6/15/2014

Team Reference Document

SUST_ATTOPROTTOYEE

Syed Shahriar Manzoor, Nafis Ahmed, Imtiaz Shakil Siddique
SUST_ATTOPROTTOYEE

/* Pre Code */

```
#include <map>
#include <queue>
#include <stack>
#include <cmath>
#include <cctype>
#include <set>
#include <bitset>
#include <algorithm>
#include <list>
#include <vector>
#include <sstream>
#include <iostream>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <assert.h>
#include  <time.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> paii;
typedef vector<int> VI;

#define PI (2.0*acos(0))
#define ERR  1e-5
#define SZ(x) (int)x.size()
#define oo (1<<25)
#define FOREACH(it,x) for(__typeof((x).begin())
it=(x.begin()); it!=(x).end(); ++it)
#define popc(i) (__builtin_popcountll(i))
#define EQ(a,b)     (fabs(a-b)<ERR)
#define csprnt printf("Case %d: ", ++cas);
#define mp          make_pair
#define round(i,a)  i = ( a < 0 ) ? a - 0.5 : a +
0.5;
#define pb push_back
#define mem(a,b)    memset(a,b,sizeof(a))
#define BE(a)       a.begin(),a.end()
#define rev(a)      reverse(BE(a));
#define sorta(a)    sort(BE(a))

template<class T1,class T2> void deb(T1 e1,T2
e2){cout<<e1<<" "<<e2<<endl;}
ll Pow(ll B,ll P){      ll R=1; while(P>0)
{if(P%2==1)     R=(R*B);P/=2;B=(B*B);}return R;}
ll BigMod(ll B,ll P,ll M){     ll R=1; while(P>0)
{if(P%2==1){R=(R*B)%M;}P/=2;B=(B*B)%M;} return R;}
//freopen("in.txt","r",stdin);
```

/* BCC with Graph Build */

```
vector<int>adj[MAX], nadj[MAX], gg[MAX];
int T[MAX], low[MAX];
ull cmp_tot[MAX];
int CMP, ID;
set<int>SS;
stack<pii>S;
int deg[MAX], used[MAX], finish[MAX];
ull ans, sol;

void calc_bcc(int u, int v)
{
    int i, j, uu, vv, cur;
    pii now; int tot=0;
    SS.clear();
    while(!S.empty())
    {
        now = S.top(); S.pop();
        uu = now.first, vv = now.second;
        tot++;
        SS.insert(uu), SS.insert(vv);
        if(u==uu && v==vv) break;
        if(u==vv && v==uu) break;
    }
//    if(tot<=1) return;
    CMP++;
    set<int>::iterator it;
    for(it=SS.begin();it!=SS.end();it++)
    {
        j = (*it);
        used[j]=1;
        gg[j].pb(CMP);
        finish[j] = CMP;
        cmp_tot[CMP]++;
    }
    return;
}

void dfs_bcc(int u, int p)
{
    int i, j, v;
    T[u] = low[u] = ID++;
    for(i=0;i<SZ(adj[u]);i++)
    {
        v = adj[u][i];
        if(v==p) continue;
        if(T[v]==0)
        {
            S.push(MP(u, v));
            dfs_bcc(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v]>=T[u])
                calc_bcc(u, v);
        }
        else if(T[v] < T[u])
        {
            S.push(MP(u, v));
            low[u] = min(low[u], T[v]);
        }
    }
    return;
}

void sol_dfs(int u, int root)
{
    int i, j, v;
    int ret=0;
    for(i=0;i<SZ(nadj[u]);i++)
    {
        v = nadj[u][i];
        if(used[v]) continue;
        used[v] = 1;
        sol_dfs(v, root);
        ret++;
    }
    if(ret==0 || (u==root && ret<=1))
    {
        sol++;
        ans*=(cmp_tot[u]-1);
    }
    return;
}

int main()
{
    int t, cas=1;
    scanf("%d", &t);

while(t--)
```

```
    {
        int n, m, u, v;
        int i, j, k;
        scanf("%d%d", &n, &m);
        for(i=0;i<2*n;i++) adj[i].clear(),
nadj[i].clear(), gg[i].clear();

        for(i=0;i<m;i++)
        {
            scanf("%d%d", &u, &v);
            adj[u].pb(v);
            adj[v].pb(u);
        }
        memset(T, false, sizeof T);
        memset(cmp_tot, false, sizeof cmp_tot);
        memset(used, false, sizeof used);
        memset(finish, false, sizeof finish);
        CMP=0, ID=1;

        for(i=0;i<n;i++)
        {
            if(T[i]==0)
            {
                while(!S.empty()) S.pop();
                dfs_bcc(i, i);
            }
        }

        for(i=0;i<n;i++)
        {
            if(used[i]==0)
            {
                CMP++;
                finish[i] = CMP;
                gg[i].pb(CMP);
                cmp_tot[CMP]++;
            }
        }

//        cout<<"cmp "<<CMP<<endl;

        for(i=0;i<n;i++)
        {
            if(SZ(gg[i])<2) continue;
            CMP++;
            for(j=0;j<SZ(gg[i]);j++)
            {
                nadj[CMP].pb(gg[i][j]);
                nadj[gg[i][j]].pb(CMP);
            }
        }

        memset(used, false, sizeof used);
        memset(deg, false, sizeof deg);
        ans = 1, sol=0;
        for(i=1;i<=CMP;i++)
        {
            if(used[i]==0)
            {
                used[i]=1;
                sol_dfs(i, i);
            }
        }

        csprnt;
        if(sol==1)
        {
//            cout<<"cmp "<<cmp_tot[1]<<endl;
            ans = (cmp_tot[1]*(cmp_tot[1]-1))/2;
            sol = 2;
        }
        cout<<sol<<" "<<ans<<endl;
    }
    return 0;
}
```

/* Dinic Flow */

```
int dx[]={1,0,-1,0};int dy[]={0,1,0,-1};

using namespace std;

struct node{
        int u, v, cap, revind;
        node(int c=0, int a=0, int b=0, int d=0)
        {
                u=c, v=a, cap=b, revind=d;
        }
}; node edge[400000];
vector<int>adj[20010];
vector<pii>ed;
int source, sink, pind[20010], flow, edge_no;
char board[105][105];

void addedge(int u, int v, int cap)
{
        edge[edge_no] = node(u, v, 1, edge_no+1);
        adj[u].pb(edge_no);
        edge[edge_no+1] = node(v, u, 0, edge_no);
        adj[v].pb(edge_no+1);
        edge_no+=2;
}

bool aug_path()
{
        int i, u, used=0;
        queue<int>Q;
        Q.push(source);
        memset(pind, -1, sizeof pind);
        pind[source] = -2;
        while(!Q.empty())
        {
                u = Q.front(), Q.pop();
                for(i=0;i<SZ(adj[u]);i++)
                {
                        node now = edge[adj[u][i]];
                        if(pind[now.v]!=-1) continue;
                        if(now.cap<=0) continue;
                        pind[now.v] = adj[u][i];;
                        if(now.v!=sink)
Q.push(now.v);
                }
        }
        return (pind[sink]!=-1);
}

void path_upd(int v)
{
        node now = edge[pind[v]];
        flow = min(flow, now.cap);
        if(now.u!=source) path_upd(now.u);
        edge[pind[v]].cap -= flow;
        edge[now.revind].cap += flow;
        return;
}

bool maxflow(int cnt)
{
        int i, u, v, j, ret=0;

        while(aug_path())
        {
                for(i=0;i<SZ(ed);i++)
                {
                        u = ed[i].first, j =
ed[i].second;
```

```
                        if(pind[u]==-1) continue;
                        if(edge[j].cap<=0) continue;
                        pind[sink] = j;
                        flow=INF;
                        path_upd(sink);
                        ret+=flow;
                        if(ret==cnt) return true;
                }
        }
        return false;
}

int main()
{
    int t, cas=1;
    scanf("%d", &t);
    while(t--)
    {
        edge_no=0;
        int N, M, i, j, k, nodes, nx, ny, u, v;
        int req=0;

                scanf("%d%d ", &N, &M);
                for(i=0;i<N;i++)
                        scanf("%s", board[i]);
                nodes = N*M; ed.clear();
                source=0, sink = 2*nodes + 1;
                for(i=0;i<=sink;i++) adj[i].clear();

                for(i=0;i<N;i++)
                {
                        for(j=0;j<M;j++)
                        {
                                u = (i*M)+j+1;
                                addedge(u, u+nodes,
1);
                                if(board[i][j]=='*')
                                {

        addedge(source, u, 1);
                                        req++;
                                }
                                if(i==0 || i==N-1 ||
j==0 || j==M-1)
                                {

        addedge(u+nodes, sink, 1);

        ed.pb(MP(u+nodes, edge_no-2));
                                }
                                for(k=0;k<4;k++)
                                {
                                        nx = i+dx[k],
ny = j+dy[k];
                                        if(nx<0 ||
nx==N || ny<0 || ny==M) continue;
                                        v =
(nx*M)+ny+1;

        addedge(u+nodes, v, 1);
                                }
                        }
                }

                int sol = maxflow(req);
                csprnt;
                if(sol) printf("yes\n");
                else printf("no\n");
    }
    return 0;
}
```

/* SCC WITH BPM */

```
vector<int>G[1005], GT[1005], adj[1005], grp[1005];
vector<pii>tmp;
int col[1005], arr[1005], fns[1005], FTIME, cmpN,
CL;

void fdfs(int u)
{
    int v, i;
    FTIME++;
    fns[u]=FTIME;
    for(i=0;i<SZ(G[u]);i++)
    {
        v = G[u][i];
        if(fns[v]!=-1) continue;
        fdfs(v);
    }
    FTIME++;
    fns[u]=FTIME;
    return;
}

void bdfs(int u)
{
    int v, i;
    for(i=0;i<SZ(GT[u]);i++)
    {
        v = GT[u][i];
        if(fns[v]!=-1)  continue;
        fns[v]=cmpN;
        bdfs(v);
    }
    return;
}

bool bpm(int u)
{
    int v, i;
    if(col[u]==CL) return false;
    col[u]=CL;
    for(i=0;i<SZ(adj[u]);i++)
    {
        v = adj[u][i];
        if(arr[v]==-1 || bpm(arr[v]))
        {
            arr[v] = u;
            return true;
        }
    }
    return false;
}

int mtc()
{
    int i, ret=0; CL=0;
    memset(arr, -1, sizeof arr); memset(col, 0,
sizeof col);
    for(i=0;i<cmpN;i++)
    {
        CL++;
        if(bpm(i))
            ret++;
    }
    return ret;
}

void bld(int par, int u)
{
    int v, i;
    for(i=0;i<SZ(grp[u]);i++)
    {
        v = grp[u][i];
        if(col[v]) continue;
```

```
        col[v]=1;
        adj[par].pb(v);
        bld(par, v);
    }
    return;
}

int main()
{
    int t, cas=1;
    scanf("%d", &t);
    while(t--)
    {
        int n, m, i, j, k, u, v;
        scanf("%d%d", &n, &m);
        for(i=0;i<=n;i++) adj[i].clear(),
G[i].clear(), GT[i].clear(), grp[i].clear();
        for(i=0;i<m;i++)
        {
            scanf("%d%d", &u, &v); u--, v--;
            G[u].pb(v); GT[v].pb(u);
        }
        FTIME=0; memset(fns, -1, sizeof fns);
        for(i=0;i<n;i++)
        {
            if(fns[i]==-1)
                fdfs(i);
        }
        tmp.clear();
        for(i=0;i<n;i++)
            tmp.pb(MP(fns[i], i));
        sort(tmp.rbegin(), tmp.rend());
        cmpN=0; memset(fns, -1, sizeof fns);
        for(i=0;i<n;i++)
        {
            if(fns[tmp[i].second]==-1)
            {
                cmpN++;
                fns[tmp[i].second]=cmpN;
                bdfs(tmp[i].second);
            }
        }
        for(i=0;i<n;i++)
        {
            for(j=0;j<SZ(G[i]);j++)
            {
                v = G[i][j];
                if(fns[i]!=fns[v])
                    grp[fns[i]].pb(fns[v]);
            }
        }
        for(i=0;i<cmpN;i++)
        {
            memset(col, 0, sizeof col);
            col[i] = 1;
            bld(i, i);
        }
        int ans = mtc();
//        cout<<"cmpN "<<cmpN<<" "<<ans<<endl;
        csprnt;
        ans = cmpN-ans;
        printf("%d\n", ans);
    }
    return 0;
}
```

/* 2-SAT , SCC */

```
VI adj[2*lim]; //2*lim for true and false
argument(only adj should be cleared)
int col[2*lim],low[2*lim],tim[2*lim],timer;
int group_id[2*lim],components;//components=number
of components, group_id = which node belongs to
which node
bool ans[lim]; //boolean assignment ans
stack<int>S;

void scc(int u)
{
    int i,v,tem;
    col[u]=1;
    low[u]=tim[u]=timer++;
    S.push(u);
    fr(i,0,SZ(adj[u])-1)
    {
        v=adj[u][i];
        if(col[v]==1)
            low[u]=min(low[u],tim[v]);
        else if(col[v]==0)
        {
            scc(v);
            low[u]=min(low[u],low[v]);
        }
    }

    //SCC checking...
    if(low[u]==tim[u])
    {
        do
        {
            tem=S.top();S.pop();
            group_id[tem]=components;
            col[tem]=2; //Completed...
        }while(tem!=u);
        components++;
    }
}

int TarjanSCC(int n) //n=nodes (some change may be
required here)
{
    int i;
    timer=components=0;
    mem(col,0);
    while(!S.empty()) S.pop();
    fr(i,0,n-1) if(col[i]==0) scc(i);
    return components;
}

bool TwoSAT(int n) //n=nodes (some change may be
required here)
{
    TarjanSCC(n);
    int i;
    for(i=0;i<n;i+=2)
    {
        if(group_id[i]==group_id[i+1])
            return false;
        if(group_id[i]<group_id[i+1]) //Checking who
is lower in Topological sort
            ans[i/2]=true;
        else ans[i/2]=false;
    }
    return true;
}

void add(int ina,int inb)
{
    adj[ina].pb(inb);
}

int complement(int n)
{
    if(n%2) return n-1;
```

```
    return n+1;
}

void initialize(int n)
{
    for(int i=0;i<n;i++)
        adj[i].clear();
}
```

/* Articulation Point */

```
#define lim          1005
//no problem in multiple edge
int tim[lim],low[lim];// low means the last
dependent node (tim should be memset)
bool flag[lim];// should be memset, flag true means
articulation point
int timer;//timer=0 initially should be made
VI adj[lim];

void dfs(int u,int par)// par=-1 dhore call dite
hobe(root ar parent nai)
{
    tim[u] = low[u] = ++timer;
    int subtree = 0;
    for(int i = 0 ; i<SZ(adj[u]) ; i++)
    {
        int v = adj[u][i];
        if(v==par) continue; //parent check is
needed
        if(!tim[v])
        {
            subtree++;
            dfs(v,u);
            low[u] = min(low[u],low[v]);
            if(low[v]>=tim[u] && par!=-1) flag[u] =
true; //attention greater equals for bridge and
articulation point
        }
        else //determining back edge
        {
            low[u] = min(low[u],tim[v]);
        }
    }

    if(par==-1 && subtree>1) flag[u] = true; //for
root
    return;
}

//sometimes change needed here
void articulationPoint(int n)
{
    mem(tim,0);
    mem(flag,0);
    timer=0;
    for(int i=1;i<=n;i++)
    if(!tim[i])
        dfs(i,-1);
}
```

/* Center of Tree */

```
// split the tree so that any subtree has maximum
n/2(floor) nodes
// recursively traverse tree by center of tree
approach reqires O(nlogn) time
#define lim 100010
VI adj[lim];
int next[lim]; //next node in the longest path
void getoneend(int node,int par,int h,int
&maxhei,int &ret) //any one of the two(maybe more)
side nodes of the longest path
{
    if(maxhei<=h)
    {
        maxhei=h;
        ret=node;
    }
    int i;
    for(i=0;i<SZ(adj[node]);i++)
    {
        int tem=adj[node][i];
        if(tem==par) continue;
        getoneend(tem,node,h+1,maxhei,ret);
    }
}

int getlongestpath(int node,int par)
{
    int i;
    int ret=0;
    for(i=0;i<SZ(adj[node]);i++)
    {
        int tem=adj[node][i];
        if(tem==par) continue;
        int val=getlongestpath(tem,node)+1;
        if(ret<val)
        {
            ret=val;
            next[node]=tem;
        }
    }
    return ret;
}

int getcenteroftree(int node,int rem)
{
    if(rem==0) return node;
    return getcenteroftree(next[node],rem-1);
}

int centeroftree(int node)
{
    int maxhei=0;
    int oneend;
    getoneend(node,-1,0,maxhei,oneend);
    maxhei=getlongestpath(oneend,-1);
    return getcenteroftree(oneend,maxhei/2);
}
```

/* MST */

```
#define mo          100010
//Just need to insert input in arredge.
//The vertices order is not important that means you
just have to enter every edge only once.

typedef struct edges{
    int n1;
    int n2;
    int w;
}ed;
ed arredge[mo];
int p[mo];
int rank[mo];
int settree;

void makeset(int node)  //initialize
{
    p[node]=node;
}
```

```
void link(int x,int y)
{
    if(rank[x]>rank[y])
        p[y]=x;
    else
        p[x]=y;
    if(rank[x]==rank[y])
        rank[y]++;
}

int findset(int node)  //giving value
recursively(once done then query O(1))
{
    if(node!=p[node])
        p[node]=findset(p[node]);
    return p[node];
}

bool comp(ed x,ed y)
{
    return x.w<y.w;
}

int mst(int st,int end,int node)//look at the sort
for information about the parameters
{
        if(st>end)
        sort(&arredge[st],&arredge[end+1],comp);
        int in=st,i;
        for(i=1;i<=node;i++)//1 indexed node
            makeset(i);
        mem(rank,0);
        int edgecost=0;
        settree=node;
        while(settree!=1)/*change in this can change
mst*/
        {
            int c,d;
            c=findset(arredge[in].n1);
            d=findset(arredge[in].n2);
            if(c!=d)
            {
                settree--;
                link(c,d);
                edgecost+=arredge[in].w;
            }
            in++;
        }
        return edgecost;
}
```

/* LCA */

```
#define MAXN 100100
#define step 18    // step=log(n)
//Each time in preprocess T=0 should be set and adj
should be cleared and col should be set 0
//normally call dfs(1,1)

int
n,parent[MAXN][step+1],start[MAXN],finish[MAXN],T;//
T = time
VI adj[MAXN];

void dfs(int u,int p)
{
    int i,v;

    start[u]=T++;

    parent[u][0]=p;  //recursively defined
    for(i=1;i<=step;i++)
        parent[u][i]=parent[parent[u][i-1]][i-1];


    fr(i,0,SZ(adj[u])-1)
    {
        v=adj[u][i];
        if(v==p) continue;
        dfs(v,u);
    }
    finish[u]=T++;
}

bool IsAnchestor(int u,int v) //Is u ancestor of v
including himself
{
    if(start[u]<=start[v] && finish[u]>=finish[v])
return true;
    return false;
}

int lca_query(int u,int v)
{
    int w=-1,temp=u;

    if(IsAnchestor(u,v)) w=u;
    if(IsAnchestor(v,u)) w=v;

    if(w==-1)
    {
        for(int i=step;i>=0;i--)
            if(!IsAnchestor(parent[temp][i],v))
                temp=parent[temp][i];
        w=parent[temp][0];
    }
    return w;
}
```

/* Aho Corasick */

```
#define wnum  510
#define wsize 510
#define strsize 1000010 //size of the main string
#define bacca   27     //number of child
struct state
{
    int child[bacca],link,depth;
    vector<int>matched;
    void initialize(int dep)
    {
        mem(child,0);
        matched.clear();
        link=0;
        depth=dep;
    }
};
state T[wnum*wsize]; //normally total character
char words[wnum][wsize]; //1 based
int sz,last;  //sz=node no(1 based)(0 is root),last
is used while iteration
char str[strsize];
int height[wsize],top[wnum*wsize]; //for topological
sort
int freq[wsize*wnum],ans[wnum]; //O based.
ans=occurance of the word in the string(not always
needed). freq=substring frequency

void Initialize()  //normally only 1st node
(initialize all for safety)
{
    T[0].initialize(0);
    sz=1;
    mem(height,0);
```

```
}

void Build_Aho_Corasick(int N) //how many node
{
    Initialize();
    int i,j,len,u,v,p;
    char ch;
    queue<int>Q;

    fr(i,1,N)
    {
        last=0;
        len=strlen(words[i]);
        REP(j,len)
        {
            ch=words[i][j]-'a'; //sometimes change
here
            if(T[last].child[ch]==0){
                T[sz].initialize(j+1);
                T[last].child[ch]=sz++;
            }
            last=T[last].child[ch];
        }
        T[last].matched.pb(i);
    }

    fr(i,0,bacca-1)
    {
        if(T[0].child[i])
        {
            Q.push(T[0].child[i]);
            T[T[0].child[i]].link=0;
        }
    }

    while(!Q.empty())
    {
        u=Q.front();Q.pop();

        REP(i,bacca)
        {
            if(T[u].child[i])
            {
                p=T[u].link;
                v=T[u].child[i];
                while(p!=0 && T[p].child[i]==0)
                      p=T[p].link;
                T[v].link=T[p].child[i];
                Q.push(v);
            }
            else
                T[u].child[i] =
T[T[u].link].child[i];
        }
    }

   for(i=0;i<sz;i++) height[T[i].depth]++;
   for(i=1;i<wsize;i++) height[i]+=height[i-1];
   for(i=0;i<sz;i++) top[--height[T[i].depth]] = i;
}

void Search()
{
    int i,j,len,cur,p,v;
    char ch;

    cur=0;
    len=strlen(str);
    mem(freq,0);

    for(i=0;i<len;i++)
    {
        ch=str[i]-'a';
        if(T[cur].child[ch]==0)
        {
            p=T[cur].link;
            while(p>0 && T[p].child[ch]==0)
                p=T[p].link;
            cur=T[p].child[ch];
        }
        else
            cur=T[cur].child[ch];
        freq[cur]++;
    }

    for(i=sz-1;i>=0;i--)
    {
        v=top[i];
        freq[T[v].link]+=freq[v];
    }

    for(i=1;i<sz;i++)
    {
        if(SZ(T[i].matched))
        {
            REP(j,SZ(T[i].matched))
                ans[T[i].matched[j]]=freq[i];
        }
    }
}
```

/* BIT */

```
#define MAX 1010
//normally 1 based index
int tree[MAX];
int MaxVal;//always should be set(size of the set
len)

//cumulitive sum
int query(int idx){
    if(idx<=0) return 0;
        int sum = 0;
        idx =min(idx,MaxVal);
        while (idx > 0){
                sum += tree[idx];
                idx -= (idx & -idx);
        }
        return sum;
}

void update(int idx ,int val){
    if(idx<=0) return;
        while (idx <= MaxVal){
                tree[idx] += val;
                idx += (idx & -idx);
        }
}
```

/* Priority Queue */

```
struct pq
{
        int cost,node;
        bool operator<(const pq &b)const
        {
                return cost>b.cost;    // Min
Priority Queue(b is curret)
        }
};
```

/* Suffix Automata */

```
//Link = Longest Proper suffix in suffix
automata(already exist). (next clear can be needed)
//Depth means the highest substring attainable
towards these node. Some strings are already
attained by the link of the node(the total depth of
the link)
//Preprocess complexity nlogk (k=number of child)

struct state {
        int depth, link ;
        map < char , int > next ;  //by sacrificing
memory we can make it linear
        void initialize(){
          next.clear();
          link=-1;
      depth=0;
        }
};

const int MAXLEN = 100010; //stringsize
state st [ MAXLEN * 2 ] ;
int sz, last ;
/* when topological sort is needed (insert frequeny)
int height[MAXLEN],top[2*MAXLEN]; //for topological
sort
int maxhei;
for(i=0;i<sz;i++) height[st[i].depth]++;
for(i=1;i<=maxhei;i++) height[i]+=height[i-1];
for(i=0;i<sz;i++) top[--height[st[i].depth]] = i;
for(i=sz-1;i>=1;i--)
{
    int now=top[i];
    st[st[now].link].freq+=st[now].freq;
}
*/

void sam_init ( ) {
    //topological sort
        //mem(height,0);
        //maxhei=0;
        st[0].initialize(0,0);
        sz = last = 0 ;
        ++ sz ;

}

void sam_extend ( char c ) {
        int cur = sz ++ ;
        st[cur].initialize();
        st [cur].depth = st [ last ] . depth + 1 ;
        int p;

    for (p =last; p!= -1 && !st [p].next[c] ;p=st [
p ] . link )
                st [p] . next [ c ] = cur ;
        if ( p == - 1 )
                st [ cur ] . link = 0 ;
        else {
                int q = st [ p ] . next [ c ] ;
                if(st[p].depth + 1==st[q].depth )
                        st [ cur ] . link = q ;
                else {
                                int clone = sz ++ ;
                                //clone of q
                        st[clone].initialize();
                        st[clone].depth=st[p].depth +
1 ;
                        st[clone].next=st[q].next ;
                        st[clone].link=st[q].link ;
                        for ( ; p!= -1 && st [ p ] .
next [ c ] == q ; p = st [ p ] . link )
                                st[p].next[c]=clone;
                        st[q].link=st[cur].link=
clone ;
                }
        }
        last = cur ;
}

string lcs (string s, string t) { //longest common
substring with length
        sam_init();
        for (int i=0; i<(int)s.length(); ++i)
                sam_extend (s[i]);

        int v = 0,  l = 0,
                best = 0,  bestpos = 0;
        for (int i=0; i<(int)t.length(); ++i) {
                while (v && ! st[v].next.count(t[i]))
{
                        v = st[v].link;
                        l = st[v].depth;
                }
                if (st[v].next.count(t[i])) {
                        v = st[v].next[t[i]];
                        ++l;
                }
                if (l > best)
                        best = l,  bestpos = i;
        }
        return t.substr (bestpos-best+1, best);
}

void all_occurences ( int v, int p_length ) {

        while(true) {
            if ( ! st [ v ] . isclone )
                noverlap.pb(st [ v ] . in - p_length
);
        for ( int i = 0 ; i < st [ v ] . inv_link .
size ( ) ; ++ i )
                    all_occurences ( st [ v ] .
inv_link [ i ] , p_length ) ;
        }
}
```

/* Rectangle Union(General) */

```
//complexity k*nlgn

struct cord
{
        int x, y1, y2, val; //val for starting or
ending
        cord(int _x=0, int _y1=0, int _y2=0, int
_val=0)
        {
                x=_x, y1=_y1, y2=_y2, val=_val;
        }
};
cord pnt[MAX];
int ans[4*MAX], upd[4*MAX];
vector<int>y;

bool cmp(cord a, cord b)
{
        if(a.x==b.x) return (a.val > b.val);
        return (a.x < b.x);
}
int k;

int update(int node, int st, int end, int i, int j,
int val)
```

```
{
        if(j<=y[st] || i>=y[end]) return ans[node];
        if(y[st]>=i && y[end]<=j)
        {
                upd[node]+=val;
                if(upd[node]>=k)
                        return ans[node] = y[end]-
y[st];
                else
                {
                        if(end-st==1)
                                return ans[node]=0;
                        int mid=(st+end)>>1, ret1,
ret2;
                        //lp
                        if(upd[node]==k-1&&val==-1)
upd[2*node]++;
                        if(upd[node]==k-1&&val==-1)
upd[2*node+1]++;

             ret1 = update(2*node, st, mid, i, j,
val);
             ret2 = update(2*node+1, mid, end, i, j,
val);
             return ans[node] = ret1+ret2;
                }
        }

        int mid=(st+end)>>1, ret1, ret2;
        ret1 = update(2*node, st, mid, i, j, val);
        ret2 = update(2*node+1, mid, end, i, j,
val); //special attention to mid
        if(upd[node]<k) ans[node] = ret1+ret2;
        else ans[node]=y[end]-y[st];
        return ans[node];
}


int main()
{
    int t, cas=1;
    scanf("%d", &t);
    while(t--)
    {
                y.clear();
                int i, j, n, x1, y1, x2, y2, cnt=0,
m;
                scanf("%d %d", &n,&k);
                for(i=0;i<n;i++)
                {
                        scanf("%d%d%d%d", &x1, &y1,
&x2, &y2);
                    pnt[cnt++] = cord(x1, y1, y2, 1);
                        pnt[cnt++] = cord(x2, y1, y2,
-1);
                        y.pb(y1), y.pb(y2);
                }
                sort(y.begin(), y.end());
                y.resize(unique(y.begin(), y.end())-
y.begin());
                n = SZ(y);
                sort(&pnt[0], &pnt[0]+cnt, cmp);
                memset(ans, 0, sizeof ans);
memset(upd, 0, sizeof upd);

                ll sum=0, now; x1=-1; //any value
                for(i=0;i<cnt;i++)
                {
                        x2 = pnt[i].x;
                        now = x2-x1;
                        sum+=now*ans[1];

        //print2(pnt[i].y1,pnt[i].y2);
                        update(1, 0, n-1, pnt[i].y1,
pnt[i].y2, pnt[i].val);
                        x1 = x2;
                }
                csprnt;
                printf("%lld\n", sum);
    }
    return 0;
}
```

/* MAT EXPO */

```
#define mod         1000000007
struct matrix{
     LL x[6][6];
};
matrix base,ret,power;

void copy(matrix &a,matrix &b,int n)
{
    int i,j;
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
        a.x[i][j]=b.x[i][j];
}

void matmult(matrix &xx,matrix &a,matrix &b,int
n)//m*n and n*r matrix  //1 based
{
    int i,j,k;
    fr(i,1,n)
    fr(j,1,n)
    {
        ret.x[i][j]=0;
        fr(k,1,n)

ret.x[i][j]=ret.x[i][j]+(a.x[i][k]*b.x[k][j])%mod;
//we can reduce complexity here
        ret.x[i][j]%=mod;
    }
    copy(xx,ret,n);
}

void bigmod(matrix &xx,matrix &b,long long p,int n)
//have to pass n
{
    int i,j;
    //making it identity
    fr(i,1,n)
    fr(j,1,n)
    if(i!=j)  xx.x[i][j]=0;
    else xx.x[i][j]=1;
    copy(power,b,n);
    while(p)
    {
        if((p&1)==1) matmult(xx,xx,power,n);
        matmult(power,power,power,n);
        p/=2;
    }
}
```

/* MANACHER */

```
#define lim  100100
//0 based
int m[2*lim+1]; //length of the longest palindrome
centered at the index

int manacher(string &s)
{
    int len = s.size();
    if(len == 0) return -1;
```

```
    mem(m,0);
    m[0] = 0;
    m[1] = 1;
    // "cur" is the current center
    // "r" is the right bound of the palindrome
    // that centered at current center
    int cur, r;
    r = 2;
    cur = 1;
    int ma=1;
    for(int p2=2; p2<2*len+1; p2++)
    {
        int p1 = cur- (p2-cur);
        //if p1 is negative, we need to
        //move "cur" forward
        while(p1 < 0)
        {
            cur++;
            r = m[cur] + cur;
            p1 = cur- (p2-cur);
        }
        //If the first character of t is
        //strictly on the right of the
        // first character of s
        if(m[p1] < r - p2)
            m[p2] = m[p1];
        //otherwise
        else
        {
            //reset "cur"
            cur = p2;
            int k = r-p2;
            if(k<0) k = 0;
            while(1)
            {
                if((p2+k+1)&1)
                {
                    if(p2+k+1 < 2*len+1 && p2-k-1
>=0 && s[(p2+k)/2] == s[(p2-k-2)/2])
                        k++;
                    else break;
                }
                else
                {
                    if(p2+k+1 < 2*len+1 && p2-k-1
>=0)
                        k++;
                    else break;
                }

            }
            r = p2+k;
            m[p2] = k;
            ma=max(ma,k);
        }
    }
    return ma;
}
```

/* KMP */

```
string T;
int F[MAX];

void failure_func()//works on string T
{
        F[0]=0;
        int pt=0,i=1;
        while(i<(int)T.size() )
        {
                if( T[i]==T[pt] )
{F[i]=pt+1;++pt;i++;}
                else if( pt>0 ) pt=F[pt-1];
                else {F[i]=0;i++;}
        }
    return ;
}
```

/* Monotonous Queue */

```
#define MAX 100050
ll M[MAX],C[MAX]; //y=mx+c we need only m(slope) and
c(constant)

//Returns true if either line l1 or line l3 is
//always better than line l2
bool bad(int l1,int l2,int l3)
{
        /*
        intersection(l1,l2) has x-coordinate (c1-
c2)/(m2-m1)
        intersection(l1,l3) has x-coordinate (c1-
c3)/(m3-m1)
        set the former greater than the latter, and
cross-multiply to
        eliminate division
        */
        return (C[l3]-C[l1])*(M[l1]-M[l2])<=(C[l2]-
C[l1])*(M[l1]-M[l3]); /// ( <= sign) if the query x
//values is non-decreasing/increasing

///   (> sign) when value of x is decreasing
}

//Adding should be done serially
/// if x is in ascending order {
///     If we want minimum y coordinate(value) then
//maximum valued m should be inserted first
///     If we want maximum y coordinate(value) then
//minimum valued m should be inserted first
/// }
/// else {
///     if minimum y needed minimum m should be
inserted first
///     if maximum y needed maximum m should be
inserted first
/// }
void add(long long m,long long c,int &last)
{
        //First, let's add it to the end
        M[last]=m;
        C[last++]=c;
        //If the penultimate is now made irrelevant
//between the antepenultimate
        //and the ultimate, remove it. Repeat as
//many times as necessary
        //in short convex hull main convex hull
//tecnique is applied here
        while(last>=3&&bad(last-3,last-2,last-1))
        {
                M[last-2]=M[last-1];
                C[last-2]=C[last-1];
                last--;
        }
}

//Returns the minimum y-coordinate of any
//intersection between a given vertical
//line(x) and the lower/upper envelope(pointer)
//This can only be applied if the query of vertical
//line(x) is already sorted
//works better if number of query is huge
long long query(long long x,int &pointer,int last)
```

```
{
        //If we removed what was the best line for
//the previous query, then the
        //newly inserted line is now the best for
//that query
        if (pointer>=last)
                pointer=last-1;
        //Any better line must be to the right,
//since query values are
        //non-decreasing
        while (pointer<last-1 &&
M[pointer+1]*x+C[pointer+1]<=M[pointer]*x+C[pointer]
)  /// if Min Value wanted (<= sign) should be
//applied
                pointer++;
/// if Max Value wanted (> sign) should be applied
        return M[pointer]*x+C[pointer];
}

//for any kind of query(sorted or not) it can be
//used
//it works because of the hill property
//works better if number of query is few
long long bs(int st,int end,long long x,int last)
{
    int mid=(st+end)/2;
    if(mid+1<last &&
M[mid+1]*x+C[mid+1]<M[mid]*x+C[mid]) return
bs(mid+1,end,x,last); /// Min Value wanted. But for
//Max Value (reverse(> sign) )
    if(mid-1>=0 && M[mid-1]*x+C[mid-
1]<M[mid]*x+C[mid]) return bs(st,mid-1,x,last); ///
//Min Value wanted... But for Max Value (reverse(>
//sign) )
    return M[mid]*x+C[mid];
}
```

/* Suffix Array */

```
struct SAdata{
    paii val;
    int pos;
    SAdata() {}
    SAdata(paii x, int y)
    {
        val=x;pos=y;
    }
}SADT[MAX];/// temporary Data Structure for building
//SA
bool comSA(SAdata a, SAdata b)
{
    return a.val<b.val;
}
/// MAX is the Maximum String Size for which SA is
//being built
string input_text;/// Input String for SA to build
int SA[MAX],revSA[MAX];      /// Found SA and
//reverse SA
int lcp[MAX]; /// longest common prfix array for
//adjacent suffixes found in SA


/// SA returns the start positions  of the suffixes
//which are stored in lexicographical order
/// revSA returns for every suffix starting at 0 its
//position in SA
void buildSA()
{
    for(int i=0;i<SZ(input_text);i++)
revSA[i]=input_text[i];
    int prv_rank;

    for(int cnt=1;cnt<SZ(input_text);cnt*=2)
    {
        for(int i=0;i<SZ(input_text);i++)
        {
            SADT[i].pos=i;
            SADT[i].val.fs=revSA[i];
            SADT[i].val.sc= (i+cnt<SZ(input_text)) ?
revSA[i+cnt] : -1;
        }
        sort(SADT,SADT+SZ(input_text),comSA);

        for(int i=0;i<SZ(input_text);i++)
        {
            revSA[ SADT[i].pos ]= (i-1>=0 &&
SADT[i].val==SADT[i-1].val) ? prv_rank : i ;
            prv_rank=revSA[ SADT[i].pos ];
        }
    }
    for(int i=0;i<SZ(input_text);i++)
SA[revSA[i]]=i;
}

void buildLCP()
{
    int mx_mtch=0,SApos,j;
    for(int i=0;i<SZ(input_text);i++)
    {

        SApos=revSA[i];
        if(SApos==0) continue;
        if(mx_mtch>0) mx_mtch--;

        j=SA[SApos-1];

        while( (i+mx_mtch)<SZ(input_text) &&
(j+mx_mtch)<SZ(input_text) &&
input_text[i+mx_mtch]==input_text[j+mx_mtch]  )
            mx_mtch++;

        lcp[SApos]=mx_mtch;
    }
    lcp[0]=0;

    return ;
}
void printSA()
{
    for(int i=0;i<SZ(input_text);i++) printf("%d
",SA[i]);puts("");
    for(int i=1;i<SZ(input_text);i++) printf("%d
",lcp[i]);puts("");
    return ;
}
int main()
{
    char arr[MAX];
    int cas;
    scanf("%d",&cas);
    while(cas--)
    {
        scanf("%s",arr);
        input_text=arr;
        buildSA();
        buildLCP();
        printSA();
    }
```

```
    return 0;
}
```

/* KD Tree & K-Nearest Neighbour*/

```
#define dimension 3
#define lim 100010
struct co{
   LL x[dimension];
};
co arr[lim];

struct node{
   co now;
   //for left and right child
   int left;
   int right;
};
node bst[lim];
int axis;

bool comp(co p,co q)
{
    return p.x[axis]<q.x[axis]; //sort in terms of
axis direction
}

//overall complexity n(logn)^2
void kdtree(co arr[],int st,int end,int depth,int
&bstindex)
{
    if(st>end) return;
    axis=depth%dimension;
    sort(arr+st,arr+end+1,comp); //can be done in
nlogn time by making optimizing here
    //debug_array(arr,9);
    int median=(st+end)/2;
    ++bstindex;
    int previndex=bstindex;
    bst[previndex].now=arr[median];

    if(median!=st) bst[previndex].left=bstindex+1;
    else bst[previndex].left=0;
    kdtree(arr,st,median-1,depth+1,bstindex);

    if(median!=end) bst[previndex].right=bstindex+1;
    else bst[previndex].right=0;
    kdtree(arr,median+1,end,depth+1,bstindex);
}

LL dist(co p,co q)  //taking square distance
{
    int i;
    LL ret=0;
    for(i=0;i<dimension;i++)
        ret+=(p.x[i]-q.x[i])*(p.x[i]-q.x[i]);
    return ret;
}

//normally logn complexity
void KNN(int bstnode,int bstindex,int depth,co
&query,int k,priority_queue<LL> &Q) //kth nearest
{
    if(bstnode>bstindex) return;
    Q.push(dist(bst[bstnode].now,query));
    if(Q.size()>k) Q.pop();

    axis=depth%dimension;
    LL chc=bst[bstnode].now.x[axis]-query.x[axis];
    if(chc<=0) //go to left
    {

KNN(bst[bstnode].left,bstindex,depth+1,query,k,Q);
        //special attention to > sign (sometimes >=)
        if(Q.top()>chc*chc || Q.size()<k)  //there
is a chance of less

KNN(bst[bstnode].right,bstindex,depth+1,query,k,Q);
        return;
    }

    //go to right

KNN(bst[bstnode].right,bstindex,depth+1,query,k,Q);
    //special attention to > sign (sometimes >=)
    if(Q.top()>chc*chc || Q.size()<k)  //there is a
chance of less

KNN(bst[bstnode].left,bstindex,depth+1,query,k,Q);
    return;
}
```

/* Chinese Remainder theorem */

```
void chineseremaindertheorem(LL x[],LL a[],LL
r[][100],LL p[],LL k) //a=remainder, r[j][i]=p[j]^-1
(mod p[i]), p=primes (0 based)
{
    for ( LL i = 0 ; i < k ; ++ i ) {
        x [ i ] = a [ i ] ;
        for ( LL j = 0 ; j < i ; ++ j ) {
            x [ i ] = r [ j ][ i ] * ( x [ i ] -
x [ j ] ) ;

            x [ i ] = x [ i ] % p [ i ] ; //mod
value to avoid overflow
            if ( x [ i ] < 0 )  x [ i ] += p [ i
] ;
        }
        }
}
```

/* Combination */

```
#define mo 1010
unsigned long long comb[mo][mo];

void combination()
{
    int i,j;
    for(i=0;i<mo;i++)//let comb[0][0]=1
        comb[i][0]=1;
    comb[1][1]=1;
    for(i=2;i<mo;i++)
        for(j=1;j<mo;j++)
            comb[i][j]=comb[i-1][j]+comb[i-1][j-1];
    return ;
}
```

/* Inverse Modulo */

```
int extendedgcd ( int a, int b, int & x, int & y ) {
        if ( a == 0 ) {
            x = 0 ; y = 1 ;
            return b ;
        }
        int x1, y1 ;
        int d = extendedgcd( b % a, a, x1, y1 ) ;
        x = y1 - ( b / a ) * x1 ;
        y = x1 ;
        return d;
```

```
}

void findinverse(int a,int m)
{
   int x, y ;
   int g = extendedgcd( a, m, x, y );
   if ( g!=1 )
        cout << "no solution"<<endl;

   else {
    x = ( x % m + m ) % m ;
        cout << x <<endl;
    }
}
```

/* Extended Euclid */

```
#define paii pair<LL,LL>

//ax+by=1
paii egcd ( LL a, LL b )
{
        if (b == 1)
                return mp(0, 1);
        paii ret = egcd(b%a, a);
        int p = ret.second-(b/a)*ret.first, q =
ret.first;
        p %= b; //for overflow

        //cout << a << "*" << p << " + " << b << "*"
<< q << " = 1\n";
        return mp(p, -(a*p-1LL)/b);
}

//ax+by=c
bool find_any_solution( LL a , LL b, LL c, LL &x0 ,
LL &y0 , LL &g)
{
    if( !a && !b ) return !c;
    g=__gcd(a,b);
    if( (c%g)!=0 )
        return false;
    a/=g;
    b/=g;
    c/=g;
    paii ret=egcd(abs(a), abs(b));
    x0=ret.first;
    y0=ret.second;

    x0 = (x0*(c%b))%b;
    y0 = (c-a*x0)/b;
    if( a<0 ) x0*= -1;
    if( b<0 ) y0*= -1;

    return true;
}

void shift_solution( LL &x , LL &y , LL a, LL b, LL
cnt) {
    x+= cnt*b;
    y-= cnt*a;
}

// ax+by=c;
LL find_all_solutions (LL a, LL b, LL c, LL minx, LL
maxx, LL miny, LL maxy) //mainly takes the range
{
    LL x, y, g;

    if (!find_any_solution (a, b, c, x, y, g))
        return 0;
    if(!a&&!b)
       return (maxx-minx+1)*(maxy-miny+1);
    if(a&&!b)
    {
       x=c/a;
       if(x<minx||x>maxx) return 0;
       return maxy-miny+1;
    }

    if(!a&&b)
    {
       y=c/b;
       if(y<miny||y>maxy) return 0;
       return maxx-minx+1;
    }

    a /= g; b /= g;

    LL sign_a = a> 0?  1: - 1;
    LL sign_b = b> 0?  1: - 1;

    shift_solution (x, y, a, b, (minx - x) / b);
    if (x <minx)
        shift_solution (x, y, a, b, sign_b);
    if (x> maxx)
        return 0LL;
    LL lx1 = x;

    shift_solution (x, y, a, b, (maxx - x) / b);
    if (x> maxx)
        shift_solution (x, y, a, b, - sign_b);
    LL rx1 = x;

    shift_solution (x, y, a, b, - (miny - y) / a);
    if (y <miny)
        shift_solution (x, y, a, b, - sign_a);
    if (y> maxy)
        return 0LL;
    LL lx2 = x;

    shift_solution (x, y, a, b, - (maxy - y) / a);
    if (y> maxy)
        shift_solution (x, y, a, b, sign_a);
    LL rx2 = x;

    if (lx2> rx2)
        swap (lx2, rx2);
    LL lx = max (lx1, lx2);
    LL rx = min (rx1, rx2);

    return max(0LL,(rx - lx) / abs (b) + 1);
}
```

/* Baby Step Giant Step */

```
//a^x=b (mod m)

int solve ( int a, int b, int m ) {
        int n = ( int ) sqrt ( m + .0 ) + 1 ;

        int an = 1 ;
        for ( int i = 0 ; i < n ; ++ i )
                an = ( an * a ) % m ;

        map < int , int > vals ;
        for ( int i = 1 , cur = an ; i <= n ; ++ i )
{
                if ( ! vals. count ( cur ) )
                        vals [ cur ] = i ;
                cur = ( cur * an ) % m ;
        }

        for ( int i = 0 , cur = b ; i <= n ; ++ i )
{
                if ( vals. count ( cur ) ) {
```

```
                        int ans = vals [ cur ] * n -
i ;
                        if ( ans < m )
                                return ans ;
                }
                cur = ( cur * a ) % m ;
        }
        return - 1 ;
}
```

/* Geometry */

```
//Vector or point2d
#define vectorVar double  //change should be done
here for different datatype
struct Vector{
   vectorVar  x,y;
   Vector(vectorVar  x1=0,vectorVar  y1=0) { x=x1;
y=y1;}

   int scan() { return scanf("%lf %lf",&x,&y); }
   int scanint() { return scanf("%d %d",&x,&y); }
   int scanLL() { return scanf("%lld %lld",&x,&y); }
   void print() { print2(x,y); }

   Vector negate() { return Vector(-x,-y);}
   vectorVar length() { return sqrt(x*x+y*y);}
   vectorVar sqrLength() { return x*x+y*y; }
   vectorVar length(Vector b)  //from a to b and
vice versa
   {
       Vector tem(x-b.x,y-b.y);
       return tem.length();
   }
   vectorVar angle()  //(-pi to +pi)  (for all
angles)
   {
       vectorVar  ret=atan2(y,x);
       return ret;
   }
   vectorVar angle(Vector b)  //(0 to +pi)
   {
       vectorVar  ret=dot(b)/(length()*b.length());
       if(ret<-1) ret=-1;
       if(ret>1) ret=1;
       return acos(ret);
   }
   vectorVar angleWithSign(Vector b) //(-pi to +pi)
(a to b)
   {
       if(cross(b)>0) return angle(b);
       return -angle(b);
   }
   Vector add(Vector b) { return
Vector(x+b.x,y+b.y); }
   Vector substract(Vector b) { return Vector(x-
b.x,y-b.y); }
   vectorVar dot(Vector b) { return x*b.x+y*b.y; }
   //negative means b is clockwise to main vector
   vectorVar  cross(Vector b) { return x*b.y-b.x*y;
}
   //a is fixed
   vectorVar cross(Vector a,Vector b) //now to b
   {
       Vector now;
       now=substract(a);
       b=b.substract(a);
       return now.cross(b);
   }
   //for unit vector l=1
   Vector lengthTransform(vectorVar l) { vectorVar
len=length(); return  Vector(x*l/len,y*l/len); }
   Vector rotation(vectorVar theta) { return
Vector(x*cos(theta)-
y*sin(theta),x*sin(theta)+y*cos(theta)); }

   Vector shortestPoint(Vector b)  //make point a
vector //distance from point to line
   {
       vectorVar len=dot(b)/length();
       Vector ret=lengthTransform(len);
       if(ret.x>max(0.0,x)||ret.x<min(0.0,x))
       {
           ret.x=0;
           ret.y=0;
           if(b.length()<length(b))  return ret;
           ret.x=x;
           ret.y=y;
           return ret;
       }
       if(ret.y>max(0.0,y)||ret.y<min(0.0,y))
       {
           ret.x=0;
           ret.y=0;
           if(b.length()<length(b))  return ret;
           ret.x=x;
           ret.y=y;
           return ret;
       }
       return ret;
   }

   vectorVar shortestDist(Vector b)  //make point a
vector //distance from point to line
   {
       vectorVar len=dot(b)/length();
       Vector ret=lengthTransform(len);
       if(ret.x>max(0.0,x)||ret.x<min(0.0,x))
            return min(b.length(),length(b));

       if(ret.y>max(0.0,y)||ret.y<min(0.0,y))
           return min(b.length(),length(b));
       ret=ret.substract(b);
       return ret.length();
   }
};
typedef Vector Point;
//this are used for set compare
int dcmp(double x){ //precise up to ERR
        if(fabs(x)<ERR)return 0;else return x<0?-
1:1;
}
bool operator<( const Point& A,const Point& B
){return dcmp(A.x-B.x)<0||(dcmp(A.x-
B.x)==0&&dcmp(A.y-B.y)<0);}
bool operator==(const Point&a,const Point&b){return
dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0;}
bool operator!=(const Point&a,const Point&b){return
a==b?false:true;}

//line or segment2d
struct line{
     Vector p,q;
     line(Vector p1=0,Vector q1=0) { p=p1; q=q1; }
     void print()
     {
         printf("%.10lf %.10lf %.10lf
%.10lf\n",p.x,p.y,q.x,q.y);
     }
     //ax+by+c=0;
     void equation(vectorVar &a,vectorVar
&b,vectorVar &c)
     {
         a=p.y-q.y;
         b=q.x-p.x;
```

```
        c=-(a*p.x+b*p.y);
    }
    //y=m*x+c (p.x!=q.x)
    void equation(vectorVar &m,vectorVar &c)
    {
        vectorVar a=p.x-q.x;
        vectorVar b=p.y-q.y;
        m=b/a;
        c=(a*p.y-b*p.x)/a;
    }

    //some test still remaining
    //this line to l
    vectorVar interiorangle(line l)
    {
        vectorVar a1,b1,c1,a2,b2,c2;
        equation(a1,b1,c1);
        l.equation(a2,b2,c2);
        vectorVar x,y;
        y=-a2*b1+a1*b2;
        x=a1*a2+b1*b2;
        //print2(x,y);
        vectorVar ret=atan2(y,x);
        if(ret<-pi/2) ret=ret+pi;
        else if(ret>pi/2) ret=ret-pi;
        if(ret>pi/2) ret=pi/2;
        else if(ret<-pi/2) ret=-pi/2;
        return ret;
    }

    //some test still remaining
    //this line to l
    vectorVar exteriorangle(line l)
    {
        double ret=interiorangle(l);
        if(ret>0) ret=pi-ret;
        else ret=-pi-ret;
        if(ret>pi) ret=pi;
        else if(ret<-pi) ret=-pi;
        return ret;
    }
    //qpp1 angle (p is in the middle)
    vectorVar angle(Vector p1)
    {
        p1=p1.substract(p);
        Vector q1=q.substract(p);
        return q1.angle(p1);
    }
    //qpp1 angle (p is in the middle)  (from q to
p1)
    vectorVar angleWithSign(Vector p1)
    {
        p1=p1.substract(p);
        Vector q1=q.substract(p);
        return q1.angleWithSign(p1);
    }

    //a point inside a line segment
    bool inside(Vector p1)
    {
        if(p1.x>max(p.x,q.x)||p1.x<min(p.x,q.x))
return false;
        if(p1.y>max(p.y,q.y)||p1.y<min(p.y,q.y))
return false;
        return true;
    }
    vectorVar length() { Vector q1=q.substract(p);
return q1.length(); }
    vectorVar sqrLength() { Vector
q1=q.substract(p); return q1.sqrLength(); }
    //if p.x!=q.x
    vectorVar gety(double x)
    {
        Vector ret(q.x-p.x,q.y-p.y);
        x-=p.x;
        double m=1.0*ret.y/(1.0*ret.x);
        double y=m*x;
        y+=p.y;
        return y;
    }
    //if p.y!=q.y
    double getx(double y)
    {
        if(EQ(p.x,q.x)) return p.x;
        Vector ret(q.x-p.x,q.y-p.y);
        y-=p.y;
        double m=1.0*ret.y/(1.0*ret.x);
        double x=y/m;
        x+=p.x;
        return x;
    }

    Vector shortestPointOfSegment(Vector p1)
    {
        p1=p1.substract(p);
        Vector q1=q.substract(p);
        Vector ret=q1.shortestPoint(p1);
        ret=ret.add(p);
        return ret;
    }
    //point to segment
    vectorVar shortestDistOfSegment(Vector p1) {
p1=p1.substract(p); Vector q1=q.substract(p);
return q1.shortestDist(p1); }
    //segment to segment
    vectorVar shortestDistOfSegment(line l)
    {
        vectorVar ret=shortestDistOfSegment(l.p);
        ret=min(ret,shortestDistOfSegment(l.q));
        ret=min(ret,l.shortestDistOfSegment(p));
        ret=min(ret,l.shortestDistOfSegment(q));
        return ret;
    }
    //keeping p fixed
    line lengthTransform(vectorVar l)
    {
        Vector q1=q.substract(p);
        q1=q1.lengthTransform(l);
        q1=q1.add(p);
        return line(p,q1);
    }
    //keeping p fixed
    line rotation(vectorVar theta)
    {
        Vector q1=q.substract(p);
        q1=q1.rotation(theta);
        q1=q1.add(p);
        return line(p,q1);
    }
    //only shift in c in y=mx+c
    line shift(vectorVar cshift)
    {
        Vector tem(0,cshift);
        double theta=q.substract(p).angle();
        if(fabs(theta)>pi/2.0) theta+=pi;
        else if(EQ(theta,-pi/2.0)) theta+=pi; //-
pi/2 to pi/2 range(-pi/2 exclusive)
        tem=tem.rotation(theta);
        return line(tem.add(p),tem.add(q));
    }
    //slope should not be the same
    Vector lineIntersectingPoint(line l)
    {
        vectorVar a1,b1,c1,a2,b2,c2;
        equation(a1,b1,c1);
        l.equation(a2,b2,c2);
```

```
        Vector ret;
        ret.x=(b1*c2-b2*c1)/(a1*b2-a2*b1);
        ret.y=(c1*a2-c2*a1)/(a1*b2-a2*b1);
        return ret;
    }
    //if line segment intersect with each other
    //for double
    //risky to use this in case of double(special
attention to error)
    bool intersects(line l)
    {
        vectorVar a1,b1,c1,a2,b2,c2;
        equation(a1,b1,c1);
        l.equation(a2,b2,c2);
        if(EQ(a1*b2,a2*b1)) return false;
        Vector ret=lineIntersectingPoint(l);

if(ret.x>max(p.x,q.x)+ERR||ret.x<min(p.x,q.x)-ERR)
return false;

if(ret.x>max(l.p.x,l.q.x)+ERR||ret.x<min(l.p.x,l.q.x
)-ERR) return false;

if(ret.y>max(p.y,q.y)+ERR||ret.y<min(p.y,q.y)-ERR)
return false;

if(ret.y>max(l.p.y,l.q.y)+ERR||ret.y<min(l.p.y,l.q.y
)-ERR) return false;
        return true;
    }

    //determines which side of line the point is in
    vectorVar sideOfLine(Point p)
    {
        vectorVar a,b,c;
        equation(a,b,c);
        return a*p.x+b*p.y+c;
    }
};

struct triangle
{
    Point a,b,c;
    triangle(Vector a1=0,Vector b1=0, Vector c1=0) {
a=a1; b=b1; c=c1; }
    //a is fixed
    //.5 should be omitted in case of integer
counting
    vectorVar areaWithoutSign() { Vector
p=b.substract(a); Vector q=c.substract(a); return
fabs(.5*p.cross(q)); }
    //a is fixed
    //.5 should be omitted in case of integer
counting
    vectorVar areaWithSign() { Vector
p=b.substract(a); Vector q=c.substract(a); return
.5*p.cross(q); }
};

struct circle{
   Point c;//center
   vectorVar r;
   circle(vectorVar x=0,vectorVar y=0, vectorVar
r1=0) { c.x=x; c.y=y; r=r1; }
   double area() { return pi*r*r; }
   bool inside(Vector p) { p=p.substract(c); return
(!(p.sqrLength()>r*r)); }
   //change for integer
   bool onBoundary(Vector p) { p=p.substract(c);
return EQ(p.sqrLength(),r*r); }
   double areaOfArc(double theta) { return
(r*r*theta)/2.0; }
   //from p to q
   //area inside circle only
   double areaOfArc(Vector p,Vector q) {
p=p.substract(c); q=q.substract(c); return
areaOfArc(p.angleWithSign(q)); }
   //point should be on boundary
   double areaOfArcExceptTriangle(Vector p,Vector q)
{ double sub=triangle(c,p,q).areaWithSign(); return
areaOfArc(p,q)-sub; }
   //returns the point on boundary with given angle
   Point point(double a){return
Point(c.x+cos(a)*r,c.y+sin(a)*r);}
   //of linesegment
   //if it is tangent it will return twice
   vector<Vector> intersects(line l)
   {
       int i;
       l.p=l.p.substract(c);
       l.q=l.q.substract(c);
       Vector p,q;
       vector<Vector>ret;
       //vertical line
       if(EQ(l.p.x,l.q.x))
       {
           p.x=l.p.x;
           q.x=l.q.x;
           if(!quadraticequation(1,0,p.x*p.x-
r*r,p.y,q.y)) return ret;
           if(l.inside(p)) ret.pb(p);
           if(l.inside(q)) ret.pb(q);
           fr(i,0,SZ(ret)-1)
            ret[i]=ret[i].add(c);
           return ret;
       }
       vectorVar m,cc;
       l.equation(m,cc);
       if(!quadraticequation(1+m*m,2*m*cc,cc*cc-
r*r,p.x,q.x)) return ret;
       p.y=m*p.x+cc;
       q.y=m*q.x+cc;
       if(l.inside(p)) ret.pb(p);
       if(l.inside(q)) ret.pb(q);
       fr(i,0,SZ(ret)-1)
            ret[i]=ret[i].add(c);
       return ret;
   }

   //1 based
   //polygon should be simple
   //logn
   double intersectingArea(Vector poly[],int n)
   {
       int i;
       double area=0;
       for(i=1;i<=n;i++)
       {
           int j=i+1;
           if(j>n) j=1;
           vector<Vector>
ret=intersects(line(poly[i],poly[j]));
           if(inside(poly[i]) && inside(poly[j]))
//both inside

area+=triangle(c,poly[i],poly[j]).areaWithSign();
           else if(!ret.size()) //both outside
without intersection
             area+=areaOfArc(poly[i],poly[j]);
           else if(ret.size()==1) //exactly 1 point
is inside
           {
               if(inside(poly[i]))
area+=areaOfArc(ret[0],poly[j])+triangle(c,poly[i],r
et[0]).areaWithSign();
```

```
                else
area+=areaOfArc(poly[i],ret[0])+triangle(c,ret[0],po
ly[j]).areaWithSign();
            }
            else //both are outside with intersection
            {

if(poly[i].length(ret[0])>poly[i].length(ret[1]))
swap(ret[0],ret[1]);

area+=areaOfArc(poly[i],ret[0])+triangle(c,ret[0],re
t[1]).areaWithSign()+areaOfArc(ret[1],poly[j]);

            }
        }
        return fabs(area);
    }

    int circleIntersectingPoint(circle cir,Point
&p1,Point &p2)
    {
        double d=c.length(cir.c); //distance of two
center
        if(dcmp(d)==0) //same center
        {
            if(dcmp(r-cir.r)==0) return 3; //same
circle(infinite intersection point)
            return 0; //totally inside
           }
           //different center
        if(dcmp(r+cir.r-d)<0) return -1; //strictly
outside
        if(dcmp(fabs(r-cir.r)-d)>0) return 0;
//strictly inside
        double a=fabs(cir.c.substract(c).angle());
        double da=acos((r*r+d*d-
cir.r*cir.r)/(2*r*d));
        p1=point(a-da);p2=point(a+da);
        if(p1==p2) return 1; //touch in one point
        return 2;
    }

    //tested by LJ 1118
    double circleIntersectingArea(circle cir)
    {
        double d = c.length(cir.c);
        double r1=r;
        double r2=cir.r;
        if (r1 + r2 <d) return 0; //outside
        if (d >fabs (r1 - r2)+PRE) //partially
inside
        {
            double x = (d * d + r1 * r1 - r2 * r2)
/ (2 * d);
            double t1 = acos (x / r1);
            double t2 = acos ((d - x) / r2);
            return r1 * r1 * t1 + r2 * r2 * t2 - d
* r1 * sin (t1);
        }
         //totally inside
        double rr = min (r1, r2);
        return pi * rr * rr;
    }
};

//1 based
//should be clockwise or anticlockwise
//works in simple polygon
vectorVar areaOfPolygon(Vector poly[],int n)
{
    int i;
    double area=0;
    for(i=2;i<n;i++)
    {
        triangle tem(poly[1],poly[i],poly[i+1]);
        area+=tem.areaWithSign();
    }
    return fabs(area);
}

//1 based
//should be clockwise or anticlockwise
//clipping polygon should be strictly convex(no 180
degree angles) and target polygon should be simple
//twice memory is needed in worst case
//complexity 2*n*m
vectorVar areaOfClippingPolygon(Vector
clipPoly[],int n,Vector targetPoly[],int m)
{
    int i,j;
    Vector temtar[2*m+4];
    int temm;
    double
chck=clipPoly[2].cross(clipPoly[1],clipPoly[3]);
    for(i=1;i<=n;i++)
    {
        int next=i+1;
        if(next>n) next=1;
        temm=0;
        //clipping done with infinte line
        for(j=1;j<=m;j++)
        {
            int nextj=j+1;
            if(nextj>m) nextj=1;

if(clipPoly[next].cross(clipPoly[i],targetPoly[j])*c
hck>-PRE)
            {

if(clipPoly[next].cross(clipPoly[i],targetPoly[nextj
])*chck>-PRE) //both are inside

temtar[++temm]=targetPoly[nextj];
                else
temtar[++temm]=line(clipPoly[i],clipPoly[next]).line
IntersectingPoint(line(targetPoly[j],targetPoly[next
j])); //inside to outside
            }

            else
if(clipPoly[next].cross(clipPoly[i],targetPoly[nextj
])*chck>-PRE) //outside to inside
            {

temtar[++temm]=line(clipPoly[i],clipPoly[next]).line
IntersectingPoint(line(targetPoly[j],targetPoly[next
j]));
                temtar[++temm]=targetPoly[nextj];
            }
        }

        m=temm;
        for(j=1;j<=m;j++)
            targetPoly[j]=temtar[j];
    }

    return areaOfPolygon(targetPoly,m);
}

//1 based
//polygon should be disjoint and strictly convex
//should be given in clockwise or anticlockwise
//complexity m+n
vectorVar shortestDistBetweenPolygon(Vector P[],int
n,Vector Q[],int m)
{
```

```
    if(P[2].cross(P[1],P[3])>0) //anticlockwise
        reverse(P+1,P+n+1);
    if(Q[2].cross(Q[1],Q[3])>0) //anticlockwise
        reverse(Q+1,Q+m+1);
    int inP=1;
    double mi=P[1].y;
    int i,j;
    for(i=2;i<=n;i++)
    if(P[i].y<mi)
    {
        mi=P[i].y;
        inP=i;
    }

    int inQ=1;
    double ma=Q[1].y;
    for(i=2;i<=m;i++)
    if(Q[i].y>ma)
    {
        ma=Q[i].y;
        inQ=i;
    }

    i=inQ;
    j=inP;
    int cntP=1;
    int cntQ=1;
    double ans=P[inP].length(Q[inQ]);
    while(cntP<n||cntQ<m)
    {
        if(i>n) i=1;
        if(j>m) j=1;
        int nexti=i+1;
        int nextj=j+1;
        if(nexti>n) nexti=1;
        if(nextj>m) nextj=1;
        vectorVar
chck=P[nexti].substract(P[i]).angle();
        chck-=Q[nextj].substract(Q[j]).angle();
        if(chck<0) chck+=2.0*pi;
        if(fabs(chck)<ERR&&cntP<n&&cntQ<m) //segment
to segment
        {

ans=min(ans,line(P[nexti],P[i]).shortestDistOfSegmen
t(line(Q[nextj],Q[j])));
            i++;
            j++;
            cntP++;
            cntQ++;
        }

        else if(chck<pi&&cntQ<m) //Q is near
        {

ans=min(ans,line(Q[nextj],Q[j]).shortestDistOfSegmen
t(P[i]));
            j++;
            cntQ++;
        }

        else if(chck>pi&&cntP<n) //P is near
        {

ans=min(ans,line(P[nexti],P[i]).shortestDistOfSegmen
t(Q[j]));
            i++;
            cntP++;
        }

        else if(cntQ<m) //only Q left
        {

ans=min(ans,line(Q[nextj],Q[j]).shortestDistOfSegmen
t(P[i]));
            j++;
            cntQ++;
        }

        else //only P left
        {

ans=min(ans,line(P[nexti],P[i]).shortestDistOfSegmen
t(Q[j]));
            i++;
            cntP++;
        }
    }
    return ans;
}

//a*x^2+b*x+c=0
bool quadraticequation(vectorVar a,vectorVar
b,vectorVar c,double &x1,double &x2)
{
    vectorVar d=b*b-a*c*4;
    if(d<0) return false;
    d=sqrt(d);
    x1=(-b+d)/(2*a);
    x2=(-b-d)/(2*a);
    return true;
}

bool mult(Point sp,Point ep,Point op)
{
        return (sp.x-op.x)*(ep.y-op.y)>=(ep.x-
op.x)*(sp.y-op.y);
}

bool operator < (const Point& l,const Point& r)
{
        return l.y<r.y||(l.y==r.y&&l.x<r.x);
}

int graham(Point pnt[],int n,Point res[])
{
        int i,len,k=0,top=1;
        sort(pnt,pnt+n);
        if(n==0) return 0; res[0]=pnt[0];
        if(n==1) return 1; res[1]=pnt[1];
        if(n==2) return 2; res[2]=pnt[2];
        for(i=2;i<n;i++)
        {

        while(top&&mult(pnt[i],res[top],res[top-1]))
                        top--;
                res[++top]=pnt[i];
        }
        len=top; res[++top]=pnt[n-2];
        for(i=n-3;i>=0;i--)
        {

        while(top!=len&&mult(pnt[i],res[top],res[top
-1]))
                        top--;
                res[++top]=pnt[i];
        }
        return top;
}
//works for simple polygon (both convex and concave)
//returns true if it on the boundary or vertex
//1 based
bool pointInPoly(int n, Vector arr[], Vector P)
//nodes should be given in clockwise or anti-
clockwise order
```

```
{
  int i, j;
  bool c=false;
  vectorVar xx=P.x;
  vectorVar yy=P.y;
  for (i = 1, j = n; i <= n; j = i++) {
      if ( ((arr[i].y>yy) != (arr[j].y>yy)) &&
//here all corner(vertex) case are handled
        (xx < (arr[j].x-arr[i].x) * (yy-arr[i].y) /
(arr[j].y-arr[i].y) + arr[i].x) )
       c = !c;
  }
  return c;
}
```

/* Geometry 3D */

```
public class Geom3D {
  // distance from point (x, y, z) to plane aX + bY
+ cZ + d = 0
  public static double ptPlaneDist(double x, double
y, double z,
      double a, double b, double c, double d) {
    return Math.abs(a*x + b*y + c*z + d) /
Math.sqrt(a*a + b*b + c*c);
  }

  // distance between parallel planes aX + bY + cZ +
d1 = 0 and
  // aX + bY + cZ + d2 = 0
  public static double planePlaneDist(double a,
double b, double c,
      double d1, double d2) {
    return Math.abs(d1 - d2) / Math.sqrt(a*a + b*b +
c*c);
  }

  // distance from point (px, py, pz) to line (x1,
y1, z1)-(x2, y2, z2)
  // (or ray, or segment; in the case of the ray,
the endpoint is the
  // first point)
  public static final int LINE = 0;
  public static final int SEGMENT = 1;
  public static final int RAY = 2;
  public static double ptLineDistSq(double x1,
double y1, double z1,
      double x2, double y2, double z2, double px,
double py, double pz,
      int type) {
    double pd2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) +
(z1-z2)*(z1-z2);

    double x, y, z;
    if (pd2 == 0) {
      x = x1;
      y = y1;
      z = z1;
    } else {
      double u = ((px-x1)*(x2-x1) + (py-y1)*(y2-y1)
+ (pz-z1)*(z2-z1)) / pd2;
      x = x1 + u * (x2 - x1);
      y = y1 + u * (y2 - y1);
      z = z1 + u * (z2 - z1);
      if (type != LINE && u < 0) {
        x = x1;
        y = y1;
        z = z1;
      }
      if (type == SEGMENT && u > 1.0) {
        x = x2;
        y = y2;
        z = z2;
      }
    }

    return (x-px)*(x-px) + (y-py)*(y-py) + (z-
pz)*(z-pz);
  }

  public static double ptLineDist(double x1, double
y1, double z1,
      double x2, double y2, double z2, double px,
double py, double pz,
      int type) {
    return Math.sqrt(ptLineDistSq(x1, y1, z1, x2,
y2, z2, px, py, pz, type));
  }
}
```

/* String Operation */

```
#define REV(i,n) for (i=n;i>=0;i--)
#define FOR(i,j,n) for(i=j;i<n;i++)

string Multiplication(string a,string b){
    int i,j,multi,carry; string ans="0",temp;
    REV(j,SZ(b)-1){
        temp=""; carry=0;
        REV(i,SZ(a)-1){
            multi=(a[i]-'0')*(b[j]-'0')+carry;
            temp+=(multi%10+'0'); carry=multi/10;
        }
        if(carry) temp+=(carry+'0'); Reverse(temp);
        temp+=string(SZ(b)-j-1,'0');
        ans=Addition(ans,temp);
    }
    ans=cut_leading_zero(ans);
    return ans;
}

string Addition(string a,string b){
    int carry=0,i; string ans;
    if(SZ(a)>SZ(b)) b=string(SZ(a)-SZ(b),'0')+b;
    if(SZ(b)>SZ(a)) a=string(SZ(b)-SZ(a),'0')+a;

    ans.resize(SZ(a));
    REV(i,SZ(a)-1){
        int sum=carry+a[i]+b[i]-96;
        ans[i]=(char)(sum%10+'0'); carry=sum/10;
    }
    if(carry) ans.insert(0,string(1,carry+'0'));
    ans=cut_leading_zero(ans);
    return ans;
}

string Division(string a,string b){
    string mod,temp,ans="0"; int i,j;
    REP(i,SZ(a)){
        mod+=a[i]; mod=cut_leading_zero(mod);
        FOR(j,0,10){
            temp=Multiplication(b,j);
            if(compare(temp,mod)==1) break;
        }
        temp=Multiplication(b,j-1);
        mod=Subtraction(mod,temp); ans+=(j-1)+'0';
    }
    mod=cut_leading_zero(mod);
    ans=cut_leading_zero(ans);
    return ans;
}

int Div_mod(string a,int k){
  int i,sum=0;REP(i,SZ(a))sum=(sum*10+(a[i]-'0'))%k;
  return sum;
}

int compare(string a,string b){
    int i;
    a=cut_leading_zero(a);b=cut_leading_zero(b);
```

```
    if(SZ(a)>SZ(b))return 1;
    if(SZ(a)<SZ(b)) return -1;
    REP(i,SZ(a))
        if(a[i]>b[i]) return 1;
        else if(a[i]<b[i]) return -1;
    return 0;
}

string cut_leading_zero(string a){
    string s=""; int i;
    if(a[0]!='0') return a;
    REP(i,SZ(a)-1) if(a[i]!='0') break;
    FOR(i,i,SZ(a)) s+=a[i];
    return s;
}
```

/* Determinant */

```
//a is the total matrix, last column is the constant
matrix and other columns are coefficient matrix
//final ans is stored is ans matrix

int det (vector < vector < double > > a)
//determinant of a square matrix
{
    int n=( int ) a. size ();
    int i, j, k, flg = 1;
    double ans=1.0,x;
    for (i = 0; i < n; i++)
    {
        int sol=i;
        for (j = i+1; j < n; j++)
            if (abs(a[j][i])>abs(a[sol][i]))
                sol=i;
        if(abs(a[i][sol])<ERR) return -1;
//according to problem
        flg = !flg;
        for (k = i; k < n; k++)
            swap (a[i][k], a[j][k]);

        ans = ans * a[i][i];
        x=1.0/a[i][i];
        for (k = i+1; k < n; k++)
            a[i][k] = a[i][k] * x;
        for (j = i+1; j < n; j++)
            if (abs(a[j][i])<ERR) for (k = i+1; k <
n; k++)
                a[j][k] = a[j][k] - a[i][k]*a[j][i];
    }
    if (flg) return ans;
    return -ans;
}
```

/* GAUSS */

```
#define INF 1000000000
//a is the total matrix, last column is the constant
matrix and other columns are coefficient matrix
//final ans is stored is ans matrix

int gauss ( vector < vector < double > > a, vector <
double > & ans )
{
        int n = ( int ) a. size ( ) ;
        int m = ( int ) a [ 0 ] . size ( ) - 1 ;

        vector < int > where ( m, - 1 ) ;
        for ( int col = 0 , row = 0 ; col < m && row
< n ; ++ col ) {
                int sel = row ;
                for ( int i = row ; i < n ; ++ i )
                        if ( abs ( a [ i ] [ col ] )
> abs ( a [ sel ] [ col ] ) ) //maxvalued row for
this column
                                sel = i ;
                if ( abs ( a [ sel ] [ col ] ) < ERR
)
                        continue ;
                for ( int i = col ; i <= m ; ++ i )
                        swap ( a [ sel ] [ i ] , a [
row ] [ i ] ) ; //swap the rows
                where [ col ] = row ;

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

        ans. assign ( m, 0 ) ;
        for ( int i = 0 ; i < m ; ++ i )
                if ( where [ i ] != - 1 )
                        ans [ i ] = a [ where [ i ] ]
[ m ] / a [ where [ i ] ] [ i ] ;
        //checking right
        for ( int i = 0 ; i < n ; ++ i ) {
                double sum = 0 ;
                for ( int j = 0 ; j < m ; ++ j )
                        sum += ans [ j ] * a [ i ] [
j ] ;
                if ( abs ( sum - a [ i ] [ m ] ) >
ERR ) //no solution
                        return 0 ;
        }

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

/* FFT */

```
// nlogn complexity
// memory complexity 12n
/* application
   1. multiplying two arrays.
   2. multiplying two long(string) numbers.
*/
// i-th index mean coefficient2 of i-th power
typedef complex <double> base ;

void fft ( vector < base > & a, bool invert ) {
//invert=true means inverse FFT
        int n = ( int ) a. size ( ) ;

        for ( int i = 1 , j = 0 ; i < n ; ++ i ) {
                int bit = n >> 1 ;
                for ( ; j >= bit ; bit >>= 1 )
                        j -= bit ;
                j += bit ;
                if ( i < j )
                        swap ( a [ i ] , a [ j ] ) ;
        }

        for ( int len = 2 ; len <= n ; len <<= 1 ) {
```

```
                double ang = 2 * pi / len * ( invert
? - 1 : 1 ) ;
                base wlen ( cos ( ang ) , sin ( ang )
) ;
                for ( int i = 0 ; i < n ; i += len )
{
                        base w ( 1 ) ;
                        for ( int j = 0 ; j < len / 2
; ++ j ) {
                                base u = a [ i + j ] ,
v = a [ i + j + len / 2 ] * w ;
                                a [ i + j ] = u + v ;
                                a [ i + j + len / 2 ]
= u - v ;
                                w *= wlen ;
                        }
                }
        }
        if ( invert )
                for ( int i = 0 ; i < n ; ++ i )
                        a [ i ]/= n ;
}

void multiply ( vector < int > & a, vector < int > &
b, vector < int > & res ) {
        vector < base > fa ( a. begin ( ) , a. end (
) ) ,  fb ( b. begin ( ) , b. end ( ) ) ;
        size_t n = 1 ;
        while ( n < max ( a. size ( ) , b. size ( )
) )  n <<= 1 ; //making it a power of 2
        n <<= 1 ; //making double size(2*n)
        fa. resize ( n ) ,  fb. resize ( n ) ;

        fft ( fa, false ) ,  fft ( fb, false ) ;
        for ( size_t i = 0 ; i < n ; ++ i )
                fa [ i ] *= fb [ i ] ;
        fft ( fa, true ) ; //inverse fft

        res. resize ( n ) ;
        for ( size_t i = 0 ; i < n ; ++ i )
                res [ i ] = int ( fa [ i ] . real ( )
+ 0.5 ) ;
}

void carryoperation( vector < int > & res )
//multiplying two long(string) numbers.(normalizing)
{
    int n=res.size();
    int carry = 0 ;
        for ( size_t i = 0 ; i < n ; ++ i ) {
                res [ i ] += carry ;
                carry = res [ i ] / 10 ;
                res [ i ] %= 10 ;
        }
}
```

/* Infix 2 Postfix */

```
int order[300]

// order should be defined before calling this
function
// will not work for unary operators
// variables should be single character alpha
chracters (a-zA-Z)
// numbers can be of any character
// no spaces are allowed in the string
// only "()" this bracket is allowed
// "()" have highest precedence than anyhing
vector<string> infix_to_postfix(string &str)
{
    vector<string>inp;
    string val;
    for(int i=0;i<SZ(str);i++)
    {
        if(  isdigit(str[i]) )
        {
            val="";
            int j;
            for( j=i; j<SZ(str) ;j++ )
            {
                if( !isdigit(str[j]) ) break;
                val=val+str[j];
            }
            i=j-1;
            inp.pb(val);
        }
        else {
            string tt="";tt.pb(str[i]);
            inp.pb(tt);
        }
    }

    stack<string>S;
    vector<string>res;

    string tmp;
    for(int i=0;i<SZ(inp);i++)
    {
        if( isalpha(inp[i][0]) ) res.pb(inp[i]);
        else if( isdigit(inp[i][0]) )
res.pb(inp[i]);
        else {
            if(inp[i][0]=='(') S.push(inp[i]);
            else if( S.empty() ) S.push(inp[i]);
            else if(inp[i][0]==')')
            {
                tmp=S.top();
                while(tmp[0]!='(') {
                    res.pb( S.top() );
                    S.pop();
                    tmp=S.top();
                }
                S.pop();
            }
            else {
                while(true)
                {
                    if(S.empty()) break;
                    tmp=S.top();
                    if( order[tmp[0]] >=
order[inp[i][0]] ) {
                        res.pb(S.top());
                        S.pop();
                        if(S.empty()) break;
                        tmp=S.top();
                    }
                    else break;
                }
                S.push( inp[i] );
            }
        }
    }
    while(!S.empty()) {res.pb(S.top()); S.pop();}
    return res;
}

int main()
{
    order['+']=1;
    order['-']=1;
    order['*']=2;
    order['/']=3;
    string str;
```

```
    //(n+22)*c+g+f*(x+123)-(0-100)
    //((a/b)-(c*d/23)*(51-23))/f

    vector<string>inp;
    str="((a/b)-(c*d/23)*(51-23))/f";
    inp.clear();
    inp=infix_to_postfix(str);
    for(int i=0;i<SZ(inp);i++) cout<<inp[i]<<endl;

    return 0;
}
```

/* Hashing */

```
//no two level hash required for both method even
//for string related hash
//easy to code
//sometimes more time consuming even for less
//collision
struct Map
{
    static const int M = 1400007; //mod value
    long long v[M]; //the value
    int f[M]; //number of occurence
    void init()
    {
        memset(f,0,sizeof(f));
    }
    void insert(long long val)
    {
       int p = val%M;
       while(f[p] && v[p]!=val)  //check for
collision
       {
            p++;if(p==M) p = 0;
       }
       f[p]++;v[p]=val;
    }
    int query(long long val)
    {
        int p = val%M;
        while(f[p] && v[p]!=val)
        {
            p++;if(p==M) p = 0;
        }
               // if number of occurence is needed
use this code
        if(v[p]==val) return f[p];
               // if just the new index is needed
use this one
               // if(v[p]==val) return p;

        return 0;
    }
}M;
//less time consuming in case of less collision
struct HASH{
    static const int H = 1000003 , N = 1000008;
//H=mod value //N>=H
    int head[H],next[N],sz; //sz=size of a queue
//head and next for maintaining the queue
    int val[H];
    long long q[N];
    void init(){
        memset(head,-1,sizeof(head));
        sz = 0;
    }
    void insert(const long long &u) {
        int v = u % H;
        for (int i = head[v]; i != -1; i = next[i])
{ //check for exact same value
            if (q[i] == u) {
                val[i]++;
                return;
            }
        }

        //collision or no entry
        q[sz] = u;
        val[sz] = 1;
        next[sz] = head[v];
        head[v] = sz++;
    }
    int query(const long long &u) {
        int v = u % H;
        for (int i = head[v]; i != -1; i = next[i])
{
            if (q[i] == u) {
                return val[i];
            }
        }
        return 0;
    }
}H;
```

/* Euler Tour And Circuit */

```
void dfs(int u) { int
    i,v;

    for(i=0;i<SZ(adj[u]);i++) { v =
        adj[u][i]; if(v!=-1) {
            adj[u][i] = -1; dfs(v);
        }
    }
    order.pb(u);
}

bool possible() {
    int i,start,end,c=0;

    start = end = -1;
    for(i=0;i<nodes;i++) {
        if(indeg[i]==outdeg[i]) continue; else
        if(indeg[i]-outdeg[i]==1) {
            end = i; c++;
        }
        else if(outdeg[i]-indeg[i]==1) { start =
            i; c++;
        }
        else return false;
    }
    if(c>2) return false;

    if(start == -1) { //circuit probably
        for(i=0;i<nodes;i++)
            if(outdeg[i]) { start =
                i;break;
            }
    }
    order.clear(); //Here Finding the
    dfs(start);//Eulear tour orderings.
    Reverse(order); if(SZ(order)!=nodes) return
    false; //could be disconnected.
    return true;
}
```

/* Closest Pair of Points */

```
typedef pair<int,int>pii;
struct P{
    double x,y,z;
    P(double xt=0,double yt=0,int zt=0)
    { x=xt,y=yt,z=zt; }
};

struct Comparator {
```

```
    bool operator ()(const P &a,const P &b)
    const{ if(a.y!=b.y) return a.y<b.y;
        return a.x<b.x; }
};

const int S = 100000;
P p[S];

bool com(P a,P b) {
    return(a.x!=b.x)?(a.x<b.x):(a.y<b.y); }

double SD(P a,P b){
    return sqr(a.x-b.x)+sqr(a.y-b.y); }

pii ClosestPair(P p[],int n)
{
    /// Return the index's of closest points.
   int left,right,ci,cj,i;
   double dis,m;
   set<P,Comparator>st;
   P tmp;
   __typeof(st.begin()) itl,ith;
   sort(p,p+n,com);
   for(i=0;i<n;i++) p[i].z = i;
   ci=p[0].z; cj=p[1].z;
   m = SD(p[0],p[1]);
   st.insert(p[0]); st.insert(p[1]);
   left=0; right=2;

   while(right<n)
   {
     while(left<right&&sqr(p[left].x-p[right].x)>=m)
     {
        st.erase(p[left]); left++;
     }
     dis=sqrt(m)+ERR;
     itl = st.lower_bound(P(p[right].x,
                            p[right].y-dis));
     ith = st.upper_bound(P(p[right].x,
                            p[right].y+dis));
     while(itl!=ith)
     {
         dis = SD(*itl,p[right]);
         if(dis<m)
         {
             m=dis; ci=itl->z;
             cj = p[right].z;
         }
         itl++;
     }
     st.insert(p[right]); right++;
   }
   return pii(ci,cj);
}
```

Trigonometric identities

$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$ $\quad \cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$

$\sin(\alpha-\beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta$ $\quad \cos(\alpha-\beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$

$\tan(\alpha+\beta) = \frac{\tan\alpha+\tan\beta}{1-\tan\alpha\tan\beta}$ $\quad \sin 2\alpha = 2\sin\alpha\cos\alpha,\ \cos 2\alpha = \cos^2\alpha - \sin^2\alpha$

$\cos^2\alpha = \frac{1}{2}(1+\cos 2\alpha)$ $\quad \sin^2\alpha = \frac{1}{2}(1-\cos 2\alpha)$

$\sin\alpha+\sin\beta = 2\sin\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$ $\quad \cos\alpha+\cos\beta = 2\cos\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$

$\sin\alpha-\sin\beta = 2\sin\frac{\alpha-\beta}{2}\cos\frac{\alpha+\beta}{2}$ $\quad \cos\alpha-\cos\beta = -2\sin\frac{\alpha+\beta}{2}\sin\frac{\alpha-\beta}{2}$

$\tan\alpha+\tan\beta = \frac{\sin(\alpha+\beta)}{\cos\alpha\cos\beta}$ $\quad \cot\alpha+\cot\beta = \frac{\sin(\alpha+\beta)}{\sin\alpha\sin\beta}$

$\sin\alpha\sin\beta = \frac{1}{2}[\cos(\alpha-\beta)-\cos(\alpha+\beta)]$ $\quad \cos\alpha\cos\beta = \frac{1}{2}[\cos(\alpha-\beta)+\cos(\alpha+\beta)]$

$\sin\alpha\cos\beta = \frac{1}{2}[\sin(\alpha+\beta)+\sin(\alpha-beta)]$ $\quad \sin' x = \cos x,\ \cos' x = -\sin x$

Law of sines: $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R_{out}.$ Inscribed/outscribed circles: $R_{out} = \frac{abc}{4S},\ R_{in} = \frac{2S}{a+b+c}$

Law of cosines: $c^2 = a^2+b^2-2ab\cos C.$ Heron: $\sqrt{s(s-a)(s-b)(s-c)},\ s = \frac{a+b+c}{2}.$

Law of tangents: $\frac{a+b}{a-b} = \frac{\tan[\frac{1}{2}(A+B)]}{\tan[\frac{1}{2}(A-B)]}$ $\quad \Delta$'s area, given side and adjacent angles: $\frac{c^2}{2(\cot\alpha+\cot\beta)}$