Team notebook

September 27, 2021

Contents

1 BlockCutTree

```
#include<bits/stdc++.h>

using namespace std;

typedef pair<int,int> II;
typedef vector< II >   VII;
typedef vector<int>   VI;
typedef vector< VI >  VVI;
typedef long long int  LL;

#define PB push_back
#define MP make_pair
#define F first
#define S second
#define SZ(a) (int)(a.size())
#define ALL(a) a.begin(),a.end()
#define SET(a,b) memset(a,b,sizeof(a))

#define si(n) scanf("%d",&n)
#define dout(n) printf("%d\n",n)
#define sll(n) scanf("%lld",&n)
#define lldout(n) printf("%lld\n",n)
#define fast_io
    ios_base::sync_with_stdio(false);cin.tie(NULL)

#define TRACE

#ifdef TRACE
```

```
#define trace(...) __f(#__VA_ARGS__,
     __VA_ARGS__)
template <typename Arg1>
void __f(const char* name, Arg1&& arg1){
    cerr << name << " : " << arg1 << std::endl;
}
template <typename Arg1, typename... Args>
void __f(const char* names, Arg1&& arg1,
     Args&&... args){
    const char* comma = strchr(names + 1,
         ',');cerr.write(names, comma - names)
         << " : " << arg1<<" | ";__f(comma+1,
         args...);
}
#else
#define trace(...)
#endif

//FILE *fin = freopen("in","r",stdin);
//FILE *fout = freopen("out","w",stdout);
const int N = int(2e5)+1;
const int M = int(2e5)+1;
const int LOGN = 20;
VI g[N],tree[N],st;//graph in edge-list form.
     N should be 2*N
int
     U[M],V[M],low[N],ord[N],sz[N],depth[N],col[N],C,T,compNo[N],extra[N],level[N],DP[LOGN][N];
bool isArtic[N];
int arr[N],dep[N],vis[N];
int adj(int u,int e){
    return u^V[e]^U[e];
}
//everything from [1,n+C] whose extra[i]=0 is
     part of Block-Tree
//1-Based Graph Input.Everything from [1,C]
     is type B and [C,n+C] is type C.
void dfs(int i){
    low[i]=ord[i]=T++;
    for(int j=0;j<SZ(g[i]);j++){
        int ei=g[i][j],to = adj(i,ei);
        if(ord[to]==-1){
            depth[to]=depth[i]+1;
            st.PB(ei);dfs(to);
            low[i] = min(low[i],low[to]);
            if(ord[i]==0||low[to]>=ord[i]){
                if(ord[i]!=0||j>=1)
                    isArtic[i] = true;
                ++C;
                while(!st.empty()){
                    int
                         fi=st.back();st.pop_back();
                    col[fi]=C;
                    if(fi==ei)break;
                }
            }
        }else if(depth[to]<depth[i]-1){
            low[i] = min(low[i],ord[to]);
            st.PB(ei);
        }
    }
}
void run(int n){
    SET(low,-1);SET(depth,-1);
    SET(ord,-1);SET(col,-1);
    SET(isArtic,0);st.clear();C=0;
    for(int i=1;i<=n;++i)
        if(ord[i]==-1){
            T = 0;dfs(i);
        }
}
void buildTree(int n){
    SET(compNo,-1);
    VI tmpv;SET(extra,-1);
    tmpv.clear();SET(sz,0);
    for(int i=1;i<=n;i++){
        tmpv.clear();
        for(auto e:g[i])
            tmpv.PB(col[e]);
        sort(ALL(tmpv));
        tmpv.erase(unique(ALL(tmpv)),
             tmpv.end());
        //handle isolated vertics
        if(tmpv.empty()){
            compNo[i]=C+i;extra[C+i]=0;
            sz[C+i]=1;continue;
        }if(SZ(tmpv)==1){//completely in 1
             comp.
            compNo[i]=tmpv[0];
            extra[tmpv[0]]=0;
            sz[tmpv[0]]++;
        }else{ //it's an articulation vertex.
            compNo[i]=C+i;
            extra[C+i]=0;sz[C+i]++;
            for(auto j:tmpv){
                extra[j]=0;sz[j]++;
                tree[C+i].push_back(j);
                tree[j].push_back(C+i);
            }
        }
    }
}
int currComp;
void dfs2(int u,int p){
    level[u]=level[p]+1;DP[0][u]=p;
    arr[u]=++T;vis[u]=currComp;
    for(auto w:tree[u])
        if(w!=p)
            dfs2(w,u);
    dep[u]=T++;
}
int lca(int a,int b){
    if(level[a]>level[b])swap(a,b);
    int d = level[b]-level[a];
    for(int i=0;i<LOGN;i++)
        if((1<<i)&d)
            b = DP[i][b];
    if(a==b)return a;
    for(int i=LOGN-1;i>=0;i--)
        if(DP[i][a]!=DP[i][b])
            a=DP[i][a],b=DP[i][b];
    return DP[0][a];
}
bool anc(int p,int u){
    return (arr[u]>=arr[p] && dep[u]<=dep[p]);
}
int main()
{
    int n,m,q;
    si(n);si(m);si(q);
    for(int i=0;i<m;i++){
        scanf("%d %d",U+i,V+i);
        g[U[i]].PB(i);
        g[V[i]].PB(i);
    }
    buildTree(n);T=0;
    for(int i=1;i<=C+n;i++)
        if(!vis[i] && !extra[i])
            currComp++,dfs2(i,i);
    for(int i=1;i<LOGN;i++)
        for(int j=1;j<=C+n;j++)
            if(!extra[j])
                DP[i][j]=DP[i-1][DP[i-1][j]];
    while(q--){
        int u,v,w;
        si(u);si(v);si(w);
        if(u==v){
            puts(u==w?"Party":"Break-Up");
            continue;
        }
        u=compNo[u];v=compNo[v];w=compNo[w];
        if(!(vis[u]==vis[w] &&
            vis[w]==vis[v])){
            puts("Break-Up");
            continue;
        }
        int LCA = lca(u,v);
        if(level[u]>level[v])swap(u,v);
        if(sz[w]==1 && w!=LCA && w!=DP[0][LCA]
            && sz[DP[0][w]]>2) w = DP[0][w];
        if(sz[u]==1 && u!=LCA &&
            sz[DP[0][w]]>2) u = DP[0][u];
        if(sz[v]==1 && v!=LCA &&
            sz[DP[0][v]]>2) v = DP[0][v];
        bool ok=false;
        ok|=anc(w,u);
        ok|=anc(w,v);
        ok&=anc(LCA,w);
        ok|=(sz[LCA]>2 && w==DP[0][LCA]);
        puts(ok?"Party":"Break-Up");
    }
    return 0;
}
```

2 Centroid

```
#include <bits/stdc++.h>
#define X first
#define Y second
#define pb push_back
using namespace std;
typedef pair<int, int> pii;
typedef pair<pii, int> ppi;
const int maxn = 2e5 + 17, lg = 18;

int n = 1, q, par[maxn][lg], cpar[maxn],
    h[maxn], sz[maxn];
set<ppi> s[maxn];
vector<int> g[maxn], ch[maxn];
struct Q{
  int t, v, d;
} qu[maxn];
void prep(int v = 0){
  sz[v] = 1;
  for(auto u : g[v]){
    prep(u);
    sz[v] += sz[u];
  }
}
int get_cent(int root = 0){
  int v = root, size = sz[root];
  bool done = 0;
  while(done ^= 1)
    for(auto &u : g[v])
      if(sz[u] > (size >> 1)){
        v = u, done = 0;
        break;
      }
  int mysz = sz[v];
  for(int u = v; ; u = par[u][0]){
    sz[u] -= mysz;
    if(u == root) break;
  }
  for(auto &u : g[v])
    if(sz[u]){
      int x = get_cent(u);
      //cerr << v << ' ' << x << '\n';
      cpar[x] = v;
      ch[v].pb(x);
    }
  if(v != root){
    int x = get_cent(root);
    //cerr << v << ' ' << x << '\n';
    cpar[x] = v;
    ch[v].pb(x);
  }
  return v;
}
int dis(int v, int u){
    if(h[u] < h[v]) swap(v, u);
    int ans = h[v] + h[u];
    for(int i = 0; i < lg; i++)
  if((h[u] - h[v]) >> i & 1)
      u = par[u][i];
    for(int i = lg - 1; i >= 0; i--)
  if(par[v][i] != par[u][i])
      v = par[v][i], u = par[u][i];
    return v == u ? ans - 2 * h[v] : ans - 2 *
        (h[v] - 1);
}
void add(int v){
  for(int u = v; u != -1; u = cpar[u]){
    if(v == 6)
      ;//cerr << u << '\n';
    int d = dis(u, v);
    auto it = s[u].lower_bound({{d + 1, -1},
        -1});
    if(it != s[u].begin() && prev(it) -> X.Y
        >= h[v])
      continue;
    it = s[u].insert({{d, h[v]}, v}).X;
    it++;
    while(it != s[u].end() && it -> X.Y <=
        h[v])
      s[u].erase(prev(++it));
  }
}
int get(int v, int d){
  int ans = -1, cer = -1;
  for(int u = v; u != -1; u = cpar[u]){
    int di = dis(u, v);
    //cerr << u << '\n';
    auto it = s[u].lower_bound({{d - di + 1,
        -1}, -1});
    if(it != s[u].begin()){
      it--;
      if(it -> X.Y > ans)
        ans = it -> X.Y, cer = it -> Y;
    }
  }
  return cer;
}



//////////////////////////////////////////////////////////////////////////////////////////////


#include <bits/stdc++.h>
#define X first
#define Y second
#define pb push_back
using namespace std;
typedef pair<int, int> pii;
typedef pair<pii, int> ppi;
typedef long long ll;
const int maxn = 5e5 + 17, lg = 19;
const ll inf = 1e18;
int n, q, par[maxn][lg], cpar[maxn], h[maxn],
    sz[maxn], che[maxn];
ll sw[maxn][lg], ns[maxn], sd[maxn][lg];
vector<int> ch[maxn];
vector<pii> g[maxn];
bool mark[maxn];
void prep(int v = 0, int p = 0){
  sz[v] = 1;
  par[v][0] = p;
  for(auto e : g[v])
    if(e.X != p){
      h[e.X] = h[v] + 1;
      sw[e.X][0] = e.Y;
      prep(e.X, v);
      sz[v] += sz[e.X];
    }
}
void setD(int v, int lvl, int p = -1, ll cd =
    0){
  if(mark[v])
    return ;
  sd[v][lvl] = cd;
  for(auto e : g[v])
    if(e.X != p)
      setD(e.X, lvl, v, cd + e.Y);
}
int get_cent(int root = 0, int h = 0){
  int v = root, size = sz[root];
  bool done = 0;
  while(done ^= 1)
    for(auto &e : g[v])
      if(e.X != par[v][0] && sz[e.X] > (size
          >> 1)){
        v = e.X, done = 0;
        break;
      }
  che[v] = h;
  setD(v, h);
  mark[v] = 1;
  int mysz = sz[v];
  for(int u = v; ; u = par[u][0]){
    sz[u] -= mysz;
    if(u == root) break;
  }
  for(auto &e : g[v])
    if(e.X != par[v][0] && sz[e.X]){
      int x = get_cent(e.X, h + 1);
      //cerr << v << ’ ’ << x << ’\n’;
      cpar[x] = v;
      ch[v].pb(x);
    }
  if(v != root){
    int x = get_cent(root, h + 1);
    //cerr << v << ’ ’ << x << ’\n’;
    cpar[x] = v;
    ch[v].pb(x);
  }
  return v;
}
ll dis(int v, int u){
  if(h[u] < h[v]) swap(v, u);
  ll ans = 0;
  for(int i = 0; i < lg; i++)
    if(h[u] - h[v] >> i & 1){
      ans += sw[u][i];
      //cerr << "$ " << u << ’ ’ << i << ’ ’
          << sw[u][i] << ’\n’;
      u = par[u][i];
    }
  //cerr << ans << ’\n’;
  if(v == u)
    return ans;
  for(int i = lg - 1; i >= 0; i--)
    if(par[v][i] != par[u][i]){
      ans += sw[v][i], ans += sw[u][i];
      v = par[v][i], u = par[u][i];
    }
  ans += sw[v][0] + sw[u][0];
  return ans;
}
void add(int v){
  for(int u = v; u != -1; u = cpar[u])
    ns[u] = min(ns[u], sd[v][che[u]]);
}
void clear(int v){
  for(int u = v; u != -1; u = cpar[u])
    ns[u] = inf;
}
ll get(int v){
  ll ans = inf;
  for(int u = v; u != -1; u = cpar[u]){
    ans = min(ans, sd[v][che[u]] + ns[u]);
    //cerr << dis(u, v) << ’ ’ << ns[u] <<
        ’\n’;
  }
  return ans;
}
```

3 ConvexHull

```
#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
}
```

4 ConvexHullTrick

```
typedef long long int64;
typedef long double float128;

const int64 is_query = -(1LL<<62), inf = 1e18;

struct Line {
        int64 m, b;
        mutable function<const Line*()> succ;
        bool operator<(const Line& rhs) const {
                if (rhs.b != is_query) return m
                    < rhs.m;
                const Line* s = succ();
                if (!s) return 0;
                int64 x = rhs.m;
                return b - s->b < (s->m - m) *
                    x;
        }
};

struct HullDynamic : public multiset<Line> {
    // will maintain upper hull for maximum
        bool bad(iterator y) {
                auto z = next(y);
                if (y == begin()) {
                        if (z == end()) return 0;
                        return y->m == z->m &&
                            y->b <= z->b;
                }
                auto x = prev(y);
                if (z == end()) return y->m ==
                    x->m && y->b <= x->b;
                return (float128)(x->b -
                    y->b)*(z->m - y->m) >=
                    (float128)(y->b -
                    z->b)*(y->m - x->m);
        }
        void insert_line(int64 m, int64 b) {
                auto y = insert({ m, b });
                y->succ = [=] { return next(y)
                    == end() ? 0 : &*next(y); };
                if (bad(y)) { erase(y); return;
                    }
                while (next(y) != end() &&
                    bad(next(y)))
                    erase(next(y));
                while (y != begin() &&
                    bad(prev(y)))
                    erase(prev(y));
        }

        int64 eval(int64 x) {
                auto l = *lower_bound((Line) {
                    x, is_query });
                return l.m * x + l.b;
        }
};
```

5 Cut

```
stack<int> stak;
inline void add_edge(int v, int u){
    g[v].push_back(u), g[u].push_back(v);
}
int get_cut(int v = 0, int p = -1){
    if(mark[v]) return h[v];
    hi[v] = h[v] = ~p ? h[p] + 1 : 0, mark[v]
        = 1;
    stak.push(v);
    for(auto u : adj[v])
          smin(hi[v], get_cut(u, v));
    if(hi[v] + 1 == h[v]){
        while(stak.top() != v)
            add_edge(stak.top(), v + n),
                stak.pop();
        add_edge(v, v + n), stak.pop();
        add_edge(p, v + n);
    }
    return hi[v];
}
```

6 Euclid

```
// 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;
}
```

7 FFT

```
#define REP(i, n) for(int i = 0; i < (n); i++)
typedef int llint;
namespace FFT {
  const int MAX = 1 << 17;
  typedef llint value;
  typedef complex<double> comp;
  int N;
  comp omega[MAX];
  comp a1[MAX], a2[MAX];
  comp z1[MAX], z2[MAX];
  void fft(comp *a, comp *z, int m = N) {
    if (m == 1) {
      z[0] = a[0];
    } else {
      int s = N/m;
      m /= 2;
      fft(a, z, m);
      fft(a+s, z+m, m);
      REP(i, m) {
        comp c = omega[s*i] * z[m+i];
        z[m+i] = z[i] - c;
        z[i] += c;
      }
    }
  }
  void mult(value *a, value *b, value *c, int
      len) {
    N = 2*len;
    while (N & (N-1)) ++N;
    assert(N <= MAX);
    REP(i, N) a1[i] = 0;
    REP(i, N) a2[i] = 0;
    REP(i, len) a1[i] = a[i];
    REP(i, len) a2[i] = b[i];
    REP(i, N) omega[i] = polar(1.0,
        2*M_PI/N*i);
    fft(a1, z1, N);
    fft(a2, z2, N);
    REP(i, N) omega[i] = comp(1, 0) / omega[i];
    REP(i, N) a1[i] = z1[i] * z2[i] / comp(N,
        0);
    fft(a1, z1, N);
    REP(i, 2*len) c[i] = round(z1[i].real());
  }
  void mult_mod(int *a, int *b, int *c, int
      len, int mod) {
    static llint a0[MAX], a1[MAX];
    static llint b0[MAX], b1[MAX];
    static llint c0[MAX], c1[MAX], c2[MAX];
    REP(i, len) a0[i] = a[i] & 0xFFFF;
    REP(i, len) a1[i] = a[i] >> 16;
    REP(i, len) b0[i] = b[i] & 0xFFFF;
    REP(i, len) b1[i] = b[i] >> 16;
    FFT::mult(a0, b0, c0, len);
    FFT::mult(a1, b1, c2, len);
    REP(i, len) a0[i] += a1[i];
    REP(i, len) b0[i] += b1[i];
    FFT::mult(a0, b0, c1, len);
    REP(i, 2*len) c1[i] -= c0[i] + c2[i];
    REP(i, 2*len) c1[i] %= mod;
    REP(i, 2*len) c2[i] %= mod;
    REP(i, 2*len) c[i] = (c0[i] + ((long long)
        c1[i] << 16) + ((long long) c2[i] <<
        32)) % mod;
  }
}
#undef REP
```

8 Ford Fulkerson

```
// Ford Fulkerson: Runs in O(E * maxflow)
int head[maxn], to[maxm], prv[maxm],
    cap[maxm], cost[maxm], ecnt;
const int maxn = 2e3 + 17, maxm = maxn * maxn
    + 17, inf = 1e9 + 17;
void init() {
        memset(head, -1, sizeof head);
    ecnt = 0;
}
void add(int v, int u, int vu, int uv = 0) {
        to[ecnt] = u, prv[ecnt] = head[v],
            cap[ecnt] = vu, head[v] = ecnt++;
        to[ecnt] = v, prv[ecnt] = head[u],
            cap[ecnt] = uv, head[u] = ecnt++;
}
int dfs(int v, int flow = inf) {
        if (v == sink || flow == 0) return f;
        if (mark[v]) return 0;
        mark[v] = 1;
        for (int e = head[v]; e != -1; e =
            prv[e])
                if (cap[e]) {
                    int x = dfs(to[e],
                        min(flow, cap[e]));
                    if (x)
                        return cap[e] -=
                            x, cap[e ^
                            1] += x, x;
                }
        return 0;
}
int maxflow() {
        int ans = 0;
        for (int tmp; (tmp = dfs(so)); ans +=
            tmp)
                memset(mark, 0, sizeof mark);
        return ans;
}
```

9 GaussElim

```
// A[0..n-1][0..m-1]*ANS=A[1..n][m]. this
    functions will find ANS and returns
    number of different answer
// which can be 0, 1 or INF.
int gauss (vector < vector<double> > a,
    vector<double> & ans) {
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;

  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n;
      ++col) {
    int sel = row;
    for (int i=row; i<n; ++i)
      if (abs (a[i][col]) > abs (a[sel][col]))
        sel = i;
    if (abs (a[sel][col]) < EPS)
      continue;
    for (int i=col; i<=m; ++i)
      swap (a[sel][i], a[row][i]);
    where[col] = row;

    for (int i=0; i<n; ++i)
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j)
          a[i][j] -= a[row][j] * c;
      }
    ++row;
  }

  ans.assign (m, 0);
  for (int i=0; i<m; ++i)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i=0; i<n; ++i) {
    double sum = 0;
    for (int j=0; j<m; ++j)
      sum += ans[j] * a[i][j];
    if (abs (sum - a[i][m]) > EPS)
      return 0;
  }

  for (int i=0; i<m; ++i)
    if (where[i] == -1)
      return INF;
  return 1;
}
```

10 GaussJordan

```
// Gauss-Jordan elimination with full
    pivoting.
//
// Uses:
//   (1) solving systems of linear equations
    (aX=b)
//   (2) inverting matrice a (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[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

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);

  // expected: 60
  cout << "Determinant: " << det << endl;

  // expected: -0.233333 0.166667 0.133333
      0.0666667
  //           0.166667 0.166667 0.333333
      -0.333333
  //           0.233333 0.833333 -0.133333
      -0.0666667
  //           0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

  // expected: 1.63333 1.3
  //           -0.166667 0.5
  //           2.36667 1.7
  //           -1.85 -1.35
  cout << "Solution: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

11 Geometry

```
const double EPS = 1e-12;
struct P {
    double x, y;
    P operator+(const P &p) const { return {x
        + p.x, y + p.y}; }
    P operator-(const P &p) const { return {x
        - p.x, y - p.y}; }
    P operator*(double c) const { return {x *
        c, y * c}; }
    double operator*(P q) const { return x *
        q.y - y * q.x; }
    P operator/(double c) const { return {x /
        c, y / c}; }
    double angle() const {
        return atan2(y, x);
    }
    P RotateCCW90() const { return {-y, x}; }
    P RotateCW90() const { return {y, -x}; }
    P RotateCCW(double t) const {
        return {x * cos(t) - y * sin(t), x *
            sin(t) + y * cos(t)};
    }
    double size2() const {
        return x * x + y * y;
    }
    double size() const {
        return sqrt(size2());
    }
};
double dot(P p, P q) { return p.x * q.x + p.y
    * q.y; }
double dist2(P p, P q) { return (p -
    q).size2(); }
double dist(P p, P q) { return (p -
    q).size(); }
ostream &operator<<(ostream &os, const P &p) {
    return os << "(" << p.x << "," << p.y <<
        ")";
}
// project point c onto line through a and b
// assuming a != b
P project_point_line(P a, P b, P c) {
    return a + (b - a) * dot(c - a, b - a) /
        (b - a).size2();
}
// project point c onto line segment through
    a and b
P project_point_segment(P a, P b, P c) {
    double r = dot(b - a, b - a);
    if (abs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b - a) * r;
}
// compute distance from c to segment between
    a and b
double distance_point_segment(P a, P b, P c) {
    return sqrt(dist2(c,
        project_point_segment(a, b, c)));
}
// compute distance between point (x,y,z) and
    plane ax+by+cz=d
```

```
double distance_point_plane(double x, double
    y, double z,
                          double a, double b,
                              double c, double
                              d) {
    return abs(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 lines_parallel(P a, P b, P c, P d) {
    return abs((b - a) * (c - d)) < EPS;
}
bool lines_collinear(P a, P b, P c, P d) {
    return lines_parallel(a, b, c, d)
        && abs((a - b) * (a - c)) < EPS
        && abs((c - d) * (c - a)) < EPS;
}
// determine if line segment from a to b
    intersects with
// line segment from c to d
bool segments_intersect(P a, P b, P c, P d) {
    if (lines_collinear(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 (((d - a) * (b - a)) * ((c - a) * (b -
        a)) > 0 || ((a - c) * (d - c)) * ((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
P line_intersection(P a, P b, P c, P d) {
    b = b - a;
    d = c - d;
    c = c - a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b * (c * d) / (b * d);
}
// compute center of circle given three points
P circle_center(P a, P b, P c) {
    b = (a + b) / 2;
    c = (a + c) / 2;
    return line_intersection(b, b + (a -
        b).RotateCW90(), c, c + (a -
        c).RotateCW90());
}
// 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 point_in_polygon(const vector<P> &p, P
    q) {
    bool c = false;
    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 point_on_polygon(const vector<P> &p, P
    q) {
    for (int i = 0; i < p.size(); i++)
        if (dist2(project_point_segment(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
// going from a to b, t[1] is the first
    intersection and t[0] is the second
vector<P> circle_line_intersection(P a, P b,
    P c, double r) {
    vector<P> 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
// order is counter clock wise
vector<P> circle_circle_intersection(P a, P
    b, double r, double R) {
    vector<P> 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);
    P v = (b - a) / d;
    ret.push_back(a + v * x + v.RotateCCW90()
        * y);
    if (y > 0)
        ret.push_back(a + v * x -
            v.RotateCCW90() * 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 signed_area(const vector<P> &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; // TODO
    }
    return area / 2.0;
}
double area(const vector<P> &p) {
    return abs(signed_area(p));
}
P centroid(const vector<P> &p) {
    P c{0, 0};
    double scale = 6.0 * signed_area(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 is_simple(const vector<P> &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 (segments_intersect(p[i], p[j],
                p[k], p[l]))
                return false;
        }
    }
    return true;
}
double coef_on_line(P a, P b, P c) {
    if (abs(a.x - c.x) < EPS)
        return (b.y - a.y) / (c.y - a.y);
    return (b.x - a.x) / (c.x - a.x);
}
void seg_union(vector<pair<double, double >>
    &segs) {
    sort(segs.begin(), segs.end());
    int sz = 0;
    for (auto[l, r] : segs)
        if (l <= r)
            if (!sz || l > segs[sz - 1].second
                + EPS)
                segs[sz++] = {l, r};
            else
                segs[sz - 1].second =
                    max(segs[sz - 1].second, r);
    segs.resize(sz);
}
vector<pair<double, double> >
    polygon_segment_intersection(vector<P>
    &pol, P a, P b) {
    vector<pair<double, double> > segs;
    vector<P> impos({a, b});
    for (int k = 0; k < pol.size(); k++)
        if (segments_intersect(a, b, pol[k],
            pol[(k + 1) % pol.size()]))
            impos.push_back(line_intersection(a,
                b, pol[k], pol[(k + 1) %
                pol.size()]));
    sort(impos.begin(), impos.end(), [&](P x,
        P y) {
        return coef_on_line(a, x, b) <
            coef_on_line(a, y, b);
    });
    for (int k = 0; k < impos.size() - 1; k++)
        {
        P mid = (impos[k] + impos[k + 1]) / 2;
        if (point_in_polygon(pol, mid))
            segs.emplace_back(coef_on_line(a,
                impos[k], b), coef_on_line(a,
                impos[k + 1], b));
    }
    return segs;
}
pair<double, double>
    circle_segment_intersection(P a, P b, P
    c, double r) {
    vector<P> ret =
        circle_line_intersection(a, b, c, r);
    if (ret.size() < 2)
        return {0, 0};
    return {max<double>(0, min(coef_on_line(a,
        ret[0], b), coef_on_line(a, ret[1],
        b))),
            min<double>(1, max(coef_on_line(a,
                ret[0], b), coef_on_line(a,
                ret[1], b)))};
}
bool cmp_angle(const P &a, const P &b) {
    if (a.y * b.y < 0)
        return a.y < b.y;
    return a * b > 0;
}
```

12 HLD

```
const int maxn = 1e5 + 17, lg = 17;
int n, q, col[maxn], head[maxn],
    par[lg][maxn], h[maxn], st[maxn],
    ft[maxn], iman[maxn << 2], sina[maxn <<
    2];
vector<int> g[maxn];
pair<int, int> qu[maxn];
```

```
int prep(int v = 0, int p = -1){
  if(g[v].empty() || g[v].size() == 1 &&
      g[v][0] == p){
    col[v] = head[v] = v;
    return 1;
  }
  int sz = 1, big, mx = 0;
  for(int i = 0; i < g[v].size(); i++){
    int u = g[v][i];
    if(u == p) continue;
    par[0][u] = v;
    h[u] = h[v] + 1;
    int s = prep(u, v);
    sz += s;
    if(s > mx)
      mx = s, big = i;
  }
  col[v] = col[ g[v][big] ];
  head[ col[v] ] = v;
  swap(g[v][0], g[v][big]);
  return sz;
}
void get_st(int v = 0){
  static int time = 0;
  st[v] = time++;
  for(auto u : g[v])
    if(u != par[0][v])
      get_st(u);
  ft[v] = time;
}
int lca(int v, int u){
  if(h[u] < h[v])
    swap(v, u);
  for(int i = 0; i < lg; i++)
    if(h[u] - h[v] >> i & 1)
      u = par[i][u];
  for(int i = lg - 1; i >= 0; i--)
    if(par[i][v] != par[i][u])
      v = par[i][v], u = par[i][u];
  return v == u ? v : par[0][v];
}
int dis(int v, int u){
  return h[v] + h[u] - 2 * h[lca(v, u)];
}
void sadra(int id){
  if(sina[id] == -1)
    return;
  iman[id << 1] = iman[id << 1 | 1] = sina[id
      << 1] = sina[id << 1 | 1] = sina[id];
  sina[id] = -1;
}
void majid(int s, int e, int x, int l = 0,
    int r = n, int id = 1){
  if(s <= l && r <= e){
    iman[id] = sina[id] = x;
    return ;
  }
  if(e <= l || r <= s) return ;
  sadra(id);
  int mid = l + r >> 1;
  majid(s, e, x, l, mid, id << 1);
  majid(s, e, x, mid, r, id << 1 | 1);
  iman[id] = max(iman[id << 1], iman[id << 1 |
      1]);
}
int hamid(int s, int e, int l = 0, int r = n,
    int id = 1){
  if(s <= l && r <= e) return iman[id];
  if(e <= l || r <= s) return 0;
  sadra(id);
  int mid = l + r >> 1;
  return max(hamid(s, e, l, mid, id << 1),
      hamid(s, e, mid, r, id << 1 | 1));
}
void change(int v, int u, int x){
  //cerr << "changeing " << v << ' ' << u << '
      ' << x << '\n';
  if(col[v] == col[u]){
    majid(st[u], st[v] + 1, x);
    return ;
  }
  if(col[v] != col[ par[0][v] ]){
    majid(st[v], st[v] + 1, x);
    change(par[0][v], u, x);
    return ;
  }
  majid(st[ head[ col[v] ] ], st[v] + 1, x);
  change(par[0][ head[ col[v] ] ], u, x);
}
void Change(int v, int u, int x){
  int p = lca(v, u);
  change(v, p, x);
  change(u, p, x);
}
int get_max(int v, int u){
  if(col[v] == col[u])
    return hamid(st[u], st[v] + 1);
  if(col[v] != col[ par[0][v] ])
    return max(hamid(st[v], st[v] + 1),
        get_max(par[0][v], u));
  return max(hamid(st[ head[ col[v] ] ], st[v]
      + 1), get_max(par[0][ head[ col[v] ] ],
      u));
}
int Get_max(int v, int u){
  int p = lca(v, u);
  return max(get_max(v, p), get_max(u, p));
}
int main(){
  ios::sync_with_stdio(0), cin.tie(0);
  memset(sina, -1, sizeof sina);
  cin >> n >> q;
  for(int i = 1, v, u; i < n; i++){
    cin >> v >> u;
    v--, u--;
    g[v].push_back(u);
    g[u].push_back(v);
  }
  prep();
}
```

13 Hungarian

```
typedef long long ll;
const ll INFL = (1 << 60);
using Weight = ll;
const Weight InfWeight = INFL;

Weight hungarianMin(const vector
    <vector<Weight>> &A) {
        if (A.empty()) return 0;
        int h = A.size(), n = A[0].size();
if (h > n) return InfWeight;
vector <Weight> fx(h), fy(n);
vector<int> x(h, -1), y(n, -1);
vector<int> t(n), s(h + 1);
for (int i = 0; i < h;) {
        fill(t.begin(), t.end(), -1);
        s[0] = i;
        int q = 0;
        for (int p = 0; p <= q; ++p) {
                for (int k = s[p], j =
                    0; j < n; ++j) {
                        if (fx[k] + fy[j]
                            == A[k][j]
                            && t[j] < 0)
                            {
                                s[++q] =
                                    y[j];
                                t[j] = k;
                                if (s[q]
                                    < 0) {
                                        for
                                            (p
                                            =
                                            j;
                                            p
                                            >=
                                            0;
                                            j
                                            =
                                            p)
                                            {
                                                y[j]
                                                    =
                                                    k
                                                    =
                                                    t[j];
                                                p
                                                    =
                                                    x[k];
                                                x[k]
                                                    =
                                                    j;
                                        }
                                        ++i;
                                        goto
                                            continue_;
                                }
                        }
                }
        }
        if (0) {
        continue_:;
        } else {
                Weight d = InfWeight;
                for (int j = 0; j < n;
                    j++)
                        if (t[j] < 0) {
                                for (int
                                    k =
                                    0; k
                                    <= q;
                                    ++k)
                                        if
                                            (A[s[k]][j]
                                            !=
                                            InfWeight)
                                                d
                                                    =
                                                    min(d,
                                                    A[s[k]][j]
                                                    -
                                                    fx[s[k]]
                                                    -
                                                    fy[j]);
                        }
                if (d == InfWeight)
                        return InfWeight;
                for (int j = 0; j < n;
                    ++j) {
                        if (t[j] >= 0)
                                fy[j] -=
                                    d;
                }
                for (int k = 0; k <= q;
                    ++k)
                        fx[s[k]] += d;
        }
}
Weight res = 0;
for (int i = 0; i < h; ++i)
        res += A[i][x[i]];
return res;
}
```

14 KDTree

```
// number type for coordinates, and its
    maximum value
typedef long long ntype;
const ntype sentry =
    numeric_limits<ntype>::max();

// point structure for 2D-tree, can be
    extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx),
        y(yy) {}
};

bool operator==(const point &a, const point
    &b)
{
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b)
{
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b)
{
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b)
```

```
{
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox
{
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry),
        y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of
        points
    void compute(const vector<point> &v) {
        for (int i = 0; i < v.size(); ++i) {
            x0 = min(x0, v[i].x); x1 = max(x1,
                v[i].x);
            y0 = min(y0, v[i].y); y1 = max(y1,
                v[i].y);
        }
    }

    // squared distance between a point and
        this bbox, 0 if inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)      return
                pdist2(point(x0, y0), p);
            else if (p.y > y1) return
                pdist2(point(x0, y1), p);
            else               return
                pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)      return
                pdist2(point(x1, y0), p);
            else if (p.y > y1) return
                pdist2(point(x1, y1), p);
            else               return
                pdist2(point(x1, p.y), p);
        }
        else {
```

```
            if (p.y < y0)      return
                pdist2(point(p.x, y0), p);
            else if (p.y > y1) return
                pdist2(point(p.x, y1), p);
            else               return 0;
        }
    }
};

// stores a single node of the kd-tree,
    either internal or leaf
struct kdnode
{
    bool leaf;     // true if this is a leaf
        node (has one point)
    point pt;      // the single point of this
        is a leaf
    bbox bound;    // bounding box for set of
        points in children

    kdnode *first, *second; // two children of
        this kd-node

    kdnode() : leaf(false), first(0),
        second(0) {}
    ~kdnode() { if (first) delete first; if
        (second) delete second; }

    // intersect a point with this node
        (returns squared distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a
        given cloud of points
    void construct(vector<point> &vp)
    {
        // compute bounding box for points at
            this node
        bound.compute(vp);

        // if we're down to one point, then
            we're a leaf node
        if (vp.size() == 1) {
```

```
            leaf = true;
            pt = vp[0];
        }
        else {
            // split on x if the bbox is wider
                than high (not best
                heuristic...)
            if (bound.x1-bound.x0 >=
                bound.y1-bound.y0)
                sort(vp.begin(), vp.end(),
                    on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(),
                    on_y);

            // divide by taking half the array
                for each child
            // (not best performance if many
                duplicates in the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(),
                vp.begin()+half);
            vector<point> vr(vp.begin()+half,
                vp.end());
            first = new kdnode();
                first->construct(vl);
            second = new kdnode();
                second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and
    handle queries
struct kdtree
{
    kdnode *root;

    // constructs a kd-tree from a points
        (copied here, as it sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared
        distance to nearest point
    ntype search(kdnode *node, const point &p)
    {
        if (node->leaf) {
            // commented special case tells a
                point not to find itself
//            if (p == node->pt) return sentry;
//            else
                return pdist2(p, node->pt);
        }

        ntype bfirst =
            node->first->intersect(p);
        ntype bsecond =
            node->second->intersect(p);

        // choose the side with the closest
            bounding box to search first
        // (note that the other side is also
            searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best,
                    search(node->second, p));
            return best;
        }
        else {
            ntype best = search(node->second,
                p);
            if (bfirst < best)
                best = min(best,
                    search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
```

```
};
```

15 MaxFlowMinCost

```
// Running Complexity is about O(SPFA() *
    Max_flow) but better. Can be O(N^3M)
    using dijkstra.
const int maxn = 1e2 + 17, maxm = 1e4 + 17,
    so = maxn - 1, sink = maxn - 2;
int head[maxn], to[maxm], prv[maxm],
    cap[maxm], cost[maxm], q[maxm * maxn],
    ecnt;
void init(){
    memset(head, -1, sizeof head);
    ecnt = 0;
}
void add(int v, int u, int cst = 0, int vu =
    1, int uv = 0){
    prv[ecnt] = head[v], to[ecnt] = u,
        cap[ecnt] = vu, cost[ecnt] = cst,
        head[v] = ecnt++;
    prv[ecnt] = head[u], to[ecnt] = v,
        cap[ecnt] = uv, cost[ecnt] = -cst,
        head[u] = ecnt++;
}
int d[maxn], par[maxn];
bool mark[maxn];
bool spfa(){
    memset(d, 63, sizeof d);
    d[so] = 0;
    int h = 0, t = 0;
    q[t++] = so, par[so] = -1;
    while(h < t){
        int v = q[h++];
        mark[v] = 0;
        for(int e = head[v]; ~e; e = prv[e])
            if(cap[e] && d[to[e]] > d[v] +
                cost[e]){
                d[to[e]] = d[v] + cost[e];
                if(!mark[to[e]]){
                    mark[to[e]] = 1;
                    q[t++] = to[e];
                }
                par[to[e]] = e;
            }
    }
    return d[sink] < 1e9;
}
int mincost(){
    int ans = 0;
    while(spfa())
        for(int e = par[sink]; ~e; e =
            par[to[e ^ 1]])
            cap[e]--, cap[e ^ 1]++, ans +=
                cost[e];
    return ans;
}
```

16 MaxMatchingAndIndependent

```
int mat[N][2];
bool mark[N];

// I hope this is HopcroftKarp algorithm. O(E
    * sqrt(V)).
// for sparse random graphs, runs in O(E *
    log(v)) with high probability.

bool dfs(int v){
    if(mark[v]) return 0;
    mark[v] = 1;
    for(auto u : adj[v][0])
        if(mat[u][1] == -1 || dfs(mat[u][1]))
            return mat[v][0] = u, mat[u][1] =
                v, 1;
    return 0;
}
void dfs(int v, int part){
    seen[v][part] = 1;
    for(auto u : adj[v][part])
        if(!seen[u][!part]){
            bad[u] = 1;
            seen[u][!part] = 1;
            dfs(mat[u][!part], part);
        }
}
void maximum_matching() { // can be used to
     find max independent set
    memset(mat, -1, sizeof mat);
    bool br = 0;
    int ans = n;
    while(br ^= 1) {
                memset(mark, 0, sizeof mark);
                for(int i = 0; i < n; i++)
                    if(mat[i][0] == -1 &&
                         dfs(i))
                                ans--, br = 0;
    }
    for(int i = 0; i < n; i++)
                for(int j = 0; j < 2; j++)
                if(seen[i][j] == 0 && mat[i][j]
                     == -1)
                                dfs(i, j);

    cout << ans << '\n';
    for(int i = 0; i < n; i++)
        if(bad[i] == 0 && seen[i][0] == 1)
            cout << i + 1 << ' ';
    cout << '\n';
}
```

17 NTT

```
namespace NTT {
    const int maxn = 1 << 18;
    const int p = 998244353;
    const int g = 3;
    int R[maxn], tmp[maxn];
    int pm(int a, int b) {
        int res = 1;
        while (b) {
            if (b & 1)
                res = (ll) res * a % p;
            a = (ll) a * a % p;
            b >>= 1;
        }
    return res;
}
void NTT(int * a, int n, int on) {
    for (int i = 0; i < n; i++)
        if (i < R[i])
            swap(a[i], a[R[i]]);
    int wn, u, v;
    for (int i = 1, m = 2; i < n; i = m, m
         <<= 1) {
        wn = pm(g, (p - 1) / m);
        if (on == -1)
            wn = pm(wn, p - 2);
        for (int j = 0; j < n; j += m) {
            for (int k = 0, w = 1; k < i;
                 k++, w = (ll) w * wn % p) {
                u = a[j + k], v = (ll) w *
                    a[i + j + k] % p;
                a[j + k] = (u + v) % p;
                a[i + j + k] = (u - v + p)
                    % p;
            }
        }
    }
    if (on == -1)
        for (int i = 0, k = pm(n, p - 2); i
             < n; i++)
            a[i] = (ll) a[i] * k % p;
}
vector < int > operator * (vector < int >
     & A, vector < int > & B) {
    vector < int > C;
    int n = A.size(), m = B.size();
    int l1 = n, l2 = m, L = 0;
    m += n, n = 1;
    while (n <= m)
        n <<= 1, L++;
    for (int i = 0; i < n; i++)
        R[i] = (R[i >> 1] >> 1) | ((i & 1)
             << (L - 1));
    A.resize(n);
    B.resize(n);
    NTT(A.data(), n, 1);
    NTT(B.data(), n, 1);
    for (int i = 0; i < n; i++)
        tmp[i] = (ll) A[i] * B[i] % p;
        NTT(tmp, n, -1);
        return vector < int > (tmp, tmp + l1 +
             l2 - 1);
    }
}
```

18 OrderedSet

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<int, null_type, less<int>,
     rb_tree_tag,tree_order_statistics_node_update>
     os;
```

19 Simpson$_integration$

```
// The error in approximating an integral by
     Simpson's formula is: 1/90* ((ba)/2)^5 *
     f(4)()
// f(4) is the forth derivative of f.
//   is some number between a and b.
const int N = 1000000; // number of steps
     (already multiplied by 2)
double simpson_integration(double a, double
     b){ // Find integration in [a, b] range.
    double h = (b - a) / N;
    double s = f(a) + f(b); // a = x_0 and b =
         x_2n
    for (int i = 1; i <= N - 1; ++i) { //
         Refer to final Simpson's formula
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3;
    return s;
}
```

20 SuffixArray

```cpp
int sa[maxl], pos[maxl], tmp[maxl], lcp[maxl];
void buildSA(string s) {
    int n = s.size();
    for (int i = 0; i < n; i++)
        sa[i] = i, pos[i] = s[i];
    for (int gap = 1;; gap *= 2) {
        auto sufCmp = [&n, &gap](int i, int j) {
            if (pos[i] != pos[j])
                return pos[i] < pos[j];
            i += gap; j += gap;
            return (i < n && j < n) ? pos[i] < pos[j] : i > j;
        };
        sort(sa, sa + n, sufCmp);
        for (int i = 0; i < n - 1; i++)
            tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i + 1]);
        for (int i = 0; i < n; i++)
            pos[sa[i]] = tmp[i];
        if (tmp[n - 1] == n - 1) break;
    }
    for (int i = 0, k = 0; i < n; ++i)
        if (pos[i] != n - 1) {
            for (int j = sa[pos[i] + 1]; s[i + k] == s[j + k];)
                ++k;
            lcp[pos[i] + 1] = k;
            if (k)--k;
        }
}
```

21 aho

```cpp
int nxt[maxn][z], q[maxn], f[maxn], sz = 1;
int insert(string &s){
  int v = 0;
  for(auto c : s){
    if(!nxt[v][c - 'a'])
      nxt[v][c - 'a'] = sz++;
    v = nxt[v][c - 'a'];
  }
  return v;
}
void aho_corasick(){
  int head = 0, tail = 0;
  for(int i = 0; i < z; i++)
    if(nxt[0][i])
      q[tail++] = nxt[0][i];
  while(head < tail){
    int v = q[head++];
    for(int i = 0; i < z; i++)
      if(nxt[v][i]){
        f[ nxt[v][i] ] = nxt[ f[v] ][i];
        q[tail++] = nxt[v][i];
      }
      else
        nxt[v][i] = nxt[ f[v] ][i];
  }
}
```

22 berlekamp-massey

```cpp
#include<cassert>
#include<vector>
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;

const int MOD = 1000000007;

int inverse(int a) {
  return a == 1 ? 1 : (long long)(MOD - MOD / a) * inverse(MOD % a) % MOD;
}

// Berlekamp-Massey Algorithm
// Requirement: const MOD, inverse(int)
// Input: vector<int> the first elements of the sequence
// Output: vector<int> the recursive equation of the given sequence
// Example: In: {1, 1, 2, 3} Out: {1, 1000000006, 1000000006} (MOD = 1e9+7)

struct Poly {
  vector<int> a;
  Poly() { a.clear(); }
  Poly(vector<int> &a): a(a) {}
  int length() const { return a.size(); }
  Poly move(int d) {
    vector<int> na(d, 0);
    na.insert(na.end(), a.begin(), a.end());
    return Poly(na);
  }
  int calc(vector<int> &d, int pos) {
    int ret = 0;
    for (int i = 0; i < (int)a.size(); ++i) {
      if ((ret += (long long)d[pos - i] * a[i] % MOD) >= MOD) {
        ret -= MOD;
      }
    }
    return ret;
  }
  Poly operator - (const Poly &b) {
    vector<int> na(max(this->length(), b.length()));
    for (int i = 0; i < (int)na.size(); ++i) {
      int aa = i < this->length() ? this->a[i] : 0,
        bb = i < b.length() ? b.a[i] : 0;
      na[i] = (aa + MOD - bb) % MOD;
    }
    return Poly(na);
  }
  friend Poly operator * (const int &c, const Poly &p) {
    vector<int> na(p.length());
    for (int i = 0; i < (int)na.size(); ++i) {
      na[i] = (long long)c * p.a[i] % MOD;
    }
    return na;
  }
};
```

```
vector<int> solve(vector<int> a) {
  int n = a.size();
  Poly s, b;
  s.a.push_back(1), b.a.push_back(1);
  for (int i = 1, j = 0, ld = a[0]; i < n;
       ++i) {
    int d = s.calc(a, i);
    if (d) {
      if ((s.length() - 1) * 2 <= i) {
        Poly ob = b;
        b = s;
        s = s - (long long)d * inverse(ld) %
             MOD * ob.move(i - j);
        j = i;
        ld = d;
      } else {
        s = s - (long long)d * inverse(ld) %
             MOD * b.move(i - j);
      }
    }
  }
  return s.a;
}

//end of template

int main() {
  int T = 1000;
  for (int i = 0; i < T; ++i) {
    cout << "Test " << i + 1 << endl;
    int n = rand() % 1000 + 1;
    vector<int> s;
    for (int i = 0; i < n; ++i) {
      s.push_back(rand() % (MOD - 1) + 1);
    }
    vector<int> a;
    for (int i = 0; i < n; ++i) {
      a.push_back(rand() % MOD);
    }
    for (int i = 0; i < n; ++i) {
      int na = 0;
      for (int j = 0; j < n; ++j) {
        if ((na += (long long)a[n + i - 1 - j]
             * s[j] % MOD) >= MOD) {
          na -= MOD;
        }
      }
      a.push_back(na);
    }
    vector<int> ss = solve(a);
    /*
    for (int i = 0; i < n; ++i) {
      printf("%d%c", s[i], i == n - 1 ? '\n' :
           ' ');
    }
    cout << endl;
    for (int i = 0; i < n; ++i) {
      printf("%d%c", ss[i + 1], i == n - 1 ?
           '\n' : ' ');
    }
    */
    assert((int)ss.size() == n + 1);
    assert(ss[0] == 1);
    for (int i = 0; i < n; ++i) {
      assert((ss[i + 1] + s[i]) % MOD == 0);
    }
  }
  cout << "All tests OK!!!" << endl;
  return 0;
}
```

23 charpoly-matrix

```
// see modnum.cpp file for defining num type.
// Compute the characteristic polynomial of a
     square matrix A over some field.
// Not numerically stable at all.
// Takes argument by value, use std::move if
     you can.
// at the end of method, res[i] will be c_i
     in sigma(i:0->n): c_i*A^i which is
     characteristic polynomial of A and equal
     to det(tI-A).
// so (-1)^n*c_0 is the determinant of A.
// c_n will be 1. for more information, go to
     Parsa Abdollahi :)
template <typename num> std::vector<num>
     charPoly(std::vector<std::vector<num>> A)
     {
    int N = int(A.size());
    std::vector<num> res; res.reserve(N+1);
    res.push_back(num(1));
    for (int i = 0, deg = 0; i < N; i++) {
        auto& Ai = A[i];

        int c = i+1;
        while (c < N && Ai[c] == num(0)) c++;
        if (c == N) {
            res.resize(i+2, num(0));
            for (int x = deg; x >= 0; x--) {
                num v = res[x];
                for (int y = x+1, z = i; z >=
                     deg; z--, y++) {
                    res[y] -= v * Ai[z];
                }
            }
            deg = i+1;
            continue;
        }

        num vc = Ai[c];
        num ivc = inv(vc);

        Ai[c] = Ai[i+1];
        Ai[i+1] = 0;

        std::swap(A[i+1], A[c]);
        auto& Ai1 = A[i+1];
        for (int k = deg; k < N; k++) {
            Ai1[k] *= vc;
        }

        for (int k = i+1; k < N; k++) {
            auto& Ak = A[k];
            {
                auto& x = Ak[i+1];
                auto& y = Ak[c];
                num tmp = y;
                y = x;
                x = tmp * ivc;
            }
            {
                num v = Ak[i+1];
                for (int j = deg; j < N; j++) {
                    Ak[j] -= v * Ai[j];
                }
            }
            if (k > i+1) {
                num v = Ai[k];
                for (int j = deg; j < N; j++) {
                    Ai1[j] += v * Ak[j];
                }
            }
        }

        for (int k = deg; k <= i; k++) {
            Ai1[k+1] += Ai[k];
        }
    }
    reverse(res.begin(), res.end());
    return res;
}

// Compute the characteristic polynomial of a
    square matrix A over F2.
// Takes argument by value, use std::move if
    you can.
template <std::size_t MAXS> std::bitset<MAXS>
    charPoly(std::vector<std::bitset<MAXS>>
    A) {
    using bs = std::bitset<MAXS>;
    int N = int(A.size());
    bs ans; ans[0] = 1;
    int deg = 0;
    for (int i = 0; i < N; i++) {
        {
            int j = int(A[i]._Find_next(i));
            if (j >= N) {
                bs nans;
                for (; deg <= i; ans <<= 1,
                    deg++) {
                    if (A[i][deg]) nans ^= ans;
                }
                ans ^= nans;
                continue;
            }
            if (j != i+1) {
                swap(A[j], A[i+1]);
                for (auto& a : A) {
                    bool tmp = a[j];
                    a[j] = a[i+1];
                    a[i+1] = tmp;
                }
            }
        }
        assert(A[i][i+1]);
        bs msk = A[i]; msk.flip(i+1);
        for (int k = 0; k < N; k++) {
            if (msk[k]) A[i+1] ^= A[k];
        }
        for (auto& a : A) {
            if (a[i+1]) a ^= msk;
        }
    }
    return ans;
}
```

24 dinic with low scaling

```
// MaxFlow Dinic algorithm with scaling.
// O(N * M * log(MC)), where MC is maximum
    edge capacity.
// Based on problem
    http://informatics.mccme.ru/mod/statements/view3.php?chapterid=2784#1
// For not using long long, make all "ll"s
    int and change infs.
struct Edge {
    int a, b;
    ll f, c;
    Edge (int a, int b, ll f, ll c) : a(a),
        b(b), f(f), c(c) {};
};
const int INF_BFS=1e9;
const int MAXN = 550;
const ll INF_CAP = (ll)1e16;
int d[MAXN], source=MAXN-2, sink=MAXN-1;
int pt[MAXN]; // very important performance
    trick
vector <Edge> e;
vector <ll> g[MAXN];
ll lim;

void add_edge(int a, int b, ll ab_cap, ll
    ba_cap=0) {
    //keep edges in vector: e[ind] - direct
        edge, e[ind ^ 1] - back edge
    g[a].emplace_back(e.size());
    e.emplace_back(Edge(a, b, 0, ab_cap));

    g[b].emplace_back(e.size());
    e.emplace_back(Edge(b, a, 0, ba_cap));
}

bool bfs() {
    fill(d, d+MAXN, INF_BFS); // be cautious
        about using this.
    d[source] = 0;
    queue <int> q;
    q.push(source);
    while (!q.empty() && d[sink] == INF_BFS) {
        int cur = q.front(); q.pop();
        for (size_t i = 0; i < g[cur].size();
            i++) {
            int id = g[cur][i];
            int to = e[id].b;
            //printf("cur = %d id = %d a = %d b
                = %d f = %d c = %d\n", cur,
                id, e[id].a, e[id].b, e[id].f,
                e[id].c);
            if (d[to] == INF_BFS && e[id].c -
                e[id].f >= lim) {
                d[to] = d[cur] + 1;
                q.push(to);
            }
        }
    }
    while (!q.empty())
        q.pop();
    return d[sink] != INF_BFS;
}

bool dfs(int v, ll flow) {
    if (flow == 0)
```

```
        return false;
    if (v == sink) {
        //cerr << v << endl;
        return true;
    }
    for (; pt[v] < g[v].size(); pt[v]++) {
        int id = g[v][pt[v]];
        int to = e[id].b;
        //printf("v = %d id = %d a = %d b = %d
            f = %d c = %d\n", v, id, e[id].a,
            e[id].b, e[id].f, e[id].c);
        if (d[to] == d[v] + 1 && e[id].c -
            e[id].f >= flow) {
            bool pushed = dfs(to, flow);
            if (pushed) {
                e[id].f += flow;
                e[id ^ 1].f -= flow;
                return true;
            }
        }
    }
    return false;
}

ll dinic() {
    ll flow=0;
    for (lim = (1LL << 62); lim >= 1;) {
        if (!bfs()) {
            lim >>= 1;
            continue;
        }
        fill(pt, pt + MAXN, 0); // be cautious
            about this one.
        while (dfs(source, lim)) {
            flow = flow + lim;
        }
        //cerr << flow << endl;
    }
    return flow;
}
```

25 dinic

```
// Dinic: O(V^2*E).
// Runs in O(E * sqrt(V)) for finding
    matching in bipartite graph (for more
    specification go to parsa’s talabarg).
// For not using long long, make all "ll"s
    int and change infs.
const int maxn = 2e3 + 17, maxm = 5e4 + 17;
const ll INF_CAP = (ll)1e10, INF = (ll)1e17;
int ptr[maxn], head[maxn], prv[maxm],
    to[maxm], d[maxn], q[maxn], dis[maxn],
    source = maxn - 1, sink = maxn - 2, ecnt;
ll cap[maxm];
void init(){
    memset(head, -1, sizeof head);
    ecnt = 0;
}
void add_edge(int v, int u, ll vu, ll uv = 0){
    to[ecnt] = u, prv[ecnt] = head[v],
        cap[ecnt] = vu, head[v] = ecnt++;
    to[ecnt] = v, prv[ecnt] = head[u],
        cap[ecnt] = uv, head[u] = ecnt++;
}
bool bfs(){
    memset(dis, 63, sizeof dis);
    dis[source] = 0;
    int h = 0, t = 0;
    q[t++] = source;
    while(h < t){
        int v = q[h++];
        for(int e = head[v]; e >= 0; e =
            prv[e])
            if(cap[e] && dis[ to[e] ] > dis[v]
                + 1){
                dis[ to[e] ] = dis[v] + 1,
                    q[t++] = to[e];
                if(to[e] == sink)
                    return 1;
            }
    }
    return 0;
}
ll dfs(int v, ll f = INF){
    if(v == sink || f == 0)
        return f;
    ll ret = 0;
    for(int &e = ptr[v]; e >= 0; e = prv[e])
        if(dis[v] == dis[ to[e] ] - 1){
            ll x = dfs(to[e], min(f, cap[e]));
            f -= x, ret += x;
            cap[e] -= x, cap[e ^ 1] += x;
            if(!f)
                break;
        }
    return ret;
}
ll mf(){
    ll ans = 0;
    while(bfs()){
        memcpy(ptr, head, sizeof ptr);
        ans += dfs(source);
    }
    return ans;
}

// Some of the code of finding cut
bool visited[maxn];
void dfs_cut(int v) {
    visited[v] = true;
    for (int e = head[v]; e >= 0; e = prv[e]) {
        if (cap[e] and !visited[to[e]]) {
            dfs_cut(to[e]);
        }
    }
}
void find_cut() {
    cerr << "Left part of cut: ";
    dfs_cut(source);
    for(int i = 0; i < maxn; i++)
        if(visited[i])
            cerr << i << ' ';
}
```

26 fwt

```
template <typename T>
struct FWT {
  void fwt(T io[], int n) {
    for (int d = 1; d < n; d <<= 1) {
      for (int i = 0, m = d<<1; i < n; i += m)
          {
        for (int j = 0; j < d; j++) { ///
            Don’t forget modulo if required
          T x = io[i+j], y = io[i+j+d];
          io[i+j] = (x+y), io[i+j+d] = (x-y);
              // xor
//         io[i+j] = x+y; // and
//         io[i+j+d] = x+y; // or
        }
      }
    }
  }
  void ufwt(T io[], int n) {
    for (int d = 1; d < n; d <<= 1) {
      for (int i = 0, m = d<<1; i < n; i += m)
          {
        for (int j = 0; j < d; j++) { ///
            Don’t forget modulo if required
          T x = io[i+j], y = io[i+j+d];
           /// Modular inverse if required here
          io[i+j] = (x+y)>>1, io[i+j+d] =
              (x-y)>>1; // xor
//         io[i+j] = x-y; // and
//         io[i+j+d] = y-x; // or
        }
      }
    }
  }
  // a, b are two polynomials and n is size
      which is power of two
  void convolution(T a[], T b[], int n) {
    fwt(a, n);
    fwt(b, n);
    for (int i = 0; i < n; i++)
      a[i] = 1ll * a[i] * b[i]; //% MOD;
    ufwt(a, n);
  }
  // for a*a
  void self_convolution(T a[], int n) {
    fwt(a, n);
    for (int i = 0; i < n; i++)
      a[i] = a[i] * a[i]; //% MOD;
    ufwt(a, n);
  }
};
```

27 interactive$_runner$

```
from __future__ import print_function
import sys, subprocess, threading

class SubprocessThread(threading.Thread):
  def __init__(self,
              args,
              stdin_pipe=subprocess.PIPE,
              stdout_pipe=subprocess.PIPE,
              stderr_pipe=subprocess.PIPE):
    threading.Thread.__init__(self)
    self.p = subprocess.Popen(
        args,
        stdin=stdin_pipe,
        stdout=stdout_pipe,
        stderr=stderr_pipe)

  def run(self):
    try:
      self.return_code = self.p.wait()
      self.stdout = "" if self.p.stdout is
          None else self.p.stdout.read()
      self.stderr = "" if self.p.stderr is
          None else self.p.stderr.read()
    except (SystemError, OSError):
      self.return_code = -1
      self.stdout = ""
      self.stderr = "The process crashed or
          produced too much output."

assert sys.argv.count("--") == 1, (
    "There should be exactly one instance of
        ’--’ in the command line.")
sep_index = sys.argv.index("--")
judge_args = sys.argv[1:sep_index]
sol_args = sys.argv[sep_index + 1:]

t_sol = SubprocessThread(sol_args)
t_judge = SubprocessThread(judge_args,
    stdin_pipe=t_sol.p.stdout,
                         stdout_pipe=t_sol.p.stdin)
t_sol.start()
t_judge.start()
t_sol.join()
t_judge.join()
print("Judge return code:",
    t_judge.return_code)
print("Judge standard error:",
    t_judge.stderr.decode())
print("Solution return code:",
    t_sol.return_code)
print("Solution standard error:",
    t_sol.stderr.decode())
```

28 j

```
for ((i = 1; i <= 10000; i++)); do
  echo $i -------
  cmake-build-debug/gen 5 $i >in
  cmake-build-debug/code <in >out
  #  cmake-build-debug/naive <in >out2
  #  diff out out2 >/dev/null
  if [ $? != 0 ]; then
    echo WA
    exit
  fi
done
```

29 kmp

```
int k = 0;
for(int i = 1; i < p.size(); i++){
        while(k && p[k] != p[i]) k = f[k];
        if(p[k] == p[i]) k++;
        f[i + 1] = k;
}
```

30 polard-rho

```
#define MAXL (50000>>5)+1
#define GET(x) (mark[x>>5]>>(x&31)&1)
#define SET(x) (mark[x>>5] |= 1<<(x&31))
int mark[MAXL];
int P[50000], Pt = 0;
void sieve() {
    register int i, j, k; // clang++ >=17
        compile error
    SET(1);
    int n = 46340;
    for (i = 2; i <= n; i++) {
        if (!GET(i)) {
            for (k = n/i, j = i*k; k >= i; k--,
                j -= i)
                SET(j);
            P[Pt++] = i;
        }
    }
}
long long mul(unsigned long long a, unsigned
    long long b, unsigned long long mod) { //
    can be handled with int128
    long long ret = 0;
    for (a %= mod, b %= mod; b != 0; b >>= 1,
        a <<= 1, a = a >= mod ? a - mod : a) {
        if (b&1) {
            ret += a;
            if (ret >= mod)   ret -= mod;
        }
    }
    return ret;
}
void exgcd(long long x, long long y, long
    long &g, long long &a, long long &b) {
    if (y == 0)
        g = x, a = 1, b = 0;
    else
        exgcd(y, x%y, g, b, a), b -= (x/y) * a;
}
long long llgcd(long long x, long long y) {
    if (x < 0)   x = -x;
    if (y < 0)   y = -y;
    if (!x || !y)   return x + y;
    long long t;
    while (x%y)
        t = x, x = y, y = t%y;
    return y;
}
long long inverse(long long x, long long p) {
    long long g, b, r;
    exgcd(x, p, g, r, b);
    if (g < 0) r = -r;
    return (r%p + p)%p;
}
long long mpow(long long x, long long y, long
    long mod) { // mod < 2^32
    long long ret = 1;
    while (y) {
        if (y&1)
            ret = (ret * x)%mod;
        y >>= 1, x = (x * x)%mod;
    }
    return ret % mod;
}
long long mpow2(long long x, long long y,
    long long mod) {
    long long ret = 1;
    while (y) {
        if (y&1)
            ret = mul(ret, x, mod);
        y >>= 1, x = mul(x, x, mod);
    }
    return ret % mod;
}
int isPrime(long long p) { // implements by
    miller-rabin
    if (p < 2 || !(p&1))     return 0;
    if (p == 2)                     return 1;
    long long q = p-1, a, t;
    int k = 0, b = 0;
    while (!(q&1))    q >>= 1, k++;
    for (int it = 0; it < 2; it++) {
        a = rand()%(p-4) + 2;
        t = mpow2(a, q, p);
        b = (t == 1) || (t == p-1);
        for (int i = 1; i < k && !b; i++) {
            t = mul(t, t, p);
            if (t == p-1)
                b = 1;
        }
        if (b == 0)
            return 0;
    }
    return 1;
}
long long pollard_rho(long long n, long long
    c) {
    long long x = 2, y = 2, i = 1, k = 2, d;
    while (true) {
        x = (mul(x, x, n) + c);
        if (x >= n)    x -= n;
        d = llgcd(x - y, n);
        if (d > 1) return d;
        if (++i == k) y = x, k <<= 1;
    }
    return n;
}
void factorize(int n, vector<long long> &f) {
    for (int i = 0; i < Pt && P[i]*P[i] <= n;
        i++) {
        if (n%P[i] == 0) {
                while (n%P[i] == 0)
                    f.push_back(P[i]), n /=
                        P[i];
        }
    }
    if (n != 1) f.push_back(n);
}
void llfactorize(long long n, vector<long
    long> &f) {
    if (n == 1)
        return ;
    if (n < 1e+9) {
        factorize(n, f);
        return ;
    }
    if (isPrime(n)) {
        f.push_back(n);
        return ;
    }
    long long d = n;
    for (int i = 2; d == n; i++)
```

```
        d = pollard_rho(n, i);
    llfactorize(d, f);
    llfactorize(n/d, f);
}
```

31 poly

```
const int inf=1e9, magic=500;// threshold for
    sizes to run the naive algo
template<typename T>
struct poly {
    vector<T> a;
    // get rid of leading zeroes
    void normalize() { while(!a.empty() &&
        a.back() == T(0)) a.pop_back(); }
    poly(){}
    poly(T a0) : a{a0}{normalize();}
    poly(vector<T> t) : a(t){normalize();}
    poly operator += (const poly &t) {
        a.resize(max(a.size(), t.a.size()));
        for(size_t i = 0; i < t.a.size(); i++)
            a[i] += t.a[i];
        normalize();
        return *this;
    }
    poly operator -= (const poly &t) {
        a.resize(max(a.size(), t.a.size()));
        for(size_t i = 0; i < t.a.size(); i++)
            a[i] -= t.a[i];
        normalize();
        return *this;
    }
    poly operator + (const poly &t) const
        {return poly(*this) += t;}
    poly operator - (const poly &t) const
        {return poly(*this) -= t;}
    // get same polynomial mod x^k
    poly mod_xk(size_t k) const {k = min(k,
        a.size()); return vector<T>(begin(a),
        begin(a) + k);}
    // multiply by x^k
    poly mul_xk(size_t k) const {poly
        res(*this); res.a.insert(begin(res.a),
        k, 0); return res;}
    // divide by x^k, dropping coefficients
    poly div_xk(size_t k) const {k = min(k,
        a.size()); return vector<T>(begin(a) +
        k, end(a));}
    poly substr(size_t l, size_t r) const { //
        return mod_xk(r).div_xk(l)
        l = min(l, a.size()); r = min(r,
            a.size());
        return vector<T>(begin(a) + l,
            begin(a) + r);
    }
    poly inv(size_t n) const { // get inverse
        series mod x^n in O(nlgn)
        assert(!is_zero());
        poly ans = a[0].inv();
        size_t a = 1;
        while(a < n) {
            poly C = (ans * mod_xk(2 *
                a)).substr(a, 2 * a);
            ans -= (ans *
                C).mod_xk(a).mul_xk(a);
            a *= 2;
        }
        return ans.mod_xk(n);
    }
    poly operator *= (const poly &t)
        {/*fft::mul(a, t.a);*/ normalize();
        return *this;}
    poly operator * (const poly &t) const
        {return poly(*this) *= t;}
    poly reverse(size_t n, bool rev = 0) const
        { // reverses and leaves only n terms
        poly res(*this);
        if(rev) // If rev = 1 then tail goes
            to head
            res.a.resize(max(n, res.a.size()));
        std::reverse(res.a.begin(),
            res.a.end());
        return res.mod_xk(n);
    }
    // when divisor or quotient is small
    pair<poly, poly> divmod_slow(const poly
        &b) const {
        vector<T> A(a);
        vector<T> res;
        while(A.size() >= b.a.size()) {
            res.push_back(A.back() /
                b.a.back());
            if(res.back() != T(0))
                for(size_t i = 0; i <
                    b.a.size(); i++)
                    A[A.size() - i - 1] -=
                        res.back() *
                        b.a[b.a.size() - i - 1];
            A.pop_back();
        }
        std::reverse(begin(res), end(res));
        return {res, A};
    }
    // returns quotient and remainder of a mod
        b
    pair<poly, poly> divmod(const poly &b)
        const {
        if(deg() < b.deg())
            return {poly{0}, *this};
        int d = deg() - b.deg();
        if(min(d, b.deg()) < magic)
            return divmod_slow(b);
        poly D = (reverse(d + 1) * b.reverse(d
            + 1).inv(d + 1)).mod_xk(d +
            1).reverse(d + 1, 1);
        return {D, *this - D * b};
    }
    poly operator / (const poly &t) const
        {return divmod(t).first;}
    poly operator % (const poly &t) const
        {return divmod(t).second;}
    poly operator /= (const poly &t) {return
        *this = divmod(t).first;}
    poly operator %= (const poly &t) {return
        *this = divmod(t).second;}
    poly operator *= (const T &x) {
        for(auto &it: a) it *= x;
        normalize();
        return *this;
    }
```

```
poly operator /= (const T &x) {
    for(auto &it: a) it /= x;
    normalize();
    return *this;
}
poly operator * (const T &x) const {return
    poly(*this) *= x;}
poly operator / (const T &x) const {return
    poly(*this) /= x;}
T& lead() { return a.back(); } // leading
    coefficient
int deg() const {return a.empty() ? -inf :
    a.size() - 1;} // degree
bool is_zero() const { // is polynomial
    zero
    return a.empty();
}
T operator [](int idx) const {return idx
    >= (int)a.size() || idx < 0 ? T(0) :
    a[idx];}
T& coef(size_t idx) { // mutable reference
    at coefficient
    return a[idx];
}
bool operator == (const poly &t) const
    {return a == t.a;}
bool operator != (const poly &t) const
    {return a != t.a;}
poly deriv() { // calculate derivative
    vector<T> res;
    for(int i = 1; i <= deg(); i++) {
        res.push_back(T(i) * a[i]);
    }
    return res;
}
poly integr() { // calculate integral with
    C = 0
    vector<T> res = {0};
    for(int i = 0; i <= deg(); i++) {
        res.push_back(a[i] / T(i + 1));
    }
    return res;
}
size_t leading_xk() const { // Let p(x) =
    x^k * t(x), return k
```

```
    if(is_zero()) {
        return inf;
    }
    int res = 0;
    while(a[res] == T(0)) {
        res++;
    }
    return res;
}
poly log(size_t n) { // calculate log p(x)
    mod x^n
    assert(a[0] == T(1));
    return (deriv().mod_xk(n) *
        inv(n)).integr().mod_xk(n);
}
poly exp(size_t n) { // calculate exp p(x)
    mod x^n
    if(is_zero()) {
        return T(1);
    }
    assert(a[0] == T(0));
    poly ans = T(1);
    size_t a = 1;
    while(a < n) {
        poly C = ans.log(2 * a).div_xk(a) -
            substr(a, 2 * a);
        ans -= (ans *
            C).mod_xk(a).mul_xk(a);
        a *= 2;
    }
    return ans.mod_xk(n);

}
poly pow_slow(size_t k, size_t n) { // if
    k is small
    return k ? k % 2 ? (*this * pow_slow(k
        - 1, n)).mod_xk(n) : (*this *
        *this).mod_xk(n).pow_slow(k / 2,
        n) : T(1);
}
poly pow(size_t k, size_t n) { //
    calculate p^k(n) mod x^n in O(nlgnk)
    if(is_zero()) {
        return *this;
    }
```

```
    if(k < magic) {
        return pow_slow(k, n);
    }
    int i = leading_xk();
    T j = a[i];
    poly t = div_xk(i) / j;
    return bpow(j, k) * (t.log(n) *
        T(k)).exp(n).mul_xk(i *
        k).mod_xk(n);
}
poly mulx(T x) { // component-wise
    multiplication with x^k
    T cur = 1;poly res(*this);
    for(int i = 0; i <= deg();
        i++){res.coef(i) *= cur;cur *= x;}
    return res;
}
poly mulx_sq(T x) { // component-wise
    multiplication with x^{k^2}
    T cur = x, total = 1, xx = x * x;
    poly res(*this);
    for(int i = 0; i <= deg();
        i++){res.coef(i) *= total;total *=
        cur;cur *= xx;}
    return res;
}
vector<T> chirpz_even(T z, int n) { //
    P(1), P(z^2), P(z^4), ..., P(z^2(n-1))
    int m = deg();
    if(is_zero()) return vector<T>(n, 0);
    vector<T> vv(m + n);
    T zi = z.inv(), zz = zi * zi, cur =
        zi, total = 1;
    for(int i = 0; i <= max(n - 1, m);
        i++) {
        if(i <= m) {vv[m - i] = total;}
        if(i < n) {vv[m + i] = total;}
        total *= cur;cur *= zz;
    }
    poly w = (mulx_sq(z) * vv).substr(m, m
        + n).mulx_sq(z);
    vector<T> res(n);
    for(int i = 0; i < n; i++) res[i] =
        w[i];
    return res;
```

```
}
// calculate P(1), P(z), P(z^2), ...,
    P(z^(n-1)) in O(nlgn)
vector<T> chirpz(T z, int n) {
  auto even = chirpz_even(z, (n + 1) /
      2);
  auto odd = mulx(z).chirpz_even(z, n /
      2);
  vector<T> ans(n);
  for(int i = 0; i < n / 2; i++){ans[2 *
      i] = even[i];ans[2 * i + 1] =
      odd[i];}
  if(n % 2 == 1) ans[n - 1] =
      even.back();
  return ans;
}
template<typename iter> // auxiliary
    evaluation function
vector<T> eval(vector<poly> &tree, int v,
    iter l, iter r) {
  if(r - l == 1) {
    return {eval(*l)};
  } else {
    auto m = l + (r - l) / 2;
    auto A = (*this % tree[2 *
        v]).eval(tree, 2 * v, l, m);
    auto B = (*this % tree[2 * v +
        1]).eval(tree, 2 * v + 1, m,
        r);
    A.insert(end(A), begin(B), end(B));
    return A;
  }
}
// evaluate polynomial in (x1, ..., xn) in
    O(nlg2n)
vector<T> eval(vector<T> x) {
  int n = x.size();
  if(is_zero()) return vector<T>(n,
      T(0));
  vector<poly> tree(4 * n);build(tree,
      1, begin(x), end(x));
  return eval(tree, 1, begin(x), end(x));
}
template<typename iter>
  poly inter(vector<poly> &tree, int v, iter
      l, iter r, iter ly, iter ry) { //
      auxiliary interpolation function
    if(r - l == 1) return {*ly / a[0]};
    else {
      auto m = l + (r - l) / 2;
      auto my = ly + (ry - ly) / 2;
      auto A = (*this % tree[2 *
          v]).inter(tree, 2 * v, l, m,
          ly, my);
      auto B = (*this % tree[2 * v +
          1]).inter(tree, 2 * v + 1, m,
          r, my, ry);
      return A * tree[2 * v + 1] + B *
          tree[2 * v];
    }
  }
};
template<typename T>
T resultant(poly<T> a, poly<T> b) { //
    computes resultant of a and b
  if(b.is_zero()) return 0;
  else if(b.deg() == 0) return
      bpow(b.lead(), a.deg());
  else {
    int pw = a.deg();a %= b;pw -= a.deg();
    T mul = bpow(b.lead(), pw) *
        T((b.deg() & a.deg() & 1) ? -1 :
        1);
    T ans = resultant(b, a);
    return ans * mul;
  }
}
template<typename iter> // computes
    (x-a1)(x-a2)...(x-an) without building
    tree
poly<typename iter::value_type> kmul(iter L,
    iter R) {
  if(R - L == 1) {
    return vector<typename
        iter::value_type>{-*L, 1};
  } else {
    iter M = L + (R - L) / 2;
    return kmul(L, M) * kmul(M, R);
  }
}
template<typename T, typename iter> // builds
    evaluation tree for (x-a1)(x-a2)...(x-an)
poly<T> build(vector<poly<T>> &res, int v,
    iter L, iter R) {
  if(R - L == 1) {
    return res[v] = vector<T>{-*L, 1};
  } else {
    iter M = L + (R - L) / 2;
    return res[v] = build(res, 2 * v, L,
        M) * build(res, 2 * v + 1, M, R);
  }
}
template<typename T> // interpolates minimum
    polynomial from (xi, yi) pairs in O(nlg2n)
poly<T> inter(vector<T> x, vector<T> y) {
  int n = x.size();
  vector<poly<T>> tree(4 * n);
  return build(tree, 1, begin(x),
      end(x)).deriv().inter(tree, 1,
      begin(x), end(x), begin(y), end(y));
}
```

32 sat

```
struct Sat {
 int n = maxn, col[maxn] = {}, cnt, ver[maxn]
     = {}, versz, cer[maxn];
 vector<int> g[maxn], rg[maxn];
 bool mrk[maxn] = {};
 void addE(int x, int y) {
  g[x].push_back(y);
  rg[y].push_back(x);
 }
 void addOr(int x, int y) {
  addE(x ^ 1, y);
  addE(y ^ 1, x);
 }
 void dfsadd(int v){
  mrk[v]=1; for(auto
      &u:g[v])if(!mrk[u])dfsadd(u);
  ver[versz++]=v;
  }
  void dfsset(int v){
    col[v]=cnt;
    for(auto &u:rg[v])
      if(col[u]==-1)
        dfsset(u);
  }
  bool ok() {
    memset(mrk, 0, n);
    memset(col, -1, n * sizeof col[0]);
    for(int v = 0; v < n; v++)
      if(!mrk[v])
        dfsadd(v);
      while(versz)if(col[ver[--versz]]==-1)
           dfsset(ver[versz]), cnt++;
    for(int v = 0; v < n; v += 2)
      if(col[v]==col[v^1])
        return 0;
      else
        cer[v] = col[v^1] < col[v];
    return 1;
  }
}  sat;
```

33 treap

```
const int maxn = 2e5 + 17;
struct Node{
  int k, p;
  Node *l, *r;
};
typedef Node* Ni;
void split(Ni t, int k, Ni& l, Ni& r){
  if(!t)
    l = r = 0;
  else if(k < t -> k)
    split(t -> l, k, l, t -> l), r = t;
  else
    split(t -> r, k, t -> r, r), l = t;
}
void insert(Ni &t, Ni it){
  if(!t)
    t = it;
  else if(it -> p < t -> p)
    insert(it -> k < t -> k ? t -> l : t -> r,
         it);
  else
    split(t, it -> k, it -> l, it -> r), t =
         it;
}
// Implicit treap // GSS6
const int maxn = 1e6 + 17, mod = 998244353;
int nxP(){
  static int cur = 1;
  cur = (ll) cur * 3 % mod;
  return cur;
}
struct Store{
  int pre, suf, sum, ans;
  Store (int val = -mod){
    pre = suf = sum = ans = val;
  }
  Store(int a, int b, int c, int d): pre(a),
       suf(b), sum(c), ans(d) {}
};
Store operator +(Store l, Store r){
  if(l.sum == -mod)
    return r;
  if(r.sum == -mod)
    return l;
  return Store(max(l.pre, l.sum + r.pre),
       max(r.suf, l.suf + r.sum), l.sum +
       r.sum, max({l.ans, r.ans, l.suf +
       r.pre}));
}
struct Node{
  int k, p, val;
  Store ans;
  Node *l, *r;
};
typedef Node* Ni;
int cnt(Ni i){
  return i ? i -> k : 0;
}
Store ans(Ni i){
  return i ? i -> ans : Store();
}
void upd(Ni t){
  if(!t) return;
  t -> k = cnt(t -> l) + cnt(t -> r) + 1;
  t -> ans = ans(t -> l) + t -> val + ans(t ->
       r);
}
void split(Ni t, int k, Ni& l, Ni& r){
  if(!t)
    l = r = 0;
  else{
    if(k <= cnt(t -> l))
      split(t -> l, k, l, t -> l), r = t;
    else
      split(t -> r, k - 1 - cnt(t -> l), t ->
           r, r), l = t;
    upd(t);
  }
}
void merge(Ni &t, Ni l, Ni r){
  if(!l || !r)
    t = l ? l : r;
  else if(l -> p > r -> p)
    merge(l -> r, l -> r, r), t = l;
  else
    merge(r -> l, l, r -> l), t = r;
  upd(t);
}
Ni root;
void insert(int k, int v){
  Ni r;
  split(root, k, root, r);
  Ni nw = new Node({0, nxP(), v, v});
  merge(root, root, nw);
  merge(root, root, r);
}
void erase(int k){
  // removes kth element
  k++;
  Ni tmp, r;
  split(root, k, root, r);
  split(root, k - 1, root, tmp);
  merge(root, root, r);
}
int get(int l, int r){
  Ni qans, ri;
  split(root, r, root, ri);
  split(root, l, root, qans);
  int ret = ans(qans).ans;
  merge(root, root, qans);
  merge(root, root, ri);
  return ret;
}
void replace(int k, int v){
  erase(k);
  insert(k, v);
}
```

34 z-function

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i
            + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```