0 Contents

1 Flow

1.1 Dinic

```
#define INF 0x3f3f3f3f
#define LINF 0x3f3f3f3f3f3f3f3fLL
struct Dinic {
  typedef long long int T;
  struct edge{
    int u, v;
    T c, f;
    edge(int _u, int _v, T _c, T _f): u(_u),v(_v),c(_c)
        ,f(_f){}
  };
  int n, s, t;
  vector<vector<int> > G;
  vector<edge> E;
  vector<int> cur, vis, d;
  Dinic(int _n):n(_n){
    G.resize(n+1);
    vis.resize(n+1); cur.resize(n+1); d.resize(n+1);
    for(int i=0; i<=n; i++)d[i] = INF;
  }
  void pb(int u, int v, T cap){
    G[u].push_back(E.size());
    E.push_back(edge(u, v, cap, 0));
    G[v].push_back(E.size());
    E.push_back(edge(v, u, 0, 0));
  }
  int bfs() {
    queue<int> q;
    for(int i=0; i<=n; i++)vis[i] = 0;
    q.push(s); d[s] = 0;
    while(!q.empty()) {
      int u = q.front(); q.pop();
      vis[u] = 1;
      for(int i=0; i<(int)G[u].size(); i++) {
        edge e = E[G[u][i]];
        if(e.c - e.f > 0 && !vis[e.v]) {
          d[e.v] = d[u] + 1;
          q.push(e.v);
        }
      }
    }
    return vis[t];
  }
  T dfs(int u, T a) {
    if(u == t || !a)return a;
    T totf = 0, f;
    for(int &i=cur[u]; i<(int)G[u].size(); i++) {
      edge &e = E[G[u][i]], &r=E[G[u][i]^1];
      if(d[e.v] != d[u]+1)continue;
      f = dfs(e.v, min(a, e.c - e.f));
      if(f<=0)continue;
      e.f += f; r.f -= f;
      totf += f;
      a -= f; if(!a)break;
    }
    return totf;
  }

  T operator()(int _s, int _t) {
    s = _s, t = _t;
    T maxf = 0;
    while(bfs()) {
      for(int i=0; i<=n; i++)cur[i] = 0;
      maxf += dfs(s, LINF);
    }
    return maxf;
  }
};
```

1.2 Min Cost Flow

```
#define ll long long int
#define LINF 2147483647000000LL
#define INF 2147483647
using namespace std;
struct MCF {
  struct edge
```

```
  {
    int u, v, c, f;
    ll co;
    edge(int _u, int _v, int _c, ll _co){ u = _u, v =
        _v, c = _c; co = _co; f = 0; }
  };
  vector<vector<int> > G;
  vector<edge> E;
  vector<ll> d;
  vector<int> inq, arg, p;
  int N, s, t;
  MCF(int _n) {
    N = _n;
    G.resize(_n+1);
    d.resize(_n+1); inq.resize(_n+1);
    arg.resize(_n+1); p.resize(_n+1);
    E.clear();
  }
  void pb(int u, int v, int c, ll co) {
    G[u].push_back(E.size());
    E.push_back(edge(u, v, c, co));
    G[v].push_back(E.size());
    E.push_back(edge(v, u, 0, -co));
  }
  bool BF(int &flow, ll &cost) {
    for(int i=0;i<=N;i++)p[i] = 0, inq[i] = 0, d[i] =
        LINF;
    queue<int> Q;
    Q.push(s);
    d[s]=0; inq[s] = 1; arg[s] = INF;
    while(!Q.empty()) {
      int x=Q.front(); Q.pop(); inq[x] = 0;
      for(int i=0; i<(int)G[x].size(); i++) {
        edge &e=E[G[x][i]];
        if(d[x] + e.co < d[e.v] && e.c > e.f) {
          d[e.v] = d[x]+e.co;
          p[e.v] = G[x][i];
          arg[e.v] = min(arg[x], e.c - e.f);
          if(!inq[e.v])Q.push(e.v), inq[e.v] = 1;
        }
      }
    }
    if(d[t] == LINF)return 0;
    int a = arg[t];
    for(int now = t; now != s; now = E[p[now]].u) {
      E[p[now]].f += a;
      E[p[now]^1].f -= a;
    }
    cost += arg[t] * d[t];
    flow += a;
    return 1;
  }

  pair<int, ll> operator ()(int _s, int _t) {
    s = _s, t = _t;
    int flow=0;
    ll cost=0;
    while(BF(flow, cost)){}
    return pair<int, ll>(flow, cost);
  }
};
```

1.3 Common Modeling Technique

Minimum Path Covering on DAG

1. Path covering without path intersection: For each vertex v, we may construct two vertices v_i and v_o, then for each edge $u \to v$, connect $u_o \to v_i$.
   This forms a bipartite graph. Each selected edge means a "join" of paths. Therefore the cardinality of the minimum path covering on the original graph will be $|V| - m$, where m is the cardinality of the maximum bipartite matching.
2. Covering that allows intersection: Perform Floyd-Warshall to obtain trasitive closure first, then make edge for each pair that are connected, the problem subsequently reduces to the non-intersecting case.

Euler Circuit on Undirected Graph

1. Give undirected edges directions arbitrary. Add corresponding arc with same direction and capacity 1 in the network.
   Filling an edge means "adjust" the direction
2. For each vertex u, calculate number of edges need to be changed by direction $d(u) = (deg_{in}(u) - deg_{out}(u))/2$.
3. Add arc from s to each u with $d(u) < 0$, to t from each u with $d(u) > 0$. The capacity of arc is $|d(u)|$.
4. Check if there exist full flow.

Network with Capacity Lower Bounds Todo.

2 Math

2.1 ExtGCD

```
typedef long long int ll;
#define mod 1000000007
void gcd(ll a, ll b, ll &x, ll &y, ll &d) {
  if(!b){ x = 1; y = 0; d = a; return ; }
  gcd(b, a%b, y, x, d); y -= (a/b)*x;
}
ll inv(ll a) {
  ll x, y, d;
  gcd(a, mod, x, y, d);
  return d==1 ? (x+mod)%mod : 0;
}
```

2.2 FFT

1. When convert back to integer, use `LL` can be safer.
2. `eps` are `0.5` generally, but sometime need adjustments.
3. the array `A` and `B` will be changed after DFT, and the result `AB` has been devided by `_n`.

```
#include <stdlib.h>
#include <math.h>
#include <complex>
#include <string.h>
#define MAXN 1048576
#define eps 0.5
#define PI
     3.14159265358979323846264338327950288419716939937510
#define max(a,b) (((a) > (b)) ? (a) : (b))

typedef std::complex<double> comp;

struct FFT{
  int _n;
  comp ww[MAXN], rw[MAXN];
  void init(int n, int m){ // n terms in polynomial
    _n=1; while(_n<n+m)_n<<=1;
    ww[0] = rw[0] = comp(1.0, 0.0);
    for(int k=1; k<_n; k++){
      ww[k]=comp(cos(2*k*PI/_n), sin(2*k*PI/_n));
      rw[_n-k]=ww[k];
    }
  }
  int rev(int n,int x){int res=0;while(n){res<<=1;res|=
      x&1;x>>=1;n>>=1;}return res;}
  void dft(int n, comp *res, comp *w){
    for(int i=0; i<n; i++){int j=rev(n>>1,i);if(i<j){
        comp tmp=res[j];res[j]=res[i];res[i]=tmp;}}
    for(int m=1; m<=n; m<<=1){
      if(m==1)continue;
      int mp = m>>1;
      for(int o = 0; o<n; o+=m){
        for(int i=0; i<mp; i++){
          comp tmp = w[i*(n/m)]*res[o+i+mp];
          res[o+i + mp] = res[o+i] - tmp;
          res[o+i] = res[o+i] + tmp;
        }
```

```
      }
    }
  }

  void mult(comp *A, comp *B, comp *AB){
    dft(_n, A, ww); dft(_n, B, ww);
    for(int i=0; i<_n; i++)AB[i] = A[i]*B[i];
    dft(_n, AB, rw);
    for(int i=0; i<_n; i++)AB[i]/=_n;
  }
} fft;

comp A[MAXN], B[MAXN];
comp AB[MAXN];
```

2.3 Common Theorems

Josephus Problem Let $f(i)$ be the survivor in the round with i people, in the numbering from $0 \sim i-1$. Then we have $f(1) = 0$ and $f(i+1) = (f(i)+k) mod (i+1)$. The $+k$ term is to restore the numbering, but stepping through k people. Note that $f(i)$ and $f(i+1)$ used distinct numbering, where kth in $f(i+1)$'s is the 0th in $f(i)$'s.

Pick's Theorem For a polygon consist integral-coordinate vertices. Let the number of integral points on the border of the polygon be a, and the number of integral points inside the polygon be b, then we have the area of the polygon:

$$A = a + \frac{b}{2} - 1$$

Burnside's Lemma

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X_g|$$

Mobius Inversion For $n \in \mathbb{N}$, if

$$g(n) = \sum_{d|n} f(d)$$

then

$$f(n) = \sum_{d|n} \mu(d) g(n/d)$$

3 Graph

3.1 Cut Vertex and BCC

Determining Bridge $low[v] > pre[u] \Rightarrow v$ is a cut vertex and (u, v) is a bridge

```
#define MAXN 1005
using namespace std;
struct edge {
  int u,v;
  edge(int _u,int _v){u=_u;v=_v;}
};
vector<edge> E;
vector<int> G[MAXN];
int N,M;
void pb(int u,int v) {
  G[u].push_back(E.size());
  E.push_back(edge(u,v));
  G[v].push_back(E.size());
  E.push_back(edge(v,u));
}

stack<edge> S;
int pre[MAXN],low[MAXN],bccno[MAXN];
int iscut[MAXN];
int stamp,bcc_cnt;
vector<int> bcc[MAXN];

int dfs(int u,int fa) {
  low[u]=pre[u]=++stamp;
  int ch=0;
  iscut[u]=0;
  for(int i=0;i<(int)G[u].size();i++) {
    edge e=E[G[u][i]];
    int v=e.v;
    if(!pre[v]) {
      ch++;
      S.push(e);
      low[u]=min(low[u],dfs(v,u));
      if(low[v]>=pre[u]) {
        iscut[u]=true;
        bcc_cnt++;
        bcc[bcc_cnt].clear();
        while(1) {
          edge x=S.top();S.pop();
          if(bccno[x.u]!=bcc_cnt)bcc[bcc_cnt].push_back
              (x.u),bccno[x.u]=bcc_cnt;
          if(bccno[x.v]!=bcc_cnt)bcc[bcc_cnt].push_back
              (x.v),bccno[x.v]=bcc_cnt;
          if(x.u==u&&x.v==v)break;
        }
      }
    } else if(pre[v]<pre[u]&&v!=fa) {
      S.push(e);
      low[u]=min(low[u],pre[v]);
    }
  }
  if(fa<0&&ch==1)iscut[u]=false;
  return low[u];
}
```

3.2 Kosaraju

```
#define MAXN 100005
int N;
bool vis[MAXN];
vector<int> G[MAXN];
vector<int> R[MAXN];
vector<int> SCC[MAXN];
int sccno[MAXN];
int scc_cnt;
int owner[MAXN];
int dfs_stamp;

queue<int> Q;
void dfs_for_stamp(int now) {
  vis[now]=true;
  for(int i=0;i<(int)R[now].size();i++)
  {
    int v=R[now][i];
    if(!vis[v]) {
      dfs_for_stamp(v);
    }
  }
  owner[++dfs_stamp]=now;
}

void dfs_for_scc(int now) {
  vis[now]=true;
  sccno[now]=scc_cnt;
  SCC[scc_cnt].push_back(now);
  for(int i=0;i<(int)G[now].size();i++)
  {
    int v=G[now][i];
    if(!vis[v])dfs_for_scc(v);
  }
}

int main() {
  dfs_stamp=0;
  for(int i=1;i<=N;i++) {
    owner[i]=0;
  }
  for(int i=1;i<=N;i++)if(!vis[i])dfs_for_stamp(i);
  for(int i=1;i<=N;i++)vis[i]=false;
  scc_cnt=0;
  for(int i=dfs_stamp;i>=1;i--) {
    if(!vis[owner[i]]){//cout<<i<<" "<<owner[i]<<endl;
      dfs_for_scc(owner[i]),scc_cnt++;
```

```
    }
  }
  return 0;
}
```

3.3 2-SAT Model

Problem Satisfy the boolean expression like $(x_0 \vee x_1) \wedge (x_1 \vee \neg x_3) \wedge ...(x_5 \vee x_2)$

Model For the expression $(x_0 \vee x_1)$, make edge $\neg x_0 \to x_1$, then none of the statements x_i and $\neg x_i$ can be in the same SCC.

3.4 KM

```
#define MAXN 1005
#define LL __int128_t
int t,N,K;
LL w[MAXN][MAXN];
LL x[MAXN],y[MAXN];
LL Lx[MAXN],Ly[MAXN];
bool S[MAXN],T[MAXN];
int Left[MAXN];
LL U,L;
const LL INF=(((LL)0x7fffffffffffLL)<<50)|((LL)0
    xffffffffffffffLL);

void getLL(LL &x){
  x=0;
  char c=getchar();
  while(c>'9'||c<'0')c=getchar();
  while(c<='9'&&c>='0'){x*=(LL)10;x+=(LL)(c-'0');c=
      getchar();}
}
void printLL(LL x){
  if(!x){printf("0"); return ;}
  vector<int> res;
  while(x) {
    res.push_back((int)(x%10));
    x/=(LL)10;
  }
  for(int i=res.size()-1;i>=0;i--)printf("%d",res[i]);
}

void initKM(){
  for(int i=1;i<=N;i++) {
    S[i]=T[i]=false;
    Left[i]=0;
    Lx[i]=Ly[i]=(LL)0LL;
    for(int j=1;j<=N;j++){
      if(w[i][j]==-INF)continue;
      if(x[i]+y[j]>U) {
        w[i][j]=L-U;
      } else if(x[i]+y[j]>L) {
        w[i][j]=L-x[i]-y[j];
      } else w[i][j]=0;
    }
  }
}

bool dfs(int i){
  S[i]=true;
  for(int j=1;j<=N;j++) {
    if(T[j])continue;
    if(Lx[i]+Ly[j]==w[i][j]) {
      T[j]=true;
      if(!Left[j]||dfs(Left[j])) {
        Left[j]=i;
        return true;
      }
    }
  }
  return false;
}

void KM(LL &ANS){
  for(int i=1;i<=N;i++){
    while(true) {
      for(int j=1;j<=N;j++)S[j]=T[j]=0;
      if(dfs(i))break;

      LL d=INF;
      for(int x=1;x<=N;x++){
        if(S[x])
        for(int y=1;y<=N;y++){
          if(!T[y]&&w[x][y]!=-INF)d=min(d,Lx[x]+Ly[y]-w
              [x][y]);
        }
      }
      if(d==INF){ANS=INF; return ;}

      for(int i=1;i<=N;i++){
        if(S[i])Lx[i]-=d;
        if(T[i])Ly[i]+=d;
      }
    }
  }

  for(int i=1;i<=N;i++) {
    ANS-=w[Left[i]][i];
  }
}

int main(){
  //cout<<INF*2<<endl;
  scanf("%d",&t);
  while(t--){
    scanf("%d",&N);
    getLL(L);getLL(U);
    scanf("%d",&K);
    for(int i=0;i<=N;i++)for(int j=0;j<=N;j++)w[i][j
        ]=0;
    for(int i=0;i<K;i++){
      int u,v;
      scanf("%d%d",&u,&v);
      w[u][v]=-INF;
    }
    for(int i=1;i<=N;i++)getLL(x[i]);
    for(int i=1;i<=N;i++)getLL(y[i]);
    initKM();
    LL ANS=0;
    KM(ANS);
    if(ANS==INF)puts("no");
    else printLL(ANS),puts("");
  }
  return 0;
}
```

4 String

4.1 Aho-Corasick Automata

```
#define MAXN 1000005
template<typename T>
struct AutoAC{
  struct Node {
    int v;
    map<T, Node*> ch;
    typename map<T, Node*>::iterator find(T k){ return
        ch.find(k); }
    typename map<T, Node*>::iterator begin(){ return ch
        .begin(); }
    typename map<T, Node*>::iterator end(){ return ch.
        end(); }
    Node *at(T k){ return ch.at(k); }
    Node *& operator [](T k){ return this->ch[k]; }
    void insert(T k, Node* v){ ch.insert(pair<T, Node
        *>(k, v)); }

    Node *fail;
  } nodes[MAXN];
  int n;
  Node *root;
  Node *newNode(){ nodes[n].v=0; nodes[n].fail=nullptr
      ; nodes[n].ch.clear(); return nodes+(n++); }
  AutoAC() { n=0; root=newNode(); root->v=0; root->fail
      =nullptr; }
  void init() { n=0; root=newNode(); root->v=0; root->
      fail=nullptr; }
```

```
  void insert( const T *s , int k ) {
    Node *now = root;
    for(int i=0; s[i]; i++){
      typename map<T, Node*>::iterator it = now->find(s
          [i]);
      if(it == now->end()){
        now->insert(s[i], newNode());
      }
      now = now->at(s[i]);
    }
    now->v = k;
  }
  void buildFail() {
    queue<Node*> q;
    q.push(root);
    while(!q.empty()) {
      Node *x = q.front(); q.pop();
      for(typename map<T, Node*>::iterator it = x->
          begin(); it!=x->end(); it++){
        T next = it->first;
        Node *cur = x->fail;
        while(cur&&cur->find(next) == cur->end())cur =
            cur->fail;
        it->second->fail = cur ? cur->at(next) : root;
        q.push(it->second);
      }
    }
  }
  int search( const T *s ) {
    int res=0;
    Node *cur = root;
    for(int i=0; s[i]; i++){
      while(cur && cur->find(s[i]) == cur->end())cur =
          cur->fail;
      cur = cur ? cur->at(s[i]) : root;
      if(cur->v)cnt[cur->v]++;
      res = max(cnt[cur->v], res);
    }
    return res;
  }
};
```

4.2 KMP

```
char s[10005], t[10005];
int f[10005];
// t is 1-based
void buildFail() {
  f[1]=0; f[0]=-1;
  for(int i=2; t[i]; i++){
    int now = f[i-1];
    while(now!=-1 && t[now+1] != t[i])now = f[now];
    f[i] = now+1;
  }
}

int search(char *s, int m) {
  int now = 0, res = 0;
  for(int i=0; s[i]; i++){
    while(now!=-1 && s[i] != t[now+1]) now = f[now];
    now++;
    if(now == m)res++;
  }
  return res;
}
```

4.3 Suffix Array

```
#define SIGSZ 130
#define MAXN 1000005
struct SA {
  int c[MAXN];
  int r1[MAXN], r2[MAXN], sa[MAXN], h[MAXN];
  int *rx = r1, *y = r2;
  int neq(int *r, int a, int b, int step, int n){
    return r[a] != r[b] || a+step>=n || b+step>=n || r[
        a+step] != r[b+step];
  }
  void build(int *s, int n, int *_rank, int *_hei, int
       *_h) {
    for(int i=0; i<SIGSZ; i++) c[i] = 0;
    for(int i=0; i<n; i++)c[rx[i] = s[i]]++;
    for(int i=1; i<SIGSZ; i++)c[i] = c[i-1] + c[i];
    for(int i=n-1; i>=0; i--)sa[--c[s[i]]] = i;
    int m = SIGSZ, p = 0;
    for(int step = 1; step<n; step<<=1, p=0) {
      // storing index of rx[i] based on sorted y[i] to
           y[i],
      // using the previously calculated sa[i] array.
      for(int i=n-step; i<n; i++)y[p++] = i;
      for(int i=0; i<n; i++)if(sa[i] >= step)y[p++] =
          sa[i] - step;
      // sorting rx[i] in the order of sorted y[i](aka
          . rx[y[i]])
      for(int i=0; i<m; i++)c[i] = 0;
      for(int i=0; i<n; i++)c[rx[y[i]]]++;
      for(int i=1; i<m; i++)c[i] = c[i-1] + c[i];
      for(int i=n-1; i>=0; i--)sa[ --c[rx[y[i]]] ] = y[
          i];
      m = 1; swap(rx, y); rx[sa[0]] = 0;
      for(int i=1; i<n; i++)rx[sa[i]] = neq(y, sa[i],
          sa[i-1], step, n) ? m++ : m-1;
      if(m == n) break;
    }
    int ph = 0;
    for(int i=0; i<n; i++) h[i] = 0;
    for(int i=0; i<n; i++) {
      if(rx[i] == 0) { h[i] = 0; continue; }
      if(i == 0 || h[i-1] <= 1) {
        for(ph = 0; i+ph<n && s[i+ph] == s[sa[rx[i
            ]-1] + ph]; ph++);
      } else {
        for(ph = h[i-1]-1; i+ph<n && s[i+ph] == s[sa[rx
            [i]-1] + ph]; ph++);
      }
      h[i] = ph;
    }
    if(_rank){ for(int i=0; i<n; i++)_rank[i] = rx[i
        ]; }
    if(_hei){ for(int i=0; i<n; i++)_hei[i] = h[sa[i
        ]]; }
    if(_h){ for(int i=0; i<n; i++)_h[i] = h[i]; }
  }
  inline int operator [](int i){ return sa[i]; }
};
```

5 Geometry

5.1 Convex Hull

```
// Remember to check if the first point need to be
    repeated.
#define MAXN 100005
#define ll long long int
struct poi {
  ll x, y;
  bool operator <(const poi &rhs)const {
    return x == rhs.x ? (y < rhs.y) : (x < rhs.x);
  }
};
int test(poi &pi, poi &pj, poi &pk) {
  ll dx1 = pj.x - pi.x, dy1 = pj.y - pi.y;
  ll dx2 = pk.x - pi.x, dy2 = pk.y - pi.y;
  return dx1*dy2 - dx2*dy1 >= 0;
}
void ConvexHull(poi *po, int n, vector<poi> &hull) {
  vector<poi> p;
  for(int i=0; i<n; i++)p.push_back(po[i]);
  sort(p.begin(), p.end());
  hull.push_back(p[0]);
  for(int i=1; i<n; i++) {
    while(hull.size() > 1 && !test(hull[hull.size()
        -2], hull[hull.size()-1], p[i])) hull.pop_back
        ();
    hull.push_back(p[i]);
  }
  unsigned int h1 = hull.size();
  for(int i=n-2; i>=0; i--) {
```

```
    while(hull.size() > h1 && !test(hull[hull.size()
        -2], hull[hull.size()-1], p[i])) hull.pop_back
        ();
    hull.push_back(p[i]);
  }
  hull.pop_back();
}
```

6 Data Structure

6.1 Splay Tree

```
#define MAXN 200005
#define SZ(o) (o?(o->sz):0)
#define MI(o) (o?(o->minv):2147483647)

struct Node {
  int v,sz;
  int add, minv, rev;
  Node *ch[2];
}NODES[MAXN];
int nodecnt;
Node *newNode() {
  NODES[nodecnt].v=NODES[nodecnt].add=NODES[nodecnt].
      minv=0;
  NODES[nodecnt].rev = 0;
  NODES[nodecnt].sz=1;
  NODES[nodecnt].ch[0] = NODES[nodecnt].ch[1] = NULL;
  return NODES + (nodecnt++);
}
Node *newNode(int x) { Node *res = newNode(); res->minv
     = res->v = x; return res; }

void push(Node *&o) {
  if(!o)return ;
  if(o->rev) {
    o->rev = 0;
    swap(o->ch[0], o->ch[1]);
    if(o->ch[0])o->ch[0]->rev ^= 1;
    if(o->ch[1])o->ch[1]->rev ^= 1;
  }
  if(o->add) {
    o->minv += o->add;
    o->v += o->add;
    if(o->ch[0])o->ch[0]->add += o->add;
    if(o->ch[1])o->ch[1]->add += o->add;
    o->add = 0;
  }
}
void pull(Node *&o) {
  if(!o)return ;
  push(o);
  push(o->ch[0]); push(o->ch[1]);
  o->sz = 1;
  o->sz += SZ(o->ch[0]) + SZ(o->ch[1]);
  o->minv = min(o->v, min(MI(o->ch[0]), MI(o->ch[1])));
}
void rotate(Node *&o, int d) {
  push(o);
  Node *c = o->ch[d^1];
  //cout<<o<<" "<<c<<" "<<o->v<<" "<<c->v<<endl;
  push(c);
  o->ch[d^1] = c->ch[d];
  c->ch[d] = o;
  pull(o); pull(c);
  o = c;
}
void splay(Node *&o, int k) {
  if(!o)return ;
  push(o);
  int i = SZ(o->ch[0]) + 1;
  int d1, d2;
  Node *p;
  //cout<<i<<" "<<k<<endl;
  //cout<<o<<" "<<o->ch[0]<<" "<<o->ch[1]<<endl;
  if(i == k)return ;
  else if(i < k) {
    k -= i;
    d1 = 0;
    p = o->ch[1];
  } else {
    d1 = 1;
    p = o->ch[0];
  }
  push(p);
  i = SZ(p->ch[0]) + 1;
  //cout<<"sec "<<i<<" "<<k<<endl;
  if(i == k){ rotate(o, d1); return ;}
  else if(i < k) {
    k -= i;
    d2 = 0;
    splay(p->ch[1], k);
  } else {
    d2 = 1;
    splay(p->ch[0], k);
  }
  if(d1^d2) { rotate(o->ch[d1^1], d2); rotate(o, d1); }
  else { rotate(o, d1); rotate(o, d2); }
  pull(o);
}
void split(Node *o, int x, Node *&l, Node *&r) {
  if(x == 0) { l = NULL; r = o; return ; }
  push(o);
  splay(o, x);
  r = o->ch[1];
  o->ch[1] = NULL;
  l = o;
  pull(l); pull(r);
}
void merge(Node *&l, Node *r) {
  //cout<<"l->sz: "<<SZ(l)<<endl;
  if(!l){ l = r; return ; }
  splay(l, SZ(l));
  //cout<<"l r "<<SZ(l)<<" "<<SZ(r)<<endl;
  l->ch[1] = r;
  pull(r); pull(l);
}
```

6.2 Link Cut Tree

```
#define MAXN 50005
struct Node
{
    int tag,sum,color,sz,id;
    Node *p;
    Node *ch[2];
    Node(){tag=sum=color=0;id=0;sz=1;p=ch[0]=ch[1]=NULL
        ;}
    void init(){tag=sum=color=0;id=0;sz=1;p=ch[0]=ch
        [1]=NULL;}
};
int NODE_ID;
Node NODES[MAXN];
Node* newNode(){NODES[NODE_ID].init();return &NODES[
    NODE_ID++];}

inline int SZ(Node *o){return o ? o->sz : 0;}
inline int SUM(Node *o){return o ? o->sum : 0;}

void putTag(Node *o, int c){if(!o)return ;o->sum=o->
    color=o->tag=(1<<c);}

void pull(Node *o){
    if(!o)return ;
    assert(!o->tag);
    o->sz=1+SZ(o->ch[1])+SZ(o->ch[0]); o->sum=o->color|
        SUM(o->ch[0])|SUM(o->ch[1]);
}

void push(Node *o){
    if(!o)return ;
    if(o->tag){
        o->sum=o->color=o->tag;
        if(o->ch[0])o->ch[0]->color=o->ch[0]->sum=o->ch
            [0]->tag=o->tag;
        if(o->ch[1])o->ch[1]->color=o->ch[1]->sum=o->ch
            [1]->tag=o->tag;
        o->tag=0;
    }
}
```

```
inline bool isroot(Node *o){return o?(o->p ? (o->p->ch
    [0]!=o&&o->p->ch[1]!=o) : 1):0;}

void deal(Node *o){
    if(!isroot(o))deal(o->p);
    push(o);
}

Node *A[MAXN];
int ID(Node *o){return o ? o->id : 0;}

void rotate(Node *o, int d){
    Node *t=o->ch[d^1];
    assert(t);
    o->ch[d^1]=t->ch[d]; if(t->ch[d])t->ch[d]->p=o;
    bool notroot=(!isroot(o));
    if(notroot)o->p->ch[o->p->ch[1]==o]=t;
    t->p=o->p;
    t->ch[d]=o; o->p=t;
    pull(o);pull(t);if(notroot)pull(t->p);
}

void splay(Node *o){
    if(!o||isroot(o)){push(o);return ;}
    deal(o);
    int d1,d2;
    while(o->p&&!isroot(o->p)){
        assert(o->p->p);
        d1=(o==o->p->ch[0]);
        d2=(o->p==o->p->p->ch[0]);
        if(d1^d2)rotate(o->p,d1),rotate(o->p,d2);
        else rotate(o->p->p,d2),rotate(o->p,d1);
        if(isroot(o))return ; //bug : o might be root
            of aux-tree after rotation
    }
    pull(o);
    d1=(o==o->p->ch[0]); //bug : forgot to change the
        direction
    rotate(o->p,d1);
    assert(isroot(o));
}

void access(Node *o){
    if(!o)return ;

    Node *currentPreferredChain=NULL;
    for(Node *t=o;t;t=t->p){
        splay(t);assert(!t->tag);
        t->ch[1]=currentPreferredChain;
        pull(t);
        currentPreferredChain=t;
    }
    splay(o);
}

Node* find(Node *o){
    access(o);
    for(;o->ch[0];o=o->ch[0]){push(o);}
    return o;
}

Node* cut(Node *o){
    access(o);
    Node *res=o->p ? o->p :NULL;
    if(o->ch[0]){
        for(res=o->ch[0];res->ch[1];res=res->ch[1]);
        o->ch[0]->p=o->p;
        o->ch[0]=o->p=NULL;
    }
    return res;
}

void link_to(Node *x, Node *y, int c){
    if(x==y)return ;
    Node *v=cut(x);
    if(x==find(y)){
        x->p=v; //link back if y is in the subtree of x
    }
    else{
        x->p=y;
        x->color=(1<<c);
    }
}

void query(Node *x, Node *y, int c, int &ans1, int &
    ans2){
    if(x==y||find(x)!=find(y)){ans1=ans2=0;return ;}
    access(x);
    Node *currentPreferredChain=NULL;

    for(Node *t=y;t;t=t->p){
        splay(t);
        if(!t->p){
            if(c!=-1)putTag(t->ch[1], c),putTag(
                currentPreferredChain, c);
            else ans1=SZ(t->ch[1])+SZ(
                currentPreferredChain),ans2=SUM(t->ch
                [1])|SUM(currentPreferredChain);
        }
        t->ch[1]=currentPreferredChain;
        currentPreferredChain=t;
        pull(t);
    }
    splay(y);
}
```

6.3 Leftist Tree

```
struct LeftistTree{
  struct Node{
    int v, d;
    Node *l, *r;
    Node(int _v = 0){
      v = _v, d = 1;
      l = r = NULL;
    }
    inline int deep(){
      return d;
    }
    inline void pull(){
      if (!l) {l = r; r=NULL;return ;}
      if (l->deep() < r->deep())
        swap(l, r);
      d = 1 + r->deep();
    }
  }*rt;
  int sz;
  LeftistTree(){
    sz = 0; rt = NULL;
  }
  ~LeftistTree(){
    remove(rt);
  }
  Node* merge(Node *L1, Node *L2){
    if (!L1 || !L2) return L1 ? L1 : L2;
    if (L1->v < L2->v){
      L1->r = merge(L1->r, L2);
      L1->pull();
      return L1;
    }else{
      L2->r = merge(L2->r, L1);
      L2->pull();
      return L2;
    }
  }
  void push(int v){
    rt = merge(rt, new Node(v));
    sz++;
  }
  void pop(){
    Node *tmp = rt;
    rt = merge(rt->l, rt->r);
    delete tmp;
    sz--;
  }
  void join(LeftistTree *L){
    rt = merge(rt, L->rt);
    L->rt = NULL;
    sz += L->sz;
```

```
    L->sz = 0;
  }
  int size(){ return sz; }
  int top(){ return rt->v; }
  bool empty(){ return !sz; }
  void remove(Node *u){ if (u) remove(u->l), remove(u->
      r), delete u; }
};
```

6.4 Parallel Binary Search

```
#define MAXN 100005
#define LL long long int

using namespace std;

int N,M,Q;
LL V[MAXN];
LL S[MAXN];
int A[MAXN];
int ANS[MAXN];

struct query
{
    int l,r,id;LL c;
    bool operator <(const query &r)const{
        return l<r.l;
    }
};

vector<query> QUERIES;
vector<int> FARM[MAXN];
int nxt[MAXN];

LL BIT[MAXN];
void bit_add(int pos,LL v){while(pos<=MAXN)BIT[pos]+=v,
    pos+=(pos&(-pos));}
LL bit_query(int pos){LL res=0;while(pos>0)res+=BIT[pos
    ],pos-=(pos&(-pos));return res;}

void tot_bs(int s,int e,vector<int>& people)
{
    //cout<<s<<" "<<e<<endl;
    if(s==e){for(auto p:people)ANS[p]=s;return ;}
    vector<query> queries;
    vector<int> farms;
    int mid=(s+e)/2;
    for(int i=0;i<people.size();i++)
    {
        //cout<<"YEE "<<i<<endl;
        int p=people[i];
        for(int j=0;j<FARM[p].size();j++)
        {
            //cout<<"YEE2 "<<j<<endl;
            farms.push_back(FARM[p][j]);
        }
    }
    for(int i=s;i<=mid;i++)
    {
        queries.push_back(QUERIES[i]);
    }
    sort(farms.begin(),farms.end());
    sort(queries.begin(),queries.end());
    map<int,LL> changes;
    int k=0;
    for(auto f:farms)
    {
        for(k;k<queries.size()&&f>=queries[k].l;k++)
            bit_add(queries[k].r,queries[k].c);
        LL change=bit_query(nxt[f]-1)-bit_query(f-1);
        S[A[f]]+=change;
        if(changes.find(A[f])==changes.end())changes[A[
            f]]=change;
        else changes[A[f]]+=change;
    }
    vector<int> finished,notyet;
    for(auto p:people)
    {
        if(S[p]>=V[p])finished.push_back(p),S[p]-=
            changes[p];
        else notyet.push_back(p);
    }
    for(k=k-1;k>=0;k--)bit_add(queries[k].r,-queries[k
        ].c);
    tot_bs(s,mid,finished);
    tot_bs(mid+1,e,notyet);
}

int main()
{
    while(scanf("%d%d%d",&N,&M,&Q)==3)
    {
        vector<int> people;
        for(int i=1;i<=M;i++)
        {
            scanf("%d",A+i);
            FARM[A[i]].push_back(i);
        }
        for(int i=1;i<=N;i++)
        {
            people.push_back(i);
            for(int j=0;j<FARM[i].size();j++)
            {
                nxt[FARM[i][j]]=(j+1==FARM[i].size() ?
                    M+1 : FARM[i][j+1]);
            }
            scanf("%lld",V+i);
        }
        for(int i=0;i<Q;i++)
        {
            int l,r;LL c;
            scanf("%d%d%lld",&l,&r,&c);
            QUERIES.push_back((query){l,r,i,c});
        }
        tot_bs(0,Q,people);
        for(int i=1;i<=N;i++)
        {
            printf("%d\n",ANS[i]==Q?-1:ANS[i]+1);
        }
    }
    return 0;
}
```