostream &operator<<(ostream &os, const PT &p) {
26     return os << "(" << p.x << "," << p.y << ")";
27 }
28
29 // if movement from a to b to c is done in a CW path returns 1
30 // else if it's CCW returns -1 and if they make a line returns 0
31 int IsCWTurn(PT a, PT b, PT c) {
32     double r = cross((b - a), (c - a));
33     return (fabs(r) < EPS)? 0: (r > 0)? 1: -1;

```

```

34 }
35
36 // rotate a point CCW or CW around the origin
37 PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
38 PT RotateCW90(PT p) { return PT(p.y,-p.x); }
39 PT RotateCCW(PT p, double t) {
40     return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
41 }
42
43 // project point c onto line through a and b
44 // assuming a != b
45 PT ProjectPointLine(PT a, PT b, PT c) {
46     return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
47 }
48
49 // project point c onto line segment through a and b
50 PT ProjectPointSegment(PT a, PT b, PT c) {
51     double r = dot(b-a,b-a);
52     if (fabs(r) < EPS) return a;
53     r = dot(c-a, b-a)/r;
54     return (r < 0)? a: (r > 1)? b: a + (b - a)*r;
55 }
56
57 // compute distance from c to segment between a and b
58 double DistancePointSegment(PT a, PT b, PT c) {
59     return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
60 }
61
62 // compute distance between point (x,y,z) and plane ax+by+cz=d
63 double DistancePointPlane(double x, double y, double z, double a,
64     double b,
65     double c, double d) {
66     return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
67 }
68
69 // determine if lines from a to b and c to d are parallel or
70 // collinear
71 bool LinesParallel(PT a, PT b, PT c, PT d) {
72     return fabs(cross(b-a, c-d)) < EPS;
73 }

```

```

73 bool LinesCollinear(PT a, PT b, PT c, PT d) {
74     return LinesParallel(a, b, c, d)
75     && fabs(cross(a-b, a-c)) < EPS
76     && fabs(cross(c-d, c-a)) < EPS;
77 }
78
79 // determine if line segment from a to b intersects with
80 // line segment from c to d
81 bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
82     if (LinesCollinear(a, b, c, d)) {
83         if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
84             dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
85         if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-
86             b) > 0)
87             return false;
88         return true;
89     }
90     if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
91     if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
92     return true;
93 }
94
95 // compute intersection of line passing through a and b
96 // with line passing through c and d, assuming that unique
97 // intersection exists; for segment intersection, check if
98 // segments intersect first
99 PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
100     b=b-a; d=c-d; c=c-a;
101     assert(dot(b, b) > EPS && dot(d, d) > EPS);
102     return a + b*cross(c, d)/cross(b, d);
103 }
104
105 // compute center of circle given three points
106 PT ComputeCircleCenter(PT a, PT b, PT c) {
107     b=(a+b)/2;
108     c=(a+c)/2;
109     return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+
110         RotateCW90(a-c));
111 }
112
113 // determine if point is in a possibly non-convex polygon (by

```

```

112 // Randolph Franklin); returns 1 for strictly interior points, 0
    for
113 // strictly exterior points, and 0 or 1 for the remaining points.
114 // Note that it is possible to convert this into an *exact* test
    using
115 // integer arithmetic by taking care of the division
    appropriately
116 // (making sure to deal with signs properly) and then by writing
    exact
117 // tests for checking point on polygon boundary
118 bool PointInPolygon(const vector<PT> &p, PT q) {
119     bool c = 0;
120     for (int i = 0; i < p.size(); i++) {
121         int j = (i+1)%p.size();
122         if (((p[i].y <= q.y && q.y < p[j].y) || (p[j].y <= q.y &&
            q.y < p[i].y)) &&
123             q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j]
            ].y - p[i].y))
124             c = !c;
125     }
126     return c;
127 }
128
129 // determine if point is on the boundary of a polygon
130 bool PointOnPolygon(const vector<PT> &p, PT q) {
131     for (int i = 0; i < p.size(); i++)
132         if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q)
            < EPS)
133             return true;
134     return false;
135 }
136
137 // compute intersection of line through points a and b with
138 // circle centered at c with radius r > 0
139 vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
140     vector<PT> ret;
141     b = b-a;
142     a = a-c;
143     double A = dot(b, b);
144     double B = dot(a, b);

```

```

145     double C = dot(a, a) - r*r;
146     double D = B*B - A*C;
147     if (D < -EPS) return ret;
148     ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
149     if (D > EPS)
150         ret.push_back(c+a+b*(-B-sqrt(D))/A);
151     return ret;
152 }
153
154 // compute intersection of circle centered at a with radius r
155 // with circle centered at b with radius R
156 vector<PT> CircleCircleIntersection(PT a, PT b, double r, double
    R) {
157     vector<PT> ret;
158     double d = sqrt(dist2(a, b));
159     if (d > r+R || d+min(r, R) < max(r, R)) return ret;
160     double x = (d*d-R*R+r*r)/(2*d);
161     double y = sqrt(r*r-x*x);
162     PT v = (b-a)/d;
163     ret.push_back(a+v*x + RotateCCW90(v)*y);
164     if (y > 0)
165         ret.push_back(a+v*x - RotateCCW90(v)*y);
166     return ret;
167 }
168
169 // This code computes the area or centroid of a (possibly
    nonconvex)
170 // polygon, assuming that the coordinates are listed in a
    clockwise or
171 // counterclockwise fashion. Note that the centroid is often
    known as
172 // the "center of gravity" or "center of mass".
173 double ComputeSignedArea(const vector<PT> &p) {
174     double area = 0;
175     for(int i = 0; i < p.size(); i++) {
176         int j = (i+1) % p.size();
177         area += p[i].x*p[j].y - p[j].x*p[i].y;
178     }
179     return area / 2.0;
180 }
181

```

```

182 double ComputeArea(const vector<PT> &p) {
183     return fabs(ComputeSignedArea(p));
184 }
185
186 PT ComputeCentroid(const vector<PT> &p) {
187     PT c(0,0);
188     double scale = 6.0 * ComputeSignedArea(p);
189     for (int i = 0; i < p.size(); i++){
190         int j = (i+1) % p.size();
191         c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
192     }
193     return c / scale;
194 }
195
196 // tests whether or not a given polygon (in CW or CCW order) is
197 // simple
198 bool IsSimple(const vector<PT> &p) {
199     for (int i = 0; i < p.size(); i++) {
200         for (int k = i+1; k < p.size(); k++) {
201             int j = (i+1) % p.size();
202             int l = (k+1) % p.size();
203             if (i == l || j == k) continue;
204             if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
205                 return false;
206         }
207     }
208     return true;

```

3.2 Convex Hull

```

1 // Compute the 2D convex hull of a set of points using the
2 // monotone chain
3 // algorithm. Eliminate redundant points from the hull if
4 // REMOVE_REDUNDANT is
5 // #defined.
6 //
7 // Running time: O(n log n)
8 //
9 // INPUT: a vector of input points, unordered.
10 // OUTPUT: a vector of points in the convex hull,
11 // counterclockwise, starting

```

```

9 // with bottommost/leftmost point
10
11 #include <cstdio>
12 #include <cassert>
13 #include <vector>
14 #include <algorithm>
15 #include <cmath>
16
17 using namespace std;
18
19 #define REMOVE_REDUNDANT
20
21 typedef double T;
22 const T EPS = 1e-7;
23 struct PT {
24     T x, y;
25     PT() {}
26     PT(T x, T y) : x(x), y(y) {}
27     bool operator<(const PT &rhs) const { return make_pair(y,x) <
28         make_pair(rhs.y,rhs.x); }
29     bool operator==(const PT &rhs) const { return make_pair(y,x)
30         == make_pair(rhs.y,rhs.x); }
31 };
32
33 T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
34 T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) +
35     cross(c,a); }
36
37 #ifdef REMOVE_REDUNDANT
38 bool between(const PT &a, const PT &b, const PT &c) {
39     return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0
40         && (a.y-b.y)*(c.y-b.y) <= 0);
41 }
42 #endif
43
44 void ConvexHull(vector<PT> &pts) {
45     sort(pts.begin(), pts.end());
46     pts.erase(unique(pts.begin(), pts.end()), pts.end());
47     vector<PT> up, dn;
48     for (int i = 0; i < pts.size(); i++) {
49         while (up.size() > 1 && area2(up[up.size()-2], up.back(),

```

```

pts[i]] >= 0) up.pop_back();
46 while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(),
pts[i]] <= 0) dn.pop_back();
47 up.push_back(pts[i]);
48 dn.push_back(pts[i]);
49 }
50 pts = dn;
51 for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(
up[i]);
52
53 #ifdef REMOVE_REDUNDANT
54 if (pts.size() <= 2) return;
55 dn.clear();
56 dn.push_back(pts[0]);
57 dn.push_back(pts[1]);
58 for (int i = 2; i < pts.size(); i++) {
59     if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn
.pop_back();
60     dn.push_back(pts[i]);
61 }
62 if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
63     dn[0] = dn.back();
64     dn.pop_back();
65 }
66 pts = dn;
67 #endif
68 }

```

4 Data Structures

4.1 Fenwick1

```

1 const int MAXN = 1 * 1000 + 10;
2
3 int fen[MAXN]; // 0-based, []
4
5 void add(int x, int val = 1) {
6     for (int i = x + 1; i < MAXN; i += i & (-i))
7         fen[i] += val;
8 }
9

```

```

10 int get(int x) {
11     int ans = 0;
12     for (int i = x; i > 0; i -= i & (-i))
13         ans += fen[i];
14     return ans;
15 }
16
17 int sum(int x, int y) {
18     return get(y) - get(x);
19 }

```

4.2 Fenwick2

```

1 int fen[MAXN]; // 0-based, []
2
3 void add(int x, int val) {
4     for (int i = x; i > 0; i -= i & (-i))
5         fen[i] += val;
6 }
7
8 int get(int x) {
9     int ans = 0;
10    for (int i = x + 1; i < MAXN; i += i & (-i))
11        ans += fen[i];
12    return ans;
13 }
14
15 void update(int l, int r, int val) {
16     add(r, +val);
17     add(l, -val);
18 }

```

4.3 Segment Tree

```

1 int v1[4 * MAXN];
2
3 // lo, hi -> []
4 // s = 0, e = n, x = 1
5
6 void update1(int lo, int hi, int s, int e, int x, int val) {
7     if (lo == s && hi == e) {
8         v1[x] = val;

```

```

9     return ;
10 }
11 int mid = (s + e) / 2;
12 if (lo < mid) update1(lo, min(hi, mid), s, mid, x + x + 0, val)
13 ;
14 if (hi > mid) update1(max(lo, mid), hi, mid, e, x + x + 1, val)
15 ;
16 }
17 int get1(int k, int s, int e, int x) {
18     if (e - s < 2) return v1[x];
19     int mid = (s + e) / 2;
20     return max(v1[x], ((k < mid) ? get1(k, s, mid, x + x + 0) :
21         get1(k, mid, e, x + x + 1)));
22 }
23 int v2[4 * MAXN];
24 void update2(int k, int s, int e, int x, int val) {
25     if (e - s < 2) {
26         v2[x] = val;
27         return ;
28     }
29     int mid = (s + e) / 2;
30     if (k < mid) update2(k, s, mid, x + x + 0, val);
31     else update2(k, mid, e, x + x + 1, val);
32     v2[x] = max(v2[x + x + 0], v2[x + x + 1]);
33 }
34 int get2(int lo, int hi, int s, int e, int x) {
35     if (lo == s && hi == e) return v2[x];
36     int mid = (s + e) / 2, ans = 0;
37     if (lo < mid) ans = max(ans, get2(lo, min(hi, mid), s, mid, x +
38         x + 0));
39     if (hi > mid) ans = max(ans, get2(max(lo, mid), hi, mid, e, x +
40         x + 1));
41     return ans;
42 }

```

4.4 Segment Tree Lazy Propagation

```

1 int seg[4 * MAXN], add[4 * MAXN];

```

```

2
3 inline void shift(int x, int s, int e) {
4     int lc = x + x + 0, rc = x + x + 1;
5
6     int mid = (s + e) / 2;
7     int l1 = mid - s, l2 = e - mid;
8
9     seg[lc] += l1 * add[x];
10    seg[rc] += l2 * add[x];
11    add[lc] += add[x];
12    add[rc] += add[x];
13
14    add[x] = 0;
15 }
16
17 // lo, hi -> []
18 // s = 0, e = n, x = 1
19
20 void update(int lo, int hi, int s, int e, int x, int delta) {
21     if (lo == s && hi == e) {
22         int len = (e - s);
23         seg[x] = seg[x] + len * delta;
24         add[x] = add[x] + delta;
25         return ;
26     }
27     shift(x, s, e);
28
29     int mid = (s + e) / 2;
30     int lc = x + x + 0, rc = x + x + 1;
31     if (lo < mid) update(lo, min(mid, hi), s, mid, lc, delta);
32     if (hi > mid) update(max(lo, mid), hi, mid, e, rc, delta);
33
34     seg[x] = seg[lc] + seg[rc];
35 }
36
37 int get(int lo, int hi, int s, int e, int x) {
38     if (lo == s && hi == e) return seg[x];
39     shift(x, s, e);
40
41     int mid = (s + e) / 2;
42     int res = 0;

```

```

43 if (lo < mid) res += get(lo, min(mid, hi), s, mid, x + x + 0);
44 if (hi > mid) res += get(max(lo, mid), hi, mid, e, x + x + 1);
45
46 return res;
47 }

```

4.5 RMQ

```

1 const int N = 1000 * 100 + 5, LOG = 20;
2
3 class RMQ{
4     int f[LOG][N], Lgl[N], S;
5 public:
6     RMQ() {
7         for(int i = 1, p = 0; i < N; ++i) {
8             if(i == 1 << (p + 1))
9                 ++p;
10            Lgl[i] = p;
11        }
12    }
13    void build(int a[], int n) {
14        for(int i = 0; i < n; ++i)
15            f[0][i] = a[i];
16
17        for(int j = 1, p = 1; j < LOG; ++j, p *= 2)
18            for(int i = 0; i < n; ++i) {
19                f[j][i] = f[j - 1][i];
20                if(i + p < n)
21                    f[j][i] = min(f[j - 1][i], f[j - 1][i + p]);
22            }
23    }
24    int find(int s, int e) {
25        int l = Lgl[e - s + 1];
26        return min(f[l][s], f[l][e + 1 - (1 << l)]);
27    }
28 };

```

4.6 Trie

```

1 struct Node {
2     char x;
3     vector<Node*> adj;

```

```

4
5     Node () {
6         x = 0;
7     }
8
9     Node (char a) {
10        x = a;
11    }
12
13    Node* add_edge(char a) {
14        for (int i = 0; i < SZ(adj); i++)
15            if (adj[i] -> x == a)
16                return adj[i];
17        adj.pb(new Node(a));
18        return adj.back();
19    }
20 };
21
22 struct Trie {
23     Node* root;
24
25     Trie() {
26         root = new Node();
27     }
28
29     void add(string &s) {
30         add(s, 0, root);
31     }
32
33     void add(string &s, int pos, Node* node) {
34         if (pos == SZ(s)) {
35             return ;
36         } else {
37             Node* next = node -> add_edge(s[pos]);
38             add(s, pos + 1, next);
39         }
40     }
41 };

```


5 String

5.1 Hash

```

1 ll p[MAXN], hash[MAXN];
2
3 int main () {
4     p[0] = 1;
5     for (int i = 1; i < MAXN; i++)
6         p[i] = p[i - 1] * BASE;
7
8     string s;
9     getline(cin, s);
10
11    for (int i = 1; i <= SZ(s); i++)
12        hash[i] = hash[i - 1] * BASE + s[i - 1];
13
14
15    // hash in [i, j], 1-based
16    ll h = hash[j] - (hash[i - 1] * p[j - i + 1]);
17 }

```

5.2 KMP

```

1 #define SZ(x) ((int)((x).size()))
2
3 const int M = 1000 * 100 + 4;
4 int f[M];
5 string s, t;
6 bool match[M];
7 void kmp() {
8     f[0] = -1;
9     int pos = -1;
10    for (int i = 1; i <= SZ(t); i++) {
11        while(pos != -1 && t[pos] != t[i - 1]) pos = f[pos];
12        f[i] = ++pos;
13    }
14    pos = 0;
15    for (int i = 0; i < SZ(s); i++) {
16        while(pos != -1 && (pos == SZ(t) || s[i] != t[pos])) pos = f[pos];
17        pos++;

```

```

18     if (pos == SZ(t)) match[i] = 1;
19     else match[i] = 0;
20 }
21 }

```

5.3 Suffix Array

```

1 const int N = 1000 * 100 + 5;
2
3 namespace Suffix{
4     int sa[N], rank[N], lcp[N], gap, S;
5     bool cmp(int x, int y) {
6         if(rank[x] != rank[y])
7             return rank[x] < rank[y];
8         x += gap, y += gap;
9         return (x < S && y < S)? rank[x] < rank[y]: x > y;
10    }
11    void Sa_build(const string &s) {
12        S = Size(s);
13        int tmp[N] = {0};
14        for(int i = 0; i < S; ++i)
15            rank[i] = s[i],
16            sa[i] = i;
17        for(gap = 1; gap <= 1) {
18            sort(sa, sa + S, cmp);
19            for(int i = 1; i < S; ++i)
20                tmp[i] = tmp[i - 1] + cmp(sa[i - 1], sa[i]);
21            for(int i = 0; i < S; ++i)
22                rank[sa[i]] = tmp[i];
23            if(tmp[S - 1] == S - 1)
24                break;
25        }
26    }
27    void Lcp_build() {
28        for(int i = 0, k = 0; i < S; ++i, —k)
29            if(rank[i] != S - 1) {
30                k = max(k, 0);
31                while(s[i + k] == s[sa[rank[i] + 1] + k])
32                    ++k;
33                lcp[rank[i]] = k;
34            }
35        else

```

```

36     k = 0;
37 }
38 };

```

6 Number Theory

6.1 Phi

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 const int N = 1000 * 1000;
6
7 vector<int> pr;
8 int lp[N], phi[N];
9
10 void Sieve(int n){
11     for (int i = 2; i < n; ++i) {
12         if (lp[i] == 0)
13             lp[i] = i,
14             pr.push_back(i);
15
16         for (int j = 0; j < pr.size() && pr[j] <= lp[i] && i * pr[j] <
17             n; ++j)
18             lp[i * pr[j]] = pr[j];
19     }
20
21 void Find_Phi(int n) {
22     phi[1] = 1;
23     for(int i = 2; i < n; ++i) {
24         if(lp[i] == i)
25             phi[i] = i - 1;
26         else {
27             phi[i] = phi[lp[i]] * phi[(i / lp[i])];
28             if(lp[i / lp[i]] == lp[i])
29                 phi[i] *= lp[i], phi[i] /= (lp[i] - 1)
30         }
31     }
32 }

```

6.2 370 SGU

```

1 bool mark[MAXN];
2 vector<int> dv[MAXN];
3 int n, m;
4
5 // counts positive integers up to n that are relatively prime to
6 // x
7 int f(int x) {
8     int res = 0;
9     for (int mask = 0; mask < (1 << SZ(dv[x])); mask += 1) {
10         int t = __builtin_popcount(mask), a = n - 1;
11         for (int i = 0; i < SZ(dv[x]); i += 1)
12             if (mask & (1 << i))
13                 a /= dv[x][i];
14         if (t & 1) res -= a;
15         else res += a;
16     }
17     return res;
18 }
19
20 int main () {
21     for (int i = 2; i < n; i += 1)
22         if (!mark[i]) {
23             for (int j = i; j < m; j += i) {
24                 mark[j] = true;
25                 dv[j].pb(i);
26             }
27         }
28     ll ans = 2;
29     for (int i = 1; i < m; i += 1) ans += f(i);
30     cout << ans << endl;
31     return 0;
32 }

```

6.3 Euclid

```

1 typedef vector<int> VI;
2 typedef pair<int, int> PII;
3
4 // computes gcd(a,b)
5 int gcd(int a, int b) {

```

```

6  while (b) { int t = a%b; a = b; b = t; }
7  return a;
8  }
9
10 // returns g = gcd(a, b); finds x, y such that d = ax + by
11 int extended_euclid(int a, int b, int &x, int &y) {
12     int xx = y = 0;
13     int yy = x = 1;
14     while (b) {
15         int q = a / b;
16         int t = b; b = a%b; a = t;
17         t = xx; xx = x - q*xx; x = t;
18         t = yy; yy = y - q*yy; y = t;
19     }
20     return a;
21 }
22
23 // finds all solutions to ax = b (mod n)
24 VI modular_linear_equation_solver(int a, int b, int n) {
25     int x, y;
26     VI ret;
27     int g = extended_euclid(a, n, x, y);
28     if (!(b%g)) {
29         x = mod(x*(b / g), n);
30         for (int i = 0; i < g; i++)
31             ret.push_back(mod(x + i*(n / g), n));
32     }
33     return ret;
34 }
35
36 // Chinese remainder theorem (special case): find z such that
37 // z % m1 = r1, z % m2 = r2. Here, z is unique modulo M = lcm(m1
38 // , m2).
39 // Return (z, M). On failure, M = -1.
40 PII chinese_remainder_theorem(int m1, int r1, int m2, int r2) {
41     int s, t;
42     int g = extended_euclid(m1, m2, s, t);
43     if (r1%g != r2%g) return make_pair(0, -1);
44     return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2) / g, m1*m2 / g);
45 }

```

```

46
47 // Chinese remainder theorem: find z such that
48 // z % m[i] = r[i] for all i. Note that the solution is
49 // unique modulo M = lcm_i (m[i]). Return (z, M). On
50 // failure, M = -1. Note that we do not require the a[i]'s
51 // to be relatively prime.
52 PII chinese_remainder_theorem(const VI &m, const VI &r) {
53     PII ret = make_pair(r[0], m[0]);
54     for (int i = 1; i < m.size(); i++) {
55         ret = chinese_remainder_theorem(ret.second, ret.first, m[i],
56             r[i]);
57         if (ret.second == -1) break;
58     }
59     return ret;
60 }

```

6.4 C(n, r)

```

1  ll bin_pow(ll x, ll y) {
2      if (y == 0) return 1;
3
4      ll tmp = bin_pow(x, y / 2);
5      ll res = SQR(tmp) % MOD;
6      if (y & 1) res = (res * x) % MOD;
7      return res;
8  }
9
10 ll fct[MAXN], rev[MAXN], fct_rev[MAXN];
11
12 void init(int n) {
13     fct[0] = 1;
14     for (int i = 1; i <= n; i++)
15         fct[i] = (fct[i - 1] * i) % MOD;
16
17     rev[0] = 1;
18     for (int i = 1; i <= n; i++)
19         rev[i] = bin_pow(i, MOD - 2);
20
21     fct_rev[0] = 1;
22     for (int i = 1; i <= n; i++)
23         fct_rev[i] = (fct_rev[i - 1] * rev[i]) % MOD;
24 }

```

```

25
26 int C(int n, int r) {
27     return (((fct[n] * fct_rev[r]) % MOD) * fct_rev[n - r]) % MOD;
28 }

```

6.5 FFT

```

1 typedef long double DOUBLE;
2 typedef complex<DOUBLE> COMPLEX;
3 typedef vector<DOUBLE> VD;
4 typedef vector<COMPLEX> VC;
5
6 struct FFT {
7     VC A;
8     int n, L;
9
10    int ReverseBits(int k) {
11        int ret = 0;
12        for (int i = 0; i < L; i++) {
13            ret = (ret << 1) | (k & 1);
14            k >>= 1;
15        }
16        return ret;
17    }
18
19    void BitReverseCopy(VC a) {
20        for (n = 1, L = 0; n < a.size(); n <= 1, L++) ;
21        A.resize(n);
22        for (int k = 0; k < n; k++)
23            A[ReverseBits(k)] = a[k];
24    }
25
26    VC DFT(VC a, bool inverse) {
27        BitReverseCopy(a);
28        for (int s = 1; s <= L; s++) {
29            int m = 1 << s;
30            COMPLEX wm = exp(COMPLEX(0, 2.0 * M_PI / m));
31            if (inverse) wm = COMPLEX(1, 0) / wm;
32            for (int k = 0; k < n; k += m) {
33                COMPLEX w = 1;
34                for (int j = 0; j < m/2; j++) {
35                    COMPLEX t = w * A[k + j + m/2];

```

```

36                    COMPLEX u = A[k + j];
37                    A[k + j] = u + t;
38                    A[k + j + m/2] = u - t;
39                    w = w * wm;
40                }
41            }
42        }
43        if (inverse) for (int i = 0; i < n; i++) A[i] /= n;
44        return A;
45    }
46
47    // c[k] = sum_{i=0}^k a[i] b[k-i]
48    VD Convolution(VD a, VD b) {
49        int L = 1;
50        while ((1 << L) < a.size()) L++;
51        while ((1 << L) < b.size()) L++;
52        int n = 1 << (L+1);
53
54        VC aa, bb;
55        for (size_t i = 0; i < n; i++) aa.push_back(i < a.size()
56        ? COMPLEX(a[i], 0) : 0);
57        for (size_t i = 0; i < n; i++) bb.push_back(i < b.size()
58        ? COMPLEX(b[i], 0) : 0);
59
60        VC AA = DFT(aa, false);
61        VC BB = DFT(bb, false);
62        VC CC;
63        for (size_t i = 0; i < AA.size(); i++) CC.push_back(AA[i]
64        * BB[i]);
65        VC cc = DFT(CC, true);
66
67        VD c;
68        for (int i = 0; i < a.size() + b.size() - 1; i++) c.
69        push_back(cc[i].real());
70    }
71 };

```

6.6 Gauss Jordan

```

1 // Gauss-Jordan elimination with full pivoting.

```

```

2 //
3 // Uses:
4 // (1) solving systems of linear equations (AX=B)
5 // (2) inverting matrices (AX=I)
6 // (3) computing determinants of square matrices
7 //
8 // Running time: O(n^3)
9 //
10 // INPUT:   a[][] = an nxn matrix
11 //          b[][] = an nxm matrix
12 //
13 // OUTPUT:  X      = an nxm matrix (stored in b[][])
14 //          A-1 = an nxn matrix (stored in a[][])
15 //          returns determinant of a[][]
16
17 const double EPS = 1e-10;
18
19 typedef vector<int> VI;
20 typedef double T;
21 typedef vector<T> VT;
22 typedef vector<VT> VVT;
23
24 T GaussJordan(VVT &a, VVT &b) {
25     const int n = a.size();
26     const int m = b[0].size();
27     VI irow(n), icol(n), ipiv(n);
28     T det = 1;
29
30     for (int i = 0; i < n; i++) {
31         int pj = -1, pk = -1;
32         for (int j = 0; j < n; j++) if (!ipiv[j])
33             for (int k = 0; k < n; k++) if (!ipiv[k])
34                 if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
35         if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
36         ipiv[pj]++;
37         swap(a[pj], a[pk]);
38         swap(b[pj], b[pk]);
39         if (pj != pk) det *= -1;
40         irow[i] = pj;

```

```

41         icol[i] = pk;
42
43         T c = 1.0 / a[pk][pk];
44         det *= a[pk][pk];
45         a[pk][pk] = 1.0;
46         for (int p = 0; p < n; p++) a[pk][p] *= c;
47         for (int p = 0; p < m; p++) b[pk][p] *= c;
48         for (int p = 0; p < n; p++) if (p != pk) {
49             c = a[p][pk];
50             a[p][pk] = 0;
51             for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
52             for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
53         }
54     }
55
56     for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
57         for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
58     }
59
60     return det;
61 }
62
63 int main() {
64     const int n = 4;
65     const int m = 2;
66     double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
67     double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
68     VVT a(n), b(n);
69     for (int i = 0; i < n; i++) {
70         a[i] = VT(A[i], A[i] + n);
71         b[i] = VT(B[i], B[i] + m);
72     }
73     double det = GaussJordan(a, b);
74 }

```

7 Other

7.1 Read Input

```

1 inline int read() {

```

```

2 bool minus = false;
3 int result = 0;
4 char ch;
5 ch = getchar();
6 while (true) {
7     if (ch == '-') break;
8     if (ch >= '0' && ch <= '9') break;
9     ch = getchar();
10 }
11 if (ch == '-') minus = true; else result = ch-'0';
12 while (true) {
13     ch = getchar();
14     if (ch < '0' || ch > '9') break;
15     result = result*10 + (ch - '0');
16 }
17 if (minus)
18     return -result;
19 else
20     return result;
21 }

```

7.2 LIS

```

1 int c[MAXN], a[MAXN];
2
3 int main() {
4     int n;
5     cin >> n;
6     for (int i = 0; i < n; i++) cin >> a[i];
7     for (int i = 0; i <= n; i++) c[i] = 1e9;
8
9     int ans = 0;
10    for (int i = 0; i < n; i++) {
11        int l = 0, r = i + 1;
12        while (r - l > 1) {
13            int mid = (l + r) / 2;
14            if (c[mid] <= a[i]) l = mid;
15            else r = mid;
16        }
17        ans = max(ans, l + 1);
18        if (c[l + 1] > a[i]) c[l + 1] = a[i];
19    }

```

```

20 cout << ans << endl;
21 }

```

7.3 Divide and Conquer Tree

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 const int N = 1000 * 100 + 5;
6 vector <int> adj[N];
7
8 int is_av[N], _sz[N]; //XXX initiate is_av to 1
9
10 void set_size(int v, int p) {
11     _sz[v] = 1;
12     for(int u:adj[v])
13         if(u != p && is_av[u]) {
14             set_size(u, v);
15             _sz[v] += _sz[u];
16         }
17 }
18
19 void divide(int v) {
20     set_size(v, v);
21     int S = _sz[v], p = v;
22     sign:
23     for(int u:adj[v])
24         if(is_av[u] && u != p && _sz[u] > S / 2) {
25             p = v;
26             v = u;
27             goto sign;
28         }
29
30     // now v is the centroid of the tree
31     // Enter your code here
32
33     is_av[v] = 0;
34     for(int u:adj[v])
35         if(is_av[u])
36             divide(u);
37 }

```

```
38
39 int main() {
40     ios::sync_with_stdio(false);
41     int n;
42     cin >> n;
43     for(int i = 1; i < n; ++i) {
44         int a, b;
45         cin >> a >> b;
```

```
46     -a, -b;
47     adj[a].push_back(b);
48     adj[b].push_back(a);
49     }
50     fill(is_av, is_av + N, 1);
51     divide(0);
52     return 0;
53 }
```