Team notebook

December 12, 2015

Contents

1 DS

1.1 LazySegtree

```
//to be coded
```

1.2 PersistentSegtree

```
const int N = int(1e5)+10;
const int MX = int(2e9)+10;
const int LOGN = 60;
int L[N*LOGN],R[N*LOGN],ST[N*LOGN],blen,root[N],A[N];
//sparse persistent-segment tree
//stores range sum. initially full range is 0
int update(int pos,int add,int l,int r,int id){
	if(l>pos || r<=pos)return id;
	int ID = ++blen;
	if(l==r-1){
```

```
                ST[ID]=ST[id]+add;
                return ID;
        }
        int m=l+(r-l)/2;
        L[ID]=update(pos,add,l,m,L[id]);
        R[ID]=update(pos,add,m,r,R[id]);
        ST[ID]=ST[L[ID]]+ST[R[ID]];
        return ID;
}
root[0]=++blen;
for(int i=1;i<=n;i++)
root[i]=update(A[i],1,0,MX,root[i-1]);
```

1.3 TreapBst

```
#include<bits/stdc++.h>
using namespace std;
typedef struct node{
        int val,prior,size;
        struct node *l,*r;
}node;
typedef node* pnode;
int sz(pnode t){
        return t?t->size:0;
}
void upd_sz(pnode t){
        if(t)t->size = sz(t->l)+1+sz(t->r);
}
void split(pnode t,pnode &l,pnode &r,int key){
        if(!t)l=r=NULL;
        else if(t->val<=key)split(t->r,t->r,r,key),l=t;//elem=key comes in l
        else split(t->l,l,t->l,key),r=t;
        upd_sz(t);
}
void merge(pnode &t,pnode l,pnode r){
        if(!l || !r)t=l?l:r;
        else if(l->prior > r->prior)merge(l->r,l->r,r),t=l;
        else merge(r->l,l,r->l),t=r;
        upd_sz(t);
}
void insert(pnode &t,pnode it){
        if(!t) t=it;
        else if(it->prior>t->prior)split(t,it->l,it->r,it->val),t=it;
        else insert(t->val<it->val?t->r:t->l,it);
        upd_sz(t);
}
void erase(pnode &t,int key){
        if(!t)return;
        else if(t->val==key){pnode temp=t;merge(t,t->l,t->r);free(temp);}
        else erase(t->val<key?t->r:t->l,key);
        upd_sz(t);
}
void unite (pnode &t,pnode l, pnode r) {
        if (!l || !r) return void(t = l ? l : r);
        if (l->prior < r->prior) swap (l, r);
        pnode lt, rt;
        split (r,lt, rt,l->val);
        unite (l->l,l->l, lt);
        unite (l->r,l->r, rt);
        t=l;upd_sz(t);
}
pnode init(int val){
        pnode ret = (pnode)malloc(sizeof(node));
        ret->val=val;ret->size=1;ret->prior=rand();ret->l=ret->r=NULL;
        return ret;
}
insert(init(x),head);
```

1.4 TreapIntervalTree

```
#include<bits/stdc++.h>
using namespace std;
typedef struct node{
        int prior,size;
        int val;//value stored in the array
        int sum;//whatever info you want to maintain in segtree for each node
        int lazy;//whatever lazy update you want to do
        struct node *l,*r;
}node;
typedef node* pnode;
int sz(pnode t){
        return t?t->size:0;
}
void upd_sz(pnode t){
        if(t)t->size=sz(t->l)+1+sz(t->r);
}
void lazy(pnode t){
        if(!t || !t->lazy)return;
        t->val+=t->lazy;//operation of lazy
        t->sum+=t->lazy*sz(t);
        if(t->l)t->l->lazy+=t->lazy;//propagate lazy
        if(t->r)t->r->lazy+=t->lazy;
        t->lazy=0;
}
void reset(pnode t){
        if(t)t->sum = t->val;//no need to reset lazy coz when we call this lazy would itself be propagated
}
void combine(pnode& t,pnode l,pnode r){//combining two ranges of segtree
        if(!l || !r)return void(t = l?l:r);
        t->sum = l->sum + r->sum;
}
void operation(pnode t){//operation of segtree
        if(!t)return;
        reset(t);//reset the value of current node assuming it now represents a single element of the array
        lazy(t->l);lazy(t->r);//imp:propagate lazy before combining t->l,t->r;
        combine(t,t->l,t);
        combine(t,t,t->r);
}
void split(pnode t,pnode &l,pnode &r,int pos,int add=0){
        if(!t)return void(l=r=NULL);
        lazy(t);
        int curr_pos = add + sz(t->l);
        if(curr_pos<=pos)//element at pos goes to left subtree(l)
                split(t->r,t->r,r,pos,curr_pos+1),l=t;
        else    split(t->l,l,t->l,pos,add),r=t;
        upd_sz(t);
        operation(t);
}
void merge(pnode &t,pnode l,pnode r){//l->leftarray,r->rightarray,t->resulting array
        lazy(l);lazy(r);
        if(!l || !r) t = l?l:r;
        else if(l->prior>r->prior)merge(l->r,l->r,r),t=l;
        else    merge(r->l,l,r->l),t=r;
        upd_sz(t);
        operation(t);
}
pnode init(int val){
        pnode ret = (pnode)malloc(sizeof(node));
        ret->prior=rand();ret->size=1;
        ret->val=val;
        ret->sum=val;ret->lazy=0;
        return ret;
}
int range_query(pnode t,int l,int r){//[l,r]
        pnode L,mid,R;
        split(t,L,mid,l-1);
        split(mid,t,R,r-l);//note: r-l!!
        int ans = t->sum;
        merge(mid,L,t);
        merge(t,mid,R);
        return ans;
}
void range_update(pnode t,int l,int r,int val){//[l,r]
        pnode L,mid,R;
        split(t,L,mid,l-1);
        split(mid,t,R,r-l);//note: r-l!!
        t->lazy+=val; //lazy_update
        merge(mid,L,t);
        merge(t,mid,R);
}
```

2 Geometry

2.1 ConvexHull

```
//to be done
```

2.2 GeometryTemplate

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

using namespace std;

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

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

#define rep(i, begin, end) for (__typeof(end) i = (begin) - ((begin) >
    (end)); i != (end) - ((begin) > (end)); i += 1 - 2 * ((begin) >
    (end)))
//Works for forward as well as backward iteration

#define gu getchar
#define pu putchar
#define si(n) scanf("%d",&n)
#define dout(n) printf("%d\n",n)
#define sll(n) scanf("%lld",&n)
#define lldout(n) printf("%lld\n",n)

#define DRT() int t; si(t); while(t--)

#define PlUSWRAP(index,n) index = (index+1)%n     //index++;
    if(index>=n) index=0
#define MINUSWRAP(index,n) index = (index + n -1)%n  //index--;
    if(index<0) index=n-1
#define ROUNDOFFINT(d) d = (int)((double)d + 0.5)  //Round off d to
    nearest integer

#define FLUSHN while(gu()!=’\n’)
#define FLUSHS while(gu()!=’ ’)

#define TRACE

#ifdef TRACE
#define trace1(x)              cerr << #x << ": " << x << endl;
#define trace2(x, y)           cerr << #x << ": " << x << " | " << #y <<
    ": " << y << endl;
#define trace3(x, y, z)       cerr << #x << ": " << x << " | " << #y <<
    ": " << y << " | " << #z << ": " << z << endl;
#define trace4(a, b, c, d)    cerr << #a << ": " << a << " | " << #b <<
    ": " << b << " | " << #c << ": " << c << " | " << #d << ": " << d <<
    endl;
#define trace5(a, b, c, d, e) cerr << #a << ": " << a << " | " << #b <<
    ": " << b << " | " << #c << ": " << c << " | " << #d << ": " << d <<
    " | " << #e << ": " << e << endl;
#define trace6(a, b, c, d, e, f) cerr << #a << ": " << a << " | " << #b
    << ": " << b << " | " << #c << ": " << c << " | " << #d << ": " << d
    << " | " << #e << ": " << e << " | " << #f << ": " << f << endl;

#else

#define trace1(x)
#define trace2(x, y)
#define trace3(x, y, z)
#define trace4(a, b, c, d)
#define trace5(a, b, c, d, e)
#define trace6(a, b, c, d, e, f)

#endif

//FILE *fin = freopen("in","r",stdin);
//FILE *fout = freopen("out","w",stdout);

const double EPS = 1e-9;
const double PI = 22.0/7.0;
double DEG_to_RAD(double theta)
{
       return ((theta*PI)/180.0);
}
//**************** Point *****************
class point{
       public:
              double x,y;
              point(){x=0;y=0;}
              point(double _x,double _y) : x(_x),y(_y){}
              bool operator < (point other) const
              {
                     if(fabs(x - other.x)>EPS)
                            return x<other.x;
                     return y<other.y;
              }
              bool operator == (point other) const
```

```
                {
                        return ((fabs(x - other.x)<EPS )&& (fabs(y - other.y)<EPS));
                }
};
double dist(point p1,point p2){
        return hypot(p1.x-p2.x,p1.y-p2.y);
}
//rotate p by theta "degrees" counter clockwise wrt origin (0,0)
point rotate(point p,double theta){
        double rad = DEG_to_RAD(theta);
        return point(p.x*cos(rad) - p.y*sin(rad) , p.x*sin(rad) + p.y*cos(rad));
}
// ********************** LINE *************************

// We represent it as ax+by+c=0 where b = 1 always. i.e. a/b x + y + c/b =0 or y = -a/b x + (-c/b) or y = (-A)x + (-C)

class line{
        public :
                double a,b,c;
                line(point p1,point p2)
                {
                        if(fabs(p1.x-p2.x)<EPS) //vertical line
                        {
                                a = 1.0;
                                b = 0.0;
                                c = -p1.x;
                        }
                        else
                        {
                                a = -(double)(p2.y-p1.y)/(p2.x-p1.x);
                                b = 1;
                                c = -(p1.y + a*p1.x);
                        }
                }
};

bool areParallel(line l1,line l2)
{
        return ((fabs(l1.a-l2.a)<EPS) && (fabs(l1.b-l2.b)<EPS));
}
bool areSame(line l1,line l2)
{
        return (areParallel(l1,l2) && (fabs(l1.c-l2.c)<EPS));
}
bool areIntersect(line l1,line l2,point &p)
{
        if(areParallel(l1,l2)) return false;

        //solve system of 2 linear algebraic equations with 2 unknowns
        p.x = (l2.b*l1.c - l1.b*l2.c)/(l2.a*l1.b - l1.a*l2.b);
        //special case for vertical line to avoid division by zero.
        if(fabs(l1.b)>EPS)
                p.y = -(l1.a*p.x + l1.c);
        else
                p.y = -(l2.a*p.x + l2.c);

        return true;
}

int main()
{



        return 0;
}
```

3 Graphs

3.1 AuxillaryTree

```
//Q[]: array containing k nodes of auxillary tree
//arr[] : arrival time of nodes
//anc(p,u) : returns true if p is ancestor of u
//VI tree[N] : final auxillary tree with O(k) nodes
bool cmp(int u,int v){
        return arr[u]<arr[v];
}
int create_tree(){//return root of tree
        set<int> S;//get distinct nodes
        for(int i=0;i<k;i++)
                S.insert(Q[i]);
        k=0;
        for(auto it=S.begin();it!=S.end();it++)
```

```
                Q[k++]=*it;
        sort(Q,Q+k,cmp);
        int kk = k;//number of distinct initial nodes
        //add lca of adjacent pairs
        for(int i=0;i<kk-1;i++){
                int x = lca(Q[i],Q[i+1]);
                if(S.count(x))continue;
                Q[k++]=x;
                S.insert(x);
        }
        sort(Q,Q+k,cmp);
        stack<int> s;
        s.push(Q[0]);
        for(int i=1;i<k;i++){
                while(!anc(s.top(),Q[i]))s.pop();
                tree[s.top()].PB(Q[i]);
                tree[Q[i]].PB(s.top());
                s.push(Q[i]);
        }
        return Q[0];
}
```

3.2 BCC

```
const int N = int(2e5) + 10;
VI g[N],tree[N];//graph is stored in edge list form
int U[N],V[N],vis[N],numComp,compNo[N],arr[N],t;
stack<int> S;
bool isArtic[N];
set<int> compNodes[N];//contains indexes of nodes in i'th BCC
//compNo[i] : if node i (original graph) if an articulation point, then
    corresponding Ci in block tree
//           else the Bi in the tree where the node lies
int dfs(int u,int p){
  arr[u] = t++;vis[u]=1;
  int dbe=arr[u],dfsChild=0;
  for(int i=0;i<SZ(g[u]);i++){
    int e = g[u][i],w =(U[e]==u?V[e]:U[e]);
    if(!vis[w]){
      S.push(e);dfsChild++;
      int wbe = dfs(w,u); //Back edge coming from w;
      dbe = min(dbe,wbe);
      if((arr[u]==0 && dfsChild>1) || (wbe >= arr[u] && arr[u]>0)){
        // u is an articulation vertex "wrt" to the subtree of w .
        isArtic[u]=true;
        if(compNo[u]==-1){
          compNo[u]=numComp;
          compNodes[numComp].insert(u);
          numComp++;
          compNodes[numComp].insert(u);
        }
        while(S.top()!=e){
          compNodes[numComp].insert(U[S.top()]);
          compNodes[numComp].insert(V[S.top()]);
          S.pop();
        }
        compNodes[numComp].insert(U[S.top()]);
        compNodes[numComp].insert(V[S.top()]);
        S.pop();
        for(set<int>::iterator
            it=compNodes[numComp].begin();it!=compNodes[numComp].end();it++)
          if(isArtic[*it]){
            tree[numComp].PB(compNo[*it]);
            tree[compNo[*it]].PB(numComp);
          }
          else
            compNo[*it]=numComp;
        numComp++;
      }
    }
    else if(w!=p && arr[w]<dbe) {
      dbe = min(dbe,arr[w]);
      S.push(e);
    }
  }
  return dbe;
}
void buildTree(int n)
{
  SET(compNo,-1);SET(vis,0);
  for(int i=0;i<n;i++)
  {
    if(vis[i])continue;
    t=0;dfs(i,i);
    if(!S.empty()){
      while(!S.empty()){
        compNodes[numComp].insert(U[S.top()]);
        compNodes[numComp].insert(V[S.top()]);
    S.pop();
   }
   for(set<int>::iterator it =
     compNodes[numComp].begin();it!=compNodes[numComp].end();it++)
    if(isArtic[*it]){
     tree[numComp].PB(compNo[*it]);
     tree[compNo[*it]].PB(numComp);
    }
    else
     compNo[*it]=numComp;
   numComp++;
  }
 }
}
//buildTree(n);//builds the block cut tree;
```

3.3 BridgeTree

```
VI tree[N],graph[N];//edge list representation of graph
int U[M],V[M],vis[N],arr[N],T,cmpno;
bool isbridge[M]; // if i'th edge is a bridge edge or not
queue<int> Q[N];
int adj(int u,int e){
 return U[e]==u?V[e]:U[e];
}
int dfs0(int u,int edge){ //mark bridges
 vis[u]=1;arr[u]=T++;
 int dbe = arr[u];
 for(int i=0;i<SZ(graph[u]);i++){
  int e = graph[u][i];
  int w = adj(u,e);
  if(!vis[w])dbe = min(dbe,dfs0(w,e));
  else if(e!=edge)dbe = min(dbe,arr[w]);
 }
 if(dbe == arr[u] && edge!=-1)isbridge[edge]=true;
 return dbe;
}
void dfs1(int v){//Build the bridge tree
 int currcmp = cmpno;
 Q[currcmp].push(v);vis[v]=1;
 while(!Q[currcmp].empty()){
  int u = Q[currcmp].front();
  Q[currcmp].pop();
```

```
  for(int i=0;i<SZ(graph[u]);i++){
   int e = graph[u][i];
   int w = adj(u,e);
   if(vis[w])continue;
   if(isbridge[e]){
    cmpno++;
    tree[currcmp].push_back(cmpno);
    tree[cmpno].push_back(currcmp);
    dfs1(w);
   }
   else{
    Q[currcmp].push(w);
    vis[w]=1;
   }
  }
 }
}
```

3.4 CentroidDecomposition

```
set<int> g[N];int sub[N];
//decompose
int nn;
void dfs1(int u,int p){
    sub[u]=1;nn++;
    for(auto it=g[u].begin();it!=g[u].end();it++)
        if(*it!=p){
            dfs1(*it,u);
            sub[u]+=sub[*it];
        }
}
int dfs2(int u,int p){
    for(auto it=g[u].begin();it!=g[u].end();it++)
        if(*it!=p && sub[*it]>nn/2)
            return dfs2(*it,u);
    return u;
}
void decompose(int root,int p){
    nn=0;
    dfs1(root,root);
    int centroid = dfs2(root,root);
    if(p==-1)p=centroid;
    //fuck the centroid here :)
    for(auto it=g[centroid].begin();it!=g[centroid].end();it++)
    {
        g[*it].erase(centroid);
        decompose(*it,centroid);
    }
    g[centroid].clear();
}
```

3.5 Dinics

```
// INPUT:
//     - graph, constructed using AddEdge()
//     - source and sink
// OUTPUT:
//     - maximum flow value
//     - To obtain actual flow values, look at edges with capacity > 0
//       (zero capacity edges are residual edges).
struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
  LL rcap() { return cap - flow; }
};
struct Dinic {
  int N;
  vector<vector<Edge> > G;
  vector<vector<Edge *> > Lf;
  vector<int> layer;
  vector<int> Q;
  Dinic(int N) : N(N), G(N), Q(N) {}
  void AddEdge(int from, int to, int cap) {
    if (from == to) return;
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }
  LL BlockingFlow(int s, int t) {
    layer.clear(); layer.resize(N, -1);
    layer[s] = 0;
    Lf.clear(); Lf.resize(N);
    int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail) {
      int x = Q[head++];
      for (int i = 0; i < G[x].size(); i++) {
        Edge &e = G[x][i]; if (e.rcap() <= 0) continue;
        if (layer[e.to] == -1) {
          layer[e.to] = layer[e.from] + 1;
          Q[tail++] = e.to;
        }
        if (layer[e.to] > layer[e.from]) {
          Lf[e.from].push_back(&e);
        }
      }
    }
    if (layer[t] == -1) return 0;
    LL totflow = 0;
    vector<Edge *> P;
    while (!Lf[s].empty()) {
      int curr = P.empty() ? s : P.back()->to;
      if (curr == t) { // Augment
        LL amt = P.front()->rcap();
        for (int i = 0; i < P.size(); ++i) {
          amt = min(amt, P[i]->rcap());
        }
        totflow += amt;
        for (int i = P.size() - 1; i >= 0; --i) {
          P[i]->flow += amt;
          G[P[i]->to][P[i]->index].flow -= amt;
          if (P[i]->rcap() <= 0) {
            Lf[P[i]->from].pop_back();
            P.resize(i);
          }
        }
      } else if (Lf[curr].empty()) { // Retreat
        P.pop_back();
        for (int i = 0; i < N; ++i)
          for (int j = 0; j < Lf[i].size(); ++j)
            if (Lf[i][j]->to == curr)
              Lf[i].erase(Lf[i].begin() + j);
      } else { // Advance
        P.push_back(Lf[curr].back());
      }
    }
    return totflow;
  }
  LL GetMaxFlow(int s, int t) {
    LL totflow = 0;
    while (LL flow = BlockingFlow(s, t))
    totflow += flow;
  return totflow;
 }
};
```

3.6 HLD

```
VI g[N];
int U[N],V[N],W[N];//edge list graph.graph is 1-based.
int
    DP[LOGN][N],chainParent[N],chainHead[N],blen,chainNo[N],pos[N],sub[N],nchain,level[N]
int baseArray[N],ST[4*N];//for segtree
int adj(int u,int e)
{
   return U[e]==u?V[e]:U[e];
}
void HLD(int u,int ee)
{
   baseArray[blen]=W[ee];
   pos[u]=blen;blen++;
   chainNo[u]=nchain;
   int sc=-1,mx=0;
   for(int i=0;i<SZ(g[u]);i++)
   {
      int e = g[u][i];
      if(e==ee)continue;
      int w = adj(u,e);
      if(sub[w]>mx)
         sc = e,mx = sub[w];
   }
   if(sc==-1)return;
   HLD(adj(u,sc),sc);
   for(int i=0;i<SZ(g[u]);i++)
   {
      int e = g[u][i];
      if(e==ee || e==sc)continue;
      int w = adj(u,e);
      nchain++;chainParent[nchain]=u;chainHead[nchain]=w;
      HLD(w,e);
   }
}
void dfs(int u,int ee)
{
   sub[u]=1;
   for(int i=0;i<SZ(g[u]);i++)
      if(g[u][i]!=ee)
      {
         int w = adj(u,g[u][i]);
         level[w]=level[u]+1;DP[0][w]=u;
         dfs(w,g[u][i]);
         sub[u]+=sub[w];
      }
}
void preprocess()
{
   //hang the tree
   DP[0][1]=1;dfs(1,0);
   //HLD
   chainHead[nchain]=chainParent[nchain]=1;
   HLD(1,0);
   //LCA & segtree :)
}
```

3.7 HopcroftKarp

```
const int MAXN1 = 50000;
const int MAXN2 = 50000;
const int MAXM = 150000;
int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
int matching[MAXN2], dist[MAXN1], Q[MAXN1];
bool used[MAXN1], vis[MAXN1];
//init : takes no of vertices on left and right
//addEdge(u,v) : node u on left and v on right(0-based)
void init(int _n1, int _n2) {
   n1 = _n1;
   n2 = _n2;
   edges = 0;
   fill(last, last + n1, -1);
}
void addEdge(int u, int v) {
   head[edges] = v;
   prev[edges] = last[u];
   last[u] = edges++;
}
void bfs() {
   fill(dist, dist + n1, -1);
```

```
    int sizeQ = 0;
    for (int u = 0; u < n1; ++u) {
        if (!used[u]) {
            Q[sizeQ++] = u;
            dist[u] = 0;
        }
    }
    for (int i = 0; i < sizeQ; i++) {
        int u1 = Q[i];
        for (int e = last[u1]; e >= 0; e = prev[e]) {
            int u2 = matching[head[e]];
            if (u2 >= 0 && dist[u2] < 0) {
                dist[u2] = dist[u1] + 1;
                Q[sizeQ++] = u2;
            }
        }
    }
}
bool dfs(int u1) {
    vis[u1] = true;
    for (int e = last[u1]; e >= 0; e = prev[e]) {
        int v = head[e];
        int u2 = matching[v];
        if (u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2)) {
            matching[v] = u1;
            used[u1] = true;
            return true;
        }
    }
    return false;
}
int maxMatching() {
    fill(used, used + n1, false);
    fill(matching, matching + n2, -1);
    for (int res = 0;;) {
        bfs();
        fill(vis, vis + n1, false);
        int f = 0;
        for (int u = 0; u < n1; ++u)
            if (!used[u] && dfs(u))
                ++f;
        if (!f)
            return res;
        res += f;
    }
```

```
}
```

3.8 Hungarian

```
// Min cost bipartite matching via shortest augmenting paths
// In practice, it solves 1000x1000 problems in around 1
// second.
//   cost[i][j] = cost for pairing left node i with right node j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
// The values in cost[i][j] may be positive or negative. To perform
// maximization, simply negate the cost[][] matrix.
typedef vector<double> VD;
typedef vector<VD> VVD;
double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());
  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
  }
  // construct primal solution satisfying complementary slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
        Lmate[i] = j;
        Rmate[j] = i;
        mated++;
        break;
      }
    }
  }
  VD dist(n);
  VI dad(n),seen(n);
  // repeat until primal solution is feasible
  while (mated < n) {
    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;
    // initialize Dijkstra
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++)
      dist[k] = cost[s][k] - u[s] - v[k];
    int j = 0;
    while (true) {
      // find closest
      j = -1;
      for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        if (j == -1 || dist[k] < dist[j]) j = k;
      }
      seen[j] = 1;
      // termination condition
      if (Rmate[j] == -1) break;
      // relax neighbors
      const int i = Rmate[j];
      for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
        if (dist[k] > new_dist) {
          dist[k] = new_dist;
          dad[k] = j;
        }
      }
    }
    // update dual variables
    for (int k = 0; k < n; k++) {
      if (k == j || !seen[k]) continue;
      const int i = Rmate[k];
      v[k] += dist[k] - dist[j];
      u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];
    // augment along path
    while (dad[j] >= 0) {
      const int d = dad[j];
      Rmate[j] = Rmate[d];
      Lmate[Rmate[j]] = j;
      j = d;
    }
    Rmate[j] = s;
    Lmate[s] = j;
    mated++;
  }
  double value = 0;
  for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];
  return value;
}
```

3.9 MinCostMaxFlow

```
//INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
// OUTPUT:
//     - (maximum flow value, minimum cost value(can be double)
//     - To obtain the actual flow, look at positive values only.
const LL INF = numeric_limits<LL>::max() / 4;
struct MinCostMaxFlow {
  int N;
  vector<vector<LL> > cap, flow, cost;
  VI found;
  vector<LL> dist, pi, width;
  VII dad;
  MinCostMaxFlow(int N) :
    N(N), cap(N, vector<LL>(N)), flow(N, vector<LL>(N)), cost(N,
        vector<LL>(N)),
    found(N), dist(N), pi(N), width(N), dad(N) {}
  void AddEdge(int from, int to, LL cap, LL cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
  }
  void Relax(int s, int k, LL cap, LL cost, int dir) {
    LL val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }
  LL Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;
    while (s != -1) {
      int best = -1;
      found[s] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best = k;
      }
      s = best;
    }
    for (int k = 0; k < N; k++)
      pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }
  pair<LL, LL> GetMaxFlow(int s, int t) {
    LL totflow = 0, totcost = 0;
    while (LL amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        } else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
};
```

3.10 SCC

```
VI order, component;
int vis[N];
//g:graph,rg:reverse graph
void dfs1(int u){
        vis[u] = 1;
        for(int i=0;i<SZ(g[u]);i++)
                if(!vis[g[u][i]])
                        dfs1(g[u][i]);
        order.PB(v);
}
void dfs2(int u) {
        vis[u] = 1;
        component.push_back (u);
        for(int i=0;i<SZ(rg[u]);i++)
                if (!vis[rg[u][i]])
                        dfs2(rg[u][i]);
}
//main
SET(vis,0);
for (int i=0;i<n;i++)
        if(!vis[i])
                dfs1 (i);
SET(vis,0);
for (int i=0; i<n; ++i) {
        int u = order[n-1-i];
        if (!vis[u]) {
                dfs2 (u);
                component.clear();
        }
}
```

4 Math

4.1 ExtendedGCD

```
LL coeffA, coeffB, G;
void extGcd(LL a, LL b){
  if( b == 0 ){
    G = a;
    coeffA = 1; coeffB = 0;
  }
  else{
    extGcd(b, a % b);
    coeffA -= coeffB * (a/b);
    swap(coeffA, coeffB);
  }
}
/* If this system { x = r1 mod m1 ; x = r2 mod m2 }
 * has a solution, this methods returns the
 * smallest non negative solution.
 * If there is no solution, -1 is returned.*/
LL congruence( LL r1, LL m1, LL r2, LL m2){
  extGcd(m1,m2);
  if( (r1 - r2 ) % G != 0) return -1;
  LL M = m1 * m2 / G;
  // Solution exists and is unique on LCM(m1,m2) = M
  LL K = ( r2 - r1) / G;
  LL ans = ((K * m1 * coeffA ) + r1) % M;
  //Note that K * (m1*coeffA + m2*coeffB) = r2-r1
  if( ans < 0) ans += M;
  return ans;
}
```

4.2 FFT

```
namespace FFT {
#define fore(i, a, b) for(int i = (int)(a); i <= (int)(b); ++i)
#define forn(i, n) for(int i = 0; i < (int)(n); ++i)
        typedef long double ld;
        struct base {
                typedef double T; T re, im;
                base() :re(0), im(0) {}
                base(T re) :re(re), im(0) {}
                base(T re, T im) :re(re), im(im) {}
                base operator + (const base& o) const { return base(re +
                    o.re, im + o.im); }
                base operator - (const base& o) const { return base(re -
                    o.re, im - o.im); }
                base operator * (const base& o) const { return base(re *
                    o.re - im * o.im, re * o.im + im * o.re); }
                base operator * (ld k) const { return base(re * k, im * k)
                    ;}
                base conj() const { return base(re, -im); }
        };
        const int N = 21;
        const int MAXN = (1<<N);
        base w[MAXN],f1[MAXN];
        int rev[MAXN];
        void build_rev(int k) {
                static int rk = -1;
                if( k == rk )return ; rk = k;
                for(int i=1;i<=(1<<k);i++){
                        int j = rev[i-1],t = k-1;
                        while(t >= 0 && ((j>>t)&1)){j ^= 1<<t;--t;}
                        if(t >= 0){j ^= 1<< t;--t;}
                        rev[i] = j;
                }
        }
        void fft(base *a, int k) {
                build_rev(k);int n = 1<< k;
                forn(i, n) if( rev[i] > i ) swap(a[i], a[rev[i]]);
                for(int l = 2, ll = 1; l <= n; l += l, ll += ll) {
                        if( w[ll].re == 0 && w[ll].im == 0 ) {
                                ld angle = M_PI / ll;
                                base ww( cosl(angle), sinl(angle) );
                                if( ll > 1 ) for(int j = 0; j < ll; ++j) {
                                        if( j & 1 ) w[ll + j] = w[(ll+j)/2] *
                                            ww;
                                        else w[ll + j] = w[(ll+j)/2];
                                } else w[ll] = base(1, 0);
                        }
                        for(int i = 0; i < n; i += l) forn(j, ll) {
                                base v = a[i + j], u = a[i + j + ll] * w[ll
                                    + j];
                                a[i + j] = v + u; a[i + j + ll] = v - u;
                        }
                }
        }
        VI mul(const VI& a, const VI& b) {
                int k = 1;
                while( (1<<k) < (SZ(a) + SZ(b)) ) ++k;
                int n = (1<<k);
                for(int i=0;i<n;i++)f1[i] = base(0,0);
                for(int i=0;i<SZ(a);i++)f1[i] = f1[i]+base(a[i], 0);
                for(int i=0;i<SZ(b);i++)f1[i] = f1[i]+base(0, b[i]);
                fft(f1, k);
                for(int i=1;i<=1+n/2;i++){
                        base p = f1[i] + f1[(n-i)%n].conj();
                        base _q = f1[(n-i)%n] - f1[i].conj();
                        base q(_q.im, _q.re);
                        f1[i] = (p * q) * 0.25;
                        if(i>0) f1[(n - i)] = f1[i].conj();
                }
                for(int i=0;i<n;i++)f1[i] = f1[i].conj();
                fft(f1, k);
                VI r(SZ(a) + SZ(b));
                for(int i=0;i<SZ(r);i++){
                        r[i] = LL (f1[i].re / n + 0.5);
                }
                /*Uncomment for multiplication of two numbers
                  int carry = 0;
                  for (int i=0; i<SZ(r); ++i) {
                  r[i] += carry; carry = r[i] / 10;
                  r[i] %= 10;
                  }
                  */
                return r;
        }
} // end of FFT namespace
```

4.3 MatrixDeterModP

4.4 MatrixInverse

4.5 MatrixRank

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

using namespace std;

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

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

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

#define TRACE

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

//FILE *fin = freopen("in","r",stdin);
//FILE *fout = freopen("out","w",stdout);


int main()
{



        return 0;
}
```

4.6 MillerRabin

```
/* this function calculates (a*b)%c taking into account that a*b might
    overflow */
long long mulmod(long long a,long long b,long long c){
  long long x = 0,y=a%c;
  while(b > 0){
    if(b%2 == 1)x = (x+y)%c;
    y = (y*2)%c;
    b /= 2;
  }
  return x%c;
}
/* Miller-Rabin primality test, iteration signifies the accuracy of the
    test */
bool Miller(long long p,int iteration){
  if(p<2){
    return false;
  }
  if(p!=2 && p%2==0){
    return false;
  }
  long long s=p-1;
  while(s%2==0)s/=2;
  for(int i=0;i<iteration;i++){
    long long a=rand()%(p-1)+1,temp=s;
    long long mod=power(a,temp,p);
    while(temp!=p-1 && mod!=1 && mod!=p-1){
      mod=mulmod(mod,mod,p);
      temp *= 2;
    }
    if(mod!=p-1 && temp%2==0){
      return false;
    }
  }
  return true;
}
```

4.7 QuadraticResidue

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#define M(x) ((x)%p)
using namespace std;
typedef long long int ll;
ll p;
ll pwm(ll v,int in)
{
        ll x=v;v=1;
        while(in>0)
        {
                if((in&1)==1)
                        v=M(v*x);
                in=(in>>1);
                x=M(x*x);
        }
        return v;
}
ll sqrt(ll a)
{
        ll r=0,s=p-1,n,m,x,b,g,coff,t;
        if(pwm(a,((p-1)>>1))==p-1)
                return -1;
        while((s&1)==0){     //generating s and r////
                s=(s>>1);
                r++;
        }
        for(ll i=2;i<p;i++)
                if(pwm(i,((p-1)>>1))==p-1){     //finding n//
                        n=i;
                        break;
                }
        b=pwm(a,s);
        g=pwm(n,s);
        x=pwm(a,((s+1)>>1));
        while(r>0)
        {
                t=b;
                for(m=0;m<r;m++){
                        if(M(t)==1)
                                break;
                        t=M(t*t);
                }
                if(m>0)
                {
                        coff=pwm(g,(1<<(r-(m+1))));
                        x=M(x*coff);
```

```
			g=M(coff*coff);
			b=M(b*g);
		}
		r=m;
	}
	return x;
}


//////////////////////////////////////////////////////////
///note do not use this variable again////
void ren(ll &a,ll &b,ll q)
{
	ll temp;
	temp=b;
	b=(a-(b*q));
	a=temp;
}
ll gcd(ll a,ll b)
{
	ll olds=1,s=0,t=1,oldt=0;
	ll q;
	while(b!=0)
	{
		q=a/b;
		ren(a,b,q);
		ren(olds,s,q);
		ren(oldt,t,q);
		oldt=M(oldt);
	}
	return oldt;
}
int main()
{
	ll a,q,test,b,c,sqr1,sqr2,ans1,ans2,oldt;
	scanf("%lld",&test);
	while(test--)
	{
		scanf("%lld %lld %lld %lld",&a,&b,&c,&p);
		if(p==2){
			int cnt=0;
			if(M(a+b+c)==0) cnt+=2;
			if(c==0) cnt++;
			if(cnt==3) printf("2 0 1\n");
			else if(cnt==1) printf("1 0\n");
			else if(cnt==2) printf("1 1\n");
			else printf("0\n ");
			continue;
		}
		q=M(M(b*b)-M(4*a*c)+p);
		if(b==0 && c==0){
			printf("1 0\n");
			continue;
		}
		if(q==0)
			sqr1=0;
		else
			sqr1=sqrt(q);
		sqr2=p-sqr1;
		if(sqr1==-1){
			printf("0\n");
			continue;
		}
		//		printf("%lld %lld\n",sqr1,sqr2);
		oldt=gcd(p,M(2*a));
		if(oldt<0)
			oldt+=p*((oldt/p)+1);
		oldt=M(oldt);
		//		printf("%lld\n",oldt);
		ans1=M(oldt*M(sqr1+p-b));
		ans2=M(oldt*M(sqr2+p-b));
		if(ans1<ans2)
			printf("2 %lld %lld\n",ans1,ans2);
		else if(ans2<ans1)
			printf("2 %lld %lld\n",ans2,ans1);
		else
			printf("1 %lld\n",ans1);

	}
	return 0;
}
```

4.8 Simplex

4.9 nCrLarge

```
const int MOD = int(1e6) + 3;
long long invert_mod(long long k, long long m){
    if(m == 0)return(k == 1 || k == -1) ? k : 0;
    if(m < 0) m = -m;
    k %= m;
    if(k < 0) k += m;
    int neg = 1;
    long long p1 = 1, p2 = 0, k1 = k, m1 = m, q, r, temp;
    while(k1 > 0) {
        q = m1 / k1;r = m1 % k1;
        temp = q*p1 + p2;
        p2 = p1;p1 = temp;
        m1 = k1;k1 = r;
        neg = !neg;
    }
    return neg ? m - p2 : p2;
}
// Preconditions: 0 <= k <= n; p > 1 prime
long long choose_mod_one(long long n, long long k, long long p){
    // For small k, no recursion is necessary
    if (k < p) return choose_mod_two(n,k,p);
    long long q_n, r_n, q_k, r_k, choose;
    q_n = n / p;r_n = n % p;
    q_k = k / p;r_k = k % p;
    choose = choose_mod_two(r_n, r_k, p);
    // If the exponent of p in choose(n,k) isn't determined to be 0
    // before the calculation gets serious, short-cut here:
    /* if (choose == 0) return 0; */
    choose *= choose_mod_one(q_n, q_k, p);
    return choose % p;
}
// Preconditions: 0 <= k <= min(n,p-1); p > 1 prime
long long choose_mod_two(long long n, long long k, long long p){
    // reduce n modulo p
    n %= p;
    if (n < k) return 0;
    if (k == 0 || k == n) return 1;
    // Now 0 < k < n, save a bit of work if k > n/2
    if (k > n/2) k = n-k;
    // calculate numerator and denominator modulo p
    long long num = n, den = 1;
    for(n = n-1; k > 1; --n, --k){
        num = (num * n) % p;
        den = (den * k) % p;
    }
    // Invert denominator modulo p
    den = invert_mod(den,p);
    return (num * den) % p;
}
long long factorial_exponent(long long n, long long p){
    long long ex = 0;
    do{
        n /= p;
        ex += n;
    }while(n > 0);
    return ex;
}
//returns nCk % p in O(p). n and k can be large.
long long choose_mod(long long n, long long k, long long p){
    // We deal with the trivial cases first
    if (k < 0 || n < k) return 0;
    if (k == 0 || k == n) return 1;
    // Now check whether choose(n,k) is divisible by p
    if (factorial_exponent(n) > factorial_exponent(k) +
        factorial_exponent(n-k)) return 0;
    // If it's not divisible, do the generic work
    return choose_mod_one(n,k,p);
}
```

5 String

5.1 KMP

```
// to be done
```

5.2 SuffixArray

```
//LCP[0][i] = length of LCP of SA[i] && SA[i+1] (sorted suffixes).
//RA[i][j] = Rank of suffix S[j...j+2^i] i.e. 2^j length substring
    starting at i.
//SA[i] = i'th Lexicographically smallest suffix's index.
int RA[LOGN][N],SA[N],tempSA[N],cnt[N],msb[N],LCP[LOGN][N];
void countingSort(int l,int k,int n){
  SET(cnt,0);
  for(int i=0;i<n;i++){
    int idx = i+k<n?RA[l][i+k]:0;
    cnt[idx]++;
  }
  int maxi = max(300,n);
  for(int i=0,sum=0;i<maxi;i++){
    int t = cnt[i];
    cnt[i] = sum;
    sum+=t;
  }
  for(int i=0;i<n;i++){
    int idx = SA[i]+k<n?RA[l][SA[i]+k]:0;
    tempSA[cnt[idx]++]=SA[i];
  }
  for(int i=0;i<n;i++)
    SA[i]=tempSA[i];
}
void build_SA(string &s){
  int n = SZ(s);
  for(int i=0;i<n;i++)RA[0][i]=s[i];
  for(int i=0;i<n;i++)SA[i]=i;
  for(int i=0;i<LOGN-1;i++){
    int k = (1<<i);
    if(k>=n)break;
    countingSort(i,k,n);
    countingSort(i,0,n);
    int rank=0;
    RA[i+1][SA[0]]=rank;
    for(int j=1;j<n;j++)
      if(RA[i][SA[j]] == RA[i][SA[j-1]] &&
          RA[i][SA[j]+k]==RA[i][SA[j-1]+k])RA[i+1][SA[j]]=rank;
      else RA[i+1][SA[j]]=++rank;
  }
}
void build_msb(){
  int mx=-1;
  for(int i=0;i<N;i++){
    if(i>=(1<<(mx+1)))mx++;
    msb[i]=mx;
  }
}
void build_LCP(string& s){
  int n =SZ(s);
  for(int i=0;i<n-1;i++){//Build the LCP array in O(NlogN)
    int x = SA[i],y=SA[i+1];
    int k,ret=0;
    for(k=LOGN-1;k>=0 && x<n && y<n;k--){
      if((1<<k)>=n)continue;
      if(RA[k][x]==RA[k][y])x+=1<<k,y+=1<<k,ret+=1<<k;
    }
    LCP[0][i]=ret;
  }
  LCP[0][n-1]=10*N;
  for(int i=1;i<LOGN;i++){//Build the O(1) RMQ structure in O(NlogN)
    int add = (1<<(i-1));
    if(add>=n)break; //small optimization
    for(int j=0;j<n;j++)
      if(j+add<n)LCP[i][j] = min(LCP[i-1][j],LCP[i-1][j+add]);
      else LCP[i][j] = LCP[i-1][j];
  }
}
int lcp(int x,int y){
//O(1) LCP.x and y are indexes of the suffix in SUFFIX array..!!
  if(x==y)return 0;
  if(x>y)swap(x,y);y--;
  int idx = msb[y-x+1],sub = (1<<idx);
  return min(LCP[idx][x],LCP[idx][y-sub+1]);
}
bool equal(int i,int j,int p,int q){
  if(j-i!=q-p)return false;
  int idx = msb[j-i+1],sub = (1<<idx);
  return RA[idx][i]==RA[idx][p] && RA[idx][j-sub+1]==RA[idx][q-sub+1];
}
//Note : Do not forget to add a terminating '$'
```

5.3 ZAlgo

```
//compute Z array
int L=0,R=0;
for(int i = 1; i < n; i++) {
  if (i > R) {
    L = R = i;
    while (R < n && s[R-L] == s[R]) R++;
    z[i] = R-L; R--;
  } else {
    int k = i-L;
    if (z[k] < R-i+1) z[i] = z[k];
    else {
      L = i;
      while (R < n && s[R-L] == s[R]) R++;
      z[i] = R-L; R--;
    }
  }
}
//usage
int maxz = 0, res = 0;
for (int i = 1; i < n; i++) {
    if (z[i] == n-i && maxz >= n-i) { res = n-i; break; }
      maxz = max(maxz, z[i]);
}
```

6 Theory

7 template

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

using namespace std;

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

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

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

```
#define TRACE

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

//FILE *fin = freopen("in","r",stdin);
//FILE *fout = freopen("out","w",stdout);


int main()
{



        return 0;
}
```