************************** Ready Code **************************

```
#include <bits/stdc++.h>
using namespace std;    typedef long long ll;    typedef long long LL;
#define oo (1<<28)    #define mp make_pair
#define FRCH(it,x) for(__typeof((x).begin()) it=(x.begin()); it!=(x).end(); ++it)
#define mem(a,b) memset(a,b,sizeof a)    #define SZ(x) (int)x.size()
#define fr(i,a,b)   for(i=a;i<=b;i++)
#define round(i,a)  i = ( a < 0 ) ? a - 0.5 : a + 0.5;
#define __ std::ios_base::sync_with_stdio (false); cin.tie(0);
template<class T1> void deb(T1 e){cout<<e<<endl;}
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;
    }
```

************************ Dinic's Max Flow **********************

```
    ///V^2*E Complexity ///Base doesn't matter
    const int MAXN = 100;///total nodes
    const int MAXM = 10000;///total edges
    int N,edges;
    int last[MAXN],prev[MAXM],head[MAXM];
    int Cap[MAXM],Flow[MAXM];    int dist[MAXN];
    int nextEdge[MAXN];///used for keeping track of next edge of ith node
    queue<int> Q;
    void init(int N) { edges=0; memset(last,-1,sizeof(int)*N); }
    inline void addEdge(int u,int v,int cap,int flow){
        head[edges]=v;prev[edges]=last[u];Cap[edges]=cap;Flow[edges]=flow;
        last[u]=edges++;
        head[edges]=u;prev[edges]=last[v];Cap[edges]=0;Flow[edges]=0;
        last[v]=edges++;
}
inline bool dinicBfs(int S,int E,int N){
    int from=S,to,cap,flow;
    memset(dist,0,sizeof(int)*N);    dist[from]=1;
    while(!Q.empty()) Q.pop();    Q.push(from);
    while(!Q.empty()){
        from=Q.front();Q.pop();
        for(int e=last[from];e>=0;e=prev[e]){
            to=head[e];           cap=Cap[e];           flow=Flow[e];
            if(!dist[to] && cap>flow) {dist[to]=dist[from]+1; Q.push(to); }
        }
    }
    return (dist[E]!=0);
}
inline int dfs(int from,int minEdge,int E){
    if(!minEdge) return 0;    if(from==E) return minEdge;
    int to,e,cap,flow,ret;
    for( ; nextEdge[from]>=0;nextEdge[from]=prev[e]){
        e=nextEdge[from]; to=head[e]; cap=Cap[e]; flow=Flow[e];
        if( dist[to] != dist[from]+1) continue;
        ret = dfs(to,min(minEdge,cap-flow),E);
        if(ret) { Flow[e]+=ret;      Flow[e^1]-=ret;     return ret;       }
    }
    return 0;
}
int dinicUpdate(int S,int E){
    int flow=0;
    while(int minEdge = dfs(S,INF,E)) {
        if(minEdge==0) break; flow+=minEdge;
    }
        return flow;
    }
    int maxFlow(int S,int E,int N){
        int totFlow=0;
        while(dinicBfs(S,E,N)) {
            for(int i=0;i<=N;i++) nextEdge[i]=last[i];/// update last edge of ith node
            totFlow+=dinicUpdate(S,E);
        }
        return totFlow;
    }
```

*********************** Hopcroft BPM ************************

```
    //Esqrt(V) Complexity,0 Based,Edge from set a to set b
    const int MAXN1 = 50010; //nodes in set a
    const int MAXN2 = 50010; //nodes in set b
    const int MAXM = 150010; //number of edges
    int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
    int matching[MAXN2], dist[MAXN1], Q[MAXN1];
    bool used[MAXN1], vis[MAXN1]; //vis is cleared in each dfs
    // n1 = number of nodes in set a, n2 = number of nodes in set b
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],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 augmentPath() {
    bfs();    fill(vis, vis + n1, false);    int f = 0;
    for (int u = 0; u < n1; ++u) if (!used[u] && dfs(u)) ++f;
    return f;
}
int maxMatching() {
    fill(used, used + n1, false);    fill(matching, matching + n2, -1);    int res=0,f;
    while(f= augmentPath()) res+=f;     return res;
}
```

*********************** **Hungarian BPM** ***********************

```
//return minimum cost (multiply -1 in each entry for maximum cost)
//1 based (0 is used for algorithm),Complexity O(n^2*m) or O(n^3)
int arr[rows][clms]; //main matrix
int  u[rows], v[clms]; //used for labeling
int p[clms] , way[clms]; //p = match , way = the augmenting path
//n = number of rows,m = number of columns,n<=m
int hungarian(int n,int m) {
    mem(p,0); mem(u,0);mem(v,0);
    for ( int i = 1 ; i <= n ; ++ i ) {
        p [ 0 ] = i ;     int j0 = 0 ;
        vector < int > minv ( m + 1 , INF ) ;  vector < bool > used ( m + 1 , false ) ;
         do { //works like bfs
            used [ j0 ] = true ;   int i0 = p [ j0 ] ,  delta = INF,  j1 ;
            for ( int j = 1 ; j <= m ; ++ j )
                if ( ! used [ j ] ) {
                    int cur = arr [ i0 ] [ j ] - u [ i0 ] - v [ j ] ;
                    if ( cur < minv [ j ] ) minv [ j ] = cur,  way [ j ] = j0 ;
                        if ( minv [ j ] < delta ) delta = minv [ j ] ,  j1 = j ;
                }
                //matrix doesn't change here
                for ( int j = 0 ; j <= m ; ++ j )
                    if ( used [ j ] ) u [ p [ j ] ] += delta,  v [ j ] -= delta ;
                    else minv [ j ] -= delta ;
                    j0 = j1 ;
            } while ( p [ j0 ] != 0 ) ;///End of do while 1
            do {
                    int j1 = way [ j0 ] ;   p [ j0 ] = p [ j1 ] ;    j0 = j1 ;
            } while ( j0 ) ; ///End of do while 2
        }
        return -v[0]; //minimum cost is stored here
    }
```

********************* **Min Cost Max Flow** *********************

```
///V*E^2 Complexity,Base doesn't matter
const int MAXN = 350;///total nodes
const int MAXM = 120200;///total edges
int edges;
int last[MAXN], prev[MAXM],head[MAXM],Cap[MAXM],Cost[MAXM];
int Flow[MAXN],edgeNo[MAXN],dist[MAXN],par[MAXN];
bool visited[MAXN];
void init(int N) {    memset(last,-1,sizeof(int)*N);        edges=0;     }
void addEdge(int u,int v,int cap,int cost){
    head[edges]=v;prev[edges]=last[u];Cap[edges]=cap;Cost[edges]=cost;
    last[u]=edges++;
    head[edges]=u; prev[edges]=last[v]; Cap[edges]=0; Cost[edges]=-cost;
    last[v]=edges++;
}
queue<int> Q;
pair<int,int> SPFA(int S,int E,int N){
    int totFlow=0,totCost=0;   int u,v,cap,cost;
    while(!Q.empty()) Q.pop();
        while(true){
        Flow[S]=oo;        memset(dist,oo,sizeof(int)*N);
        dist[S]=0;         memset(visited,false,sizeof(bool)*N);
        visited[S]=1;      Q.push(S);
        while(!Q.empty()){
            u=Q.front();Q.pop();         visited[u]=false;
            for(int e=last[u];e>=0;e=prev[e]){
                v=head[e]; cap=Cap[e]; cost=Cost[e];
                if(cap&&dist[v]>dist[u]+cost){
                    dist[v]=dist[u]+cost;    Flow[v]=min(Flow[u],cap);
                    edgeNo[v]=e;             par[v]=u;
                    if(!visited[v]) {      visited[v]=true; Q.push(v);       }
                }
            }
        }
        if(dist[E]==oo) break;  totCost+=dist[E]*Flow[E];  totFlow+=Flow[E];
        for(int i=E ; i != S; i = par[i] )  {
            Cap[edgeNo[i]]-=Flow[E];           Cap[edgeNo[i]^1]+=Flow[E];
            }
        }
        return mp(totFlow,totCost);
    }
```

    *************************** **Blossom** *************************

```
    //0 based,complexity O(VE)
    const int MAXN = 505; // number of elements.
    vector<int> g[MAXN];
    int match[MAXN];     int p[MAXN]; //array of ancestors.
    int base[MAXN]; //Node numbering after compression
    int q[MAXN]; /*Queue*/    bool used[MAXN], blossm[MAXN];
    void initialize(int n) {
        for(int i=0;i<n;i++) g[i].clear();
        mem(blossm,false);
    }
    int lca (int a, int b){
        bool used[MAXN] = { 0 };
      // From the node a climb up to the roots, marking all even vertices
        for (;;){
            a = base[a]; used[a] = true;
            if (match[a] == -1) break; /* Got the root */        a = p[match[a]];
        }
    // Climb from node b,until we find the marked vertex
        for (;;){  b = base[b];  if (used[b]) return b; b = p[match[b]]; }
    }
    void mark_path (int v, int b, int children) {
        while (base[v] != b){
            blossm[base[v]]=blossm[base[match[v]]]=true;
            p[v] = children;     children = match[v];      v = p[match[v]];
    }
}
int find_path (int root,int n){
    mem(used,0); mem(p,-1);
    for (int i=0; i<n; ++i) base[i] = i;
    used[root] = true;    int qh=0, qt=0;    q[qt++] = root;
    while (qh < qt){
        int v = q[qh++];
        for (int i=0; i<g[v].size(); ++i){
            int to = g[v][i];
            if (base[v] == base[to] || match[v] == to) continue;
            if (to == root || match[to] != -1 && p[match[to]] != -1){
                int curbase = lca (v, to);               mem(blossm,0);
                mark_path (v, curbase, to);          mark_path (to, curbase, v);
                for (int i=0; i<n; ++i)
                    if (blossm[base[i]])  {
                        base[i] = curbase;     if (!used[i]){ used[i] = true; q[qt++] = i; }
                    }
            }
            else if (p[to] == -1){
                p[to] = v;               if (match[to] == -1) return to;
                to = match[to];       used[to] = true;              q[qt++] = to;
            }
        }
    }
    return -1;
}
int graph_match(int n){
    int ret = 0;    mem(match,-1);
    for (int i=0; i<n; ++i)
        if (match[i] == -1){
                int v = find_path (i,n);                if(v!=-1) ret++;
                while (v != -1){
                    int pv = p[v], ppv = match[pv];
                    match[v] = pv, match[pv] = v;          v = ppv;
                }
            }
        return ret;
    }
    int main(){
    int i,j,n,m;        scanf("%d %d",&n,&m);        initialize(n);
    while(m--){  scanf("%d %d",&i,&j);     i--,j--;   g[i].psb(j);    g[j].psb(i);  }
    int ans = graph_match(n);  printf("%d\n",ans*2);
    for(i=0; i<n; i++)
    if(match[i]>-1) {  printf("%d %d\n",i+1,match[i]+1); match[match[i]] = -1;  }
        return 0;
    }
   ******************** 2-SAT and TarjanSCC********************
    //0 based
    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]; 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]);       }
    }
    if(low[u]==tim[u]){
        do{
             tem=S.top();S.pop();group_id[tem]=components; col[tem]=2;
```

```
        } 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;
}
//double nodes needed normally
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])           ans[i/2]=true;
        else                                    ans[i/2]=false;
        }
        return true;
    }
    ******************** BCC and Bridge *********************
    //1 Based,no problem in multiple edge and self loop for BCC
    //in multiple edge bridge won't work
    int tim[lim],low[lim];    int timer;    VI adj[lim]; //only adj should be cleared
    //******** used for BCC ********
    stack<pii>S;    pii ed[2*lim];//because one edge can be part of two BCC
    void calc_bcc(int u, int v){
        int i, j, uu, vv, cur;        pii now; int tot=0;
        while(!S.empty()){
            now = S.top(); S.pop();           uu = now.first, vv = now.second;
            ed[tot++] = MP(uu, vv);
            if(u==uu && v==vv) break;           if(u==vv && v==uu) break;
        }
        if(tot<=1) return;
        //doing according to problem
        return;
    }
    //******** used for BCC ********
    //******** used for bridge ********
    struct edge{        int u;       int v;      };
    vector<edge> bridge;//the ans(should be cleared)
    //******** used for bridge ********
    void bcc(int u,int par)// par=-1 dhore call dite hobe(root er parent nai)
    {
        tim[u] = low[u] = ++timer;
        for(int i = 0 ; i<SZ(adj[u]) ; i++) {
        int v = adj[u][i];       if(v==par) continue;
        if(tim[v]==0){
            S.push(MP(u, v));           bcc(v,u);           low[u] = min(low[u],low[v]);
            if(low[v]>=tim[u]) calc_bcc(u, v);
            //******** used for bridge ********
            if(low[v]>tim[u]) {  edge tem;   tem.u=u;tem.v=v;   bridge.pb(tem);    }
            //******** used for bridge ********
        }
        else if(tim[v] < tim[u]){  low[u] = min(low[u],tim[v]); S.push(MP(u, v));
}
    }
    return;
}
void BCC(int n){  timer=0;  mem(tim,0);    int i;    fr(i,1,n)    if(!tim[i]) bcc(i,-1);
}
********************** Articulation Point **********************
//no problem in multiple edge
int tim[lim],low[lim];  bool flag[lim];  int timer;
VI adj[lim];//only adj should be cleared
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;
            }
            else low[u] = min(low[u],tim[v]);
        }
        if(par==-1 && subtree>1) flag[u] = true; //for root
    }
    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);
    }
```

```
******************** Dijkstra PQ ***********************
    struct pq{
                int cost,node;
                bool operator<(const pq &b)const{
                        return cost>b.cost;    // Min Priority Queue(b is curret)
                }
    };
    ********************  MST Union  **********************
    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]++;
    }
    ********************HLD & LCA & BIT *****************
    //Works in every based(0/1) index,compexity n(logn)^2
    #define MAXN 32000
    #define step 17    // step=log(n)
    //normally call dfs(1,1),/normally call hld(1,1,1).used for LCA
    int n,parent[MAXN][step+1],start[MAXN],finish[MAXN],T;// T = time
    VI adj[MAXN];
    //used for HLD
    int child[MAXN],nodeId,firstNode[MAXN],currentIndex[MAXN];
   //used for segment tree,1 based index is used here
   int tree[MAXN];
   int MaxVal;//always should be set(size of the set len)
   //cumulitive sum
   int queryBIT(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 updateBIT(int idx ,int val)  {
     if(idx<=0) return;
     while (idx <= MaxVal){   tree[idx] += val; idx += (idx & -idx);    }
    }
    void initialize(int n) {
    for(int i=0;i<=n;i++) adj[i].clear();   T=0; nodeId=0; MaxVal = n;
    }
int 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];
```

```
child[u]=1;
fr(i,0,SZ(adj[u])-1){   v=adj[u][i];  if(v==p) continue;  child[u]+=dfs(v,u);    }
   finish[u]=T++;        return child[u];
}
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;
   }
   int hld(int u,int p,int lastNode)    {
      currentIndex[u] = (++nodeId);        firstNode[u] = lastNode;
      int ind = -1,ma = -1,i;
      fr(i,0,SZ(adj[u])-1)        {
         int v=adj[u][i];                              if(v==p) continue;
         if(ma>=child[v]) continue;            ma = child[v];ind = i;
      }
      if(ind >= 0) hld( adj[u][ind],u,lastNode ); //same segment
      fr(i,0,SZ(adj[u])-1)  {
         int v=adj[u][i];           if(v==p) continue;
         if(i==ind) continue;           hld(v,u,v);
   }
}
int arr[MAXN];
//u should be upper node (close to root)
int query_up(int u, int v){
   int ans = 0;
   while(true){
      int y = currentIndex[v];       int x = currentIndex[firstNode[v]];
      int z = currentIndex[u];
      if(z>=x && z<=y) {
         x=z;           ans+=queryBIT(y)-queryBIT(x-1);           break;
      }
      ans+=queryBIT(y)-queryBIT(x-1);       v = parent[firstNode[v]][0];
   }
   return ans;
}
int query(int u,int v) {
 int lcanode=lca_query(u,v);
 return query_up(lcanode,u)+query_up(lcanode,v)-query_up(lcanode,lcanode);
}
void update(int u,int x) {
   int curval = queryBIT(currentIndex[u])-queryBIT(currentIndex[u]-1);
   x-=curval;    updateBIT(currentIndex[u],x);
}
************************** Suffix Array************************
#define REP(i,s,t) for(int i=(s);i<(t);i++)
#define FILL(x,v) memset(x,v,sizeof(x))  #define MAXN 100009
int N; int cnt[MAXN];  char str[MAXN]; int phi[MAXN], lcp[MAXN];
int RA[MAXN], SA[MAXN], rSA[MAXN], tmpRA[MAXN], tmpSA[MAXN];
void buildLCP(){
    int L=0;      phi[SA[0]] = -1;
       for(int i=1;i<N;i++) phi[SA[i]] = SA[i-1];
       for(int i=0;i<N;i++) {    if(phi[i]==-1) { lcp[rSA[i]]=0; continue; }
           while(str[i+L]==str[phi[i]+L]) L++;    lcp[rSA[i]] = L;        if(L) L--;
       }
   }
void csort(int k){
    int cub = max(N, 128);          FILL(cnt, 0);
    for(int i=0;i<N;i++) cnt[i+k<N?RA[i+k]:0]++;
    REP(i,1,cub) cnt[i] += cnt[i-1];     for(int i=cub-1;i>=1;i--) cnt[i] = cnt[i-1];
    cnt[0] = 0;
    for(int i=0;i<N;i++) tmpSA[ cnt[SA[i]+k<N?RA[SA[i]+k]:0]++ ] = SA[i];
    for(int i=0;i<N;i++) SA[i] = tmpSA[i];
}
void buildSA(){
   REP(i,0,N) {              RA[i] = str[i];             SA[i] = i;         }
   int k = 1;
   while(k<N){
     csort(k);       csort(0);          int r = 0;          tmpRA[SA[0]] = 0;
         REP(i,1,N) {
            if(RA[SA[i]]!=RA[SA[i-1]] || RA[SA[i]+k]!=RA[SA[i-1]+k]) r++;
            tmpRA[SA[i]] = r;
           }
```

```
            REP(i,0,N) RA[i] = tmpRA[i];    if(RA[SA[N-1]]==N-1) break;
            k <<= 1;
        }
        REP(i,0,N)   rSA[SA[i]] = i;
    }
```

************************* Suffix Automata *******************

```
//Preprocess complexity nlogk (k=number of child)
struct state {
        int depth, link ;           map < char , int > next ;
        void initialize() {next.clear(); link=-1; depth=0;}
};
const int MAXLEN = 100010; state st [ MAXLEN * 2 ] ;  int sz, last, maxhei;
/* when topological sort is needed (insert frequeny)
int height[MAXLEN],top[2*MAXLEN]; //for topological sort
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();    sz = last = 0 ;  ++ sz ;
}
//intially clone frequency is 0 and regular node frequency 1
void sam_extend ( char c ) {
    //it is needed for more than 1 string
    if(st[last].next[c]) {
    int q = st [ last ] . next [ c ],p = last;
    if ( st [ p ] . depth + 1 == st [ q ] . depth ) last = 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 = clone ; last = clone;
    }
           return;
}
int cur = sz ++,p ;   st[cur].initialize();   st [ cur ] . depth = st [ last ] . depth + 1 ;
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 ;
  }
  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 ) ;
          }
  }
```

******************* Manacher Algorithm ***********************

```
//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=1, r=2,ma=1;
for(int p2=2; p2<2*len+1; p2++) {
   int p1 = cur- (p2-cur);
   while(p1 < 0){  cur++; r = m[cur] + cur;   p1 = cur- (p2-cur);     }
   if(m[p1] < r - p2) m[p2] = m[p1];
   else{
      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;
}
```

```
*************************** Aho Corasick *********************
#define wnum  510  #define wsize 510 #define bacca 26     //number of child
struct state {
      int child[bacca],link; bool matched;
      void initialize() {  mem(child,0);    link=0;   matched=false;   }
  };
  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
  void Initialize() {  //normally only 1st node (initialize all for safety)
      T[0].initialize(); sz=1;
  }
 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]);
       fr(j,0,len-1) {
         ch=words[i][j]-'a'; //sometimes change here
         if(T[last].child[ch]==0) {  T[sz].initialize(); T[last].child[ch]=sz++;    }
            last=T[last].child[ch];
         }
         T[last].matched=true;
      }
      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();

         fr(i,0,bacca-1)  {
            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];
               if(T[T[v].link].matched) T[v].matched=true;          Q.push(v);
            }
            else  T[u].child[i] = T[T[u].link].child[i];
         }
      }
}
```

```
*************************  Convex Hull Trick *********************
LL M[lim],C[lim]; //y=mx+c we need only m(slope) and c(constant)
bool bad(int l1,int l2,int l3)  {
            return (C[l3]-C[l1])*(M[l1]-M[l2])<=(C[l2]-C[l1])*(M[l1]-M[l3]);
//query x values is non-decreasing (reverse(> sign) for vice verse)
}
//Adding should be done serially. If we want minimum y coordinate(value) then
//maximum valued m should be inserted first and if we want maximum y
//coordinate(value) then minimum valued m should be inserted first
void add(long long m,long long c,int &last) {
            M[last]=m;   C[last++]=c;
            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
//This can only be applied if the query of vertical line(x) is already sorted
long long query(long long x,int &pointer,int last) {
            if (pointer>=last)   pointer=last-1;
            //non-decreasing
            while (pointer<last-1 &&
            M[pointer+1]*x+C[pointer+1]<=M[pointer]*x+C[pointer])  // Min
            //Value wanted... (reverse(> sign) for max value)
                        pointer++;
            return M[pointer]*x+C[pointer];
}
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... (reverse(> sign) for max value)
      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... (reverse(> sign) for max
value)
      return M[mid]*x+C[mid];
   }
   ************************* Matrix Expo **********************
   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;
         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;
      }
   }
   ******************* Divide and Conquer*******************
   LL dp[810][8100];  LL arr[8100];  LL cum[8100];
   //complexity nlogn here
void divideAndConquer(int k,int l,int r,int optl,int optr)  {
         if(l>r) return;   LL m = (l+r)/2;          int best;  LL mi=INF;
         for(LL i=optl;i<=min(m-1,(LL)optr);i++) {
         if(mi>dp[k-1][i]+(m-i)*(cum[m]-cum[i])) {
         mi=dp[k-1][i]+(m-i)*(cum[m]-cum[i]); best=i;
      }
   }
   dp[k][m]=mi;    divideAndConquer(k, l,m-1, optl, best);
  divideAndConquer(k, m+1, r, best, optr);
}
```

************************ **DP Notes** *************************

Divide & Conquer : $dp[i][j] = min_{k<j} \{dp[i-1][k] + C[k][j]\}$

Sufficient Condition: $A[i][j] \le A[i][j + 1]$, or quadrilateral inequality

Knuth Optimization : $dp[i][j] = min_{i<k<j} \{dp[i][k] + dp[k][j] + C[i][j]\}$

Sufficient Condition: $A[i, j - 1] \le A[i, j] \le A[i + 1, j]$

```
************************ Fenwick Tree *********************
//for range query and range update in BIT (0 based)
int dataMul[lim]; int dataAdd[lim];
void internalUpdate(int at, int mul, int add) {
   while (at < lim) { dataMul[at] += mul; dataAdd[at] += add;   at |= (at + 1);    }
}
void update(int left, int right, int by) {
   if(left>right) return;    internalUpdate(left, by, -by * (left - 1));
   internalUpdate(right, -by, by * right);
}
int query(int at) {
   int mul = 0;    int add = 0;   int start = at;
   while (at >= 0) {
      mul += dataMul[at];  add += dataAdd[at];        at = (at & (at + 1)) - 1;
   }
   return mul * start + add;
}
************************* Implicit Treap ********************
```

```
   //pass root as starting node
   typedef struct item *pitem;
   struct item {
              int value, prior,cnt; bool revv;
              pitem  l,  r;
              item () {}
           item (int value, int prior): value (value), prior (prior), l (NULL), r
          (NULL) ,    cnt (0), revv(false) {}
           item (int value): value (value), prior (rand()), l (NULL), r (NULL), cnt
          (0), revv (false) {}
   };
   int cnt (pitem t) {  return t?  t->cnt:0;  }
   void upd_cnt (pitem t) {
              if (t)  t-> cnt = 1 + cnt (t-> l) + cnt (t-> r);
   }
   void pushup(pitem it) {   upd_cnt(it);   }
   void pushdown(pitem it)  {
             if (it && it->revv) {
                      it->revv = false;  swap (it->l, it->r);
                      if (it->l)  it->l->revv ^= true; if (it->r)  it->r->revv ^= true;
              }
   }
//spliting the treap into two treaps with one treap greater than key, and other
smaller than or equals key, l is the root of the treap of smaller or equals key,r is
the root of //the treap of greater key
   void split (pitem t, int key, pitem & l, pitem & r) {
             if (!t)          return void( l = r = 0 );
             pushdown (t);
             int cur_key = cnt(t->l)+1;
             if (key < cur_key) split (t->l, key ,l, t->l),  r = t;
             else split (t->r, key-cur_key,t->r, r),  l = t;
             pushup(t);
   }

   void insert (pitem & t, pitem it,int in) {
              pushdown(t);
              if (!t) t = it;
             else if (it-> prior> t-> prior) split (t, in , it-> l, it-> r), t = it;
             else if(in-cnt(t->l)-1>=0) insert (t-> r, it,in-cnt(t->l)-1);
          else insert(t-> l, it,in);     pushup(t);
}
//merging two treaps where one treap key values are greater than other treap key
//values,merge l and r making the root t l has smaller values and r has greater
//values
void merge (pitem & t, pitem l, pitem r) {
         pushdown (l);  pushdown (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;   pushup(t);
}
//erase the item of a given index in an array (if it is present)
void erase (pitem & t, int key) {
    pushdown (t);     if(!t) return;
    int cur_key = cnt(t->l);
    if (cur_key + 1 == key)   merge (t, t-> l, t-> r);
   else  erase (cur_key + 1 > key? t-> l: t-> r, cur_key + 1 > key? key: key -
         cur_key -1 );     pushup(t);
}
//t1 smaller than l, t2 between l and r inclusive, t3 greater than r
//l and r should be legal index
void erasesegment(pitem t, int l, int r) {
         if(l>r) return;   pitem t1, t2, t3;
         split (t, l-1,t1, t2); /*1 based*/split (t2, r-l+1,t2, t3);  merge (t, t1, t3);
}
//t1 smaller than l, t2 between l and r inclusive, t3 greater than r
//l and r should be legal index
void reverse (pitem t, int l, int r) {
         if(l>r) return;   pitem t1, t2, t3;
         split (t, l-1,t1, t2); /*1 based*/ split (t2, r-l+1,t2, t3); t2->revv ^= true;
         merge (t, t1, t2);  merge (t, t, t3);
}
//rotating a l to r k time to the left/right, t1 smaller than l, t2 between l and r
inclusive, t3 greater than r.l,r,k should be legal index
void rightrotate(pitem t, int l, int r,int k)  {
         reverse(t,l,r);   reverse(t,l,l+k-1); reverse(t,l+k,r);
}
void leftrotate(pitem t, int l, int r,int k)  {
         reverse(t,l,r); reverse(t,r-k+1,r); reverse(t,l,r-k);
   }
```

********************** **Explicit Treap** *********************

```
//spliting the treap into two treaps with one treap greater than key, and other
//smaller than or equals key,l is the root of the treap of smaller or equals key,r is
//the root of the treap of greater key
   void split (pitem t, int key, pitem & l, pitem & r) {
             if (!t)           l = r = NULL;
             else if (key <t-> key)          split (t-> l, key, l, t-> l), r = t;
             else split (t-> r, key, t-> r, r), l = t;             upd_cnt(t);
   }
   void insert (pitem & t, pitem it) {
             if (!t)           t = it;
             else if (it-> prior> t-> prior) split (t, it-> key, it-> l, it-> r), t = it;
             else insert (it-> key <t-> key? t-> l: t-> r, it);   upd_cnt(t);
   }
   void erase (pitem & t, int key) {
       if(!t) return;    if (t-> key == key)           merge (t, t-> l, t-> r);
       else          erase (key <t-> key? t-> l: t-> r, key);   upd_cnt(t);
   }
   ***************** Rectangle Union(Single) *******************
   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 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])          return ans[node] = y[end]-y[st];
                     else  {
                              if(end-st==1)          return ans[node]=0;
                              else  return ans[node] = ans[2*node]+ans[2*node+1];
                     }
            }
         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]==0) 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", &n);
                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=0; //any value
                for(i=0;i<cnt;i++)  {
                         x2 = pnt[i].x;   now = x2-x1;   sum+=now*ans[1];
                         update(1, 0, n-1, pnt[i].y1, pnt[i].y2, pnt[i].val);
                         x1 = x2;
                }
                csprnt; printf("%lld\n", sum);
   }
   return 0;
}
******************** Rectangle Union(General) *******************
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;
                         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];
}
```

**************Center of Tree and Longest Path in Tree **************

```
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;    }
   for(int 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 ret=0;
   for(int 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);
}
```

************* K-D Tree & KNN & Closest Pair of Points **********

```
//dimension 0 based all distance are euclidian distance
#define dimension 3
struct co {   LL x[dimension];  };
co arr[lim];
struct node  {    co now;   int left;   int right;  };
node bst[lim];  int axis;
bool comp(co p,co q)  {    return p.x[axis]<q.x[axis];  }
//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);    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)  {
   LL ret=0;
   for(int i=0;i<dimension;i++)  ret+=(p.x[i]-q.x[i])*(p.x[i]-q.x[i]);
   return ret;
}
//normally klogn 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);
 }
```

********************** Gauss (row order) ********************

```
//row order is kept and assigned the given (intended) value to first row then
//second row ans so on
long long gauss ( vector < vector < long long > > a, vector < long long > & ans,
long long mod)  {
         int n = ( int ) a. size ( ) ;   int m = ( int ) a [ 0 ] . size ( ) - 1 ;
         vector < int > where ( n, - 1 ) ;
         for ( int col = 0 , row = 0 ; col < m && row < n ; ++ row )  {
                  int sel = col ;
                  for ( int i = col ; i < m ; ++ i )
                  if ( abs ( a [ row ] [ i ] ) > abs ( a [ row ] [ sel ] ) )     sel = i ;
                  if ( abs ( a [ row ] [ sel ] ) == 0 )        continue ;
                  for ( int i = 0 ; i < n ; ++ i )     swap ( a [ i ] [ col ] , a [ i ] [ sel ] ) ;
                  where [ row ] = col ;
                  for ( int i = 0 ; i < n ; ++ i )
                           if ( i != row )  {
                                    long long c = a [ row ] [ col ];
                                    long long d = a [ i ][ col ];
                                    for ( int j = col ; j <= m ; ++ j ) {
                                             a [ i ] [ j ] = (c*a[i][j]-d*a[row][j])%mod ;
                                             a [i][j]=(a[i][j]+mod)%mod;
                                    }
                           }
                  ++ col ;
         }
         ans. assign ( m, 0 ) ;
         for ( int i = 0 ; i < n ; ++ i )     if ( where [ i ] != - 1 )
         ans [ where[i] ] = (a [  i  ] [ m ]* bigmod( a [ i ] [ where[i] ],mod-
         2,mod))%mod ;
         for ( int i = 0 ; i < n ; ++ i ) {
                  long long sum = 0 ;
                  for ( int j = 0 ; j < m ; ++ j ){
                           sum += (ans [ j ] * a [ i ] [ j ])%mod ; sum %= mod;
                  }
                  if ( abs ( sum - a [ i ] [ m ] ) != 0 )      return 0 ;
         }
   long long totalans=1;
         for ( int i = 0 ; i < m ; ++ i )
                  if ( where [ i ]== - 1 ) //use mod if necessary
                           totalans=(totalans* mod)%1000000007;
   return totalans ;
}
```

************************* Gauss **************************

```
int gauss ( vector < vector < double > > a, vector < double > & ans )
{
         int n = ( int ) a. size ( ) ;         int m = ( int ) a [ 0 ] . size ( ) - 1 ;
         vector < int > where ( m, - 1 ) ;
         for ( int col = 0 , row = 0 ; col < m && row < n ; ++ col ) {
            int sel = row ;
            for ( int i = row ; i < n ; ++ i )
            if ( abs ( a [ i ] [ col ] ) > abs ( a [ sel ] [ col ] ) )    sel = i ;
            if ( abs ( a [ sel ] [ col ] ) < ERR )  continue ;
            for ( int i = col ; i <= m ; ++ i )swap ( a [ sel ] [ i ] , a [ row ] [ i ] ) ;
            where [ col ] = row ;
             for ( int i = 0 ; i < n ; ++ i )
                  if ( i != row ) {
                           double c = a [ i ] [ col ] / a [ row ] [ col ] ;
                           for ( int j = col ; j <= m ; ++ j )
                                    a [ i ] [ j ] -= a [ row ] [ j ] * c ;
                  }
                  ++ row ;
         }
ans. assign ( m, 0 ) ;
for ( int i = 0 ; i < m ; ++ i )
if ( where [ i ] != - 1 ) ans [ i ] = a [ where [ i ] ] [ m ] / a [ where [ i ] ] [ i ] ;
for ( int i = 0 ; i < n ; ++ i ) {
         double sum = 0 ;
         for ( int j = 0 ; j < m ; ++ j )    sum += ans [ j ] * a [ i ] [ j ] ;
         if ( abs ( sum - a [ i ] [ m ] ) > ERR )  return 0 ;
}
         for ( int i = 0 ; i < m ; ++ i )    if ( where [ i ]== - 1 ) return INF;
         return 1 ; //unique solution
}
```

*************************Gauss Mod 2********************

```
//complexity (n^3)/64
long long gauss ( vector < vector < long long > > a, vector < long long > &
ans,int sz)   {  //sz=number of variables+1
```

```
        int n = ( int ) a. size ( ) ;          int m = sz-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 (((a[i][col/64])&(1LL<<(col%64)))  >
((a[sel][col/64])&(1LL<<(col%64))))                          sel = i ;
       if ( ((a[sel][col/64])&(1LL<<(col%64)))==0 ) continue ;
      for ( int i = col/64 ; i <= m/64 ; ++ i ) swap ( a [ sel ] [ i ] , a [ row ] [ i ] ) ;
                where [ col ] = row ;
                for ( int i = 0 ; i < n ; ++ i )
                        if ( i != row ) {
                                if((a[i][col/64])&(1LL<<(col%64))) //if set
                                for ( int j = col/64 ; j <= m/64 ; ++ j ){
                                        a [ i ][ j ] ^= a[row][j];
                                }
                        }
                ++ row ;
        }
   ans. assign ( m, 0 ) ;
        for ( int i = 0 ; i < m ; ++ i )
                if ( where [ i ] != - 1 ) {
                        ans [ i ] =  (a [ where [ i ] ] [ m/64 ]& (1LL<<(m%64)));
                   if(ans[i]) ans[i]=1;
                }
        for ( int i = 0 ; i < n ; ++ i ) {
                bool sum = 0 ;
                for ( int j = 0 ; j < m ; ++ j ) {
                   int gun=(a [ i ] [ j/64 ]& (1LL<<(j%64)));
                   if(gun) gun=1;        sum += ans [ j ] *gun;
                }
                 if( sum!= (bool)(a[i][m/64]&(1LL<<(m%64)) ))
        return 0;
        }
        long long totalans=1;
        for ( int i = 0 ; i <= m ; ++ i )
                if ( where [ i ]== - 1 ) //use mod if necessary
                        totalans=(totalans* 2)%1000000007;
        return totalans;
}
```

```
********************Determinant(modular)*********************
int det (vector < vector < long long > > a,int mod)    //determinant of a square
matrix
{
   int n=( int ) a. size ();    int i, j, k, ans = 1, x, y, flg = 1;
   for (i = 0; i < n; i++)    {
      if (a[i][i] == 0)      {
         for (j = i+1; j < n; j++)   if (a[j][i])   break;
         if (j == n) return -1;    flg = !flg;
         for (k = i; k < n; k++)    swap (a[i][k], a[j][k]);
      }
      ans = ans * a[i][i] % mod;   Egcd (a[i][i], mod, x, y); //inverse modulo
      x = (x%mod + mod) % mod;
      for (k = i+1; k < n; k++)   a[i][k] = a[i][k] * x % mod;
      for (j = i+1; j < n; j++)
           if (a[j][i] != 0) for (k = i+1; k < n; k++)
            a[j][k] = ((a[j][k] - a[i][k]*a[j][i])%mod + mod) % mod;
   }
   if (flg) return ans;    return mod-ans;
}
**********************Determinant**********************
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;  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;
 }
```

****************************FFT** ***********************

```
// memory complexity 12n
// i-th index mean coefficient 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 ;
    n <<= 1 ;  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 ) ;   res. resize ( n ) ;
    for ( size_t i = 0 ; i < n ; ++ i )      res [ i ] = int ( fa [ i ] . real ( ) + 0.5 ) ;
}
```

********************** **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*/       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){
   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);
}
```

*****************Chinese Remainder Theorem*******************

```
//a=x0+x1*p0+x2*p0*p1+x3*p0*p1*p2+....+x(k-1)*p0*p1*p2**..p(k-2) (mod
p0*p1*p2*...p(k-1))
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 ] ;        if ( x [ i ] < 0 )  x [ i ] += p [ i ] ;
            }
      }
}
```

***********************Burnside Lemma*********************

```
//most of the change were done here
LL lemmaFunction(int n,int d,int k,int m)  {
    LL ans=relPrime(n);    ans*=bigmod(k,d,m);    ans%=m;    return ans;
}
//burnside lemma(from emaxx),n and mod should be relative prime
LL burnside(int n,int k,int m) { //n=group size, k=number of color
    int i;    LL ans=0;
    for(i=1;i*i<n;i++)    if(n%i==0)    {
            ans=(ans+lemmaFunction(n/i,i,k,m))%m;
            ans=(ans+lemmaFunction(i,n/i,k,m))%m;
    }
    if(n==i*i) ans=(ans+lemmaFunction(i,i,k,m))%m; //for ignoring double count
    ans=(ans*bigmod(n,m-2,m))%m;    return ans;
}
```

**********************Number Theory Notes*****************

1.Summation of relative Prime=(n*phi(n))/2.

2.Summation of divisors sigma(n) = multiplication of (p^(x+1)-1)/(p-1) for all p where x is the power of p.

3.mobious function mu(n)={0, if n has one or more repeated prime (not square free) factors;   1   if n=1;   (-1)^k   if n is a product of k distinct primes;} Counted using seive with initialize all with 1.

4. Lucas Theorem: Find C(n,k)%p where p is prime and n and k are converted into base p numbers and now inidividually multiplying the digit combination.

5. A ^ x = A ^ (x% Phi (C) + Phi (C)) (mod C) (X>=Phi(C))

6. Catalan Number : $C_0 = 1$ & $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$ $for\ n \geq 0$;

7. Stirling Num Frst Kind: $S(m,n) = S(m-1,n-1) - (m-1)S(m-1,n)$

8. Stirling Num Scnd Kind: $S(m,n) = S(m-1,n-1) + nS(m-1,n)$

************************* Inverse Modulo *********************

```
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;    }
}
void inverseofall(int m)  {
    int r[m];    r [ 1 ] = 1 ;
  for ( int i = 2 ; i < m ; ++ i )
    r [ i ] = ( m - ( m / i ) * r [ m % i ] % m ) % m ; //0 means no inverse modulo
 }
```

********************* Baby Step and 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 Template*******************

```
#define vectorVar double
struct Vector{
  vectorVar  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) {
     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) {
    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();
     }
};
struct line{
   Vector p,q;
   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);
    }
   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;
    }
     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;
      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;
   }
   //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;
    }
    //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;
    //.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)); }
   vectorVar areaWithSign() { Vector p=b.substract(a); Vector q=c.substract(a);
return .5*p.cross(q); }
```

```
};
struct circle{
  Point c;//center   vectorVar r;
  double area() { return pi*r*r; }
  bool inside(Vector p) { p=p.substract(c); return (!(p.sqrLength()>r*r)); }
  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;
     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)   {
     double area=0;
     for(int 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())   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],ret[0]).areaWithSign();
           else
area+=areaOfArc(poly[i],ret[0])+triangle(c,ret[0],poly[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],ret[1]).areaWithSign()+areaOf
Arc(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) {  if(dcmp(r-cir.r)==0) return 3; return 0;  }
     if(dcmp(r+cir.r-d)<0) return -1; if(dcmp(fabs(r-cir.r)-d)>0) return 0;
     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;   return 2;
  }
  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,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])*chck>-PRE)  {
if(clipPoly[next].cross(clipPoly[i],targetPoly[nextj])*chck>-PRE)
               temtar[++temm]=targetPoly[nextj];
else
temtar[++temm]=line(clipPoly[i],clipPoly[next]).lineIntersectingPoint(line(targetPoly[j],targetPoly[nextj]));
}
else if(clipPoly[next].cross(clipPoly[i],targetPoly[nextj])*chck>-PRE) {
temtar[++temm]=line(clipPoly[i],clipPoly[next]).lineIntersectingPoint(line(targetPoly[j],targetPoly[nextj]));      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)   reverse(P+1,P+n+1);
   if(Q[2].cross(Q[1],Q[3])>0) 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) {
ans=min(ans,line(P[nexti],P[i]).shortestDistOfSegment(line(Q[nextj],Q[j])));
         i++;           j++;           cntP++;           cntQ++;
      }
      else if(chck<pi&&cntQ<m)   {
         ans=min(ans,line(Q[nextj],Q[j]).shortestDistOfSegment(P[i]));
         j++;           cntQ++;
      }
      else if(chck>pi&&cntP<n)  {
         ans=min(ans,line(P[nexti],P[i]).shortestDistOfSegment(Q[j]));
         i++;           cntP++;
      }
      else if(cntQ<m)  {
         ans=min(ans,line(Q[nextj],Q[j]).shortestDistOfSegment(P[i]));
         j++;           cntQ++;
      }
      else {
         ans=min(ans,line(P[nexti],P[i]).shortestDistOfSegment(Q[j]));
         i++;       cntP++;
      }
   }
   return ans;
}
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,Must required floating point values
bool pointInPoly(int n, Vector arr[], Vector P)  {
  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)) &&(xx < (arr[j].x-arr[i].x) * (yy-arr[i].y) /
(arr[j].y-arr[i].y) + arr[i].x) )
      c = !c;
  }
  return c;
}
```

***************** **Line Segment Intersection(integer)** *************

```
typedef struct {ll x,y;void scan(){cin>>x>>y;}}P;
P MV(P a,P b){ P r; r.x = b.x-a.x; r.y = b.y-a.y; return r;}
ll CV(P a,P b){return a.x*b.y - a.y*b.x;}
bool onsegment(P a,P b,P c){
   return ( min(a.x,b.x)<=c.x && c.x<=max(a.x,b.x) && min(a.y,b.y)<=c.y &&
c.y<=max(a.y,b.y) ) ;
}
bool segment_intersect(P p1,P p2,P p3,P p4) {
   ll d1,d2,d3,d4;
   d1 = CV(MV(p3,p4),MV(p3,p1));    d2 = CV(MV(p3,p4),MV(p3,p2));
   d3 = CV(MV(p1,p2),MV(p1,p3));    d4 = CV(MV(p1,p2),MV(p1,p4));
   if(d1*d2<0 && d3*d4<0)  return true;
   if(!d1 && onsegment(p3,p4,p1))  return true;
   if(!d2 && onsegment(p3,p4,p2))  return true;
   if(!d3 && onsegment(p1,p2,p3))  return true;
   if(!d4 && onsegment(p1,p2,p4))  return true;
   return false;
}
```

********************* **String Processing** *******************

```
struct Bigint {
   string a;  int sign; // sign = -1 for negative numbers, sign = 1 otherwise
   Bigint() {} // default constructor
   Bigint( string b ) { (*this) = b; } // constructor for string
  int size() {    return a.size();    }
  Bigint inverseSign() {     sign *= -1;        return (*this);    }
  Bigint normalize( int newSign ) { // removes leading 0, fixes sign
     for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )       a.erase(a.begin() + i);
     sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;        return (*this);
  }

  void operator = ( string b ) { // assigns a string to Bigint
     a = b[0] == '-' ? b.substr(1) : b;        reverse( a.begin(), a.end() );
     this->normalize( b[0] == '-' ? -1 : 1 );
  }
  bool operator < ( const Bigint &b ) const { // less than operator
     if( sign != b.sign ) return sign < b.sign;
     if( a.size() != b.a.size() )
        return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
     for( int i = a.size() - 1; i >= 0; i-- ) if( a[i] != b.a[i] )
        return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
     return false;
  }
bool operator == (const Bigint &b ) const{ return a == b.a && sign == b.sign;}
  Bigint operator + ( Bigint b ) { // addition operator overloading
     if( sign != b.sign ) return (*this) - b.inverseSign();       Bigint c;
     for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry; i++ ) {
        carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.size() ? b.a[i]-48 : 0);
        c.a += (carry % 10 + 48);           carry /= 10;
     }
     return c.normalize(sign);
  }
 Bigint operator - ( Bigint b ) { // subtraction operator overloading
 if(sign != b.sign) return (*this)+b.inverseSign(); int s = sign; sign = b.sign = 1;
 if( (*this) < b ) return ((b - (*this)).inverseSign()).normalize(-s);      Bigint c;
     for( int i = 0, borrow = 0; i < a.size(); i++ ) {
        borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);
        c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
        borrow = borrow >= 0 ? 0 : 1;
     }
     return c.normalize(s);
  }
  Bigint operator * ( Bigint b ) { // multiplication operator overloading
     int MAXN=a.size()+b.size()+5; int tmp[MAXN];
```

```
    memset(tmp,0,sizeof(tmp));
    for(int i=0; i<a.size(); i++)
      for(int j=0, p=i; j<b.size(); j++)  {
        tmp[p++] += (a[i]-'0')*(b.a[j]-'0');
      }
    Bigint c;
    for(int i=0; i<MAXN-1; i++)  {
      tmp[i+1] += tmp[i]/10;
       tmp[i] %= 10;    c.a.push_back(tmp[i]+'0');
    }
    return c.normalize(sign*b.sign);
  }
Bigint operator / ( Bigint b ) { // division operator overloading
    if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
    Bigint c("0"), d;        for( int j = 0; j < a.size(); j++ ) d.a += "0";
    int dSign = sign * b.sign; b.sign = 1;
    for( int i = a.size() - 1; i >= 0; i-- ) {
      c.a.insert( c.a.begin(), '0');            c = c + a.substr( i, 1 );
      while( !( c < b ) ) c = c - b, d.a[i]++;
    }
    return d.normalize(dSign);
}
Bigint operator % ( Bigint b ) { // modulo operator overloading
    if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
    Bigint c("0");        b.sign = 1;
    for( int i = a.size() - 1; i >= 0; i-- ) {
      c.a.insert( c.a.begin(), '0');            c = c + a.substr( i, 1 );
      while( !( c < b ) ) c = c - b;
    }
    return c.normalize(sign);
 }
  void print() {
    if( sign == -1 ) putchar('-');
    for( int i = a.size() - 1; i >= 0; i-- ) putchar(a[i]);
  }
};
```

********************** **Infix 2 Postfix** *********************

```
struct data {
  int coeff,val;
  data(int a, int b) ///3*x + 7,--> 3 coeff , 7 value
  {   coeff=a;val=b; }
  data() {}
};
vector<string> infix_to_postfix(string &str , int order[] )  {
  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  { // variable is considered one character
      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] );///variable
    else if(isdigit( inp[i][0] ) ) res.pb( inp[i] ); ///number
    else { ///operator
      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(); /// ( is removed , /// ) is not inserted into stack
      }
      else {
        while(true) {
          if(S.empty()) break;  tmp=S.top();
          if(  order[tmp[0]] >= order[ inp[i][0] ] ) { // pop until lower
```

```
                  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;
}
void init(int order[])  {
   mem(order,0); order['+']=1; order['-']=1; order['*']=2;  }
********************** Stable Marriage ********************
int prefer[2*lim][lim]; //preference for woman and man
bool wPrefersM1OverM(int N, int w, int m, int m1)
{
   for (int i = 0; i < N; i++)    {
      if (prefer[w][i] == m1)    return true;
      if (prefer[w][i] == m)      return false;
   }
}
void stableMarriage(int N)
{
    int wPartner[N];    bool mFree[N]; int freeCount = N;
   memset(wPartner, -1, sizeof(wPartner));
   memset(mFree, false, sizeof(mFree));
while (freeCount > 0)    {
   int m;
   for (m = 0; m < N; m++)  if (mFree[m] == false)          break;
   for (int i = 0; i < N && mFree[m] == false; i++)      {
   int w = prefer[m][i];
   if (wPartner[w-N] == -1){wPartner[w-N] = m;mFree[m] = true; freeCount--;}
   else  {
         int m1 = wPartner[w-N];
         if (wPrefersM1OverM(N, w, m, m1) == false)  {
            wPartner[w-N] = m;  mFree[m] = true;    mFree[m1] = false;
```

```
            }
         } // End of Else
      } // End of the for loop that goes to all women in m's list
   } // End of main while loop
  for (int i = 0; i < N; i++) //// Print the solution
     printf(" (%d %d)",wPartner[i]+1,i+1+N);
   printf("\n");
}
********************** Hashing O(1) ********************
//keep the collided values like a queue
//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;
```

*** FlowNotes **

Name	Description	Solution
Vertex cover	Minimum number of vertex required to cover all edges	Equals to Matching for bipartite otherwise NP complete
Edge cover	Minimum number of edge required to cover all vertices	V-matching for all graphs
Minimum Independent path (IP)	Minimum number of disjoint paths to cover all vertices	V-matching for all graphs
Minimum path cover (MPC)	Minimum number of paths to cover all vertices	Convert it MIP problem by finding trasitive closure
Clique	A complete subgraph	
Maximal clique	A clique which cannot be expanded	
Maximum clique	A maximal clique with highest number of vertices	Make a reverse graph then answer = V - vertex cover
Closure	A directed subgraph with no outgoing edges outside the graph	
Max/min closure	A closure with max/min sum of weighted nodes	For max join source with positive nodes,sink with negative
		nodes and capacities are absolute value,infinite capacity
		between existing edges. For min,source & sink Is reversed
		Ans = sum of positive nodes - min cut (For max)
		Ans = sum of negative nodes + min cut (For min)
Interval graph	If the nodes can be defined by intervals, and edges are built	Can be solved without flow in nlogn complexity
	based on interval overlap	
Perfect matching	Everynode can be matched	
Minimum Dominating set	Minimum number of vertex to cover all vertices	NP-complete
Set cover	Minimum number of set to cover all elements	NP-complete
Hitting set	Minimum number of element to cover all sets	NP-complete
Minimum weighted matching	Maximum matching with Minimum cost	Adding a extra column in self matching which is much
		greater than the rest but much smaller than infinity.
		Then apply hungarian algorithm
Minimum weighted IP	Minimum IP with minimum weighted	Convert it to Minimum weighted matching

*********************3D LIS*************************

```
//complexity n(logn)^2
const int MAXN = 300110;
struct node {    int x,y,z; } box[300111];
map <int, int> pos[MAXN];  map <int, int>::iterator it;
int m;
int cmp(const node & a, const node & b) {
   if(a.x != b.x) return a.x < b.x;
   if(a.y != b.y) return a.y > b.y;    return 0;
}
bool check(int a, int b) {
   if(pos[a].empty()) return false;
   it = pos[a].lower_bound(box[b].y);
   if(it != pos[a].begin()) {
      it--;        if(it->second < box[b].z) return true;
   }
   return false;
}
//y should be strictly increasing, and z should be strictly decreasing
void insert(int a, int b) {
   if(pos[a].empty()) {  pos[a][box[b].y] = box[b].z;        return ;    }
   it = pos[a].lower_bound(box[b].y);
```

```
    if(it == pos[a].end()) {
        it--;        if(it->second <= box[b].z) return ;
        pos[a][box[b].y] = box[b].z;        return ;
    }
    if(it->first == box[b].y) {
        if(it->second <= box[b].z) {      return ;        }
    }
    if(it != pos[a].begin()) {   if((--it)->second <= box[b].z) return ;   it++;    }
    while(it != pos[a].end() && it->second >= box[b].z) {  pos[a].erase(it++);    }
    pos[a][box[b].y] = box[b].z;
}
int main() {
       while(scanf("%d", &m)==1) {
        if(m ==0 ) break;
        for(int i = 1; i <= m; i++) {
            scanf("%d%d%d", &box[i].x, &box[i].y, &box[i].z);
        }
        for(int i = 0; i < MAXN; i++) pos[i].clear();
        int f_ans = 1;        sort(box + 1, box + 1 + n, cmp);
        int mx = 0;
        for(int i = 1; i <= m; i++) {
           if(i > 1 && box[i].x == box[i - 1].x && box[i].y == box[i - 1].y &&
box[i].z == box[i - 1].z) continue;
            int l = 1, r = mx, mid, ans = 0;
            while(l <= r) {
                mid = (l + r) / 2;
                if(check(mid, i)) {    l = mid + 1;     ans = mid;    }
               else {    r = mid - 1;   }
            }
            f_ans = max(f_ans, ans + 1);    insert(ans + 1, i);    mx = f_ans;
        }
        printf("%d\n", f_ans);
    }
    return 0;
}
```

******************~~~~**Tree Dp + Coin Change**********************

```
//ZOJ 3201 Tree of Tree
VI adj[110];  int val[110];  int dp[110][110];
void dfs(int node,int par)   {
    int i,j,k;    dp[node][1]=val[node];
    for(i=0;i<SZ(adj[node]);i++)   {
        int v=adj[node][i];   if(v==par) continue;
        dfs(v,node);
        for(j=100;j>=1;j--) //how many roots with these tree rooted here
        for(k=1;k<=j;k++)
            dp[node][j]=max(dp[node][j],dp[node][k]+dp[v][j-k]);
    }
}
```