

Contents

1	Base	3
1.1	vimrc	3
1.2	Template	3
2	Graph	3
2.1	LCA	3
2.2	SCC	3
2.3	Matching	4
2.4	Max Flow BFS	4
2.5	Max Flow Dinic	5
2.6	Min Cost Max Flow	6
2.7	Bellman Ford	7
2.8	Dijkstra	7
2.9	Prim	8
2.10	DSU	8
2.11	Eulerian Tour	8
2.12	Topological Sort	8
2.13	Kardak General Graph matching3	9
3	Geometry	10
3.1	Geometry	10
3.2	Convex Hull	13
3.3	Kardak Find intersection between lines	14
3.4	Kardak Half Plane Intersection	15
4	Data Structures	16
4.1	Fenwick1	16
4.2	Fenwick2	17
4.3	Segment Tree Lazy Propagation	17
4.4	RMQ	17
4.5	Trie	18
5	String	18
5.1	Hash	18
5.2	KMP	18
5.3	Suffix Array	19
5.4	Kardak AhoCorasick	19

6	Number Theory	21
6.1	Phi	21
6.2	370 SGU	22
6.3	Euclid	22
6.4	C(n, r)	23
6.5	FFT	23
6.6	Gauss Jordan	24
7	Other	25
7.1	Read Input	25
7.2	LIS	25
7.3	Divide and Conquer Tree	25

1 Base

1.1 vimrc

```

1 cd ~/Documents/Contest/
2 set sw=4
3 set ts=4
4 set si      " super indentation
5 set number  " line numbers
6 syntax on   " syntax highlighting
7 set cursorline " highlight current line
8 set guifont=consolas:h11
9 set bs=2
10 set mouse=a " mouse works normally
11 set gdefault " global replacement
12 set fdm=indent " folding method
13 set foldlevelstart=99 " at first all folds are open

```

1.2 Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define pb push_back
4 #define mp make_pair
5 #define SQR(a) ((a) * (a))
6 #define SZ(x) ((int) (x).size())
7 #define ALL(x) (x).begin(), (x).end()
8 #define CLR(x, a) memset(x, a, sizeof x)
9 #define VAL(x) #x << " = " << (x) << " "
10 #define FOREACH(i, x) for(_typeof((x).begin()) i = (x).begin(); i != (x).end(); i++)
11 #define FOR(i, n) for (int i = 0; i < (n); i++)
12 #define X first
13 #define Y second
14 typedef long long ll;
15 typedef pair<ll, ll> pll;
16 typedef pair<int, int> pii;
17 const int MAXN = 1000 + 10;
18 int main () {
19     ios::sync_with_stdio(false);
20     return 0;
21 }

```

2 Graph

2.1 LCA

```

1 vector<int> adj[MAXN];
2 int par[MAXN][MAXL], h[MAXN];
3 bool mark[MAXN];
4 void dfs(int x) {
5     mark[x] = true;
6     for (int i = 0; i < SZ(adj[x]); i++) {
7         int v = adj[x][i];
8         if (!mark[v]) {
9             par[v][0] = x, h[v] = h[x] + 1;
10            dfs(v);
11        }
12    }
13 }
14 int get_parent(int x, int k) {
15     for (int i = 0; i < MAXL; i++)
16         if ((1 << i) & k) x = par[x][i];
17     return x;
18 }
19 int lca(int x, int y) {
20     if (h[y] > h[x]) swap(x, y);
21     x = get_parent(x, h[x] - h[y]);
22     if (x == y) return x;
23     for (int i = MAXL - 1; i >= 0; i--)
24         if (par[x][i] != par[y][i])
25             x = par[x][i], y = par[y][i];
26     return par[x][0];
27 }
28 int main () {
29     par[0][0] = -1;
30     dfs(0);
31     for (int i = 1; i < MAXL; i++)
32         for (int j = 0; j < n; j++)
33             par[j][i] = par[par[j][i - 1]][i - 1];
34 }

```

2.2 SCC

```

1 vector <int> adj[N];

```

```

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

```

```

12     for (int i = 0; i < SZ(adj[x]); i++) {
13         int v = adj[x][i];
14         if (match[1][v] == -1 || dfs(match[1][v])) {
15             match[0][x] = v;
16             match[1][v] = x;
17             return true;
18         }
19     }
20     return false;
21 }
22 void bi_match() {
23     CLR(match, -1);
24     for (int i = 0; i < n; i++) {
25         CLR(mark, 0);
26         bool check = false;
27         for (int j = 0; j < n; j++)
28             if (!mark[j] && match[0][j] == -1)
29                 check |= dfs(j);
30         if (!check) break;
31     }
32 }
33 int main () {
34     cin >> n >> m >> p;
35     for (int i = 0; i < p; i++) {
36         int x, y; cin >> x >> y; x--, y--;
37         // x: a node in first part [0, n)
38         // y: a node in second part [0, m)
39         adj[x].pb(y);
40     }
41     bi_match();
42     int ans = 0;
43     FOR(i, n) ans += (match[0][i] != -1);
44     cout << ans << endl;
45     return 0;
46 }

```

2.4 Max Flow BFS

```

1 #include <queue>
2 #include <cstring>
3 const int N = 100;
4 int mat[N][N];

```

```

5 int viz[N], network[N][N], parent[N];
6 bool anotherPath(int start, int end) {
7     memset(viz, 0, sizeof viz);
8     memset(parent, -1, sizeof parent);
9     viz[start] = true;
10    queue<int> q;
11    q.push(start);
12    while (!q.empty()) {
13        int z = q.front(); q.pop();
14        viz[z] = true;
15        for (int i=0; i<N; i++) {
16            if (network[z][i] <= 0 || viz[i]) continue;
17            viz[i] = true;
18            parent[i] = z;
19            if (i == end) return true;
20            q.push(i);
21        }
22    }
23    return false;
24 }
25 int maxflow(int start, int end) {
26     memcpy(network, mat, sizeof(mat));
27     int total = 0;
28     while (anotherPath(start, end)) {
29         int flow = network[parent[end]][end];
30         int curr = end;
31         while (parent[curr] >= 0) {
32             flow = min(flow, network[parent[curr]][curr]);
33             curr = parent[curr];
34         }
35         curr = end;
36         while (parent[curr] >= 0) {
37             network[parent[curr]][curr] -= flow;
38             network[curr][parent[curr]] += flow;
39             curr = parent[curr];
40         }
41         total += flow;
42     }
43     return total;
44 }

```

2.5 Max Flow Dinic

```

1 #include <iostream>
2 #include <queue>
3 using namespace std;
4 #define REP(i,n) for((i)=0;(i)<(int)(n);(i)++)
5 typedef int F;
6 #define F_INF (1<<29)
7 #define MAXV 10000
8 #define MAXE 1000000 // E 2!
9 F cap[MAXE], flow[MAXE];
10 int to[MAXE], _prev[MAXE], last[MAXV], used[MAXV], level[MAXV];
11 struct MaxFlow {
12     int V, E;
13
14     MaxFlow(int n) {
15         int i;
16         V = n; E = 0;
17         REP(i,V) last[i] = -1;
18     }
19     void add_edge(int x, int y, F f) { //directed edge
20         cap[E] = f; flow[E] = 0; to[E] = y;
21         _prev[E] = last[x]; last[x] = E; E++;
22
23         cap[E] = 0; flow[E] = 0; to[E] = x;
24         _prev[E] = last[y]; last[y] = E; E++;
25     }
26     bool bfs(int s, int t){
27         int i;
28         REP(i,V) level[i] = -1;
29         queue<int> q;
30         q.push(s); level[s] = 0;
31         while(!q.empty()){
32             int x = q.front(); q.pop();
33             for(i=last[x]; i>=0; i=_prev[i])
34                 if(level[to[i]] == -1 && cap[i] > flow[i]) {
35                     q.push(to[i]);
36                     level[to[i]] = level[x] + 1;
37                 }
38             }
39         return (level[t] != -1);

```

```

40 }
41 F dfs(int v, int t, F f){
42     int i;
43     if(v == t) return f;
44     for(i=used[v]; i>=0; used[v]= i =_prev[i])
45         if(level[to[i]] > level[v] && cap[i] > flow[i]) {
46             F tmp = dfs(to[i], t, min(f, cap[i]-flow[i]));
47             if(tmp > 0) {
48                 flow[i] += tmp;
49                 flow[i^1] -= tmp;
50                 return tmp;
51             }
52         }
53     return 0;
54 }
55 F maxflow(int s, int t) {
56     int i;
57     while(bfs(s,t)) {
58         REP(i,V) used[i] = last[i];
59         while(dfs(s,t,F_INF) != 0);
60     }
61     F ans = 0;
62     for(i=last[s];i>=0;i=_prev[i])
63         ans += flow[i];
64     return ans;
65 }
66 };

```

2.6 Min Cost Max Flow

```

1 #include <iostream>
2 #include <queue>
3 using namespace std;
4 #define REP(i,n) for((i)=0;(i)<(int)(n);(i)++)
5 //XXX change these lines!
6 typedef int F;
7 typedef long long C;
8 #define F_INF (1<<29)
9 #define C_INF (1LL<<60)
10 #define MAXV 3000
11 #define MAXE 10000 // E 2! [or E 4 for bidirected graphs]
12 //no need to initialize these variables!

```

```

13 int V,E;
14 F cap[MAXE];
15 C cost[MAXE],dist[MAXV],pot[MAXV];
16 int to[MAXE],prv[MAXE],last[MAXV],path[MAXV];
17 bool used[MAXV];
18 priority_queue <pair <C, int> > q;
19 F flow[MAXE]; //output
20 class MinCostFlow {
21 public:
22     MinCostFlow(int n);
23     int add_edge(int x, int y, F w, C c); // zero based &&
24     pair <F, C> mincostflow(int s, int t);
25 private:
26     pair <F, C> search(int s, int t);
27     void bellman(int s);
28 };
29 //////////////////////////////////////////////////BACKEND!////////////////////////////////////
30 MinCostFlow::MinCostFlow(int n){
31     V = n; E = 0;
32     int i; REP(i,V) last[i] = -1;
33 }
34 int MinCostFlow::add_edge(int x, int y, F w, C c){
35     cap[E] = w; flow[E] = 0; cost[E] = c; to[E] = y; prv[E] =
36     last[x]; last[x] = E; E++;
37     cap[E] = 0; flow[E] = 0; cost[E] = -c; to[E] = x; prv[E] =
38     last[y]; last[y] = E; E++;
39     return E-2;
40 }
41 void MinCostFlow::bellman(int s){
42     int i,x,e;
43     REP(i,V) pot[i] = C_INF;
44     pot[s] = 0;
45     REP(i,V+10) REP(x,V) for(e=last[x];e>=0;e=prv[e]) if(cap[e] >
46     0) pot[to[e]] = min(pot[to[e]], pot[x] + cost[e]);
47 }
48 pair <F, C> MinCostFlow::search(int s, int t){
49     F ansf=0; C ansc=0;
50     int i;
51     REP(i,V) used[i] = false;
52     REP(i,V) dist[i] = C_INF;

```

```

50 dist[s] = 0; path[s] = -1; q.push(make_pair(0,s));
51 while(!q.empty()){
52     int x = q.top().second; q.pop();
53     if(used[x]) continue; used[x] = true;
54     for(int e=last[x];e>=0;e=prv[e]) if(cap[e] > 0){
55         C tmp = dist[x] + cost[e] + pot[x] - pot[to[e]];
56         if(tmp < dist[to[e]] && !used[to[e]]){
57             dist[to[e]] = tmp;
58             path[to[e]] = e;
59             q.push(make_pair(-dist[to[e]],to[e]));
60         }
61     }
62 }
63 REP(i,V) pot[i] += dist[i];
64 if(used[t]){
65     ansf = F_INF;
66     for(int e=path[t];e>=0;e=path[to[e^1]]) ansf = min(ansf,
67 cap[e]);
68     for(int e=path[t];e>=0;e=path[to[e^1]]) {ansc += cost[e]
69     ansf; cap[e] -= ansf; cap[e^1] += ansf; flow[e] += ansf;
70     flow[e^1] -= ansf;}
71 }
72 return make_pair(ansf,ansc);
73 }
74 pair <F, C> MinCostFlow::mincostflow(int s, int t){
75     F ansf=0; C ansc=0;
76     int i;
77     bellman(s);
78     while(1){
79         pair <F, C> p = search(s,t);
80         if(!used[t]) break;
81         ansf += p.first; ansc += p.second;
82     }
83     return make_pair(ansf,ansc);
84 }
85 ///////////////////////////////////////////////////
86 int main() {
87     return 0;
88 }

```

2.7 Bellman Ford

```

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

```

2.8 Dijkstra

```

1 const int MAXN = 10    1000 + 10;
2 const ll INF = 1e9;
3 ll dis[MAXN];
4 set<pii> s;
5 bool mark[MAXN];
6 vector<pii> adj[MAXN];
7 void dij(int start) {
8     for (int i = 0; i < MAXN; i++) dis[i] = INF;
9     CLR(mark, 0); s.clear();
10    mark[start] = true;
11    dis[start] = 0;
12    s.insert(mp(0, start));
13    while (SZ(s)) {
14        int x = s.begin()->Y; s.erase(s.begin());
15        for (int i = 0; i < SZ(adj[x]); i++) {
16            int v = adj[x][i].X, w = adj[x][i].Y;
17            if (dis[v] > dis[x] + w) {
18                if (mark[v]) s.erase(mp(dis[v], v));
19                else mark[v] = true;
20
21                dis[v] = dis[x] + w;
22                s.insert(mp(dis[v], v));
23            }

```

```

24     }
25 }
26 }

```

2.9 Prim

```

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

```

2.10 DSU

```

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

```

2.11 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 }

```

2.12 Topological Sort

```

1 // This function uses performs a non-recursive topological sort.
2 // Running time:  $O(|V|^2)$ . If you use adjacency lists (vector<
3 //     map<int> >),
4 //     the running time is reduced to  $O(|E|)$ .
5 // INPUT:  w[i][j] = 1 if i should come before j, 0 otherwise
6 // OUTPUT: a permutation of 0,...,n-1 (stored in a vector)
7 //         which represents an ordering of the nodes which
8 //         is consistent with w
9 // If no ordering is possible, false is returned.
10 #include <iostream>
11 #include <queue>
12 #include <cmath>
13 #include <vector>
14 using namespace std;
15 typedef double T;
16 typedef vector<T> VT;
17 typedef vector<VT> VVT;
18 typedef vector<int> VI;
19 typedef vector<VI> VVI;
20 bool TopologicalSort (const VVI &w, VI &order){
21     int n = w.size();
22     VI parents (n);
23     queue<int> q;
24     order.clear();
25     for (int i = 0; i < n; i++){
26         for (int j = 0; j < n; j++)
27             if (w[j][i]) parents[i]++;
28         if (parents[i] == 0) q.push (i);
29     }

```



```

29 while (q.size() > 0){
30     int i = q.front();
31     q.pop();
32     order.push_back (i);
33     for (int j = 0; j < n; j++) if (w[i][j]){
34         parents[j]--;
35         if (parents[j] == 0) q.push (j);
36     }
37 }
38 return (order.size() == n);
39 }

```

2.13 Kardak General Graph matching3

```

1 // GETS:
2 // V->number of vertices
3 // E->number of edges
4 // pair of vertices as edges (vertices are 1..V) O(E V^2)
5 // GIVES:
6 // output of edmonds() is the maximum matching
7 // match[i] is matched pair of i (-1 if there isn't a matched
  pair)
8 #include <bits/stdc++.h>
9 using namespace std;
10 const int M=500;
11 struct struct_edge{int v;struct_edge n;};
12 typedef struct_edge edge;
13 struct_edge pool[M M 2];
14 edge top=pool,adj[M];
15 int V,E,match[M],qh,qt,q[M],father[M],base[M];
16 bool inq[M],inb[M],ed[M][M];
17 void add_edge(int u,int v){
18     top->v=v,top->n=adj[u],adj[u]=top++;
19     top->v=u,top->n=adj[v],adj[v]=top++;
20 }
21 int LCA(int root,int u,int v){
22     static bool inp[M];
23     memset(inp,0,sizeof(inp));
24     while(1)
25     {
26         inp[u=base[u]]=true;
27         if (u==root) break;

```

```

28         u=father[match[u]];
29     }
30     while(1)
31     {
32         if (inp[v=base[v]]) return v;
33         else v=father[match[v]];
34     }
35 }
36 void mark_blossom(int lca,int u){
37     while (base[u]!=lca)
38     {
39         int v=match[u];
40         inb[base[u]]=inb[base[v]]=true;
41         u=father[v];
42         if (base[u]!=lca) father[u]=v;
43     }
44 }
45 void blossom_contraction(int s,int u,int v){
46     int lca=LCA(s,u,v);
47     memset(inb,0,sizeof(inb));
48     mark_blossom(lca,u);
49     mark_blossom(lca,v);
50     if (base[u]!=lca)
51         father[u]=v;
52     if (base[v]!=lca)
53         father[v]=u;
54     for (int u=0;u<V;u++)
55         if (inb[base[u]])
56         {
57             base[u]=lca;
58             if (!inq[u])
59                 inq[q[qt]=u]=true;
60         }
61 }
62 int find_augmenting_path(int s){
63     memset(inq,0,sizeof(inq));
64     memset(father,-1,sizeof(father));
65     for (int i=0;i<V;i++) base[i]=i;
66     inq[q[qh=qt=0]=s]=true;
67     while (qh<=qt)
68     {

```

```

69     int u=q[qh++];
70     for (edge e=adj[u];e;e=e->n)
71     {
72         int v=e->v;
73         if (base[u]!=base[v]&&match[u]!=v)
74             if ((v==s)|| (match[v]!=-1 && father[match[v]]!=-1))
75                 blossom_contraction(s,u,v);
76         else if (father[v]==-1)
77             {
78                 father[v]=u;
79                 if (match[v]==-1)
80                     return v;
81                 else if (!inq[match[v]])
82                     inq[q[++qt]=match[v]]=true;
83             }
84     }
85 }
86 return -1;
87 }
88 int augment_path(int s,int t){
89     int u=t,v,w;
90     while (u!=-1)
91     {
92         v=father[u];
93         w=match[v];
94         match[v]=u;
95         match[u]=v;
96         u=w;
97     }
98     return t!=-1;
99 }
100 int edmonds(){
101     int matchc=0;
102     memset(match,-1,sizeof(match));
103     for (int u=0;u<V;u++)
104         if (match[u]==-1)
105             matchc+=augment_path(u,find_augmenting_path(u));
106     return matchc;
107 }
108 int main(){
109     int u,v;

```

```

110     cin>>V>>E;
111     while(E-->0)
112     {
113         cin>>u>>v;
114         if (!ed[u-1][v-1])
115         {
116             add_edge(u-1,v-1);
117             ed[u-1][v-1]=ed[v-1][u-1]=true;
118         }
119     }
120     cout<<edmonds()<<endl;
121     for (int i=0;i<V;i++)
122         if (i<match[i])
123             cout<<i+1<<" "<<match[i]+1<<endl;
124 }

```

3 Geometry

3.1 Geometry

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <cassert>
5  using namespace std;
6  double INF = 1e100;
7  double EPS = 1e-12;
8  struct PT {
9      double x, y;
10     PT() {}
11     PT(double x, double y) : x(x), y(y) {}
12     PT(const PT &p) : x(p.x), y(p.y) {}
13     PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
14     PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
15     PT operator * (double c) const { return PT(x*c, y*c); }
16     PT operator / (double c) const { return PT(x/c, y/c); }
17 };

```

```

18 double dot(PT p, PT q)    { return p.x q.x+p.y q.y; }
19 double dist2(PT p, PT q)  { return dot(p-q,p-q); }
20 double cross(PT p, PT q)  { return p.x q.y-p.y q.x; }
21 ostream &operator<<(ostream &os, const PT &p) {
22     return os << "(" << p.x << "," << p.y << ")";
23 }
24 // if movement from a to b to c is done in a CW path returns 1
25 // else if it's CCW returns -1 and if they make a line returns 0
26 int IsCWTurn(PT a, PT b, PT c) {
27     double r = cross((b - c), (a - c));
28     return (fabs(r) < EPS)? 0: (r > 0)? 1: -1;
29 }
30 // rotate a point CCW or CW around the origin
31 PT RotateCCW90(PT p)    { return PT(-p.y,p.x); }
32 PT RotateCW90(PT p)     { return PT(p.y,-p.x); }
33 PT RotateCCW(PT p, double t) {
34     return PT(p.x cos(t)-p.y sin(t), p.x sin(t)+p.y cos(t));
35 }
36 // project point c onto line through a and b
37 // assuming a != b
38 PT ProjectPointLine(PT a, PT b, PT c) {
39     return a + (b-a) dot(c-a, b-a)/dot(b-a, b-a);
40 }
41 // project point c onto line segment through a and b
42 PT ProjectPointSegment(PT a, PT b, PT c) {
43     double r = dot(b-a,b-a);
44     if (fabs(r) < EPS) return a;
45     r = dot(c-a, b-a)/r;
46     return (r < 0)? a: (r > 1)? b: a + (b - a) r;
47 }
48 // compute distance from c to segment between a and b
49 double DistancePointSegment(PT a, PT b, PT c) {
50     return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
51 }
52 // compute distance between point (x,y,z) and plane ax+by+cz=d
53 double DistancePointPlane(double x, double y, double z, double a,
54     double b,
55     double c, double d) {
56     return fabs(a x+b y+c z-d)/sqrt(a a+b b+c c);
57 }
58 // determine if lines from a to b and c to d are parallel or

```

```

collinear
58 bool LinesParallel(PT a, PT b, PT c, PT d) {
59     return fabs(cross(b-a, c-d)) < EPS;
60 }
61 bool LinesCollinear(PT a, PT b, PT c, PT d) {
62     return LinesParallel(a, b, c, d)
63     && fabs(cross(a-b, a-c)) < EPS
64     && fabs(cross(c-d, c-a)) < EPS;
65 }
66 // determine if line segment from a to b intersects with
67 // line segment from c to d
68 bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
69     if (LinesCollinear(a, b, c, d)) {
70         if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
71             dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
72         if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-
73             b) > 0)
74             return false;
75         return true;
76     }
77     if (cross(d-a, b-a)    cross(c-a, b-a) > 0) return false;
78     if (cross(a-c, d-c)    cross(b-c, d-c) > 0) return false;
79     return true;
80 }
81 // compute intersection of line passing through a and b
82 // with line passing through c and d, assuming that unique
83 // intersection exists; for segment intersection, check if
84 // segments intersect first
85 PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
86     b=b-a; d=c-d; c=c-a;
87     assert(dot(b, b) > EPS && dot(d, d) > EPS);
88     return a + b cross(c, d)/cross(b, d);
89 }
90 // compute center of circle given three points
91 PT ComputeCircleCenter(PT a, PT b, PT c) {
92     b=(a+b)/2;
93     c=(a+c)/2;
94     return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+
95         RotateCW90(a-c));
96 }
97 // determine if point is in a possibly non-convex polygon (by

```

```

100 // integer arithmetic by taking care of the division
101 // appropriately
102 // (making sure to deal with signs properly) and then by writing
103 // exact
104 // tests for checking point on polygon boundary
105 bool PointInPolygon(const vector<PT> &p, PT q) {
106     bool c = 0;
107     for (int i = 0; i < p.size(); i++) {
108         int j = (i+1)%p.size();
109         if (((p[i].y <= q.y && q.y < p[j].y) || (p[j].y <= q.y &&
110             q.y < p[i].y)) &&
111             q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j]
112             ].y - p[i].y))
113             c = !c;
114     }
115     return c;
116 }
117 // determine if point is on the boundary of a polygon
118 bool PointOnPolygon(const vector<PT> &p, PT q) {
119     for (int i = 0; i < p.size(); i++)
120         if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q)
121             < EPS)
122             return true;
123     return false;
124 }
125 // compute intersection of line through points a and b with
126 // circle centered at c with radius r > 0
127 vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
128     vector<PT> ret;
129     b = b-a;
130     a = a-c;
131     double A = dot(b, b);
132     double B = dot(a, b);
133     double C = dot(a, a) - r*r;
134     double D = B*B - A*C;

```

```

129     if (D < -EPS) return ret;
130     ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
131     if (D > EPS)
132         ret.push_back(c+a+b*(-B-sqrt(D))/A);
133     return ret;
134 }
135 // compute intersection of circle centered at a with radius r
136 // with circle centered at b with radius R
137 vector<PT> CircleCircleIntersection(PT a, PT b, double r, double
138     R) {
139     vector<PT> ret;
140     double d = sqrt(dist2(a, b));
141     if (d > r+R || d+min(r, R) < max(r, R)) return ret;
142     double x = (d*d-R*R+r*r)/(2*d);
143     double y = sqrt(r*r-x*x);
144     PT v = (b-a)/d;
145     ret.push_back(a+v*x + RotateCCW90(v)*y);
146     if (y > 0)
147         ret.push_back(a+v*x - RotateCCW90(v)*y);
148     return ret;
149 }
150 // This code computes the area or centroid of a (possibly
151 // nonconvex)
152 // polygon, assuming that the coordinates are listed in a
153 // clockwise or
154 // counterclockwise fashion. Note that the centroid is often
155 // known as
156 // the "center of gravity" or "center of mass".
157 double ComputeSignedArea(const vector<PT> &p) {
158     double area = 0;
159     for (int i = 0; i < p.size(); i++) {
160         int j = (i+1) % p.size();
161         area += p[i].x*p[j].y - p[j].x*p[i].y;
162     }
163     return area / 2.0;
164 }
165 double ComputeArea(const vector<PT> &p) {
166     return fabs(ComputeSignedArea(p));
167 }
168 PT ComputeCentroid(const vector<PT> &p) {
169     PT c(0,0);

```

```

166 double scale = 6.0 ComputeSignedArea(p);
167 for (int i = 0; i < p.size(); i++){
168     int j = (i+1) % p.size();
169     c = c + (p[i]+p[j]) (p[i].x p[j].y - p[j].x p[i].y);
170 }
171 return c / scale;
172 }
173 // tests whether or not a given polygon (in CW or CCW order) is
174 // simple
175 bool IsSimple(const vector<PT> &p) {
176     for (int i = 0; i < p.size(); i++) {
177         for (int k = i+1; k < p.size(); k++) {
178             int j = (i+1) % p.size();
179             int l = (k+1) % p.size();
180             if (i == l || j == k) continue;
181             if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
182                 return false;
183         }
184     }
185     return true;
186 }

```

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 // Running time: O(n log n)
7 // INPUT: a vector of input points, unordered.
8 // OUTPUT: a vector of points in the convex hull,
9 // counterclockwise, starting
10 // with bottommost/leftmost point
11 #include <cstdio>
12 #include <cassert>
13 #include <vector>
14 #include <algorithm>
15 #include <cmath>
16 using namespace std;
17 #define REMOVE_REDUNDANT
18 typedef double T;

```

```

16 const T EPS = 1e-7;
17 struct PT {
18     T x, y;
19     PT() {}
20     PT(T x, T y) : x(x), y(y) {}
21     bool operator<(const PT &rhs) const { return make_pair(y,x) <
22         make_pair(rhs.y,rhs.x); }
23     bool operator==(const PT &rhs) const { return make_pair(y,x)
24         == make_pair(rhs.y,rhs.x); }
25 };
26 T cross(PT p, PT q) { return p.x q.y-p.y q.x; }
27 T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) +
28     cross(c,a); }
29 #ifdef REMOVE_REDUNDANT
30 bool between(const PT &a, const PT &b, const PT &c) {
31     return (fabs(area2(a,b,c)) < EPS && (a.x-b.x) (c.x-b.x) <= 0
32         && (a.y-b.y) (c.y-b.y) <= 0);
33 }
34 #endif
35 void ConvexHull(vector<PT> &pts) {
36     sort(pts.begin(), pts.end());
37     pts.erase(unique(pts.begin(), pts.end()), pts.end());
38     vector<PT> up, dn;
39     for (int i = 0; i < pts.size(); i++) {
40         while (up.size() > 1 && area2(up[up.size()-2], up.back(),
41             pts[i]) >= 0) up.pop_back();
42         while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(),
43             pts[i]) <= 0) dn.pop_back();
44         up.push_back(pts[i]);
45         dn.push_back(pts[i]);
46     }
47     pts = dn;
48     for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(
49         up[i]);
50     #ifdef REMOVE_REDUNDANT
51     if (pts.size() <= 2) return;
52     dn.clear();
53     dn.push_back(pts[0]);
54     dn.push_back(pts[1]);
55     for (int i = 2; i < pts.size(); i++) {
56         if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn

```

```

    .pop_back();
    dn.push_back(pts[i]);
}
if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
    dn[0] = dn.back();
    dn.pop_back();
}
pts = dn;
#endif
}

```

3.3 Kardak Find intersection between lines

```

1 struct Point
2 {
3     int x, y;
4 };
5
6 // A line segment with left as Point
7 // with smaller x value and right with
8 // larger x value.
9 struct Segment
10 {
11     Point left, right;
12 };
13 struct Event {
14     int x, y;
15     bool isLeft;
16     int index;
17     Event(int x, int y, bool l, int i) : x(x), y(y), isLeft(l),
18         index(i) {}
19     bool operator<(const Event& e) const {
20         if(y==e.y) return x<e.x;
21         return y < e.y;
22     }
23 };
24 // point q lies on line segment 'pr'
25 bool onSegment(Point p, Point q, Point r)
26 {
27     if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
28         q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
29         return true;

```

```

29     return false;
30 }
31 int orientation(Point p, Point q, Point r)
32 {
33     int val = (q.y - p.y) * (r.x - q.x) -
34             (q.x - p.x) * (r.y - q.y);
35     if (val == 0) return 0; // collinear
36     return (val > 0)? 1: 2; // 1clock or 2counterclock wise
37 }
38 // The main function that returns true if line segment 'p1q1'
39 // and 'p2q2' intersect.
40 bool doIntersect(Segment s1, Segment s2)
41 {
42     Point p1 = s1.left, q1 = s1.right, p2 = s2.left, q2 = s2.right;
43     int o1 = orientation(p1, q1, p2);
44     int o2 = orientation(p1, q1, q2);
45     int o3 = orientation(p2, q2, p1);
46     int o4 = orientation(p2, q2, q1);
47     if (o1 != o2 && o3 != o4)
48         return true;
49     if (o1 == 0 && onSegment(p1, p2, q1)) return true;
50     if (o2 == 0 && onSegment(p1, q2, q1)) return true;
51     if (o3 == 0 && onSegment(p2, p1, q2)) return true;
52     if (o4 == 0 && onSegment(p2, q1, q2)) return true;
53     return false; // Doesn't fall in any of the above cases
54 }
55 set<Event>::iterator pred(set<Event> &s, set<Event>::iterator it)
56 {
57     return it == s.begin() ? s.end() : --it;
58 }
59 set<Event>::iterator succ(set<Event> &s, set<Event>::iterator it)
60 {
61     return ++it;
62 }
63 // Returns true if any two lines intersect.
64 int isIntersect(Segment arr[], int n)
65 {
66     unordered_map<string, int> mp;
67     vector<Event> e;
68     for (int i = 0; i < n; ++i) {
69         e.push_back(Event(arr[i].left.x, arr[i].left.y, true, i));

```

```

68     e.push_back(Event(arr[i].right.x, arr[i].right.y, false, i));
69 }
70 sort(e.begin(), e.end(), [](Event &e1, Event &e2) {return e1.x
71 < e2.x;});
72 set<Event> s;
73 int ans=0;
74 for (int i=0; i<2n; i++)
75 {
76     Event curr = e[i];
77     int index = curr.index;
78     if (curr.isLeft)
79     {
80         auto next = s.lower_bound(curr);
81         auto prev = pred(s, next);
82         bool flag=false;
83         if (next != s.end() && doIntersect(arr[next->index], arr[
84 index])){
85             string s=to_string(next->index+1)+" "+to_string(index+1);
86             if(mp.count(s)==0){mp[s]++;ans++;} //if not already
87 checked we can increase count in map
88         }
89         if (prev != s.end() && doIntersect(arr[prev->index], arr[
90 index])){
91             string s=to_string(prev->index+1)+" "+to_string(index
92 +1);
93             if(mp.count(s)==0){mp[s]++;ans++;} //if not already
94 checked we can increase count in map
95         }
96         if (prev != s.end() && next != s.end() && next->index==prev
97 ->index)ans--;
98         s.insert(curr);
99     }
100     else
101     {
102         auto it=s.find(Event(arr[index].left.x, arr[index].left.y,
103 true, index));
104         auto next = succ(s, it);
105         auto prev = pred(s, it);
106         if (next != s.end() && prev != s.end())
107         { string s=to_string(next->index+1)+" "+to_string(prev->
108 index+1);

```

```

100         string s1=to_string(prev->index+1)+" "+to_string(next->
101 index+1);
102         if (mp.count(s)==0&&mp.count(s1)==0&&doIntersect(arr[prev
103 ->index], arr[next->index]))
104             ans++;
105             mp[s]++;
106         }
107         s.erase(it);
108     }
109 }
110 for(auto &pr:mp){
111     cout<<pr.first<<"\n";
112 }
113 return ans;
114 }
115 int main() {
116     Segment arr[] = { {{1, 5}, {4, 5}}, {{2, 5}, {10, 1}},{{3, 2},
117 {10, 3}},{{6, 4}, {9, 4}},{{7, 1}, {8, 1}}};
118     int n = sizeof(arr)/sizeof(arr[0]);
119     cout<<isIntersect(arr, n);
120     return 0;
121 }

```

3.4 Kardak Half Plane Intersection

```

1
2 const long double eps = 1e-9, inf = 1e9;
3 struct Point {
4     long double x, y;
5     explicit Point(long double x = 0, long double y = 0) : x(x),
6     y(y) {}
7     friend Point operator + (const Point& p, const Point& q) {
8         return Point(p.x + q.x, p.y + q.y);
9     }
10    friend Point operator - (const Point& p, const Point& q) {
11        return Point(p.x - q.x, p.y - q.y);
12    }
13    friend Point operator * (const Point& p, const long double& k
14    ) {
15        return Point(p.x * k, p.y * k);
16    }
17    friend long double cross(const Point& p, const Point& q) {

```

```

16     return p.x    q.y - p.y    q.x;
17 }
18 };
19 struct Halfplane {
20     // 'p' is a passing point of the line and 'pq' is the
21     // direction vector of the line.
22     Point p, pq;
23     long double angle;
24     Halfplane() {}
25     Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
26         angle = atan2l(pq.y, pq.x);
27     }
28     // Every half-plane allows the region to the LEFT of its line
29     .
30     bool out(const Point& r) {
31         return cross(pq, r - p) < -eps;
32     }
33     bool operator < (const Halfplane& e) const {
34         if (fabsl(angle - e.angle) < eps) return cross(pq, e.p -
35         p) < 0;
36         return angle < e.angle;
37     }
38     bool operator == (const Halfplane& e) const {
39         return fabsl(angle - e.angle) < eps;
40     }
41     friend Point inter(const Halfplane& s, const Halfplane& t) {
42         long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq
43         , t.pq);
44         return s.p + (s.pq    alpha);
45     }
46 };
47 vector<Point> hp_intersect(vector<Halfplane>& H) {
48     Point box[4] = { // Bounding box in CCW order
49         Point(1e9, 1e9),
50         Point(-1e9, 1e9),
51         Point(-1e9, -1e9),
52         Point(1e9, -1e9)
53     };
54     for(int i = 0; i < 4; i++) { // Add bounding box half-planes.
55         Halfplane aux(box[i], box[(i+1) % 4]);

```

```

53         H.push_back(aux);
54     }
55     sort(H.begin(), H.end());
56     H.erase(unique(H.begin(), H.end()), H.end());
57     deque<Halfplane> dq;
58     int len = 0;
59     for(int i = 0; i < int(H.size()); i++) {
60         while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2])))
61         {
62             dq.pop_back();
63             --len;
64         }
65         while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
66             dq.pop_front();
67             --len;
68         }
69         dq.push_back(H[i]);
70         ++len;
71     }
72     while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
73         dq.pop_back();
74         --len;
75     }
76     while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
77         dq.pop_front();
78         --len;
79     }
80     if (len < 3) return vector<Point>();
81     vector<Point> ret(len);
82     for(int i = 0; i+1 < len; i++) {
83         ret[i] = inter(dq[i], dq[i+1]);
84     }
85     ret.back() = inter(dq[len-1], dq[0]);
86     return ret;

```

4 Data Structures

4.1 Fenwick1

```

1 const int MAXN = 1    1000 + 10;

```



```

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

```

4.2 Fenwick2

```

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

```

4.3 Segment Tree Lazy Propagation

```

1 int seg[4 * MAXN], add[4 * MAXN];
2 inline void shift(int x, int s, int e) {
3     int lc = x + x + 0, rc = x + x + 1;
4     int mid = (s + e) / 2;
5     int l1 = mid - s, l2 = e - mid;
6     seg[lc] += l1 * add[x];
7     seg[rc] += l2 * add[x];

```

```

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

```

4.4 RMQ

```

1 const int N = 1000 * 100 + 5, LOG = 20;
2 class RMQ {
3     int f[LOG][N], Lgl[N], S;
4 public:
5     RMQ() {
6         for (int i = 1, p = 0; i < N; ++i) {
7             if (i == 1 << (p + 1))
8                 ++p;
9             Lgl[i] = p;
10        }

```

```

11 }
12 void build(int a[], int n) {
13     for(int i = 0; i < n; ++i)
14         f[0][i] = a[i];
15
16     for(int j = 1, p = 1; j < LOG; ++j, p = 2)
17         for(int i = 0; i < n; ++i) {
18             f[j][i] = f[j - 1][i];
19             if(i + p < n)
20                 f[j][i] = min(f[j - 1][i], f[j - 1][i + p]);
21         }
22 }
23 int find(int s, int e) {
24     int l = Lgl[e - s + 1];
25     return min(f[l][s], f[l][e + 1 - (1 << l)]);
26 }
27 };

```

4.5 Trie

```

1 struct Node {
2     char x;
3     vector<Node> adj;
4     Node () {
5         x = 0;
6     }
7     Node (char a) {
8         x = a;
9     }
10    Node add_edge(char a) {
11        for (int i = 0; i < SZ(adj); i++)
12            if (adj[i] -> x == a)
13                return adj[i];
14        adj.pb(new Node(a));
15        return adj.back();
16    }
17 };
18 struct Trie {
19     Node root;
20     Trie() {
21         root = new Node();
22     }

```

```

23 void add(string &s) {
24     add(s, 0, root);
25 }
26 void add(string &s, int pos, Node node) {
27     if (pos == SZ(s)) {
28         return ;
29     } else {
30         Node next = node -> add_edge(s[pos]);
31         add(s, pos + 1, next);
32     }
33 }
34 };

```

5 String

5.1 Hash

```

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

```

5.2 KMP

```

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

```

```

11     f[i] = ++pos;
12 }
13 pos = 0;
14 for (int i = 0; i < SZ(s); i++) {
15     while(pos != -1 && (pos == SZ(t) || s[i] != t[pos])) pos = f[
16         pos];
17     pos++;
18     if (pos == SZ(t)) match[i] = 1;
19     else match[i] = 0;
20 }

```

5.3 Suffix Array

```

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

```

```

29         k = max(k, 0);
30         while(s[i + k] == s[sa[rank[i] + 1] + k])
31             ++k;
32         lcp[rank[i]] = k;
33     }
34     else
35         k = 0;
36 }
37 };

```

5.4 Kardak AhoCorasick

```

1 // for string matching
2 using namespace std;
3 #include <bits/stdc++.h>
4 // Max number of states in the matching machine.
5 // Should be equal to the sum of the length of all keywords.
6 const int MAXS = 500;
7 // Maximum number of characters in input alphabet
8 const int MAXC = 26;
9 // OUTPUT FUNCTION IS IMPLEMENTED USING out[]
10 // Bit i in this mask is one if the word with index i
11 // appears when the machine enters this state.
12 int out[MAXS];
13 // FAILURE FUNCTION IS IMPLEMENTED USING f[]
14 int f[MAXS];
15 // GOTO FUNCTION (OR TRIE) IS IMPLEMENTED USING g[][]
16 int g[MAXS][MAXC];
17 // Builds the string matching machine.
18 // arr — array of words. The index of each keyword is important:
19 //     "out[state] & (1 << i)" is > 0 if we just found word[i]
20 //     in the text.
21 // Returns the number of states that the built machine has.
22 // States are numbered 0 up to the return value - 1, inclusive.
23 int buildMatchingMachine(string arr[], int k)
24 {
25     memset(out, 0, sizeof out);
26     memset(g, -1, sizeof g);
27     int states = 1;
28     for (int i = 0; i < k; ++i)
29     {
30         const string &word = arr[i];

```

```

31 int currentState = 0;
32 for (int j = 0; j < word.size(); ++j)
33 {
34     int ch = word[j] - 'a';
35     if (g[currentState][ch] == -1)
36         g[currentState][ch] = states++;
37     currentState = g[currentState][ch];
38 }
39 out[currentState] |= (1 << i);
40 }
41 for (int ch = 0; ch < MAXC; ++ch)
42     if (g[0][ch] == -1)
43         g[0][ch] = 0;
44 memset(f, -1, sizeof f);
45 queue<int> q;
46 for (int ch = 0; ch < MAXC; ++ch)
47 {
48     if (g[0][ch] != 0)
49     {
50         f[g[0][ch]] = 0;
51         q.push(g[0][ch]);
52     }
53 }
54 while (q.size())
55 {
56     int state = q.front();
57     q.pop();
58     for (int ch = 0; ch <= MAXC; ++ch)
59     {
60         if (g[state][ch] != -1)
61         {
62             int failure = f[state];
63             while (g[failure][ch] == -1)
64                 failure = f[failure];
65             failure = g[failure][ch];
66             f[g[state][ch]] = failure;
67             out[g[state][ch]] |= out[failure];
68             q.push(g[state][ch]);
69         }
70     }
71 }

```

```

72 return states;
73 }
74 // Returns the next state the machine will transition to using
75 // goto
76 // and failure functions.
77 // currentState — The current state of the machine. Must be
78 // between
79 // 0 and the number of states — 1, inclusive.
80 // nextInput — The next character that enters into the machine.
81 int findNextState(int currentState, char nextInput)
82 {
83     int answer = currentState;
84     int ch = nextInput - 'a';
85     while (g[answer][ch] == -1)
86         answer = f[answer];
87     return g[answer][ch];
88 }
89 // This function finds all occurrences of all array words
90 // in text.
91 void searchWords(string arr[], int k, string text)
92 {
93     buildMatchingMachine(arr, k);
94     int currentState = 0;
95     for (int i = 0; i < text.size(); ++i)
96     {
97         currentState = findNextState(currentState, text[i]);
98         if (out[currentState] == 0)
99             continue;
100         for (int j = 0; j < k; ++j)
101         {
102             if (out[currentState] & (1 << j))
103             {
104                 cout << "Word " << arr[j] << " appears from "
105                     << i - arr[j].size() + 1 << " to " << i << endl;
106             }
107         }
108     }
109 }
110 // Driver program to test above
111 int main()
112 {

```

```

111 string arr[] = {"he", "she", "hers", "his"};
112 string text = "ahishers";
113 int k = sizeof(arr)/sizeof(arr[0]);
114 searchWords(arr, k, text);
115 return 0;
116 }

```

6 Number Theory

6.1 Phi

```

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

```

6.2 370 SGU

```

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

```

6.3 Euclid

```

1 typedef vector<int> VI;
2 typedef pair<int, int> PII;
3 // computes gcd(a,b)
4 int gcd(int a, int b) {
5     while (b) { int t = a%b; a = b; b = t; }
6     return a;
7 }

```

```

8 // returns g = gcd(a, b); finds x, y such that d = ax + by
9 int extended_euclid(int a, int b, int &x, int &y) {
10     int xx = y = 0;
11     int yy = x = 1;
12     while (b) {
13         int q = a / b;
14         int t = b; b = a % b; a = t;
15         t = xx; xx = x - q * xx; x = t;
16         t = yy; yy = y - q * yy; y = t;
17     }
18     return a;
19 }
20 // finds all solutions to ax = b (mod n)
21 VI modular_linear_equation_solver(int a, int b, int n) {
22     int x, y;
23     VI ret;
24     int g = extended_euclid(a, n, x, y);
25     if (!(b % g)) {
26         x = mod(x * (b / g), n);
27         for (int i = 0; i < g; i++)
28             ret.push_back(mod(x + i * (n / g), n));
29     }
30     return ret;
31 }
32 // Chinese remainder theorem (special case): find z such that
33 // z % m1 = r1, z % m2 = r2. Here, z is unique modulo M = lcm(m1, m2).
34 // Return (z, M). On failure, M = -1.
35 PII chinese_remainder_theorem(int m1, int r1, int m2, int r2) {
36     int s, t;
37     int g = extended_euclid(m1, m2, s, t);
38     if (r1 % g != r2 % g) return make_pair(0, -1);
39     return make_pair(mod(s * r2 * m1 + t * r1 * m2, m1 * m2 / g), m1 * m2 / g);
40 }
41 // Chinese remainder theorem: find z such that
42 // z % m[i] = r[i] for all i. Note that the solution is
43 // unique modulo M = lcm_i (m[i]). Return (z, M). On
44 // failure, M = -1. Note that we do not require the a[i]'s
45 // to be relatively prime.
46 PII chinese_remainder_theorem(const VI &m, const VI &r) {
47     PII ret = make_pair(r[0], m[0]);

```

```

48     for (int i = 1; i < m.size(); i++) {
49         ret = chinese_remainder_theorem(ret.second, ret.first, m[i],
50             r[i]);
51         if (ret.second == -1) break;
52     }
53     return ret;

```

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 ll fct[MAXN], rev[MAXN], fct_rev[MAXN];
10 void init(int n) {
11     fct[0] = 1;
12     for (int i = 1; i <= n; i++)
13         fct[i] = (fct[i - 1] * i) % MOD;
14
15     rev[0] = 1;
16     for (int i = 1; i <= n; i++)
17         rev[i] = bin_pow(i, MOD - 2);
18
19     fct_rev[0] = 1;
20     for (int i = 1; i <= n; i++)
21         fct_rev[i] = (fct_rev[i - 1] * rev[i]) % MOD;
22 }
23 int C(int n, int r) {
24     return (((fct[n] * fct_rev[r]) % MOD) * fct_rev[n - r]) % MOD;
25 }

```

6.5 FFT

```

1 typedef long double DOUBLE;
2 typedef complex<DOUBLE> COMPLEX;
3 typedef vector<DOUBLE> VD;
4 typedef vector<COMPLEX> VC;

```

```

5 struct FFT {
6     VC A;
7     int n, L;

8     int ReverseBits(int k) {
9         int ret = 0;
10        for (int i = 0; i < L; i++) {
11            ret = (ret << 1) | (k & 1);
12            k >>= 1;
13        }
14        return ret;
15    }
16 }
17 void BitReverseCopy(VC a) {
18     for (n = 1, L = 0; n < a.size(); n <= 1, L++) ;
19     A.resize(n);
20     for (int k = 0; k < n; k++)
21         A[ReverseBits(k)] = a[k];
22 }
23 VC DFT(VC a, bool inverse) {
24     BitReverseCopy(a);
25     for (int s = 1; s <= L; s++) {
26         int m = 1 << s;
27         COMPLEX wm = exp(COMPLEX(0, 2.0 * M_PI / m));
28         if (inverse) wm = COMPLEX(1, 0) / wm;
29         for (int k = 0; k < n; k += m) {
30             COMPLEX w = 1;
31             for (int j = 0; j < m/2; j++) {
32                 COMPLEX t = w * A[k + j + m/2];
33                 COMPLEX u = A[k + j];
34                 A[k + j] = u + t;
35                 A[k + j + m/2] = u - t;
36                 w = w * wm;
37             }
38         }
39     }
40     if (inverse) for (int i = 0; i < n; i++) A[i] /= n;
41     return A;
42 }
43 // c[k] = sum_{i=0}^k a[i] b[k-i]
44 VD Convolution(VD a, VD b) {
45     int L = 1;

```

```

46     while ((1 << L) < a.size()) L++;
47     while ((1 << L) < b.size()) L++;
48     int n = 1 << (L+1);
49     VC aa, bb;
50     for (size_t i = 0; i < n; i++) aa.push_back(i < a.size()
51 ? COMPLEX(a[i], 0) : 0);
52     for (size_t i = 0; i < n; i++) bb.push_back(i < b.size()
53 ? COMPLEX(b[i], 0) : 0);
54     VC AA = DFT(aa, false);
55     VC BB = DFT(bb, false);
56     VC CC;
57     for (size_t i = 0; i < AA.size(); i++) CC.push_back(AA[i]
58 BB[i]);
59     VC cc = DFT(CC, true);
60     VD c;
61     for (int i = 0; i < a.size() + b.size() - 1; i++) c.
push_back(cc[i].real());
62     return c;
63 }

```

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 const double EPS = 1e-10;
17 typedef vector<int> VI;
18 typedef double T;
19 typedef vector<T> VT;

```

```

20 typedef vector<VT> VVT;
21 T GaussJordan(VVT &a, VVT &b) {
22     const int n = a.size();
23     const int m = b[0].size();
24     VI irow(n), icol(n), ipiv(n);
25     T det = 1;
26     for (int i = 0; i < n; i++) {
27         int pj = -1, pk = -1;
28         for (int j = 0; j < n; j++) if (!ipiv[j])
29             for (int k = 0; k < n; k++) if (!ipiv[k])
30                 if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk
31                     = k; }
32         if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." <<
33             endl; exit(0); }
34         ipiv[pk]++;
35         swap(a[pj], a[pk]);
36         swap(b[pj], b[pk]);
37         if (pj != pk) det = -1;
38         irow[i] = pj;
39         icol[i] = pk;
40         T c = 1.0 / a[pk][pk];
41         det = a[pk][pk];
42         a[pk][pk] = 1.0;
43         for (int p = 0; p < n; p++) a[pk][p] = c;
44         for (int p = 0; p < m; p++) b[pk][p] = c;
45         for (int p = 0; p < n; p++) if (p != pk) {
46             c = a[p][pk];
47             a[p][pk] = 0;
48             for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
49             for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
50         }
51     }
52     for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
53         for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]
54             ]]);
55     }
56     return det;
57 }
58 int main() {
59     const int n = 4;
60     const int m = 2;

```

```

58 double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
59 double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
60 VVT a(n), b(n);
61 for (int i = 0; i < n; i++) {
62     a[i] = VT(A[i], A[i] + n);
63     b[i] = VT(B[i], B[i] + m);
64 }
65 double det = GaussJordan(a, b);
66 }

```

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 int main() {
3     int n;
4     cin >> n;

```



```

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

```

7.3 Divide and Conquer Tree

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4 const int N = 1000 * 100 + 5;
5 vector <int> adj[N];
6 int is_av[N], _sz[N]; //XXX initiate is_av to 1
7 void set_size(int v, int p) {
8     _sz[v] = 1;
9     for(int u:adj[v])
10         if(u != p && is_av[u]) {
11             set_size(u, v);
12             _sz[v] += _sz[u];
13         }
14 }
15 void divide(int v) {

```

```

16 set_size(v, v);
17 int S = _sz[v], p = v;
18 sign:
19     for(int u:adj[v])
20         if(is_av[u] && u != p && _sz[u] > S / 2) {
21             p = v;
22             v = u;
23             goto sign;
24         }
25     // now v is the centroid of the tree
26     // Enter your code here
27 is_av[v] = 0;
28 for(int u:adj[v])
29     if(is_av[u])
30         divide(u);
31 }
32 int main() {
33     ios::sync_with_stdio(false);
34     int n;
35     cin >> n;
36     for(int i = 1; i < n; ++i) {
37         int a, b;
38         cin >> a >> b;
39         --a, --b;
40         adj[a].push_back(b);
41         adj[b].push_back(a);
42     }
43     fill(is_av, is_av + N, 1);
44     divide(0);
45     return 0;
46 }

```