Contents

1 Basic

1.1 Increase Stack

```
//stack resize
asm( "mov %0,%%esp\n" ::"g"(mem+10000000) );
//change esp to rsp if 64-bit system

//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
  const rlim_t ks = 64*1024*1024;
  struct rlimit rl;
  int res=getrlimit(RLIMIT_STACK, &rl);
  if(res==0){
    if(rl.rlim_cur<ks){
      rl.rlim_cur=ks;
      res=setrlimit(RLIMIT_STACK, &rl);
    }
  }
}
```

1.2 Default

```
#include<bits/stdc++.h>
#include<unistd.h>
using namespace std;
#define FZ(n) memset((n),0,sizeof(n))
#define FMO(n) memset((n),-1,sizeof(n))
#define F first
#define S second
#define PB push_back
#define ALL(x) begin(x),end(x)
#define SZ(x) ((int)(x).size())
#define IOS ios_base::sync_with_stdio(0); cin.tie(0)
template<typename A, typename B>
ostream& operator <<(ostream &s, const pair<A,B> &p) {
  return s<<"("<<p.first<<","<<p.second<<")";
}
template<typename T>
ostream& operator <<(ostream &s, const vector<T> &c) {
  s<<"[ ";
  for (auto it : c) s << it << " ";
  s<<"]";
  return s;
}
// Let's Fight!

int main() {
    return 0;
}
```

2 Data Structure

2.1 Heavy Light Decomposition

```
// only one segment tree / no 0/1 base issue
// getPathSeg return the segment in order u->v
// fa[root] = root
typedef pair<int,int> pii;

int N,fa[MXN],belong[MXN],dep[MXN],sz[MXN],que[MXN];
int step,line[MXN],stPt[MXN],edPt[MXN];
vector<int> E[MXN], chain[MXN];

void DFS(int u){
  vector<int> &c = chain[belong[u]];
  for (int i=c.size()-1; i>=0; i--){
    int v = c[i];
    stPt[v] = step;
    line[step++] = v;
  }
  for (int i=0; i<(int)c.size(); i++){
    u = c[i];
    for (auto v : E[u]){
      if (fa[u] == v || (i && v == c[i-1])) continue;
      DFS(v);
    }
    edPt[u] = step-1;
  }
}
void build_chain(int st){
  int fr,bk;
  fr=bk=0; que[bk++] = 1; fa[st]=st; dep[st]=0;
  while (fr < bk){
    int u=que[fr++];
    for (auto v : E[u]){
      if (v == fa[u]) continue;
      que[bk++] = v;
      dep[v] = dep[u]+1;
      fa[v] = u;
    }
  }
  for (int i=bk-1,u,pos; i>=0; i--){
    u = que[i]; sz[u] = 1; pos = -1;
    for (auto v : E[u]){
      if (v == fa[u]) continue;
      sz[u] += sz[v];
      if (pos==-1 || sz[v]>sz[pos]) pos=v;
    }
    if (pos == -1) belong[u] = u;
    else belong[u] = belong[pos];
    chain[belong[u]].PB(u);
  }
  step = 0;
  DFS(st);
}
int getLCA(int u, int v){
  while (belong[u] != belong[v]){
    int a = chain[belong[u]].back();
    int b = chain[belong[v]].back();
    if (dep[a] > dep[b]) u = fa[a];
    else v = fa[b];
  }
  return sz[u] >= sz[v] ? u : v;
}
vector<pii> getPathSeg(int u, int v){
  vector<pii> ret1,ret2;
  while (belong[u] != belong[v]){
    int a = chain[belong[u]].back();
    int b = chain[belong[v]].back();
    if (dep[a] > dep[b]){
      ret1.PB({stPt[a],stPt[u]});
      u = fa[a];
    } else {
      ret2.PB({stPt[b],stPt[v]});
      v = fa[b];
    }
  }
  if (dep[u] > dep[v]) swap(u,v);
  ret1.PB({stPt[u],stPt[v]});
  reverse(ret2.begin(), ret2.end());
  ret1.insert(ret1.end(),ret2.begin(),ret2.end());
  return ret1;
}
// Usage
void build(){
  build_chain(1); //change root
  init(0,step,0); //init segment tree
}
int get_answer(int u, int v){
  int ret = -2147483647;
  vector<pii> vec = getPathSeg(u,v);
  for (auto it : vec)
    ; // check answer with segment [it.F, it.S]
  return ret;
}
```

2.2 Bigint

```
struct Bigint{
  static const int LEN = 60;
  static const int BIGMOD = 10000;

  int s;
int vl, v[LEN];
//  vector<int> v;
Bigint() : s(1) { vl = 0; }
Bigint(long long a) {
  s = 1; vl = 0;
  if (a < 0) { s = -1; a = -a; }
  while (a) {
    push_back(a % BIGMOD);
    a /= BIGMOD;
  }
}
Bigint(string str) {
  s = 1; vl = 0;
  int stPos = 0, num = 0;
  if (!str.empty() && str[0] == '-') {
    stPos = 1;
    s = -1;
  }
  for (int i=SZ(str)-1, q=1; i>=stPos; i--) {
    num += (str[i] - '0') * q;
    if ((q *= 10) >= BIGMOD) {
      push_back(num);
      num = 0; q = 1;
    }
  }
  if (num) push_back(num);
}

int len() const {
  return vl;
  //    return SZ(v);
}
bool empty() const { return len() == 0; }
void push_back(int x) {
  v[vl++] = x;
  //    v.PB(x);
}
void pop_back() {
  vl--;
  //    v.pop_back();
}
int back() const {
  return v[vl-1];
  //    return v.back();
}
void n() {
  while (!empty() && !back()) pop_back();
}
void resize(int nl) {
  vl = nl;
  fill(v, v+vl, 0);
  //    v.resize(nl);
  //    fill(ALL(v), 0);
}

void print() const {
  if (empty()) { putchar('0'); return; }
  if (s == -1) putchar('-');
  printf("%d", back());
  for (int i=len()-2; i>=0; i--) printf("%.4d",v[i]);
}
friend std::ostream& operator << (std::ostream& out,
    const Bigint &a) {
  if (a.empty()) { out << "0"; return out; }
  if (a.s == -1) out << "-";
  out << a.back();
  for (int i=a.len()-2; i>=0; i--) {
    char str[10];
    snprintf(str, 5, "%.4d", a.v[i]);
    out << str;
  }
  return out;
}

int cp3(const Bigint &b)const {
  if (s != b.s) return s > b.s;
  if (s == -1) return -(-*this).cp3(-b);
  if (len() != b.len()) return len()>b.len()?1:-1;
  for (int i=len()-1; i>=0; i--)
    if (v[i]!=b.v[i]) return v[i]>b.v[i]?1:-1;
  return 0;
}
bool operator < (const Bigint &b)const{ return cp3(b)
    ==-1; }
bool operator == (const Bigint &b)const{ return cp3(b
    )==0; }
bool operator > (const Bigint &b)const{ return cp3(b)
    ==1; }

Bigint operator - () const {
  Bigint r = (*this);
  r.s = -r.s;
  return r;
}
Bigint operator + (const Bigint &b) const {
  if (s == -1) return -(-(*this)+(-b));
  if (b.s == -1) return (*this)-(-b);
  Bigint r;
  int nl = max(len(), b.len());
  r.resize(nl + 1);
  for (int i=0; i<nl; i++) {
    if (i < len()) r.v[i] += v[i];
    if (i < b.len()) r.v[i] += b.v[i];
    if(r.v[i] >= BIGMOD) {
      r.v[i+1] += r.v[i] / BIGMOD;
      r.v[i] %= BIGMOD;
    }
  }
  r.n();
  return r;
}
Bigint operator - (const Bigint &b) const {
  if (s == -1) return -(-(*this)-(-b));
  if (b.s == -1) return (*this)+(-b);
  if ((*this) < b) return -(b-(*this));
  Bigint r;
  r.resize(len());
  for (int i=0; i<len(); i++) {
    r.v[i] += v[i];
    if (i < b.len()) r.v[i] -= b.v[i];
    if (r.v[i] < 0) {
      r.v[i] += BIGMOD;
      r.v[i+1]--;
    }
  }
  r.n();
  return r;
}
Bigint operator * (const Bigint &b) {
  Bigint r;
  r.resize(len() + b.len() + 1);
  r.s = s * b.s;
  for (int i=0; i<len(); i++) {
    for (int j=0; j<b.len(); j++) {
      r.v[i+j] += v[i] * b.v[j];
      if(r.v[i+j] >= BIGMOD) {
        r.v[i+j+1] += r.v[i+j] / BIGMOD;
        r.v[i+j] %= BIGMOD;
      }
    }
  }
  r.n();
  return r;
}
Bigint operator / (const Bigint &b) {
  Bigint r;
  r.resize(max(1, len()-b.len()+1));
  int oriS = s;
  Bigint b2 = b; // b2 = abs(b)
  s = b2.s = r.s = 1;
  for (int i=r.len()-1; i>=0; i--) {
    int d=0, u=BIGMOD-1;
    while(d<u) {
      int m = (d+u+1)>>1;
      r.v[i] = m;
      if((r*b2) > (*this)) u = m-1;
      else d = m;
    }
    r.v[i] = d;
  }
  s = oriS;
  r.s = s * b.s;
  r.n();
  return r;
  }
  Bigint operator % (const Bigint &b) {
    return (*this)-(*this)/b*b;
  }
};
```

2.3 Leftist Heap

```
const int MAXSIZE = 10000;

struct Node{
  int num,lc,rc;
  Node () : num(0), lc(-1), rc(-1) {}
  Node (int _v) : num(_v), lc(-1), rc(-1) {}
}tree[MAXSIZE];

int merge(int x, int y){
    if (x == -1) return y;
    if (y == -1) return x;
    if (tree[x].num < tree[y].num)
        swap(x, y);
    tree[x].rc = merge(tree[x].rc, y);
    swap(tree[x].lc, tree[x].rc);
    return x;
}

/* Usage
merge: root = merge(x, y)
delmin: root = merge(root.lc, root.rc)
*/
```

2.4 Treap

```
struct Node{
  int pri,num,cnt,lc,rc;
  Node () : pri(-1), num(0), cnt(0), lc(0), rc(0) {}
  Node (int _num){
    pri = (rand()<<15) + rand();
    num = _num;
    cnt = 1;
    lc = rc = 0;
  }
}tree[MX];

int nMem;

int get_rand(){
  return (rand()<<15) + rand();
}
int get_node(){
  tree[nMem] = Node();
  if (nMem >= MX) while(1);
  return nMem++;
}
void upd_node(int rt){
  if (!rt) return ;
  int lc=tree[rt].lc;
  int rc=tree[rt].rc;
  tree[rt].cnt = tree[lc].cnt + tree[rc].cnt + 1;
}
int merge(int a, int b){
  if (!a) return b;
  if (!b) return a;
  int res=0;
  if (tree[a].pri > tree[b].pri){
    res = a; //get_node();
    tree[res] = tree[a];
    tree[res].rc = merge(tree[res].rc,b);
  } else {
    res = b; //get_node();
    tree[res] = tree[b];
    tree[res].lc = merge(a,tree[res].lc);
  }
  upd_node(res);
  return res;
}
pair<int,int> split(int a, int k){
  if (k == 0) return {0,a};
  if (k == tree[a].cnt) return {a,0};
  int lc=tree[a].lc, rc=tree[a].rc;
  pair<int,int> res;
  int np=a; //get_node();
  //tree[np] = tree[a];
  if (tree[lc].cnt >= k){
    res = split(lc,k);
    tree[np].lc = res.S;
    res.S = np;
  } else {
    res = split(rc,k-tree[lc].cnt-1);
    tree[np].rc = res.F;
    res.F = np;
  }
  upd_node(res.F);
  upd_node(res.S);
  return res;
}
```

2.5 Unordered Hash Function

```
struct Key {
  int first,second;
  Key () {}
  Key (int _x, int _y) : first(_x),  second(_y) {}
  bool operator == (const Key &b) const {
    return tie(F,S) == tie(b.F,b.S);
  }
};
struct KeyHasher {
  size_t operator()(const Key& k) const {
    return k.first + k.second*100000;
  }
};

typedef unordered_map<Key,int,KeyHasher> map_t;

int main(int argc, char** argv){
  map_t mp;
  for (int i=0; i<10; i++)
    mp[Key(i,0)] = i+1;
  for (int i=0; i<10; i++)
    printf("%d\n", mp[Key(i,0)]);

  return 0;
}
```

3 Graph

3.1 Tarjan

```
const int MAXV = 101000;

int V, E;
vector<int> el[MAXV];
int dfn[MAXV], low[MAXV], did;
bool ins[MAXV];
stack<int> st;
int scc[MAXV], scn;

void tarjan(int u){
  cout << u << endl;
  dfn[u] = low[u] = ++did;
  st.push(u); ins[u] = true;

  for(int i=0; i<(int)el[u].size(); i++){
    int v = el[u][i];
    if(!dfn[v]){
      tarjan(v);
      low[u] = min(low[u], low[v]);
    }else if(ins[v]){
      low[u] = min(low[u], dfn[v]);
    }
  }

  if(dfn[u] == low[u]){
    int v;
    do{
      v = st.top();
      st.pop();
      scc[v] = scn;
      ins[v] = false;
    }while(v != u);
    scn ++;
  }
}

void calcscc(){
  did = scn = 0;
  for(int i=0; i<V; i++){
    if(!dfn[i]) tarjan(i);
  }
}
```

3.2 Kosaraju Scc

```
struct Scc{
  int n, nScc, vst[MXN], bln[MXN];
  vector<int> E[MXN], rE[MXN], vec;
  void init(int _n){
    n = _n;
    for (int i=0; i<MXN; i++){
      E[i].clear();
      rE[i].clear();
    }
  }
  void add_edge(int u, int v){
    E[u].PB(v);
    rE[v].PB(u);
  }
  void DFS(int u){
    vst[u]=1;
    for (auto v : E[u])
      if (!vst[v]) DFS(v);
    vec.PB(u);
  }
  void rDFS(int u){
    vst[u] = 1;
    bln[u] = nScc;
    for (auto v : rE[u])
      if (!vst[v]) rDFS(v);
  }
  void solve(){
    nScc = 0;
    vec.clear();
    FZ(vst);
    for (int i=0; i<n; i++)
      if (!vst[i]) DFS(i);
    reverse(vec.begin(),vec.end());
    FZ(vst);
    for (auto v : vec){
      if (!vst[v]){
        rDFS(v);
        nScc++;
      }
    }
  }
};
```

3.3 SW Min Cut

```
struct SW{ // O(V^3)
  static const int MXN = 514;
  int n,vst[MXN],del[MXN];
  int edge[MXN][MXN],wei[MXN];
  void init(int _n){
    n = _n;
    FZ(edge);
    FZ(del);
  }
  void add_edge(int u, int v, int w){
    edge[u][v] += w;
    edge[v][u] += w;
  }
  void search(int &s, int &t){
    FZ(vst); FZ(wei);
    s = t = -1;
    while (true){
      int mx=-1, cur=0;
      for (int i=0; i<n; i++)
        if (!del[i] && !vst[i] && mx<wei[i])
          cur = i, mx = wei[i];
      if (mx == -1) break;
      vst[cur] = 1;
      s = t;
      t = cur;
      for (int i=0; i<n; i++)
        if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
    }
  }
  int solve(){
    int res = 2147483647;
    for (int i=0,x,y; i<n-1; i++){
      search(x,y);
      res = min(res,wei[y]);
      del[y] = 1;
      for (int j=0; j<n; j++)
        edge[x][j] = (edge[j][x] += edge[y][j]);
    }
    return res;
  }
}graph;
```

3.4 Dinic

```
struct Dinic{
  static const int MXN = 10000;
  struct Edge{ int v,f,re; };
  int n,s,t,level[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t){
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u].PB({v,f,SZ(E[v])});
    E[v].PB({u,0,SZ(E[u])-1});
  }
  bool BFS(){
    FMO(level);
    queue<int> que;
    que.push(s);
    level[s] = 0;
    while (!que.empty()){
      int u = que.front(); que.pop();
      for (auto it : E[u]){
        if (it.f > 0 && level[it.v] == -1){
          level[it.v] = level[u]+1;
          que.push(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
      if (it.f > 0 && level[it.v] == level[u]+1){
        int tf = DFS(it.v, min(nf,it.f));
        res += tf; nf -= tf; it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      }
    }
    if (!res) level[u] = -1;
    return res;
  }
  int flow(int res=0){
    while ( BFS() )
      res += DFS(s,2147483647);
    return res;
  }
}flow;
```

3.5 Cost Flow

```
typedef pair<long long, long long> pll;
struct CostFlow {
  static const int MXN = 205;
  static const long long INF = 102938475610293847LL;
  struct Edge {
    int v, r;
    long long f, c;
  };
  int n, s, t, prv[MXN], prvL[MXN], inq[MXN];
  long long dis[MXN], fl, cost;
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t) {
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
    fl = cost = 0;
  }
  void add_edge(int u, int v, long long f, long long c)
      {
    E[u].PB({v, SZ(E[v])  , f,  c});
    E[v].PB({u, SZ(E[u])-1, 0, -c});
  }
  pll flow() {
    while (true) {
      for (int i=0; i<n; i++) {
        dis[i] = INF;
        inq[i] = 0;
      }
      dis[s] = 0;
      queue<int> que;
      que.push(s);
      while (!que.empty()) {
        int u = que.front(); que.pop();
        inq[u] = 0;
        for (int i=0; i<SZ(E[u]); i++) {
          int v = E[u][i].v;
          long long w = E[u][i].c;
          if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
            prv[v] = u; prvL[v] = i;
            dis[v] = dis[u] + w;
            if (!inq[v]) {
              inq[v] = 1;
              que.push(v);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      long long tf = INF;
      for (int v=t; v!=s; v=u) {
        int u=prv[v], l=prvL[v];
        tf = min(tf, E[u][l].f);
      }
      for (int v=t; v!=s; v=u) {
        int u=prv[v], l=prvL[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
}flow;
```

3.6 Isap

```
struct Isap{
  static const int MXN = 10000;
  struct Edge{ int v,f,re; };
  int n,s,t,h[MXN],gap[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t){
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u].PB({v,f,SZ(E[v])});
    E[v].PB({u,0,SZ(E[u])-1});
  }
  int DFS(int u, int nf, int res=0){
    if (u == t) return nf;
    for (auto &it : E[u]){
      if (h[u]==h[it.v]+1 && it.f>0){
        int tf = DFS(it.v,min(nf,it.f));
        res += tf; nf -= tf; it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      }
    }
    if (nf){
      if (--gap[h[u]] == 0) h[s]=n;
      gap[++h[u]]++;
    }
    return res;
  }
  int flow(int res=0){
    FZ(h); FZ(gap);
    gap[0] = n;
    while (h[s] < n) res += DFS(s,2147483647);
    return res;
  }
}flow;
```

3.7 2 Com Flow

```
const int MAXN = 64;
const int INF = 1029384756;

int N;
int s1, s2, t1, t2, d1, d2, S, T;
int edge[MAXN][MAXN];
int cap[MAXN][MAXN];

int h[MAXN], gap[MAXN];
bool vis[MAXN];

int isap(int v, int f)
{
    if(v == T)return f;

    if(vis[v])return 0;
    vis[v] = true;

    for(int i=0; i<N+2; i++)
    {
        if(cap[v][i] <= 0)continue;
        if(h[i] != h[v] - 1)continue;
        int res = isap(i, min(cap[v][i], f));
        if(res > 0)
        {
            cap[v][i] -= res;
            cap[i][v] += res;
            return res;
        }
    }

    gap[h[v]]--;
    if(gap[h[v]] <= 0)h[S] = N + 4;
    h[v]++;
    gap[h[v]]++;

    return 0;
}

int get_flow()
{
    for(int i=0; i<MAXN; i++)
    {
        h[i] = gap[i] = 0;
    }
    gap[0] = N + 2;

    int flow = 0;

    while(h[S] <= N + 3)
    {
        for(int i=0; i<N+2; i++)
        {
            vis[i] = false;
        }

        int df = isap(S, INF);
        flow += df;
    }

    return flow;
}

int main()
{
    ios_base::sync_with_stdio(0);

    int TT;
    cin>>TT;
    while(TT--)
    {
        cin>>N;
        cin>>s1>>t1>>d1>>s2>>t2>>d2;

        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                edge[i][j] = 0;
            }
        }

        for(int i=0; i<N; i++)
        {
            string s;
            cin>>s;
            for(int j=0; j<N; j++)
            {
                if(s[j] == 'X')edge[i][j] = 0;
                else if(s[j] == 'O')edge[i][j] = 1;
                else if(s[j] == 'N')edge[i][j] = INF;
            }
        }

        int ans = 0;

        S = N;
        T = N + 1;

        //first
        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                cap[i][j] = edge[i][j];
            }
        }

        cap[S][s1] = cap[t1][T] = d1;
        cap[S][s2] = cap[t2][T] = d2;

        ans = get_flow();

        //second
        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                cap[i][j] = edge[i][j];
            }
        }

        cap[S][s1] = cap[t1][T] = d1;
        cap[S][t2] = cap[s2][T] = d2;

        ans = min(ans, get_flow());

        cout<<(ans == d1 + d2 ? "Yes" : "No")<<endl;
    }

    return 0;
}
```

3.8 DMST With Sol

```
const int INF = 1029384756;

struct edge_t{
  int u,v,w;
  set< pair<int,int> > add, sub;
  edge_t() : u(-1), v(-1), w(0) {}
  edge_t(int _u, int _v, int _w) {
    u = _u; v = _v; w = _w;
    add.insert({u, v});
  }
  edge_t& operator += (const edge_t& obj) {
    w += obj.w;
    FOR (it, obj.add) {
      if (!sub.count(*it)) add.insert(*it);
      else sub.erase(*it);
    }
    FOR (it, obj.sub) {
      if (!add.count(*it)) sub.insert(*it);
      else add.erase(*it);
    }
    return *this;
  }
  edge_t& operator -= (const edge_t& obj) {
    w -= obj.w;
    FOR (it, obj.sub) {
      if (!sub.count(*it)) add.insert(*it);
      else sub.erase(*it);
    }
    for (auto it : obj.add) {
      if (!add.count(it)) sub.insert(it);
      else add.erase(it);
    }
    return *this;
  }
}eg[MXN*MXN],prv[MXN],EDGE_INF(-1,-1,INF);
int N,M;
int cid,incyc[MXN],contracted[MXN];
vector<int> E[MXN];

edge_t dmst(int rt){
  edge_t cost;
  for (int i=0; i<N; i++){
    contracted[i] = incyc[i] = 0;
    prv[i] = EDGE_INF;
  }
  cid = 0;
  int u,v;
  while (true){
    for (v=0; v<N; v++){
      if (v != rt && !contracted[v] && prv[v].w == INF)
        break;
    }
    if (v >= N) break; // end
    for (int i=0; i<M; i++){
      if (eg[i].v == v && eg[i].w < prv[v].w)
        prv[v] = eg[i];
    }
    if (prv[v].w == INF) // not connected
      return EDGE_INF;
    cost += prv[v];
    for (u=prv[v].u; u!=v && u!=-1; u=prv[u].u);
    if (u == -1) continue;
    incyc[v] = ++cid;
    for (u=prv[v].u; u!=v; u=prv[u].u){
      contracted[u] = 1;
      incyc[u] = cid;
    }
    for (int i=0; i<M; i++){
      if (incyc[eg[i].u] != cid && incyc[eg[i].v] ==
          cid){
        eg[i] -= prv[eg[i].v];
      }
    }
    for (int i=0; i<M; i++){
      if (incyc[eg[i].u] == cid) eg[i].u = v;
      if (incyc[eg[i].v] == cid) eg[i].v = v;
      if (eg[i].u == eg[i].v) eg[i--] = eg[--M];
    }
    for (int i=0; i<N; i++){
      if (contracted[i]) continue;
      if (prv[i].u>=0 && incyc[prv[i].u] == cid)
        prv[i].u = v;
    }
    prv[v] = EDGE_INF;
  }
  return cost;
}

void solve(){
  edge_t cost = dmst(0);
  for (auto it : cost.add){ // find a solution
    E[it.F].PB(it.S);
    prv[it.S] = edge_t(it.F,it.S,0);
  }
}
```

3.9 Maximum Clique

```
class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
    int s[MV][MV/30+1];
    vector<int> sol;

    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }

    /* Zero Base */
    void addEdge(int u, int v) {
        if(u > v) swap(u, v);
        if(u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }

    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for(int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if(k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if(c == 0) {
            if(k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for(int i=0; i<(V+31)/32; i++) {
            for(int a = s[k][i]; a ; d++) {
                if(k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while(lb!=1) {
                    lb = (unsigned int)(lb) >> 1;
                    lg ++;
                }
                int u = i*32 + lg;
                if(k + dp[u] <= ans) return 0;
                if(dfs(u, k+1)) {
                    sol.push_back(v);
                    return 1;
                }
            }
        }
        return 0;
    }

    int solve() {
        for(int i=V-1; i>=0; i--) {
          dfs(i, 1);
          dp[i] = ans;
        }
        return ans;
    }
};
```

3.10 Minimum Mean Cycle

```
/* minimum mean cycle */
const int MAXE = 1805;
const int MAXN = 35;
const double inf = 1029384756;
const double eps = 1e-6;
struct Edge {
  int v,u;
  double c;
};
int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN];
Edge e[MAXE];
vector<int> edgeID, cycle, rho;
double d[MAXN][MAXN];
inline void bellman_ford() {
  for(int i=0; i<n; i++) d[0][i]=0;
  for(int i=0; i<n; i++) {
    fill(d[i+1], d[i+1]+n, inf);
    for(int j=0; j<m; j++) {
      int v = e[j].v, u = e[j].u;
      if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
        d[i+1][u] = d[i][v]+e[j].c;
        prv[i+1][u] = v;
        prve[i+1][u] = j;
      }
    }
  }
}
double karp_mmc() {
  // returns inf if no cycle, mmc otherwise
  double mmc=inf;
  int st = -1;
  bellman_ford();
  for(int i=0; i<n; i++) {
    double avg=-inf;
    for(int k=0; k<n; k++) {
      if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i])
          /(n-k));
      else avg=max(avg,inf);
    }
    if (avg < mmc) tie(mmc, st) = tie(avg, i);
  }
  FZ(vst); edgeID.clear(); cycle.clear(); rho.clear();
  for (int i=n; !vst[st]; st=prv[i--][st]) {
    vst[st]++;
    edgeID.PB(prve[i][st]);
    rho.PB(st);
  }
  while (vst[st] != 2) {
    int v = rho.back(); rho.pop_back();
    cycle.PB(v);
    vst[v]++;
  }
  reverse(ALL(edgeID));
  edgeID.resize(SZ(cycle));
  return mmc;
}
```

3.11 Kuhn Munkress

```
struct KM{
// Maximum Bipartite Weighted Matching (Perfect Match)
  static const int MXN = 650;
  static const int INF = 2147483647; // long long
  int n,match[MXN],vx[MXN],vy[MXN];
  int edge[MXN][MXN],lx[MXN],ly[MXN],slack[MXN];
  // ^^^^ long long
  void init(int _n){
    n = _n;
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        edge[i][j] = 0;
  }
  void add_edge(int x, int y, int w){ // long long
    edge[x][y] = w;
  }
  bool DFS(int x){
    vx[x] = 1;
    for (int y=0; y<n; y++){
      if (vy[y]) continue;
      if (lx[x]+ly[y] > edge[x][y]){
        slack[y] = min(slack[y], lx[x]+ly[y]-edge[x][y
            ]);
      } else {
        vy[y] = 1;
        if (match[y] == -1 || DFS(match[y])){
          match[y] = x;
          return true;
        }
      }
    }
    return false;
  }
  int solve(){
    fill(match,match+n,-1);
    fill(lx,lx+n,-INF);
    fill(ly,ly+n,0);
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        lx[i] = max(lx[i], edge[i][j]);
    for (int i=0; i<n; i++){
      fill(slack,slack+n,INF);
      while (true){
        fill(vx,vx+n,0);
        fill(vy,vy+n,0);
        if ( DFS(i) ) break;
        int d = INF; // long long
        for (int j=0; j<n; j++)
          if (!vy[j]) d = min(d, slack[j]);
        for (int j=0; j<n; j++){
          if (vx[j]) lx[j] -= d;
          if (vy[j]) ly[j] += d;
          else slack[j] -= d;
        }
      }
    }
    int res=0;
    for (int i=0; i<n; i++)
      res += edge[match[i]][i];
    return res;
  }
}graph;
```

3.12 Simple Graph Matching

```
struct GenMatch { // 1-base
  static const int MAXN = 250;
  int V;
  bool el[MAXN][MAXN];
  int pr[MAXN];
  bool inq[MAXN],inp[MAXN],inb[MAXN];
  queue<int> qe;
  int st,ed;
  int nb;
  int bk[MAXN],djs[MAXN];
  int ans;
  void init(int _V) {
    V = _V;
    FZ(el); FZ(pr);
    FZ(inq); FZ(inp); FZ(inb);
    FZ(bk); FZ(djs);
    ans = 0;
  }
  void add_edge(int u, int v) {
    el[u][v] = el[v][u] = 1;
  }
  int lca(int u,int v) {
    memset(inp,0,sizeof(inp));
    while(1) {
      u = djs[u];
      inp[u] = true;
      if(u == st) break;
      u = bk[pr[u]];
    }
    while(1) {
      v = djs[v];
      if(inp[v]) return v;
      v = bk[pr[v]];
    }
    return v;
  }
  void upd(int u) {
    int v;
    while(djs[u] != nb) {
      v = pr[u];
      inb[djs[u]] = inb[djs[v]] = true;
      u = bk[v];
      if(djs[u] != nb) bk[u] = v;
    }
  }
  void blo(int u,int v) {
    nb = lca(u,v);
    memset(inb,0,sizeof(inb));
    upd(u); upd(v);
    if(djs[u] != nb) bk[u] = v;
    if(djs[v] != nb) bk[v] = u;
    for(int tu = 1; tu <= V; tu++)
      if(inb[djs[tu]]) {
        djs[tu] = nb;
        if(!inq[tu]){
          qe.push(tu);
          inq[tu] = 1;
        }
      }
  }
  void flow() {
    memset(inq,false,sizeof(inq));
    memset(bk,0,sizeof(bk));
    for(int i = 1; i <= V;i++)
      djs[i] = i;

    while(qe.size()) qe.pop();
    qe.push(st);
    inq[st] = 1;
    ed = 0;
    while(qe.size()) {
      int u = qe.front(); qe.pop();
      for(int v = 1; v <= V; v++)
        if(el[u][v] && (djs[u] != djs[v]) && (pr[u] !=
            v)) {
          if((v == st) || ((pr[v] > 0) && bk[pr[v]] >
              0))
            blo(u,v);
          else if(bk[v] == 0) {
            bk[v] = u;
            if(pr[v] > 0) {
              if(!inq[pr[v]]) qe.push(pr[v]);
            } else {
              ed = v;
              return;
            }
          }
        }
    }
  }
  void aug() {
    int u,v,w;
    u = ed;
    while(u > 0) {
      v = bk[u];
      w = pr[v];
      pr[v] = u;
      pr[u] = v;
      u = w;
    }
  }
  int solve() {
    memset(pr,0,sizeof(pr));
    for(int u = 1; u <= V; u++)
      if(pr[u] == 0) {
        st = u;
        flow();
        if(ed > 0) {
          aug();
          ans ++;
        }
      }
    return ans;
  }
};

int main() {
  gp.init(V);
  for(int i=0; i<E; i++) {
    int u, v;
    cin >> u >> v;
    gp.edge(u, v);
  }
  cout << gp.solve() << endl;
}
```

3.13 Bipartite Matching

```
bool DFS(int u){
  for (auto v : E[u]){
    if (!vst[v]){
      vst[v]=1;
      if (match[v] == -1 || DFS(match[v])){
        match[v] = u; match[u] = v;
        return true;
      }
    }
  }
  return false;
}
int DoMatch(int res=0){
  memset(match,-1,sizeof(match));
  for (int i=1; i<=N; i++){
    if (match[i] == -1){
      memset(vst,0,sizeof(vst));
      DFS(i);
    }
  }
  for (int i=1; i<=N; i++)
    if (match[i] != -1) res++;
  return res;
}
```

4 Math

4.1 Simplex

```
const int maxn = 111;
const int maxm = 111;
const double eps = 1E-10;

double a[maxn][maxm], b[maxn], c[maxm], d[maxn][maxm];
double x[maxm];
int ix[maxn + maxm]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
//
// usage :
// value = simplex(a, b, c, N, M);
double simplex(double a[maxn][maxm], double b[maxn],
    double c[maxm], int n, int m) {
    ++m;
    int r = n, s = m - 1;
    memset(d, 0, sizeof(d));
    for (int i = 0; i < n + m; ++i) ix[i] = i;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i
            ][j];
        d[i][m - 1] = 1;
        d[i][m] = b[i];
        if (d[r][m] > d[i][m]) r = i;
    }
    for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
    d[n + 1][m - 1] = -1;
    for (double dd;; ) {
        if (r < n) {
            int t = ix[s]; ix[s] = ix[r + m]; ix[r + m]
                 = t;
            d[r][s] = 1.0 / d[r][s];
            for (int j = 0; j <= m; ++j) if (j != s) d[
                r][j] *= -d[r][s];
            for (int i = 0; i <= n + 1; ++i) if (i != r
                ) {
                for (int j = 0; j <= m; ++j) if (j != s
                    ) d[i][j] += d[r][j] * d[i][s];
                d[i][s] *= d[r][s];
            }
        }
        r = -1; s = -1;
        for (int j = 0; j < m; ++j) if (s < 0 || ix[s]
            > ix[j]) {
            if (d[n + 1][j] > eps || (d[n + 1][j] > -
                eps && d[n][j] > eps)) s = j;
        }
        if (s < 0) break;
        for (int i = 0; i < n; ++i) if (d[i][s] < -eps)
             {
            if (r < 0 || (dd = d[r][m] / d[r][s] - d[i
                ][m] / d[i][s]) < -eps || (dd < eps &&
                ix[r + m] > ix[i + m])) r = i;
        }
        if (r < 0) return -1; // not bounded
    }
    if (d[n + 1][m] < -eps) return -1; // not
        executable
    double ans = 0;
    for(int i=0; i<m; i++) x[i] = 0;
    for (int i = m; i < n + m; ++i) { // the missing
        enumerated x[i] = 0
        if (ix[i] < m - 1)
        {
            ans += d[i - m][m] * c[ix[i]];
            x[ix[i]] = d[i-m][m];
        }
    }
    return ans;
}
```

4.2 Ax+by=gcd

```
typedef pair<int, int> pii;

pii gcd(int a, int b){
  if(b == 0) return make_pair(1, 0);
  else{
    int p = a / b;
    pii q = gcd(b, a % b);
    return make_pair(q.second, q.first - q.second * p);
  }
}
```

4.3 Fast Fourier Transform

```
// const int MAXN = 262144;
// (must be 2^k)

typedef long double ld;
typedef complex<ld> cplx;
const ld PI = acosl(-1);
const cplx I(0, 1);

cplx omega[MAXN+1];
void pre_fft()
{
  for(int i=0; i<=MAXN; i++)
    omega[i] = exp(i * 2 * PI / MAXN * I);
}
void fft(int n, cplx a[], bool inv=false)
{
  int basic = MAXN / n;
  int theta = basic;
  for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    for (int i = 0; i < mh; i++) {
      cplx w = omega[inv ? MAXN-(i*theta%MAXN) : i*
          theta%MAXN];
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        cplx x = a[j] - a[k];
        a[j] += a[k];
        a[k] = w * x;
      }
    }
    theta = (theta * 2) % MAXN;
  }
  int i = 0;
  for (int j = 1; j < n - 1; j++) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
  if (inv)
    for (i = 0; i < n; i++)
      a[i] /= n;
}
```

4.4 Pollard Rho

```
/* pollard rho */
// does not work when n is prime
long long modit(long long x,long long mod) {
  if(x>=mod) x-=mod;
  //if(x<0) x+=mod;
  return x;
}
long long mult(long long x,long long y,long long mod) {
  long long s=0,m=x%mod;
  while(y) {
    if(y&1) s=modit(s+m,mod);
    y>>=1;
    m=modit(m+m,mod);
  }
  return s;
}
long long f(long long x,long long mod) {
  return modit(mult(x,x,mod)+1,mod);
}
long long pollard_rho(long long n) {
  long long x,x2;
  if(!(n&1)) return 2;
  //x=x2=randll()%n;
  x=x2=2;
  while(1) {
    x=f(x,n); x2=f(f(x2,n),n);
    long long d=__gcd(abs(x-x2),n);
    if(d!=1&&d!=n) return d;
  }
}
```

4.5 Polygen

```
class PolynomialGenerator {
  /* for a nth-order polynomial f(x), *
   * given f(0), f(1), ..., f(n) *
   * express f(x) as sigma_i{c_i*C(x,i)} */
  public:
    int n;
    vector<long long> coef;
    // initialize and calculate f(x), vector _fx should
        be
    // filled with f(0) to f(n)
      PolynomialGenerator(int _n,vector<long long> _fx)
          :n(_n
          ),coef(_fx) {
        for(int i=0;i<n;i++)
          for(int j=n;j>i;j--)
            coef[j]-=coef[j-1];
      }
    // evaluate f(x), runs in O(n)
    long long eval(int x) {
      long long m=1,ret=0;
      for(int i=0;i<=n;i++) {
        ret+=coef[i]*m;
        m=m*(x-i)/(i+1);
      }
      return ret;
    }
};
```

4.6 Gauss Elimination

```
const int MAX = 300;
const double EPS = 1e-8;

double mat[MAX][MAX];
void Gauss(int n) {
  for(int i=0; i<n; i++) {
    bool ok = 0;
    for(int j=i; j<n; j++) {
      if(fabs(mat[j][i]) > EPS) {
        swap(mat[j], mat[i]);
        ok = 1;
        break;
      }
    }
    if(!ok) continue;

    double fs = mat[i][i];
    for(int j=i+1; j<n; j++) {
      double r = mat[j][i] / fs;
      for(int k=i; k<n; k++) {
        mat[j][k] -= mat[i][k] * r;
      }
    }
  }
}
```

4.7 Mod Utils

```
/// _fd(a,b)  floor(a/b).
/// _rd(a,m)  a-floor(a/m)*m.
/// _pv(a,m,r) largest x s.t x<=a && x%m == r.
/// _nx(a,m,r) smallest x s.t x>=a && x%m == r.
/// _ct(a,b,m,r) |A| , A = { x : a<=x<=b && x%m == r }.

int _fd(int a,int b){ return a<0?(-~a/b-1):a/b; }
int _rd(int a,int m){ return a-_fd(a,m)*m; }
int _pv(int a,int m,int r)
{
    r=(r%m+m)%m;
    return _fd(a-r,m)*m+r;
}
int _nt(int a,int m,int r)
{
    m=abs(m);
    r=(r%m+m)%m;
    return _fd(a-r-1,m)*m+r+m;
}
int _ct(int a,int b,int m,int r)
{
    m=abs(m);
    a=_nt(a,m,r);
    b=_pv(b,m,r);
    return (a>b)?0:((b-a+m)/m);
}
```

4.8 Ntt

```
int P=605028353,root=3,MAXNUM=262144;
// Remember coefficient are mod P
/*
p=a*2^n+1
n   2^n         p             a     root
5   32          97            3     5
6   64          193           3     5
7   128         257           2     3
8   256         257           1     3
9   512         7681          15    17
10  1024        12289         12    11
11  2048        12289         6     11
12  4096        12289         3     11
13  8192        40961         5     3
14  16384       65537         4     3
15  32768       65537         2     3
16  65536       65537         1     3
17  131072      786433        6     10
18  262144      786433        3     10 (605028353,
    2308, 3)
19  524288      5767169       11    3
20  1048576     7340033       7     3
21  2097152     23068673      11    3
22  4194304     104857601     25    3
23  8388608     167772161     20    3
24  16777216    167772161     10    3
25  33554432    167772161     5     3 (1107296257, 33,
    10)
26  67108864    469762049     7     3
27  134217728   2013265921    15    31
*/
int bigmod(long long a,int b){
  if(b==0)return 1;
  return (bigmod((a*a)%P,b/2)*(b%2?a:1ll))%P;
}
int inv(int a,int b){
  if(a==1)return 1;
  return (((long long)(a-inv(b%a,a))*b+1)/a)%b;
}
std::vector<long long> ps(MAXNUM);
std::vector<int> rev(MAXNUM);
struct poly{
  std::vector<unsigned int> co;
  int n;//polynomial degree = n
  poly(int d){n=d;co.resize(n+1,0);}
  void trans2(int NN){
    int r=0,st,N;
    unsigned int a,b;
    while((1<<r)<(NN>>1))++r;
    for(N=2;N<=NN;N<<=1,--r){
      for(st=0;st<NN;st+=N){
        int i,ss=st+(N>>1);
        for(i=(N>>1)-1;i>=0;--i){
          a=co[st+i]; b=(ps[i<<r]*co[ss+i])%P;
          co[st+i]=a+b; if(co[st+i]>=P)co[st+i]-=P;
          co[ss+i]=a+P-b; if(co[ss+i]>=P)co[ss+i]-=P;
        }
      }
    }
  }
  void trans1(int NN){
    int r=0,st,N;
    unsigned int a,b;
    for(N=NN;N>1;N>>=1,++r){
      for(st=0;st<NN;st+=N){
        int i,ss=st+(N>>1);
        for(i=(N>>1)-1;i>=0;--i){
          a=co[st+i]; b=co[ss+i];
          co[st+i]=a+b; if(co[st+i]>=P)co[st+i]-=P;
          co[ss+i]=((a+P-b)*ps[i<<r])%P;
        }
      }
    }
  }
  poly operator*(const poly& _b)const{
    poly a=*this,b=_b;
    int k=n+b.n,i,N=1;
    while(N<=k)N*=2;
    a.co.resize(N,0); b.co.resize(N,0);
    int r=bigmod(root,(P-1)/N),Ni=inv(N,P);
    ps[0]=1;
    for(i=1;i<N;++i)ps[i]=(ps[i-1]*r)%P;
    a.trans1(N);b.trans1(N);
    for(i=0;i<N;++i)a.co[i]=((long long)a.co[i]*b.co[i
        ])%P
      ;
    r=inv(r,P);
    for(i=1;i<N/2;++i)std::swap(ps[i],ps[N-i]);
    a.trans2(N);
    for(i=0;i<N;++i)a.co[i]=((long long)a.co[i]*Ni)%P;
    a.n=n+_b.n; return a;
  }
};
```

4.9 Chinese Remainder

```
int pfn; // number of distinct prime factors
int pf[MAXNUM]; // prime factor powers
int rem[MAXNUM]; // corresponding remainder
int pm[MAXNUM];
inline void generate_primes() {
  int i,j;
  pnum=1;
  prime[0]=2;
  for(i=3;i<MAXVAL;i+=2) {
    if(nprime[i]) continue;
    prime[pnum++]=i;
    for(j=i*i;j<MAXVAL;j+=i) nprime[j]=1;
  }
}
inline int inverse(int x,int p) {
  int q,tmp,a=x,b=p;
  int a0=1,a1=0,b0=0,b1=1;
  while(b) {
    q=a/b; tmp=b; b=a-b*q; a=tmp;
    tmp=b0; b0=a0-b0*q; a0=tmp;
    tmp=b1; b1=a1-b1*q; a1=tmp;
  }
  return a0;
}
inline void decompose_mod() {
  int i,p,t=mod;
  pfn=0;
  for(i=0;i<pnum&&prime[i]<=t;i++) {
    p=prime[i];
    if(t%p==0) {
      pf[pfn]=1;
      while(t%p==0) {
        t/=p;
        pf[pfn]*=p;
      }
      pfn++;
    }
  }
  if(t>1) pf[pfn++]=t;
}
inline int chinese_remainder() {
  int i,m,s=0;
  for(i=0;i<pfn;i++) {
    m=mod/pf[i];
    pm[i]=(long long)m*inverse(m,pf[i])%mod;
    s=(s+(long long)pm[i]*rem[i])%mod;
  }
  return s;
}
```

4.10 Primes

```
/*
 * 12721
 * 13331
 * 14341
 * 75577
 * 123457
 * 222557
 * 556679
 * 999983
 * 1097774749
 * 1076767633
 * 100102021
 * 999997771
 * 1001010013
 * 1000512343
 * 987654361
 * 999991231
 * 999888733
 * 98789101
 * 987777733
 * 999991921
 * 1010101333
 * 1010102101
 * 1000000000039
 * 1000000000000037
 * 2305843009213693951
 * 4611686018427387847
 * 9223372036854775783
 * 18446744073709551557
 */

int mu[MX],p_tbl[MX];
vector<int> primes;
void sieve() {
  mu[1] = p_tbl[1] = 1;
  for (int i=2; i<MX; i++) {
    if (!p_tbl[i]) {
      p_tbl[i] = i;
      primes.PB(i);
      mu[i] = -1;
    }
    for (auto p : primes) {
      int x = i*p;
      if (x >= M) break;
      p_tbl[x] = p;
      mu[x] = -mu[i];
      if (i%p==0) {
        mu[x] = 0;
        break;
      }
    }
  }
}

vector<int> factor(int x) {
  vector<int> fac{1};
  while (x > 1) {
    int fn=SZ(fac), p=p_tbl[x], pos=0;
    while (x%p == 0) {
      x /= p;
      for (int i=0; i<fn; i++)
        fac.PB(fac[pos++]*p);
    }
  }
  return fac;
}
```

4.11 Miller Rabin

```
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 3,825,123,056,546,413,051  9 :  primes <= 23
long long power(long long x,long long p,long long mod){
  long long s=1,m=x;
  while(p) {
    if(p&1) s=mult(s,m,mod);
    p>>=1;
    m=mult(m,m,mod);
  }
  return s;
}
bool witness(long long a,long long n,long long u,int t)
    {
  long long x=power(a,u,n);
  for(int i=0;i<t;i++) {
    long long nx=mult(x,x,n);
    if(nx==1&&x!=1&&x!=n-1) return 1;
    x=nx;
  }
  return x!=1;
}
bool miller_rabin(long long n,int s=100) {
  // iterate s times of witness on n
  // return 1 if prime, 0 otherwise
  if(n<2) return 0;
  if(!(n&1)) return n==2;
  long long u=n-1;
  int t=0;
  // n-1 = u*2^t
  while(u&1) {
    u>>=1;
    t++;
  }
  while(s--) {
    long long a=randll()%(n-1)+1;
    if(witness(a,n,u,t)) return 0;
  }
  return 1;
}
```

5 Geometry

5.1 Half Plane Intersection

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

using namespace std;

#define PB push_back
#define _x first
#define _y second

const int MXL = 5000;
const double EPS = 1e-8;

typedef pair<double, double> pdd;
typedef pair<pdd, pdd> Line;

pdd operator + (const pdd p1, const pdd p2){
  return pdd(p1._x + p2._x, p1._y + p2._y);
}

pdd operator - (const pdd p1, const pdd p2){
  return pdd(p1._x - p2._x, p1._y - p2._y);
}

pdd operator * (const double c, const pdd p){
  return pdd(p._x * c, p._y * c);
}

double operator % (const pdd p1, const pdd p2){
  return p1._x * p2._y - p2._x * p1._y;
}


vector<Line> lnlst;
double atn[MXL];

bool lncmp(int l1, int l2){
  return atn[l1] < atn[l2];
}

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2){
  double f1 = (p2 - p1) % (q1 - p1);
  double f2 = (p2 - p1) % (p1 - q2);
  double f = (f1 + f2);

  if(fabs(f) < EPS) return pdd(nan(""), nan(""));

  return (f2 / f) * q1 + (f1 / f) * q2;
}

deque<Line> dq;

void halfPlaneInter(){
  int n = lnlst.size();
  vector<int> stlst;
  for(int i=0; i<n; i++){
    stlst.PB(i);
    pdd d = lnlst[i].second - lnlst[i].first;
    atn[i] = atan2(d._y, d._x);
  }
  sort(stlst.begin(), stlst.end(), lncmp);
  vector<Line> lst;

  for(int i=0; i<n; i++){
    if(i) {
      int j = i-1;
      Line li = lnlst[stlst[i]];
      Line lj = lnlst[stlst[j]];
      pdd di = li.second - li.first;
      pdd dj = lj.second - lj.first;
      if(fabs(di%dj) < EPS){
        if(di % (lj.second - li.second) < 0) {
          lst.pop_back();
        }else continue;
      }
    }
    lst.PB(lnlst[stlst[i]]);
  }

  dq.PB(lst[0]);
  dq.PB(lst[1]);
  for(int i=2; i<n; i++){
    int dsz = dq.size();
    Line l = lst[i];
    while(dsz >= 2){
      Line l1 = dq[dsz-1];
      Line l2 = dq[dsz-2];

      pdd it12 = interPnt(l1.first, l1.second, l2.first
          , l2.second);

      if((l.second - l.first) % (it12 - l.first) < 0){
        dq.pop_back();
        dsz --;
      } else break;
    }

    while(dsz >= 2){
      Line l1 = dq[0];
      Line l2 = dq[1];

      pdd it12 = interPnt(l1.first, l1.second, l2.first
          , l2.second);

      if((l.second - l.first) % (it12 - l.first) < 0){
        dq.pop_front();
        dsz --;
      } else break;
    }

    Line l1 = dq[dsz - 1];
    if(!std::isnan(interPnt(l.first, l.second, l1.first
        ,  l1.second)._x)){
      dq.PB(l);
    }

  }

  int dsz = dq.size();
  while(dsz >= 2){
    Line l1 = dq[dsz - 1];
    Line l2 = dq[dsz - 2];
    Line l = dq[0];
    pdd it12 = interPnt(l1.first, l1.second, l2.first,
        l2.second);
    if(std::isnan(it12._x)) {
      dq.pop_back();
      dq.pop_back();
      dsz -= 2;
    } else if((l.second - l.first) % (it12 - l.first) <
         0){
      dq.pop_back();
      dsz --;
    } else break;
  }
}


int main(){

  int N;
  cin >> N;
  for(int i=0; i<N; i++){
    double x1, x2, y1, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    lnlst.PB({pdd(x1, y1), pdd(x2, y2)});
  }

  halfPlaneInter();

  int dsz = dq.size();
  cout << dsz << endl;
  for(int i=0; i<dsz; i++){
    int j = (i+1) % dsz;
    pdd it = interPnt(dq[i].first, dq[i].second, dq[j].
        first, dq[j].second);
    cout << it._x << ' ' << it._y << endl;
  }

}
```

5.2 KD Tree

```
const INF = 1100000000;

class NODE{ public:
  int x,y,x1,x2,y1,y2;
  int i,f;
  NODE *L,*R;
};
inline long long dis(NODE& a,NODE& b){
  long long dx=a.x-b.x;
  long long dy=a.y-b.y;
  return dx*dx+dy*dy;
}
NODE node[100000];
bool cmpx(const NODE& a,const NODE& b){ return a.x<b.x;
     }
bool cmpy(const NODE& a,const NODE& b){ return a.y<b.y;
     }
NODE* KDTree(int L,int R,int dep){
  if(L>R) return 0;
  int M=(L+R)/2;
  if(dep%2==0){
    nth_element(node+L,node+M,node+R+1,cmpx);
    node[M].f=0;
  }else{
    nth_element(node+L,node+M,node+R+1,cmpy);
    node[M].f=1;
  }
  node[M].x1=node[M].x2=node[M].x;
  node[M].y1=node[M].y2=node[M].y;
  node[M].L=KDTree(L,M-1,dep+1);
  if(node[M].L){
    node[M].x1=min(node[M].x1,node[M].L->x1);
    node[M].x2=max(node[M].x2,node[M].L->x2);
    node[M].y1=min(node[M].y1,node[M].L->y1);
    node[M].y2=max(node[M].y2,node[M].L->y2);
  }
  node[M].R=KDTree(M+1,R,dep+1);
  if(node[M].R){
    node[M].x1=min(node[M].x1,node[M].R->x1);
    node[M].x2=max(node[M].x2,node[M].R->x2);
    node[M].y1=min(node[M].y1,node[M].R->y1);
    node[M].y2=max(node[M].y2,node[M].R->y2);
  }
  return node+M;
}
inline int touch(NODE* r,int x,int y,long long d){
  long long d2;
  d2 = (long long)(sqrt(d)+1);
  if(x<r->x1-d2 || x>r->x2+d2 || y<r->y1-d2 || y>r->y2+
      d2)
    return 0;
  return 1;
}
void nearest(NODE* r,int z,long long &md){
  if(!r || !touch(r,node[z].x,node[z].y,md)) return;
  long long d;
  if(node[z].i!=r->i){
    d=dis(*r,node[z]);
    if(d<md) md=d;
  }
  if(r->f==0){
    if(node[z].x<r->x){
      nearest(r->L,z,md);
      nearest(r->R,z,md);
    }else{
      nearest(r->R,z,md);
      nearest(r->L,z,md);
    }
  }else{
    if(node[z].y<r->y){
      nearest(r->L,z,md);
      nearest(r->R,z,md);
    }else{
      nearest(r->R,z,md);
      nearest(r->L,z,md);
    }
  }
}
int main(){
  int TT,n,i;
  long long d;
  NODE* root;
  scanf("%d",&TT);
  while(TT--){
    scanf("%d",&n);
    for(i=0;i<n;i++){
      scanf("%d %d",&node[i].x,&node[i].y);
      node[i].i=i;
    }
    root=KDTree(0,n-1,0);
    for(i=0;i<n;i++){
      d=9000000000000000000LL;
      nearest(root,i,d);
      ans[node[i].i]=d;
    }
  }
}
```

5.3 Minimum Covering Circle

```
struct Mcc{
  // return pair of center and r^2
  static const int MAXN = 1000100;
  int n;
  pdd p[MAXN],cen;
  double r2;

  void init(int _n, pdd _p[]){
    n = _n;
    memcpy(p,_p,sizeof(pdd)*n);
  }
  double sqr(double a){ return a*a; }
  double abs2(pdd a){ return a*a; }
  pdd center(pdd p0, pdd p1, pdd p2) {
    pdd a = p1-p0;
    pdd b = p2-p0;
    double c1=abs2(a)*0.5;
    double c2=abs2(b)*0.5;
    double d = a % b;
    double x = p0.x + (c1 * b.y - c2 * a.y) / d;
    double y = p0.y + (a.x * c2 - b.x * c1) / d;
    return pdd(x,y);
  }

  pair<pdd,double> solve(){
    random_shuffle(p,p+n);
    r2=0;
    for (int i=0; i<n; i++){
      if (abs2(cen-p[i]) <= r2) continue;
      cen = p[i];
      r2 = 0;
      for (int j=0; j<i; j++){
        if (abs2(cen-p[j]) <= r2) continue;
        cen = 0.5 * (p[i]+p[j]);
        r2 = abs2(cen-p[j]);
        for (int k=0; k<j; k++){
          if (abs2(cen-p[k]) <= r2) continue;
          cen = center(p[i],p[j],p[k]);
          r2 = abs2(cen-p[k]);
        }
      }
    }
    return {cen,r2};
  }
}mcc;
```

5.4 Minkowski Sum

```
/* convex hull Minkowski Sum*/
#define INF 1000000000000000LL
class PT{ public:
  long long x,y;
  int POS(){
    if(y==0) return x>0?0:1;
    return y>0?0:1;
  }
};
PT pt[300000],qt[300000],rt[300000];
long long Lx,Rx;
int dn,un;
inline bool cmp(PT a,PT b){
  int pa=a.POS(),pb=b.POS();
  if(pa==pb) return (a^b)>0;
  return pa<pb;
}
int minkowskiSum(int n,int m){
  int i,j,r,p,q,fi,fj;
  for(i=1,p=0;i<n;i++){
    if(pt[i].y<pt[p].y || (pt[i].y==pt[p].y && pt[i].x<
        pt[p].x)) p=i; }
  for(i=1,q=0;i<m;i++){
    if(qt[i].y<qt[q].y || (qt[i].y==qt[q].y && qt[i].x<
        qt[q].x)) q=i; }
  rt[0]=pt[p]+qt[q];
  r=1; i=p; j=q; fi=fj=0;
  while(1){
    if((fj&&j==q) || ((!fi||i!=p) && cmp(pt[(p+1)%n]-pt
        [
          p],qt[(q+1)%m]-qt[q]))){
      rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
      p=(p+1)%n;
      fi=1;
    }else{
      rt[r]=rt[r-1]+qt[(q+1)%m]-qt[q];
      q=(q+1)%m;
      fj=1;
    }
    if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))!=0)
        r
      ++;
    else rt[r-1]=rt[r];
    if(i==p && j==q) break;
  }
  return r-1;
}
void initInConvex(int n){
  int i,p,q;
  long long Ly,Ry;
  Lx=INF; Rx=-INF;
  for(i=0;i<n;i++){
    if(pt[i].x<Lx) Lx=pt[i].x;
    if(pt[i].x>Rx) Rx=pt[i].x;
  }
  Ly=Ry=INF;
  for(i=0;i<n;i++){
    if(pt[i].x==Lx && pt[i].y<Ly){ Ly=pt[i].y; p=i; }
    if(pt[i].x==Rx && pt[i].y<Ry){ Ry=pt[i].y; q=i; }
  }
  for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
  qt[dn]=pt[q]; Ly=Ry=-INF;
  for(i=0;i<n;i++){
    if(pt[i].x==Lx && pt[i].y>Ly){ Ly=pt[i].y; p=i; }
    if(pt[i].x==Rx && pt[i].y>Ry){ Ry=pt[i].y; q=i; }
  }
  for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
  rt[un]=pt[q];
}
inline int inConvex(PT p){
  int L,R,M;
  if(p.x<Lx || p.x>Rx) return 0;
  L=0;R=dn;
  while(L<R-1){ M=(L+R)/2;
    if(p.x<qt[M].x) R=M; else L=M; }
    if(tri(qt[L],qt[R],p)<0) return 0;
    L=0;R=un;
    while(L<R-1){ M=(L+R)/2;
      if(p.x<rt[M].x) R=M; else L=M; }
      if(tri(rt[L],rt[R],p)>0) return 0;
      return 1;
}
int main(){
  int n,m,i;
  PT p;
  scanf("%d",&n);
  for(i=0;i<n;i++) scanf("%I64d %I64d",&pt[i].x,&pt[i].
      y);
  scanf("%d",&m);
  for(i=0;i<m;i++) scanf("%I64d %I64d",&qt[i].x,&qt[i].
      y);
  n=minkowskiSum(n,m);
  for(i=0;i<n;i++) pt[i]=rt[i];
  scanf("%d",&m);
  for(i=0;i<m;i++) scanf("%I64d %I64d",&qt[i].x,&qt[i].
      y);
  n=minkowskiSum(n,m);
  for(i=0;i<n;i++) pt[i]=rt[i];
  initInConvex(n);
  scanf("%d",&m);
  for(i=0;i<m;i++){
    scanf("%I64d %I64d",&p.x,&p.y);
    p.x*=3; p.y*=3;
    puts(inConvex(p)?"YES":"NO");
  }
}
```

5.5 Convex Hull

```
double cross(pdd o, pdd a, pdd b){
  return (a-o) % (b-o);
}
vector<pdd> convex_hull(vector<pdd> pt){
  sort(pt.begin(),pt.end());
  int top=0;
  vector<pdd> stk(2*pt.size());
  for (int i=0; i<(int)pt.size(); i++){
    while (top >= 2 && cross(stk[top-2],stk[top-1],pt[i
        ]) <= 0)
      top--;
    stk[top++] = pt[i];
  }
  for (int i=pt.size()-2, t=top+1; i>=0; i--){
    while (top >= t && cross(stk[top-2],stk[top-1],pt[i
        ]) <= 0)
      top--;
    stk[top++] = pt[i];
  }
  stk.resize(top-1);
  return stk;
}
```

5.6 Intersection Of Two Circles

```
vector<pdd> interCircle(pdd o1, double r1, pdd o2,
    double r2) {
  ld d2 = (o1 - o2) * (o1 - o2);
  ld d = sqrt(d2);
  if (d > r1+r2) return {};
  pdd u = 0.5*(o1+o2) + ((r2*r2-r1*r1)/(2*d2))*(o1-o2);
  double A = sqrt((r1+r2+d) * (r1-r2+d) * (r1+r2-d) *
      (-r1+r2+d));
  pdd v = A / (2*d2) * pdd(o1.S-o2.S, -o1.F+o2.F);
  return {u+v, u-v};
}
```

5.7 Intersection Of Two Lines

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

using namespace std;
```

```
const double EPS = 1e-9;

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2){
  double f1 = (p2 - p1) % (q1 - p1);
  double f2 = (p2 - p1) % (p1 - q2);
  double f = (f1 + f2);

  if(fabs(f) < EPS) return pdd(nan(""), nan(""));

  return (f2 / f) * q1 + (f1 / f) * q2;
}
```

5.8 Point Operators

```
#include<bits/stdc++.h>
using namespace std;

#define _x first
#define _y second
typedef pair<double, double> pdd;

pdd operator + (const pdd p1, const pdd p2){
  return pdd(p1._x + p2._x, p1._y + p2._y);
}
pdd operator - (const pdd p1, const pdd p2){
  return pdd(p1._x - p2._x, p1._y - p2._y);
}

pdd operator * (const double c, const pdd p){
  return pdd(p._x * c, p._y * c);
}
pdd operator - (const pdd p){
  return (-1.0) * p;
}
double operator * (const pdd p1, const pdd p2){
  return p1._x * p2._x + p1._y * p2._y;
}
double operator % (const pdd p1, const pdd p2){
  return p1._x * p2._y - p2._x * p1._y;
}
```

6 Stringology

6.1 AC Automata

```
struct ACautomata{
  struct Node{
    int cnt,dp;
    Node *go[26], *fail;
    Node (){
      cnt = 0;
      dp = -1;
      memset(go,0,sizeof(go));
      fail = 0;
    }
  };

  Node *root, pool[1048576];
  int nMem;

  Node* new_Node(){
    pool[nMem] = Node();
    return &pool[nMem++];
  }
  void init(){
    nMem = 0;
    root = new_Node();
  }
  void add(const string &str){
    insert(root,str,0);
  }
  void insert(Node *cur, const string &str, int pos){
    if (pos >= (int)str.size()){
      cur->cnt++;
      return;
    }
    int c = str[pos]-'a';
    if (cur->go[c] == 0){
      cur->go[c] = new_Node();
    }
    insert(cur->go[c],str,pos+1);
  }
  void make_fail(){
    queue<Node*> que;
    que.push(root);
    while (!que.empty()){
      Node* fr=que.front();
      que.pop();
      for (int i=0; i<26; i++){
        if (fr->go[i]){
          Node *ptr = fr->fail;
          while (ptr && !ptr->go[i]) ptr = ptr->fail;
          if (!ptr) fr->go[i]->fail = root;
          else fr->go[i]->fail = ptr->go[i];
          que.push(fr->go[i]);
        }
      }
    }
  }
};
```

6.2 SAM

```
// par : fail link
// val : a topological order ( useful for DP )
// go[x] : automata edge ( x is integer in [0,26) )

struct SAM{
  struct State{
    int par, go[26], val;
    State () : par(0), val(0){ FZ(go); }
    State (int _val) : par(0), val(_val){ FZ(go); }
  };
  vector<State> vec;
  int root, tail;

  void init(int arr[], int len){
    vec.resize(2);
    vec[0] = vec[1] = State(0);
    root = tail = 1;
    for (int i=0; i<len; i++)
      extend(arr[i]);
  }
  void extend(int w){
    int p = tail, np = vec.size();
    vec.PB(State(vec[p].val+1));
    for ( ; p && vec[p].go[w]==0; p=vec[p].par)
      vec[p].go[w] = np;
    if (p == 0){
      vec[np].par = root;
    } else {
      if (vec[vec[p].go[w]].val == vec[p].val+1){
        vec[np].par = vec[p].go[w];
      } else {
        int q = vec[p].go[w], r = vec.size();
        vec.PB(vec[q]);
        vec[r].val = vec[p].val+1;
        vec[q].par = vec[np].par = r;
        for ( ; p && vec[p].go[w] == q; p=vec[p].par)
          vec[p].go[w] = r;
      }
    }
    tail = np;
  }
};
```

6.3 Palindrome

```
const int MAX = 1000;
int len;
char ip[MAX];
char op[MAX*2];
int zv[MAX*2];
int main(){

  cin >> ip;
  len = strlen(ip);

  int l2 = len*2 - 1;
  for(int i=0; i<l2; i++){
    if(i&1) op[i] = '@';
    else op[i] = ip[i/2];
  }
  int l=0, r=0;
  zv[0] = 1;

  for(int i=1; i<l2; i++){
    if( i > r ){
      l = r = i;
      while( l>0 && r<l2-1 && op[l-1] == op[r+1] ){
        l --;
        r ++;
      }
      zv[i] = (r-l+1);
    }else{
      int md = (l+r)/2;
      int j = md + md - i;
      zv[i] = zv[j];
      int q = zv[i] / 2;
      int nr = i + q;
      if( nr == r ){
        l = i + i - r;

        while( l>0 && r<l2-1 && op[l-1] == op[r+1] ){
          l --;
          r ++;
        }
        zv[i] = r - l + 1;
      }else if( nr > r ){
        zv[i] = (r - i) * 2 + 1;
      }
    }
  }

  return 0;
}
```

6.4 Z Value

```
char s[MAXLEN];
int len,z[MAXLEN];
void Z_value() {
  int i,j,left,right;
  left=right=0; z[0]=len;
  for(i=1;i<len;i++) {
    j=max(min(z[i-left],right-i),0);
    for(;i+j<len&&s[i+j]==s[j];j++);
    z[i]=j;
    if(i+z[i]>right) {
      right=i+z[i];
      left=i;
    }
  }
}
```

6.5 Smallest Rotation

```
string mcp(string s){
  int n = s.length();
  s += s;
  int i=0, j=1, k=0;
  while (j<n && k<n){
    if (s[i+k] == s[j+k]) k++;
    else {
      if (s[i+k] < s[j+k]) {
        j += k + 1;
      } else {
        i = j;
        j = max(j+1, j+k);
      }
      k = 0;
    }
  }
  return s.substr(i, n);
}
```

6.6 Suffix Array

```
const int MAX = 1020304;
int ct[MAX], he[MAX], rk[MAX], sa[MAX], tsa[MAX], tp[
    MAX][2];

void suffix_array(char *ip){

  int len = strlen(ip);
  int alp = 256;

  memset(ct, 0, sizeof(ct));
  for(int i=0;i<len;i++) ct[ip[i]+1]++;
  for(int i=1;i<alp;i++) ct[i]+=ct[i-1];
  for(int i=0;i<len;i++) rk[i]=ct[ip[i]];

  for(int i=1;i<len;i*=2){
    for(int j=0;j<len;j++){
      if(j+i>=len) tp[j][1]=0;
      else tp[j][1]=rk[j+i]+1;

      tp[j][0]=rk[j];
    }
    memset(ct, 0, sizeof(ct));
    for(int j=0;j<len;j++) ct[tp[j][1]+1]++;
    for(int j=1;j<len+2;j++) ct[j]+=ct[j-1];
    for(int j=0;j<len;j++) tsa[ct[tp[j][1]]++]=j;

    memset(ct, 0, sizeof(ct));
    for(int j=0;j<len;j++) ct[tp[j][0]+1]++;
    for(int j=1;j<len+1;j++) ct[j]+=ct[j-1];
    for(int j=0;j<len;j++) sa[ct[tp[tsa[j]][0]]++]=tsa[
        j];

    rk[sa[0]]=0;
    for(int j=1;j<len;j++){
      if( tp[sa[j]][0] == tp[sa[j-1]][0] &&
        tp[sa[j]][1] == tp[sa[j-1]][1] )
        rk[sa[j]] = rk[sa[j-1]];
      else
        rk[sa[j]] = j;
    }
  }

  for(int i=0,h=0;i<len;i++){
    if(rk[i]==0) h=0;
    else{
      int j=sa[rk[i]-1];
      h=max(0,h-1);
      for(;ip[i+h]==ip[j+h];h++);
    }
    he[rk[i]]=h;
  }
}
```

6.7 Sais

```
struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
    static const int MXN = 300010;
    bool _t[MXN*2];
    int _s[MXN*2], _sa[MXN*2], _c[MXN*2], x[MXN], _p[
        MXN], _q[MXN*2], hei[MXN], r[MXN];
    int operator [] (int i){ return _sa[i]; }
    void build(int *s, int n, int m){
        memcpy(_s, s, sizeof(int) * n);
        sais(_s, _sa, _p, _q, _t, _c, n, m);
        mkhei(n);
    }
    void mkhei(int n){
        REP(i,n) r[_sa[i]] = i;
        hei[0] = 0;
        REP(i,n) if(r[i]) {
            int ans = i>0 ? max(hei[r[i-1]] - 1, 0) :
                0;
            while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans
                ++;
            hei[r[i]] = ans;
        }
    }
    void sais(int *s, int *sa, int *p, int *q, bool *t,
         int *c, int n, int z){
        bool uniq = t[n-1] = true, neq;
        int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s +
             n, lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
        memcpy(x, c, sizeof(int) * z); \
        XD; \
        memcpy(x + 1, c, sizeof(int) * (z - 1)); \
        REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i
            ]-1]]++] = sa[i]-1; \
        memcpy(x, c, sizeof(int) * z); \
        for(int i = n - 1; i >= 0; i--) if(sa[i] && t[
            sa[i]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
        MS0(c, z);
        REP(i,n) uniq &= ++c[s[i]] < 2;
        REP(i,z-1) c[i+1] += c[i];
        if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return;
             }
        for(int i = n - 2; i >= 0; i--) t[i] = (s[i]==s
            [i+1] ? t[i+1] : s[i]<s[i+1]);
        MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[
            s[i]]]=p[q[i]=nn++]=i);
        REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1])
             {
            neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i
                ]]+1]-sa[i])*sizeof(int));
            ns[q[lst=sa[i]]]=nmxz+=neq;
        }
        sais(ns, nsa, p + nn, q + n, t + n, c + z, nn,
            nmxz + 1);
        MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[s
            [p[nsa[i]]]]] = p[nsa[i]]);
    }
}sa;

void suffix_array(int* ip, int len) {
    // should padding a zero in the back
    // s is int array, n is array length
    // s[0..n-1] != 0, and s[n] = 0
    // resulting SA will be length n+1
    ip[len++] = 0;
    sa.build(ip, len, 128);
    // original 1-base
    for (int i=0; i<l; i++) {
        hei[i] = sa.hei[i + 1];
        sa[i] = sa._sa[i + 1];
    }
}
```

7 Problems

7.1 Orange Protection

```
/*
 * Given a Tree and the power of every node.
 * Each Node can protect the nodes whose distance <=
     cover[i] with it
 * output the number of each node that it can protect.
 */
const int MXN = 100005;

int cover[MXN], ans[MXN];
int N, ok[MXN];
int fr,bk,que[MXN],vst[MXN],dis[MXN],fa[MXN],sz[MXN];
vector<int> E[MXN];

int bit[MXN];
int lb(int a){ return a & -a; }
void reset_bit(int st){
  for (int i = st+1; i < MXN; i+=lb(i))
    bit[i] = 0;
}
void update(int st){
  for (int i = st+1; i < MXN; i+=lb(i))
    bit[i]++;
}
int query(int st, int ret = 0){
  for (int i = st+1; i > 0; i-=lb(i))
    ret += bit[i];
  return ret;
}

void BFS(int st){
  fr = bk = 0;
  que[bk++] = st;
  vst[st] = 1;
  dis[st] = 0;
  while (fr < bk){
    int u = que[fr++];
    for (auto v : E[u]){
      if (!ok[v] || vst[v]) continue;
      vst[v] = 1;
      dis[v] = dis[u] + 1;
      fa[v] = u;
      que[bk++] = v;
    }
  }
  for (int i=0; i<bk; i++)
    vst[que[i]] = 0;
}
int find_centroid(int st){
  int ret=-1, cnt=MXN+100;
  BFS(st);
  for (int i = bk-1; i>=0; i--){
    int u = que[i], mx = 0;
    sz[u] = 1;
    for (auto v : E[u]){
      if (!ok[v] || v == fa[u]) continue;
      sz[u] += sz[v];
      mx = max(mx, sz[v]);
    }
    mx = max(mx, bk-sz[u]);
    if (mx < cnt){
      ret = u;
      cnt = mx;
    }
  }
  return ret;
}

void solve(int u){
  int root = find_centroid(u);
  ok[root] = 0;
  for (auto v : E[root])
    if (ok[v]) solve(v);

  for (auto v : E[root]){
    if (!ok[v]) continue;
    BFS(v);
    for (int i=0; i<bk; i++){
      dis[que[i]]++;
      update(dis[que[i]]);
    }
    for (int i=0; i<bk; i++){
      int it = que[i];
      ans[it] -= query(cover[it] - dis[it]);
    }
    for (int i=0; i<bk; i++)
      reset_bit(dis[que[i]]);
  }
  BFS(root);
  for (int i=0; i<bk; i++) update(dis[que[i]]);
  for (int i=0; i<bk; i++){
    int v = que[i];
    ans[v] += query(cover[v] - dis[v]);
  }
  for (int i=0; i<bk ;i++) reset_bit(dis[que[i]]);

  ok[root] = 1;
}
```

```
int main(int argc, char** argv){
  scanf("%d", &N);
  for (int i=0; i<N; i++){
    scanf("%d", &cover[i]);
    cover[i] = min(cover[i], N);
  }
  for (int i=0,u,v; i<N-1; i++){
    scanf("%d%d", &u, &v);
    u--; v--;
    E[u].PB(v);
    E[v].PB(u);
  }
  fill(ok,ok+N,1);
  FZ(vst); FZ(ans); FZ(bit);
  solve(0);
  for (int i=0; i<N; i++)
    printf("%d\n", ans[i]);
  return 0;
}
```

7.2 Max Tangent

```
typedef long long LL;
const int MAXN = 100010;
struct Coord{
  LL x, y;
  Coord operator - (Coord ag) const{
    Coord res;
    res.x = x - ag.x;
    res.y = y - ag.y;
    return res;
  }
}sum[MAXN], pnt[MAXN], ans, calc;

inline bool cross(Coord a, Coord b, Coord c){
  return (c.y - a.y) * (c.x - b.x) > (c.x - a.x) * (c.y
       - b.y);
}

int main(){
  int n, l, np, st, ed, now;
  scanf("%d %d\n", &n, &l);
  sum[0].x = sum[0].y = np = st = ed = 0;
  for (int i = 1, v; i <= n; i++){
    scanf("%d", &v);
    sum[i].y = sum[i - 1].y + v;
    sum[i].x = i;
  }
  ans.x = now = 1;
  ans.y = -1;
  for (int i = 0; i <= n - l; i++){
    while (np > 1 && cross(pnt[np - 2], pnt[np - 1],
        sum[i]))
      np--;
    if (np < now && np != 0) now = np;
    pnt[np++] = sum[i];
    while (now < np && !cross(pnt[now - 1], pnt[now],
        sum[i + l]))
      now++;
    calc = sum[i + l] - pnt[now - 1];
    if (ans.y * calc.x < ans.x * calc.y){
      ans = calc;
      st = pnt[now - 1].x;
      ed = i + l;
    }
  }
  double res = (sum[ed].y-sum[st].y)/(sum[ed].x-sum[st
      ].x);
  printf("%f\n", res);
  return 0;
}
```

7.3 Cot4

```
#include <bits/stdc++.h>
using namespace std;
#define FZ(n) memset((n),0,sizeof(n))
#define FMO(n) memset((n),-1,sizeof(n))
#define MC(n,m) memcpy((n),(m),sizeof(n))
#define F first
#define S second
#define MP make_pair
#define PB push_back
#define FOR(x,y) for(__typeof(y.begin())x=y.begin();x!=
    y.end();x++)
#define IOS ios_base::sync_with_stdio(0)

#ifdef ONLINE_JUDGE
#define FILEIO(name) \
  freopen(name".in", "r", stdin); \
    freopen(name".out", "w", stdout);
#else
#define FILEIO(name)
#endif
// Let's Fight!

typedef long long LL;

const int MXN = 130005;
const int MX = 300005;
const int HASH = 12721;
const int MOD = 999997771;
const int LOG = 17;
struct query_t{
  int ti,sj,id;
};
struct tTree_t{
  int tp, prvID, prvID2;
  char ch;
  int l,r,len;
}tTree[MXN];
vector<int> E[MXN];
vector<query_t> q[MXN];

int sa[MXN],rk[MXN],rk2[MXN],fa[LOG][MXN],sCh[MXN],
    cmpLv;
pair<int,int> azRange[30];
int N,nq;
int tID,sID;
int ans[MXN];

inline int getRank(int id){
  if (id == 0) return 0;
  if (id >= sID) while(1);
  return rk[id];
}
bool comp0(int a, int b){
  return sCh[a] < sCh[b];
}
bool comp1(int a, int b){
  if (getRank(a) == getRank(b)){
    return getRank(fa[cmpLv][a]) < getRank(fa[cmpLv][b
        ]);
  }
  return getRank(a) < getRank(b);
}
void suffix_array(){
  sCh[1] = 0;
  for (int i=1; i<sID; i++){
    sa[i] = i;
    rk[i] = sCh[i];
  }
  sort(sa+1,sa+sID,comp0);
  cmpLv = 0;
  for (int len = 1; len < MXN; len <<= 1){
    sort(sa+1,sa+sID,comp1);
    int r = 1;
    rk2[sa[1]] = r;
    for (int i=2; i<sID; i++){
      if (comp1(sa[i-1],sa[i])){
        rk2[sa[i]] = ++r;
      } else {
        rk2[sa[i]] = r;
      }
    }
    for (int i=1; i<sID; i++){
      rk[i] = rk2[i];
    }
    cmpLv++;
  }
}

void init(){
  nq = 0;
  tID = 2;
  sID = 2;
  FZ(fa);
}
int trash;
void input(){
  init();
  trash = scanf("%d", &N);
  for (int i=0; i<N; i++){
    int cmd;
    trash = scanf("%d", &cmd);
    if (cmd == 1){
      int prvID;
      char str[10];
      trash = scanf("%d%s", &prvID, str);
      E[prvID].PB(sID);
      fa[0][sID] = prvID;
      sCh[sID] = str[0];
      sID++;
    } else if (cmd == 2){
      int dir;
      int prvID;
      char str[5];
      trash = scanf("%d%d%s", &dir,&prvID,str);
      if (dir == 0){
        tTree[tID].tp = 0;
        tTree[tID].prvID = prvID;
        tTree[tID++].ch = str[0];
      } else {
        tTree[tID].tp = 1;
        tTree[tID].prvID = prvID;
        tTree[tID++].ch = str[0];
      }
    } else if (cmd == 3){
      int a,b;
      trash = scanf("%d%d", &a, &b);
      tTree[tID].tp = 2;
      tTree[tID].prvID = a;
      tTree[tID++].prvID2 = b;
    } else if (cmd == 4){
      int ti,sj;
      trash = scanf("%d%d", &ti, &sj);
      q[sj].PB((query_t){ti,sj,nq});
      nq++;
    }
  }
}
int moveUP(int u, int len){
  for (int i=0; i<LOG && len; i++){
    if (len & (1<<i)){
      len -= 1<<i;
      u = fa[i][u];
    }
  }
  return u;
}
int getL(int l, int r, int up, int cp){
  if (l == r) return l;
  while (l < r){
    int mid = (l+r) / 2;
    int id = moveUP(sa[mid],up);
    if (cp <= rk[id])
      r = mid;
    else
      l = mid+1;
  }
  return l;
}
int getR(int l, int r, int up, int cp){
  if (l == r) return l;
  while (l < r){
    int mid = (l+r) / 2;
    int id = moveUP(sa[mid],up);
    if (cp < rk[id])
      r = mid;
    else
      l = mid+1;
  }
  return l;
}
void prvSolve(){
  for (int i=1; i<LOG; i++){
    for (int j=1; j<sID; j++){
      fa[i][j] = fa[i-1][fa[i-1][j]];
    }
  }
  suffix_array();
  for (int i=0; i<30; i++)
    azRange[i] = MP(sID,0);
  for (int i=1; i<sID; i++){
    if (sa[i] == 1) continue;
    int ch = sCh[sa[i]] - 'a';
    azRange[ch].F = min(azRange[ch].F,i);
    azRange[ch].S = max(azRange[ch].S,i);
  }
  for (int i=0; i<30; i++){
    azRange[i].S++;
    if (azRange[i].S == 1){
      azRange[i].F = azRange[i].S = sID;
    }
  }

  tTree[1].len = 0;
  tTree[1].l = 1;
  tTree[1].r = sID;
  for (int i=2; i<tID; i++){
    int id1 = tTree[i].prvID;
    int id2 = tTree[i].prvID2;
    if (tTree[i].tp == 0){
      tTree[i].len = tTree[id1].len + 1;
      int up = tTree[id1].len;
      char ch = tTree[i].ch;
      int l,r;
      l = tTree[id1].l;
      r = tTree[id1].r;
      if (l == r){
        tTree[i].l = tTree[i].r = l;
      } else {
        while (l < r){
          int mid = (l+r)/2;
          int id = moveUP(sa[mid],up);
          if (ch <= sCh[id])
            r = mid;
          else
            l = mid+1;
        }
        tTree[i].l = l;

        l = tTree[id1].l;
        r = tTree[id1].r;
        while (l < r){
          int mid = (l+r)/2;
          int id = moveUP(sa[mid],up);
          if (ch < sCh[id])
            r = mid;
          else
            l = mid+1;
        }

        tTree[i].r = r;
      }
    } else if (tTree[i].tp == 1){
      tTree[i].len = tTree[id1].len + 1;
      int ch = tTree[i].ch;
      int l = azRange[ch-'a'].F;
      int r = azRange[ch-'a'].S;
      if (id1 == 1){
        tTree[i].l = l;
        tTree[i].r = r;
      } else {
        tTree[i].l = getL(l,r,1,tTree[id1].l);
        tTree[i].r = getR(l,r,1,tTree[id1].r-1);
      }
    } else if (tTree[i].tp == 2){
      tTree[i].len = tTree[id1].len + tTree[id2].len;
      tTree[i].len = min(tTree[i].len,100005);
      if (tTree[i].len < 100000){
        int up = tTree[id2].len;
        int l = tTree[id2].l;
        int r = tTree[id2].r;
        tTree[i].l = getL(l,r,up,tTree[id1].l);
        tTree[i].r = getR(l,r,up,tTree[id1].r-1);
      } else {
        tTree[i].l = tTree[i].r = sID-1;
      }
    }
  }
}
int bit[MXN];
int lb(int a){ return a & -a; }
void update(int st, int v){
  for (int i=st; i<sID; i+=lb(i))
    bit[i] += v;
}
int query(int st){
  int res = 0;
  for (int i=st; i>0; i-=lb(i))
    res += bit[i];
  return res;
}
void DFS(int u){
  update(rk[u],1);
  FOR(it,q[u]){
    int id = it->ti;
    if (tTree[id].len == 0 || tTree[id].len > 100000)
      ans[it->id] = 0;
    else
      ans[it->id] = query(tTree[id].r-1) - query(tTree[
          id].l-1);
  }
  FOR(it,E[u])
    DFS(*it);
  update(rk[u],-1);
}
void solve(){
  DFS(1);
  for (int i=0; i<nq; i++)
    printf("%d\n", ans[i]);
}
int main(){
  input();
  prvSolve();
  solve();

  return 0;
}
```