Contents

1 Base

1.1 vimrc

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

1.2 Template

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define mp make_pair
#define SQR(a) ((a)   (a))
#define SZ(x) ((int) (x).size())
#define ALL(x) (x).begin(), (x).end()
#define CLR(x, a) memset(x, a, sizeof x)
#define VAL(x) #x << " = " << (x) << "    "
#define FOREACH(i, x) for(__typeof((x).begin()) i = (x).begin(); i != (x).end(); i ++)
#define FOR(i, n) for (int i = 0; i < (n); i ++)
#define X first
#define Y second
typedef long long ll;
typedef pair<ll, ll> pll;
typedef pair<int, int> pii;
const int MAXN = 1   1000 + 10;
int main () {
  ios::sync_with_stdio(false);
  return 0;
}
```

2 Graph

2.1 LCA

```
vector<int> adj[MAXN];
int par[MAXN][MAXL], h[MAXN];
bool mark[MAXN];
void dfs(int x) {
  mark[x] = true;
  for (int i = 0; i < SZ(adj[x]); i ++) {
    int v = adj[x][i];
    if (!mark[v]) {
      par[v][0] = x, h[v] = h[x] + 1;
      dfs(v);
    }
  }
}
int get_parent(int x, int k) {
  for (int i = 0; i < MAXL; i ++)
    if ((1 << i) & k) x = par[x][i];
  return x;
}
int lca(int x, int y) {
  if (h[y] > h[x]) swap(x, y);
  x = get_parent(x, h[x] - h[y]);
  if (x == y) return x;
  for (int i = MAXL - 1; i >= 0; i--)
    if (par[x][i] != par[y][i])
      x = par[x][i], y = par[y][i];
  return par[x][0];
}
int main () {
  par[0][0] = -1;
  dfs(0);
  for (int i = 1; i < MAXL; i ++)
    for (int j = 0; j < n; j ++)
      par[j][i] = par[par[j][i - 1]][i - 1];
}
```

2.2 SCC

```
vector <int> adj[N];
```

```
stack <int> S, P;
int mrk[N], ind, col[N], CL;
void dfs(int v) {
  mrk[v] = ++ind;
  S.push(v);
  P.push(v);
  for(int i = 0;i < Size(adj[v]);++i) {
    int u = adj[v][i];
    if(!mrk[u])
      dfs(u);
    else
      while(mrk[u] < mrk[S.top()])
        S.pop();
  }
  if(S.top() == v) {
    mrk[v] = INF;
    col[v] = ++CL;
    while(P.top() != v) {
      col[P.top()] = CL;
      mrk[P.top()] = INF;
      P.pop();
    }
    P.pop();
    S.pop();
  }
}
//main: for(int i = 1;i <= n;++i)
//      if(!mrk[i]) dfs(i);
```

2.3 Matching

```
int match[3][MAXN]; // 0 for first part, 1 for second part
bool mark[MAXN];
vector<int> adj[MAXN]; // adjacent list for first part nodes
int n, m, p;
// n: number of nodes in first part
// m: number of nodes in second part
// p: number of edges
bool dfs(int x) {
  if (mark[x]) return false;

  mark[x] = true;
  for (int i = 0; i < SZ(adj[x]); i ++) {
    int v = adj[x][i];
    if (match[1][v] == -1 || dfs(match[1][v])) {
      match[0][x] = v;
      match[1][v] = x;
      return true;
    }
  }
  return false;
}
void bi_match() {
  CLR(match, -1);
  for (int i = 0; i < n; i ++) {
    CLR(mark, 0);
    bool check = false;
    for (int j = 0; j < n; j ++)
      if (!mark[j] && match[0][j] == -1)
        check |= dfs(j);
    if (!check) break;
  }
}
int main () {
  cin >> n >> m >> p;
  for (int i = 0; i < p; i ++) {
    int x, y; cin >> x >> y; x --, y --;
    // x: a node in first part  [0, n)
    // y: a node in second part [0, m)
    adj[x].pb(y);
  }
  bi_match();
  int ans = 0;
  FOR(i, n) ans += (match[0][i] != -1);
  cout << ans << endl;
  return 0;
}
```

2.4 Max Flow BFS

```
#include <queue>
#include <cstring>
const int N = 100;
int mat[N][N];
```

```
int viz[N], network[N][N], parent[N];
bool anotherPath(int start, int end) {
  memset(viz, 0, sizeof viz);
  memset(parent, -1, sizeof parent);
  viz[start] = true;
  queue<int> q;
  q.push(start);
  while (!q.empty()) {
    int z = q.front(); q.pop();
    viz[z] = true;
    for (int i=0;i<N;i++) {
      if (network[z][i] <= 0 || viz[i]) continue;
      viz[i] = true;
      parent[i] = z;
      if (i == end) return true;
      q.push(i);
    }
  }
  return false;
}
int maxflow(int start, int end) {
  memcpy(network, mat, sizeof(mat));
  int total = 0;
  while (anotherPath(start, end)) {
    int flow = network[parent[end]][end];
    int curr = end;
    while (parent[curr] >= 0) {
      flow = min(flow, network[parent[curr]][curr]);
      curr = parent[curr];
    }
    curr = end;
    while (parent[curr] >= 0) {
      network[parent[curr]][curr]-=flow;
      network[curr][parent[curr]]+=flow;
      curr = parent[curr];
    }
    total += flow;
  }
  return total;
}
```

2.5 Max Flow Dinic

```
#include <iostream>
#include <queue>
using namespace std;
#define REP(i,n) for((i)=0;(i)<(int)(n);(i)++)
typedef int F;
#define F_INF (1<<29)
#define MAXV 10000
#define MAXE 1000000 // E 2!
F cap[MAXE], flow[MAXE];
int to[MAXE], _prev[MAXE], last[MAXV], used[MAXV], level[MAXV];
struct MaxFlow {
    int V, E;

    MaxFlow(int n) {
        int i;
        V = n; E = 0;
        REP(i,V) last[i] = -1;
    }
    void add_edge(int x, int y, F f) { //directed edge
        cap[E] = f; flow[E] = 0; to[E] = y;
        _prev[E] = last[x]; last[x] = E; E++;

        cap[E] = 0; flow[E] = 0; to[E] = x;
        _prev[E] = last[y]; last[y] = E; E++;
    }
    bool bfs(int s, int t){
        int i;
        REP(i,V) level[i] = -1;
        queue <int> q;
        q.push(s); level[s] = 0;
        while(!q.empty()){
            int x = q.front(); q.pop();
            for(i=last[x]; i>=0; i=_prev[i])
                if(level[to[i]] == -1 && cap[i] > flow[i]) {
                    q.push(to[i]);
                    level[to[i]] = level[x] + 1;
                }
        }
        return (level[t] != -1);
```

```
    }
    F dfs(int v, int t, F f){
        int i;
        if(v == t) return f;
        for(i=used[v]; i>=0; used[v]= i =_prev[i])
            if(level[to[i]] > level[v] && cap[i] > flow[i]) {
                F tmp = dfs(to[i], t, min(f, cap[i]-flow[i]));
                if(tmp > 0) {
                    flow[i] += tmp;
                    flow[i^1] -= tmp;
                    return tmp;
                }
            }
        return 0;
    }
    F maxflow(int s, int t) {
        int i;
        while(bfs(s,t)) {
            REP(i,V) used[i] = last[i];
            while(dfs(s,t,F_INF) != 0);
        }
        F ans = 0;
        for(i=last[s];i>=0;i=_prev[i])
            ans += flow[i];
        return ans;
    }
};
```

2.6 Min Cost Max Flow

```
#include <iostream>
#include <queue>
using namespace std;
#define REP(i,n) for((i)=0;(i)<(int)(n);(i)++)
//XXX change these lines!
typedef int F;
typedef long long C;
#define F_INF (1<<29)
#define C_INF (1LL<<60)
#define MAXV 3000
#define MAXE 10000 // E 2! [or E 4 for bidirected graphs]
//no need to initialize these variables!
int V,E;
F cap[MAXE];
C cost[MAXE],dist[MAXV],pot[MAXV];
int to[MAXE],prv[MAXE],last[MAXV],path[MAXV];
bool used[MAXV];
priority_queue <pair <C, int> > q;
F flow[MAXE];//output
class MinCostFlow {
public:
    MinCostFlow(int n);
    int add_edge(int x, int y, F w, C c); // zero based && directed!
    pair <F, C> mincostflow(int s, int t);
private:
    pair <F, C> search(int s, int t);
    void bellman(int s);
};
/////////////////////////BACKEND!///////////////////////
MinCostFlow::MinCostFlow(int n){
    V = n; E = 0;
    int i; REP(i,V) last[i] = -1;
}
int MinCostFlow::add_edge(int x, int y, F w, C c){
    cap[E] = w; flow[E] = 0; cost[E] = c; to[E] = y; prv[E] = last[x]; last[x] = E; E++;
    cap[E] = 0; flow[E] = 0; cost[E] = -c; to[E] = x; prv[E] = last[y]; last[y] = E; E++;
    return E-2;
}
void MinCostFlow::bellman(int s){
    int i,x,e;
    REP(i,V) pot[i] = C_INF;
    pot[s] = 0;
    REP(i,V+10) REP(x,V) for(e=last[x];e>=0;e=prv[e]) if(cap[e] > 0) pot[to[e]] = min(pot[to[e]], pot[x] + cost[e]);
}
pair <F, C> MinCostFlow::search(int s, int t){
    F ansf=0; C ansc=0;
    int i;
    REP(i,V) used[i] = false;
    REP(i,V) dist[i] = C_INF;
```

```
    dist[s] = 0; path[s] = -1; q.push(make_pair(0,s));
    while(!q.empty()){
        int x = q.top().second; q.pop();
        if(used[x]) continue; used[x] = true;
        for(int e=last[x];e>=0;e=prv[e]) if(cap[e] > 0){
            C tmp = dist[x] + cost[e] + pot[x] - pot[to[e]];
            if(tmp < dist[to[e]] && !used[to[e]]){
                dist[to[e]] = tmp;
                path[to[e]] = e;
                q.push(make_pair(-dist[to[e]],to[e]));
            }
        }
    }
    REP(i,V) pot[i] += dist[i];
    if(used[t]){
        ansf = F_INF;
        for(int e=path[t];e>=0;e=path[to[e^1]]) ansf = min(ansf, cap[e]);
        for(int e=path[t];e>=0;e=path[to[e^1]]) {ansc += cost[e]  ansf; cap[e] -= ansf; cap[e^1] += ansf; flow[e] += ansf; flow[e^1] -= ansf;}
    }
    return make_pair(ansf,ansc);
}
pair <F, C> MinCostFlow::mincostflow(int s, int t){
    F ansf=0; C ansc=0;
    int i;
    bellman(s);
    while(1){
        pair <F, C> p = search(s,t);
        if(!used[t]) break;
        ansf += p.first; ansc += p.second;
    }
    return make_pair(ansf,ansc);
}
////////////////////////////////////////////////////
int main() {
    return 0;
}
```

2.7 Bellman Ford

```
int n, m;
int ex[MAXN], ey[MAXN], ew[MAXN], d[MAXN];
bool bellman(int start) {
  FOR(i, n) d[i] = INF;
  d[start] = 0;
  FOR(i, n - 1) FOR(j, m) {
    int x = ex[j], y = ey[j]; double w = tw[j];
    d[y] = min(d[y], d[x] + w);
  }
  // check if graph has a negative cycle
  FOR(i, m) {
    int x = ex[i], y = ey[i]; double w = tw[i];
    if (d[y] > d[x] + w) return false;
  }
  return true;
}
```

2.8 Dijkstra

```
const int MAXN = 10   1000 + 10;
const ll INF = 1e9;
ll dis[MAXN];
set<pii> s;
bool mark[MAXN];
vector<pii> adj[MAXN];
void dij(int start) {
  for (int i = 0; i < MAXN; i ++) dis[i] = INF;
  CLR(mark, 0); s.clear();
  mark[start] = true;
  dis[start] = 0;
  s.insert(mp(0, start));
  while (SZ(s)) {
    int x = s.begin()->Y; s.erase(s.begin());
    for (int i = 0; i < SZ(adj[x]); i ++) {
      int v = adj[x][i].X, w = adj[x][i].Y;
      if (dis[v] > dis[x] + w) {
        if (mark[v]) s.erase(mp(dis[v], v));
        else mark[v] = true;

        dis[v] = dis[x] + w;
        s.insert(mp(dis[v], v));
      }
```

```
    }
  }
}
```

2.9 Prim

```
const int N = 1000   100 + 5;
vector <pii> adj[N];
int ans, mrk[N];
void prim(int v) {
  int w;
  set <pii> st;
  st.insert(mp(0, v));
  while(!st.empty()) {
    v = st.begin()-> Y;
    w = st.begin()-> X;
    st.erase(st.begin());
    if(mrk[v]++) continue;
    ans += w;

    for(int i = 0;i < Size(adj[v]);++i)
      if(!mrk[adj[v][i].Y])
        st.insert(adj[v][i]);
  }
}
```

2.10 DSU

```
int par[MAXN];
pair <int, pii> e[MAXN];
int father(int x) {
  return par[x] == -1 ? x : par[x] = father(par[x]);
}
bool merge(int x, int y) {
  x = father(x);
  y = father(y);
  if (x != y) par[y] = x;
  return x != y;
}
fill(par, par + n, -1);
```

2.11 Eulerian Tour

```
void euler(int x) {
  for (int i = 0; i < SZ(graph[x]); i ++) {
    int v = graph[x][i];
    if (!vis[x][v]) {
      vis[x][v] = vis[v][x] = true;
      euler(v);
    }
  }
  tour.pb(x);
}
```

2.12 Topological Sort

```
// This function uses performs a non-recursive topological sort.
// Running time: O(|V|^2).  If you use adjacency lists (vector<map<int> >),
//                 the running time is reduced to O(|E|).
//   INPUT:   w[i][j] = 1 if i should come before j, 0 otherwise
//   OUTPUT:  a permutation of 0,...,n-1 (stored in a vector)
//            which represents an ordering of the nodes which
//            is consistent with w
// If no ordering is possible, false is returned.
#include <iostream>
#include <queue>
#include <cmath>
#include <vector>
using namespace std;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
typedef vector<int> VI;
typedef vector<VI> VVI;
bool TopologicalSort (const VVI &w, VI &order){
  int n = w.size();
  VI parents (n);
  queue<int> q;
  order.clear();
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++)
      if (w[j][i]) parents[i]++;
      if (parents[i] == 0) q.push (i);
  }
```

```
  while (q.size() > 0){
    int i = q.front();
    q.pop();
    order.push_back (i);
    for (int j = 0; j < n; j++) if (w[i][j]){
      parents[j]--;
      if (parents[j] == 0) q.push (j);
    }
  }
  return (order.size() == n);
}
```

2.13 Kardak General Graph matching3

```
// GETS:
// V->number of vertices
// E->number of edges
// pair of vertices as edges (vertices are 1..V) O(E V^2)
// GIVES:
// output of edmonds() is the maximum matching
// match[i] is matched pair of i (-1 if there isn't a matched pair)
#include <bits/stdc++.h>
using namespace std;
const int M=500;
struct struct_edge{int v;struct_edge  n;};
typedef struct_edge  edge;
struct_edge pool[M M 2];
edge top=pool,adj[M];
int V,E,match[M],qh,qt,q[M],father[M],base[M];
bool inq[M],inb[M],ed[M][M];
void add_edge(int u,int v){
  top->v=v,top->n=adj[u],adj[u]=top++;
  top->v=u,top->n=adj[v],adj[v]=top++;
}
int LCA(int root,int u,int v){
  static bool inp[M];
  memset(inp,0,sizeof(inp));
  while(1)
    {
      inp[u=base[u]]=true;
      if (u==root) break;
```

```
      u=father[match[u]];
    }
  while(1)
    {
      if (inp[v=base[v]]) return v;
      else v=father[match[v]];
    }
}
void mark_blossom(int lca,int u){
  while (base[u]!=lca)
    {
      int v=match[u];
      inb[base[u]]=inb[base[v]]=true;
      u=father[v];
      if (base[u]!=lca) father[u]=v;
    }
}
void blossom_contraction(int s,int u,int v){
  int lca=LCA(s,u,v);
  memset(inb,0,sizeof(inb));
  mark_blossom(lca,u);
  mark_blossom(lca,v);
  if (base[u]!=lca)
    father[u]=v;
  if (base[v]!=lca)
    father[v]=u;
  for (int u=0;u<V;u++)
    if (inb[base[u]])
      {
  base[u]=lca;
  if (!inq[u])
    inq[q[++qt]=u]=true;
      }
}
int find_augmenting_path(int s){
  memset(inq,0,sizeof(inq));
  memset(father,-1,sizeof(father));
  for (int i=0;i<V;i++) base[i]=i;
  inq[q[qh=qt=0]=s]=true;
  while (qh<=qt)
    {
```

```
      int u=q[qh++];
      for (edge e=adj[u];e;e=e->n)
        {
    int v=e->v;
    if (base[u]!=base[v]&&match[u]!=v)
      if ((v==s)||(match[v]!=-1 && father[match[v]]!=-1))
        blossom_contraction(s,u,v);
      else if (father[v]==-1)
        {
    father[v]=u;
    if (match[v]==-1)
      return v;
    else if (!inq[match[v]])
      inq[q[++qt]=match[v]]=true;
        }
        }
    }
  return -1;
}
int augment_path(int s,int t){
  int u=t,v,w;
  while (u!=-1)
    {
      v=father[u];
      w=match[v];
      match[v]=u;
      match[u]=v;
      u=w;
    }
  return t!=-1;
}
int edmonds(){
  int matchc=0;
  memset(match,-1,sizeof(match));
  for (int u=0;u<V;u++)
    if (match[u]==-1)
      matchc+=augment_path(u,find_augmenting_path(u));
  return matchc;
}
int main(){
  int u,v;
  cin>>V>>E;
  while(E--)
    {
      cin>>u>>v;
      if (!ed[u-1][v-1])
  {
    add_edge(u-1,v-1);
    ed[u-1][v-1]=ed[v-1][u-1]=true;
  }
    }
  cout<<edmonds()<<endl;
  for (int i=0;i<V;i++)
    if (i<match[i])
      cout<<i+1<<" "<<match[i]+1<<endl;
}
```

3 Geometry

3.1 Geometry

```
#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>
using namespace std;
double INF = 1e100;
double EPS = 1e-12;
struct PT {
    double x, y;
    PT() {}
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
    PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
    PT operator   (double c)     const { return PT(x c,   y c  ); }
    PT operator / (double c)     const { return PT(x/c,   y/c  ); }
};
```

```
double dot(PT p, PT q)     { return p.x q.x+p.y q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x q.y-p.y q.x; }
ostream &operator<<(ostream &os, const PT &p) {
    return os << "(" << p.x << "," << p.y << ")";
}
// if movement from a to b to c is done in a CW path returns 1
// else if it's CCW returns -1 and if they make a line returns 0
int IsCWTurn(PT a, PT b, PT c) {
    double r = cross((b - c), (a - c));
    return (fabs(r) < EPS)? 0: (r > 0)? 1: -1;
}
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
    return PT(p.x cos(t)-p.y sin(t), p.x sin(t)+p.y cos(t));
}
// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
    return a + (b-a) dot(c-a, b-a)/dot(b-a, b-a);
}
// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
    double r = dot(b-a,b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
    return (r < 0)? a: (r > 1)? b: a + (b - a) r;
}
// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
    return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z, double a,
     double b,
     double c, double d) {
    return fabs(a x+b y+c z-d)/sqrt(a a+b b+c c);
}
// determine if lines from a to b and c to d are parallel or
   collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
    return fabs(cross(b-a, c-d)) < EPS;
}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
    return LinesParallel(a, b, c, d)
    && fabs(cross(a-b, a-c)) < EPS
    && fabs(cross(c-d, c-a)) < EPS;
}
// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
    if (LinesCollinear(a, b, c, d)) {
        if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
        dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
        if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-
   b) > 0)
            return false;
        return true;
    }
    if (cross(d-a, b-a)   cross(c-a, b-a) > 0) return false;
    if (cross(a-c, d-c)   cross(b-c, d-c) > 0) return false;
    return true;
}
// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
    b=b-a; d=c-d; c=c-a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b cross(c, d)/cross(b, d);
}
// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
    b=(a+b)/2;
    c=(a+c)/2;
    return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+
   RotateCW90(a-c));
}
// determine if point is in a possibly non-convex polygon (by
```

```
William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an  exact  test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
    bool c = 0;
    for (int i = 0; i < p.size(); i++) {
        int j = (i+1)%p.size();
        if (((p[i].y <= q.y && q.y < p[j].y) || (p[j].y <= q.y && q.y < p[i].y)) &&
            q.x < p[i].x + (p[j].x - p[i].x)   (q.y - p[i].y) / (p[j].y - p[i].y))
            c = !c;
    }
    return c;
}
// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
    for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
    return true;
    return false;
}
// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
    vector<PT> ret;
    b = b-a;
    a = a-c;
    double A = dot(b, b);
    double B = dot(a, b);
    double C = dot(a, a) - r r;
    double D = B B - A C;
    if (D < -EPS) return ret;
    ret.push_back(c+a+b(-B+sqrt(D+EPS))/A);
    if (D > EPS)
    ret.push_back(c+a+b(-B-sqrt(D))/A);
    return ret;
}
// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
    vector<PT> ret;
    double d = sqrt(dist2(a, b));
    if (d > r+R || d+min(r, R) < max(r, R)) return ret;
    double x = (d d-R R+r r)/(2 d);
    double y = sqrt(r r-x x);
    PT v = (b-a)/d;
    ret.push_back(a+v x + RotateCCW90(v) y);
    if (y > 0)
    ret.push_back(a+v x - RotateCCW90(v) y);
    return ret;
}
// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
    double area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        area += p[i].x p[j].y - p[j].x p[i].y;
    }
    return area / 2.0;
}
double ComputeArea(const vector<PT> &p) {
    return fabs(ComputeSignedArea(p));
}
PT ComputeCentroid(const vector<PT> &p) {
    PT c(0,0);
```

```
    double scale = 6.0   ComputeSignedArea(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j]) (p[i].x p[j].y - p[j].x p[i].y);
    }
    return c / scale;
}
// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
    for (int i = 0; i < p.size(); i++) {
        for (int k = i+1; k < p.size(); k++) {
            int j = (i+1) % p.size();
            int l = (k+1) % p.size();
            if (i == l || j == k) continue;
            if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
            return false;
        }
    }
    return true;
}
```

3.2 Convex Hull

```
// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm.  Eliminate redundant points from the hull if REMOVE_REDUNDANT is
// #defined.
// Running time: O(n log n)
//   INPUT:   a vector of input points, unordered.
//   OUTPUT:  a vector of points in the convex hull, counterclockwise, starting
//            with bottommost/leftmost point
#include <cstdio>
#include <cassert>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
#define REMOVE_REDUNDANT
typedef double T;
const T EPS = 1e-7;
struct PT {
    T x, y;
    PT() {}
    PT(T x, T y) : x(x), y(y) {}
    bool operator<(const PT &rhs) const { return make_pair(y,x) < make_pair(rhs.y,rhs.x); }
    bool operator==(const PT &rhs) const { return make_pair(y,x) == make_pair(rhs.y,rhs.x); }
};
T cross(PT p, PT q) { return p.x q.y-p.y q.x; }
T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) + cross(c,a); }
#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x) (c.x-b.x) <= 0 && (a.y-b.y) (c.y-b.y) <= 0);
}
#endif
void ConvexHull(vector<PT> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<PT> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);
    #ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        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

```
struct Point
{
  int x, y;
};

// A line segment with left as Point
// with smaller x value and right with
// larger x value.
struct Segment
{
  Point left, right;
};
struct Event {
  int x, y;
  bool isLeft;
  int index;
  Event(int x, int y, bool l, int i) : x(x), y(y), isLeft(l),
   index(i) {}
  bool operator<(const Event& e) const {
      if(y==e.y)return x<e.x;
      return y < e.y;
  }
};
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r)
{
  if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
    q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
  return true;
  return false;
}
int orientation(Point p, Point q, Point r)
{
  int val = (q.y - p.y)   (r.x - q.x) -
        (q.x - p.x)   (r.y - q.y);
  if (val == 0) return 0; // collinear
  return (val > 0)? 1: 2; // 1clock or 2counterclock wise
}
// The main function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Segment s1, Segment s2)
{
  Point p1 = s1.left, q1 = s1.right, p2 = s2.left, q2 = s2.right;
  int o1 = orientation(p1, q1, p2);
  int o2 = orientation(p1, q1, q2);
  int o3 = orientation(p2, q2, p1);
  int o4 = orientation(p2, q2, q1);
  if (o1 != o2 && o3 != o4)
    return true;
  if (o1 == 0 && onSegment(p1, p2, q1)) return true;
  if (o2 == 0 && onSegment(p1, q2, q1)) return true;
  if (o3 == 0 && onSegment(p2, p1, q2)) return true;
  if (o4 == 0 && onSegment(p2, q1, q2)) return true;
  return false; // Doesn't fall in any of the above cases
}
set<Event>::iterator pred(set<Event> &s, set<Event>::iterator it)
    {
  return it == s.begin() ? s.end() : --it;
}
set<Event>::iterator succ(set<Event> &s, set<Event>::iterator it)
    {
  return ++it;
}
// Returns true if any two lines intersect.
int isIntersect(Segment arr[], int n)
{
  unordered_map<string,int> mp;
  vector<Event> e;
  for (int i = 0; i < n; ++i) {
    e.push_back(Event(arr[i].left.x, arr[i].left.y, true, i));
```

```
    e.push_back(Event(arr[i].right.x, arr[i].right.y, false, i));
  }
  sort(e.begin(), e.end(), [](Event &e1, Event &e2) {return e1.x < e2.x;});
  set<Event> s;
  int ans=0;
  for (int i=0; i<2n; i++)
  {
    Event curr = e[i];
    int index = curr.index;
    if (curr.isLeft)
    {
      auto next = s.lower_bound(curr);
      auto prev = pred(s, next);
      bool flag=false;
      if (next != s.end() && doIntersect(arr[next->index], arr[index])){
        string s=to_string(next->index+1)+" "+to_string(index+1);
        if(mp.count(s)==0){mp[s]++;ans++;} //if not already checked we can increase count in map
      }
      if (prev != s.end() && doIntersect(arr[prev->index], arr[index])){
          string s=to_string(prev->index+1)+" "+to_string(index+1);
        if(mp.count(s)==0){mp[s]++;ans++;} //if not already checked we can increase count in map
      }
      if(prev != s.end() && next != s.end() && next->index==prev->index)ans--;
      s.insert(curr);
    }
    else
    {
      auto it=s.find(Event(arr[index].left.x, arr[index].left.y, true, index));
      auto next = succ(s, it);
      auto prev = pred(s, it);
      if (next != s.end() && prev != s.end())
      { string s=to_string(next->index+1)+" "+to_string(prev->index+1);
          string s1=to_string(prev->index+1)+" "+to_string(next->index+1);
        if (mp.count(s)==0&&mp.count(s1)==0&&doIntersect(arr[prev->index], arr[next->index]))
          ans++;
          mp[s]++;
        }
        s.erase(it);
      }
    }
  for(auto &pr:mp){
    cout<<pr.first<<"\n";
  }
  return ans;
}
int main() {
  Segment arr[] = { {{1, 5}, {4, 5}}, {{2, 5}, {10, 1}},{{3, 2}, {10, 3}},{{6, 4}, {9, 4}},{{7, 1}, {8, 1}}};
  int n = sizeof(arr)/sizeof(arr[0]);
  cout<<isIntersect(arr, n);
  return 0;
}
```

3.4 Kardak Half Plane Intersection

```

const long double eps = 1e-9, inf = 1e9;
struct Point {
    long double x, y;
    explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {}
    friend Point operator + (const Point& p, const Point& q) {
        return Point(p.x + q.x, p.y + q.y);
    }
    friend Point operator - (const Point& p, const Point& q) {
        return Point(p.x - q.x, p.y - q.y);
    }
    friend Point operator   (const Point& p, const long double& k) {
        return Point(p.x   k, p.y   k);
    }
    friend long double cross(const Point& p, const Point& q) {
```

```
        return p.x   q.y - p.y   q.x;
    }
};
struct Halfplane {
    // 'p' is a passing point of the line and 'pq' is the direction vector of the line.
    Point p, pq;
    long double angle;
    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }
    // Every half-plane allows the region to the LEFT of its line.
    bool out(const Point& r) {
        return cross(pq, r - p) < -eps;
    }
    bool operator < (const Halfplane& e) const {
        if (fabsl(angle - e.angle) < eps) return cross(pq, e.p - p) < 0;
        return angle < e.angle;
    }
    bool operator == (const Halfplane& e) const {
        return fabsl(angle - e.angle) < eps;
    }
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq   alpha);
    }
};
vector<Point> hp_intersect(vector<Halfplane>& H) {

    Point box[4] = {  // Bounding box in CCW order
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };
    for(int i = 0; i<4; i++) { // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    }
    sort(H.begin(), H.end());
    H.erase(unique(H.begin(), H.end()), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
            dq.pop_back();
            --len;
        }
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }
        dq.push_back(H[i]);
        ++len;
    }
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back();
        --len;
    }
    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }
    if (len < 3) return vector<Point>();
    vector<Point> ret(len);
    for(int i = 0; i+1 < len; i++) {
        ret[i] = inter(dq[i], dq[i+1]);
    }
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

4 Data Structures

4.1 Fenwick1

```
const int MAXN = 1   1000 + 10;
```

```cpp
int fen[MAXN]; // 0-based, [)
void add(int x, int val = 1) {
  for (int i = x + 1; i < MAXN; i += i & (-i))
    fen[i] += val;
}
int get(int x) {
  int ans = 0;
  for (int i = x; i > 0; i -= i & (-i))
    ans += fen[i];
  return ans;
}
int sum(int x, int y) {
  return get(y) - get(x);
}
```

4.2 Fenwick2

```cpp
int fen[MAXN]; // 0-based, [)
void add(int x, int val) {
  for (int i = x; i > 0; i -= i & (-i))
    fen[i] += val;
}
int get(int x) {
  int ans = 0;
  for (int i = x + 1; i < MAXN; i += i & (-i))
    ans += fen[i];
  return ans;
}
void update(int l, int r, int val) {
  add(r, +val);
  add(l, -val);
}
```

4.3 Segment Tree Lazy Propagation

```cpp
int seg[4   MAXN], add[4   MAXN];
inline void shift(int x, int s, int e) {
  int lc = x + x + 0, rc = x + x + 1;
  int mid = (s + e) / 2;
  int l1 = mid - s, l2 = e - mid;
  seg[lc] += l1   add[x];
  seg[rc] += l2   add[x];
  add[lc] += add[x];
  add[rc] += add[x];
  add[x] = 0;
}
// lo, hi -> [)
// s = 0, e = n, x = 1
void update(int lo, int hi, int s, int e, int x, int delta) {
  if (lo == s && hi == e) {
    int len = (e - s);
    seg[x] = seg[x] + len   delta;
    add[x] = add[x] + delta;
    return ;
  }
  shift(x, s, e);
  int mid = (s + e) / 2;
  int lc = x + x + 0, rc = x + x + 1;
  if (lo < mid) update(lo, min(mid, hi), s, mid, lc, delta);
  if (hi > mid) update(max(lo, mid), hi, mid, e, rc, delta);
  seg[x] = seg[lc] + seg[rc];
}
int get(int lo, int hi, int s, int e, int x) {
  if (lo == s && hi == e) return seg[x];
  shift(x, s, e);
  int mid = (s + e) / 2;
  int res = 0;
  if (lo < mid) res += get(lo, min(mid, hi), s, mid, x + x + 0);
  if (hi > mid) res += get(max(lo, mid), hi, mid, e, x + x + 1);
  return res;
}
```

4.4 RMQ

```cpp
const int N = 1000   100 + 5, LOG = 20;
class RMQ{
  int f[LOG][N], Lgl[N], S;
public:
  RMQ() {
    for(int i = 1, p = 0;i < N;++i) {
      if(i == 1 << (p + 1))
        ++p;
      Lgl[i] = p;
    }
```

```
  }
  void build(int a[], int n) {
    for(int i = 0;i < n;++i)
      f[0][i] = a[i];

    for(int j = 1, p = 1; j < LOG; ++j, p  = 2)
      for(int i = 0;i < n;++i) {
        f[j][i] = f[j - 1][i];
        if(i + p < n)
          f[j][i] = min(f[j - 1][i], f[j - 1][i + p]);
      }
  }
  int find(int s, int e) {
    int l = Lgl[e - s + 1];
    return min(f[l][s], f[l][e + 1 - (1 << l)]);
  }
};
```

4.5 Trie

```
struct Node {
  char x;
  vector<Node > adj;
  Node () {
    x = 0;
  }
  Node (char a) {
    x = a;
  }
  Node  add_edge(char a) {
    for (int i = 0; i < SZ(adj); i ++)
      if (adj[i]->x == a)
        return adj[i];
    adj.pb(new Node(a));
    return adj.back();
  }
};
struct Trie {
  Node  root;
  Trie() {
    root = new Node();
  }
  void add(string &s) {
    add(s, 0, root);
  }
  void add(string &s, int pos, Node  node) {
    if (pos == SZ(s)) {
      return ;
    } else {
      Node  next = node->add_edge(s[pos]);
      add(s, pos + 1, next);
    }
  }
};
```

5 String

5.1 Hash

```
ll p[MAXN], hash[MAXN];
int main () {
  p[0] = 1;
  for (int i = 1; i < MAXN; i ++)
    p[i] = p[i - 1]   BASE;
  string s;
  getline(cin, s);
  for (int i = 1; i <= SZ(s); i ++)
    hash[i] = hash[i - 1]   BASE + s[i - 1];
  // hash in [i, j], 1-based
  ll h = hash[j] - (hash[i - 1]   p[j - i + 1]);
}
```

5.2 KMP

```
#define SZ(x) (int)((x).size())
const int M = 1000   100 + 4;
int f[M];
string s,t;
bool match[M];
void kmp() {
  f[0] = -1;
  int pos = -1;
  for (int i = 1; i <= SZ(t); i++) {
    while(pos != -1 && t[pos] != t[i - 1]) pos = f[pos];
```

```
    f[i] = ++pos;
  }
  pos = 0;
  for (int i = 0; i < SZ(s); i++) {
    while(pos != -1 && (pos == SZ(t) || s[i] != t[pos])) pos = f[pos];
    pos ++;
    if (pos == SZ(t)) match[i] = 1;
    else match[i] = 0;
  }
}
```

5.3 Suffix Array

```
const int N = 1000   100 + 5; //max string length
namespace Suffix{
  int sa[N], rank[N], lcp[N], gap, S;
  bool cmp(int x, int y) {
    if(rank[x] != rank[y])
      return rank[x] < rank[y];
    x += gap, y += gap;
    return (x < S && y < S)? rank[x] < rank[y]: x > y;
  }
  void Sa_build(const string &s) {
    S = s.size();
    int tmp[N] = {0};
    for(int i = 0;i < S;++i)
      rank[i] = s[i],
      sa[i] = i;
    for(gap = 1;;gap <<= 1) {
      sort(sa, sa + S, cmp);
      for(int i = 1;i < S;++i)
        tmp[i] = tmp[i - 1] + cmp(sa[i - 1], sa[i]);
      for(int i = 0;i < S;++i)
        rank[sa[i]] = tmp[i];
      if(tmp[S - 1] == S - 1)
        break;
    }
  }
  void Lcp_build() {
    for(int i = 0, k = 0;i < S;++i, --k)
      if(rank[i] != S - 1) {
        k = max(k, 0);
        while(s[i + k] == s[sa[rank[i] + 1] + k])
          ++k;
        lcp[rank[i]] = k;
      }
      else
        k = 0;
  }
};
```

5.4 Kardak AhoCorasick

```
// for string matching
using namespace std;
#include <bits/stdc++.h>
// Max number of states in the matching machine.
// Should be equal to the sum of the length of all keywords.
const int MAXS = 500;
// Maximum number of characters in input alphabet
const int MAXC = 26;
// OUTPUT FUNCTION IS IMPLEMENTED USING out[]
// Bit i in this mask is one if the word with index i
// appears when the machine enters this state.
int out[MAXS];
// FAILURE FUNCTION IS IMPLEMENTED USING f[]
int f[MAXS];
// GOTO FUNCTION (OR TRIE) IS IMPLEMENTED USING g[][]
int g[MAXS][MAXC];
// Builds the string matching machine.
// arr - array of words. The index of each keyword is important:
//      "out[state] & (1 << i)" is > 0 if we just found word[i]
//      in the text.
// Returns the number of states that the built machine has.
// States are numbered 0 up to the return value - 1, inclusive.
int buildMatchingMachine(string arr[], int k)
{
  memset(out, 0, sizeof out);
  memset(g, -1, sizeof g);
  int states = 1;
  for (int i = 0; i < k; ++i)
  {
    const string &word = arr[i];
```

```
    int currentState = 0;
    for (int j = 0; j < word.size(); ++j)
    {
      int ch = word[j] - 'a';
      if (g[currentState][ch] == -1)
        g[currentState][ch] = states++;
      currentState = g[currentState][ch];
    }
    out[currentState] |= (1 << i);
  }
  for (int ch = 0; ch < MAXC; ++ch)
    if (g[0][ch] == -1)
      g[0][ch] = 0;
  memset(f, -1, sizeof f);
  queue<int> q;
  for (int ch = 0; ch < MAXC; ++ch)
  {
    if (g[0][ch] != 0)
    {
      f[g[0][ch]] = 0;
      q.push(g[0][ch]);
    }
  }
  while (q.size())
  {
    int state = q.front();
    q.pop();
    for (int ch = 0; ch <= MAXC; ++ch)
    {
      if (g[state][ch] != -1)
      {
        int failure = f[state];
        while (g[failure][ch] == -1)
          failure = f[failure];
        failure = g[failure][ch];
        f[g[state][ch]] = failure;
        out[g[state][ch]] |= out[failure];
        q.push(g[state][ch]);
      }
    }
  }
  return states;
}
// Returns the next state the machine will transition to using goto
// and failure functions.
// currentState - The current state of the machine. Must be between
//       0 and the number of states - 1, inclusive.
// nextInput - The next character that enters into the machine.
int findNextState(int currentState, char nextInput)
{
  int answer = currentState;
  int ch = nextInput - 'a';
  while (g[answer][ch] == -1)
    answer = f[answer];
  return g[answer][ch];
}
// This function finds all occurrences of all array words
// in text.
void searchWords(string arr[], int k, string text)
{
  buildMatchingMachine(arr, k);
  int currentState = 0;
  for (int i = 0; i < text.size(); ++i)
  {
    currentState = findNextState(currentState, text[i]);
    if (out[currentState] == 0)
      continue;
    for (int j = 0; j < k; ++j)
    {
      if (out[currentState] & (1 << j))
      {
        cout << "Word " << arr[j] << " appears from "
          << i - arr[j].size() + 1 << " to " << i << endl;
      }
    }
  }
}
// Driver program to test above
int main()
{
```

```
  string arr[] = {"he", "she", "hers", "his"};
  string text = "ahishers";
  int k = sizeof(arr)/sizeof(arr[0]);
  searchWords(arr, k, text);
  return 0;
}
```

6 Number Theory

6.1 Phi

```
#include <iostream>
#include <vector>
using namespace std;
const int N = 1000   1000;
vector <int> pr;
int lp[N], phi[N];
void Sieve(int n){
  for (int i = 2;i < n; ++i) {
    if (lp[i] == 0)
      lp[i] = i,
      pr.push_back(i);

    for (int j = 0;j < pr.size() && pr[j] <= lp[i] && i   pr[j] < n; ++j)
      lp[i   pr[j]] = pr[j];
  }
}
void Find_Phi(int n) {
  phi[1] = 1;
  for(int i = 2;i < n;++i) {
    if(lp[i] == i)
      phi[i] = i - 1;
    else {
      phi[i] = phi[lp[i]]   phi[(i / lp[i])];
      if(lp[i / lp[i]] == lp[i])
        phi[i]  = lp[i], phi[i] /= (lp[i] - 1)
    }
  }
}
```

6.2 370 SGU

```
bool mark[MAXN];
vector<int> dv[MAXN];
int n, m;
// counts positive integers up to n that are relatively prime to x
int f(int x) {
  int res = 0;
  for (int mask = 0; mask < (1 << SZ(dv[x])); mask ++) {
    int t = __builtin_popcount(mask), a = n - 1;
    for (int i = 0; i < SZ(dv[x]); i ++)
      if (mask & (1 << i))
        a /= dv[x][i];
    if (t & 1) res -= a;
    else res += a;
  }
  return res;
}
int main () {
  for (int i = 2; i < n; i ++)
    if (!mark[i]) {
      for (int j = i; j < m; j += i) {
        mark[j] = true;
        dv[j].pb(i);
      }
    }
  ll ans = 2;
  for (int i = 1; i < m; i ++) ans += f(i);
  cout << ans << endl;
  return 0;
}
```

6.3 Euclid

```
typedef vector<int> VI;
typedef pair<int, int> PII;
// computes gcd(a,b)
int gcd(int a, int b) {
  while (b) { int t = a%b; a = b; b = t; }
  return a;
}
```

```
// returns g = gcd(a, b); finds x, y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
    int q = a / b;
    int t = b; b = a%b; a = t;
    t = xx; xx = x - q xx; x = t;
    t = yy; yy = y - q yy; y = t;
  }
  return a;
}
// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
  int x, y;
  VI ret;
  int g = extended_euclid(a, n, x, y);
  if (!(b%g)) {
    x = mod(x (b / g), n);
    for (int i = 0; i < g; i++)
      ret.push_back(mod(x + i (n / g), n));
  }
  return ret;
}
// Chinese remainder theorem (special case): find z such that
// z % m1 = r1, z % m2 = r2.  Here, z is unique modulo M = lcm(m1, m2).
// Return (z, M).  On failure, M = -1.
PII chinese_remainder_theorem(int m1, int r1, int m2, int r2) {
  int s, t;
  int g = extended_euclid(m1, m2, s, t);
  if (r1%g != r2%g) return make_pair(0, -1);
  return make_pair(mod(s r2 m1 + t r1 m2, m1 m2) / g, m1 m2 / g);
}
// Chinese remainder theorem: find z such that
// z % m[i] = r[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (m[i]).  Return (z, M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &m, const VI &r) {
  PII ret = make_pair(r[0], m[0]);
  for (int i = 1; i < m.size(); i++) {
    ret = chinese_remainder_theorem(ret.second, ret.first, m[i], r[i]);
    if (ret.second == -1) break;
  }
  return ret;
}
```

6.4 C(n, r)

```
ll bin_pow(ll x, ll y) {
  if (y == 0) return 1;

  ll tmp = bin_pow(x, y / 2);
  ll res = SQR(tmp) % MOD;
  if (y & 1) res = (res   x) % MOD;
  return res;
}
ll fct[MAXN], rev[MAXN], fct_rev[MAXN];
void init(int n) {
  fct[0] = 1;
  for (int i = 1; i <= n; i ++)
    fct[i] = (fct[i - 1]   i) % MOD;

  rev[0] = 1;
  for (int i = 1; i <= n; i ++)
    rev[i] = bin_pow(i, MOD - 2);

  fct_rev[0] = 1;
  for (int i = 1; i <= n; i ++)
    fct_rev[i] = (fct_rev[i - 1]   rev[i]) % MOD;
}
int C(int n, int r) {
  return (((fct[n]   fct_rev[r]) % MOD)   fct_rev[n - r]) % MOD;
}
```

6.5 FFT

```
typedef long double DOUBLE;
typedef complex<DOUBLE> COMPLEX;
typedef vector<DOUBLE> VD;
typedef vector<COMPLEX> VC;
```

```
struct FFT {
    VC A;
    int n, L;

    int ReverseBits(int k) {
        int ret = 0;
        for (int i = 0; i < L; i++) {
            ret = (ret << 1) | (k & 1);
            k >>= 1;
        }
        return ret;
    }
    void BitReverseCopy(VC a) {
        for (n = 1, L = 0; n < a.size(); n <<= 1, L++) ;
        A.resize(n);
        for (int k = 0; k < n; k++)
        A[ReverseBits(k)] = a[k];
    }
    VC DFT(VC a, bool inverse) {
        BitReverseCopy(a);
        for (int s = 1; s <= L; s++) {
            int m = 1 << s;
            COMPLEX wm = exp(COMPLEX(0, 2.0   M_PI / m));
            if (inverse) wm = COMPLEX(1, 0) / wm;
            for (int k = 0; k < n; k += m) {
                COMPLEX w = 1;
                for (int j = 0; j < m/2; j++) {
                    COMPLEX t = w   A[k + j + m/2];
                    COMPLEX u = A[k + j];
                    A[k + j] = u + t;
                    A[k + j + m/2] = u - t;
                    w = w   wm;
                }
            }
        }
        if (inverse) for (int i = 0; i < n; i++) A[i] /= n;
        return A;
    }
    // c[k] = sum_{i=0}^k a[i] b[k-i]
    VD Convolution(VD a, VD b) {
        int L = 1;
        while ((1 << L) < a.size()) L++;
        while ((1 << L) < b.size()) L++;
        int n = 1 << (L+1);
        VC aa, bb;
        for (size_t i = 0; i < n; i++) aa.push_back(i < a.size() ? COMPLEX(a[i], 0) : 0);
        for (size_t i = 0; i < n; i++) bb.push_back(i < b.size() ? COMPLEX(b[i], 0) : 0);
        VC AA = DFT(aa, false);
        VC BB = DFT(bb, false);
        VC CC;
        for (size_t i = 0; i < AA.size(); i++) CC.push_back(AA[i]   BB[i]);
        VC cc = DFT(CC, true);
        VD c;
        for (int i = 0; i < a.size() + b.size() - 1; i++) c.push_back(cc[i].real());
        return c;
    }
};
```

6.6 Gauss Jordan

```
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
// OUTPUT:   X      = an nxm matrix (stored in b[][])
//           A^{-1} = an nxn matrix (stored in a[][])
//           returns determinant of a[][]
const double EPS = 1e-10;
typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
```

```
typedef vector<VT> VVT;
T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;
  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
      for (int k = 0; k < n; k++) if (!ipiv[k])
    if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det  = -1;
    irow[i] = pj;
    icol[i] = pk;
    T c = 1.0 / a[pk][pk];
    det  = a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p]  = c;
    for (int p = 0; p < m; p++) b[pk][p]  = c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q]   c;
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q]   c;
    }
  }
  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
  }
  return det;
}
int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
  double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
  VVT a(n), b(n);
  for (int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }
  double det = GaussJordan(a, b);
}
```

7 Other

7.1 Read Input

```
inline int read() {
  bool minus = false;
  int result = 0;
  char ch;
  ch = getchar();
  while (true) {
    if (ch == '-') break;
    if (ch >= '0' && ch <= '9') break;
    ch = getchar();
  }
  if (ch == '-') minus = true; else result = ch-'0';
  while (true) {
    ch = getchar();
    if (ch < '0' || ch > '9') break;
    result = result 10 + (ch - '0');
  }
  if (minus)
    return -result;
  else
    return result;
}
```

7.2 LIS

```
int c[MAXN], a[MAXN];
int main() {
  int n;
  cin >> n;
```

```
  for (int i = 0; i < n; i ++) cin >> a[i];
  for (int i = 0; i <= n; i ++) c[i] = 1e9;
  int ans = 0;
  for (int i = 0; i < n; i ++) {
    int l = 0, r = i + 1;
    while (r - l > 1) {
      int mid = (l + r) / 2;
      if (c[mid] <= a[i]) l = mid;
      else r = mid;
    }
    ans = max(ans, l + 1);
    if (c[l + 1] > a[i]) c[l + 1] = a[i];
  }
  cout << ans << endl;
}
```

7.3 Divide and Conquer Tree

```
#include <iostream>
#include <vector>
using namespace std;
const int N = 1000   100 + 5;
vector <int> adj[N];
int is_av[N], _sz[N]; //XXX initiate is_av to 1
void set_size(int v, int p) {
  _sz[v] = 1;
  for(int u:adj[v])
    if(u != p && is_av[u]) {
      set_size(u, v);
      _sz[v] += _sz[u];
    }
}
void divide(int v) {
  set_size(v, v);
  int S = _sz[v], p = v;
    sign:
      for(int u:adj[v])
        if(is_av[u] && u != p && _sz[u] > S / 2) {
          p = v;
          v = u;
          goto sign;
        }
    // now v is the centroid of the tree
    // Enter your code here
  is_av[v] = 0;
  for(int u:adj[v])
    if(is_av[u])
      divide(u);
}
int main() {
  ios::sync_with_stdio(false);
  int n;
  cin >> n;
  for(int i = 1;i < n;++i) {
    int a, b;
    cin >> a >> b;
    --a, --b;
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  fill(is_av, is_av + N, 1);
  divide(0);
  return 0;
}
```