Table of Contents

Precode 3
Graph Theory 6
1. Dinic's-Maxflow 6
2. MincostMaxFlow – SPFA 7
3. Unique Min Cut 8
4. HopcroftKarp (BPM Unweighted) 9
5. Hungarian (BPM Weighted) 10
6. Blossom Algorithm 11
7. BronKerbosch (Maximum clique) 13
8. Bellman Ford 14
9. Directed MST 14
10. Dominator Tree 17
11. Euler Path Print 18
12. Articulation Points (or Cut Vertices) 18
13. Articulation Bridge 19
14. BCC 19
15. 2-SAT/ SCC 20
16. Flow/BPM Notes 22
Data Structure 24
17. Magic STL 24
18. BIT 24
19. Build BST 25
20. Persistent Segment Tree 26
21. Segment Tree (2D) 27
22. Sparse Table 29
23. Treap 29
24. LCA 1 33
25. LCA 2 34
26. HLD 35
27. MO's on Tree 38
28. Centroid Decomposition 40
29. DSU on Tree 42
30. K-D tree+ KNN (K-nearest neighbour) 43
31. Splay Tree 45
32. Rectangle Union Without Compress 48
33. Rectangle Union Compress 48
34. Li chao (Convex Hull Trick With Segment Tree) 49
35. MO's with Update 50
36. Kadane Algorithm of Maximum Sum (2-D) 52
String Related Algorithm 54
37. Trie Tree 54
38. Trie XOR(Max/Min) 54
39. Suffix Array 56
40. Suffix Automata 58

41. Suffix Tree 59
42. KMP 60
43. Minimum Expression and ExKmp 60
44. Aho Chorasic 61
45. Manachers 62
46. Palindromic Tree 62
Dynamic Programming Optimization 64
47. Notes 64
48. CovexHull Trick 1D 64
49. Covexhull Trick 2D 66
50. Divide and Conquer 66
51. Knuth Optimization 1 67
52. Knuth Optimization 2 67
53. SOS DP 68
Matrix Related Algorithm 69
54. Guass Elimination 69
55. Guass Elimination(row order) 70
56. Guass Elimination(Modular) 71
57. Guass Elimination(Mod 2) 72
58. Determinant 73
59. Determinant (modular) 74
60. FFT(without modulo) 75
61. FFT(without modulo+complexStructure) 76
62. FFT(with modulo) 78
63. NTT 79
Number Theory 83
64. Extended Euclid (ax+by=c) 83
65. Chinese Remainder Theorem(Garner's) 84
66. Burnside Lemma 84
67. Inverse Module(E-GCD) 85
68. Baby Step-Giant Step 86
69. MillerRabin Primality Test 86
70. Möbius function 87
71. Phi Function 88
72. All pair GCD 88
73. Number Theory Notes 89
Miscellaneous 90
74. Big Integer 90
75. Stable Marriage Problem 92
76. 3D LIS 93
77. Dates 95
78. Latitude Longitude 96
79. Knights Move in infinity grid 96
80. Infix to Postfix 97
81. SStream 98
82. Maximum Disjoint Segment In an Interval 98

Geometry 101
83. Convex Hull 101
84. Line Intersection Integer 101
85. Closest Pair of Point 102
86. Geometry 2D 103
87. Geometry 2D(Integer) 110
88. Geometry 3D 116
89. Vector Standard 123
90. Circle Union 136

Precode

```
#include <cstdio>
#include <sstream>
#include <cstdlib>
#include <cctype>
#include <cmath>
#include <algorithm>
#include <set>
#include <queue>
#include <stack>
#include <list>
#include <iostream>
#include <fstream>
#include <numeric>
#include <string>
#include <vector>
#include <cstring>
#include <map>
#include <iterator>
#include<complex>
//#include <bits/stdc++.h>

using namespace std;

#define HI            printf("HI\n")
#define sf            scanf
#define pf            printf
#define sf1(a)        scanf("%d",&a)
#define sf2(a,b)      scanf("%d %d",&a,&b)
#define sf3(a,b,c)    scanf("%d %d %d",&a,&b,&c)
#define sf4(a,b,c,d)  scanf("%d %d %d %d",&a,&b,&c,&d)
#define sf1ll(a)      scanf("%lld",&a)
#define sf2ll(a,b)    scanf("%lld %lld",&a,&b)
#define sf3ll(a,b,c)  scanf("%lld %lld %lld",&a,&b,&c)
#define sf4ll(a,b,c,d)  scanf("%lld %lld %lld %lld",&a,&b,&c,&d)
#define forln(i,a,n)  for(int i=a ; i<n ; i++)
#define foren(i,a,n)  for(int i=a ; i<=n ; i++)
#define forg0(i,a,n)  for(int i=a ; i>n ; i--)
#define fore0(i,a,n)  for(int i=a ; i>=n ; i--)
#define pb            push_back
#define ppb           pop_back
#define ppf           push_front
#define popf          pop_front
#define ll            long long int
#define ui            unsigned int
```

```
#define ull           unsigned long long
#define fs            first
#define sc            second
#define clr( a, b )    memset((a),b,sizeof(a))
#define jora          pair<int, int>
#define jora_d        pair<double, double>
#define jora_ll       pair<long long int, long long int>
#define mp            make_pair
#define max3(a,b,c)    max(a,max(b,c))
#define min3(a,b,c)    min(a,min(b,c))
#define PI            acos(0.0)
#define ps            pf("PASS\n")
#define popc(a)       (__builtin_popcount(a))

template<class T1> void deb(T1 e1) {
   cout<<e1<<endl;
}
template<class T1,class T2> void deb(T1 e1,T2 e2) {
   cout<<e1<<" "<<e2<<endl;
}
template<class T1,class T2,class T3> void deb(T1 e1,T2 e2,T3 e3) {
   cout<<e1<<" "<<e2<<" "<<e3<<endl;
}
template<class T1,class T2,class T3,class T4> void deb(T1 e1,T2 e2,T3
e3,T4 e4) {
   cout<<e1<<" "<<e2<<" "<<e3<<" "<<e4<<endl;
}
template<class T1,class T2,class T3,class T4,class T5> void deb(T1 e1,T2
e2,T3 e3,T4 e4,T5 e5) {
   cout<<e1<<" "<<e2<<" "<<e3<<" "<<e4<<" "<<e5<<endl;
}
template<class T1,class T2,class T3,class T4,class T5,class T6> void deb(T1
e1,T2 e2,T3 e3,T4 e4,T5 e5,T6 e6) {
   cout<<e1<<" "<<e2<<" "<<e3<<" "<<e4<<" "<<e5<<" "<<e6<<endl;
}

/// <----------------------------  For Bitmasking  ------------------------------->
//int on( int n, int pos ){
//    return n = n|( 1<<pos );
//}
//bool check( int n, int pos ){
//    return (bool)( n&( 1<<pos ) );
//}
//int off( int n, int pos ){
//    return n = n&~( 1<<pos );
//}
//int toggle( int n, int pos ){
//    return n = n^(1<<pos);
//}
//int count_bit( int n ){
//    return __builtin_popcount( n );
//}
/// <----------------------------  End of Bitmasking  ------------------------------->


/// <---------------------------- For B - Base Number System ----------------------------
--------->
//int base;
//int pw[10];
//void calPow(int b){
//    base = b;
//    pw[0] = 1;
```

```
//    for( int i = 1; i<10; i++ ){
//        pw[i] = pw[i-1]*base;
//    }
//}
//int getV(int mask, int pos){
//    mask /= pw[pos];
//    return ( mask%base );
//}
//int setV(int mask, int v, int pos){
//    int rem = mask%pw[pos];
//    mask /= pw[pos+1];
//    mask = ( mask*base ) + v;
//    mask = ( mask*pw[pos] ) + rem;
//    return mask;
//}
/// <--------------------------- End B - Base Number System -------------------------
---------->


// moves
//int dx[]= {0,0,1,-1};/*4 side move*/
//int dy[]= {-1,1,0,0};/*4 side move*/
//int dx[]= {1,1,0,-1,-1,-1,0,1};/*8 side move*/
//int dy[]= {0,1,1,1,0,-1,-1,-1};/*8 side move*/
//int dx[]={1,1,2,2,-1,-1,-2,-2};/*night move*/
//int dy[]={2,-2,1,-1,2,-2,1,-1};/*night move*/

//double Expo(double n, int p) {
//        if (p == 0)return 1;
//        double x = Expo(n, p >> 1);
//        x = (x * x);
//        return ((p & 1) ? (x * n) : x);
//}

//ll bigmod(ll a,ll b,ll m){if(b == 0) return 1%m;ll x = bigmod(a,b/2,m);x =
(x * x) % m;if(b % 2 == 1) x = (x * a) % m;return x;}
//ll BigMod(ll B,ll P,ll M){ ll R=1%M; while(P>0)
{if(P%2==1){R=(R*B)%M;}P/=2;B=(B*B)%M;} return R;} /// (B^P)%M

typedef pair<int,int> pii;
typedef pair<ll,ll> pll;

#define MXN 50
#define MXE
#define MXQ
#define SZE
#define MOD
#define EPS
#define INF 100000000
#define MX  100005
#define inf 100000000

const ll mod = 1000000007ll;

int main() {
//    ios_base::sync_with_stdio(0);
//    freopen("input.txt", "r", stdin);

   return 0;
}
```

Graph Theory

1. Dinic's-Maxflow

```
///V^2*E Complexity
///number of augment path * (V+E)
///Base doesn't matter

const int INF = 2000000000;
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);
}
//cap=capacity of edges , flow = initial flow
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)) {
        /// update last edge of ith node
        for(int i=0; i<=N; i++) nextEdge[i]=last[i];
        totFlow+=dinicUpdate(S,E);
    }
    return totFlow;
}
```

2. MincostMaxFlow – SPFA

```
///V*E^2 Complexity
///number of augment path * (V+E)
///Base doesn't matter

const int MAXN = 350;          ///total nodes
const int MAXM = 120200;       ///total edges
const int oo = 120200;         ///total edges

int edges;        ///edge info
int Last[MAXN];
int Prev[MAXM],Head[MAXM];
int Cap[MAXM];
int Cost[MAXM];

int Flow[MAXN];
int edgeNo[MAXN];
int dist[MAXN];
int 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) { //source,destination,number of
nodes (give more for safety)
    int totFlow=0,totCost=0;
    while(!Q.empty()) Q.pop();
    int u,v,cap,cost;
    while(true) {
        Flow[S]=oo;
        for(int i = 0; i <= N; i++)
            dist[i] = oo;
        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 make_pair(totFlow,totCost);
}
```

3. Unique Min Cut

```
//Dinic-Max Flow full code
int col[MAXN];
void dfs1(int now) {
```

```
    if(col[now]) return;
    //print1(now);
    col[now]=true;
    for(int e=Last[now]; e>=0; e=Prev[e]) {
        if(e&1) continue; //backward edge
        if(Cap[e]>Flow[e])
            dfs1(Head[e]);
    }
}
void dfs2(int now) {
    if(col[now]) return;
    //print1(now);
    col[now]=true;
    for(int e=Last[now]; e>=0; e=Prev[e]) {
        if((e&1)==0) continue; //forward edge
        if(Cap[e^1]>Flow[e^1])
            dfs2(Head[e]);
    }
}
int main() {
    int n,m,a,b;
    while(cin>>n>>m>>a>>b &&(n||m||a||b)) {
        init(n+10);
        int u,v,w;
        int i;
        for(i=1; i<=m; i++) {
            scanf("%d %d %d",&u,&v,&w);
            addEdge(u,v,w,0);
            addEdge(v,u,w,0);
        }
        int augmentpath=maxFlow(a,b,n+3);
        mem(col,false);
        dfs1(a);
        dfs2(b);
        for(i=1; i<=n; i++)
            if(!col[i])
                break;
        if(i>n) print1("UNIQUE");
        else print1("AMBIGUOUS");
    }
    return 0;
}
```

4. HopcroftKarp (BPM Unweighted)

```
//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];
        int u2 = matching[v];
        if (u2 < 0 || (!vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2))) {
            matching[v] = u1;
            used[u1] = true;
            return true;
        }
    }
    return false;
}
int 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);
    for (int res = 0;;) {
        int f = augmentPath();
        if (!f)
            return res;
        res += f;
    }
}
```

5. Hungarian (BPM Weighted)

```
//Andrei Lopatin
//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)
```

```
#define INF 2000000000
#define rows 110
#define clms 110
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) {
   memset(p,0, sizeof p);
   memset(u,0,sizeof u);
   memset(v,0,sizeof v);
   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 ) ;
      do {
         int j1 = way [ j0 ] ;
         p [ j0 ] = p [ j1 ] ;
         j0 = j1 ;
      } while ( j0 ) ;

   }
   return -v[0]; //minimum cost is stored here
}
```

6. Blossom Algorithm

```
//0 based
//complexity O(VE)
const int MAXN = 505; // number of elements.
vector<int> g[MAXN];
int match[MAXN]; //stores the matcings
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) {
   int i;
   for(i=0; i<n; i++)  g[i].clear();
   memset(blossm,false,sizeof blossm);
```

```
}
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) {
   memset(used,0,sizeof used);
   memset(p,-1,sizeof p);
   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);
         memset(blossm,0,sizeof blossm);
         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;
   memset(match,-1,sizeof match);
   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;
   int n,m;
   scanf("%d %d",&n,&m);
   initialize(n);
   while(m--) {
      scanf("%d %d",&i,&j);
      i--,j--;
      g[i].push_back(j);
      g[j].push_back(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;
}
```

7. BronKerbosch (Maximum clique)

```
/* Find Maximum clique in a graph .
Edges are stored using bit.
BronKerbosch(0,(1LL<<node) - 1 ,0)  */
long long n, edges[50], fans ;
void BronKerbosch(long long r,long long p,long long x) {
   if(p == 0 && x == 0) {
      fans = max(fans,(long long)__builtin_popcountll(r));
      return ;
   }
   int u = 0 ;
   while(!((1LL<<u) & (p|x)))
      u++;
   for(int v=0 ; v<n ; v++) {
      if(((1LL<<v)&p) && !((1LL<<v) & edges[u])) {
         BronKerbosch(r|(1LL<<v),p&edges[v],x&edges[v]);
         p -= (1LL<<v) ;
         x |= (1LL<<v) ;
      }
   }
}
```

8. Bellman Ford

```
//complexity VE
#define SIZE 1010
#define INF 2000000000
vector<int> adj[SIZE],cost[SIZE];
//0 based
bool BellmanFord(int source,int nodes) { //returns true if it has negative
cycle
   vector<int>dist;
   int i,j,k,w,v;
   for(i=0; i<=nodes; i++) { //distance from source
      dist.push_back(INF);
   }
   dist[source]=0;
   for(i=1; i<=nodes-1; i++) {
      for(j=1; j<=nodes; j++)
         for(k=0; k<adj[j].size(); k++) {
            v=adj[j][k];
            w=cost[j][k];
            dist[v]=min(dist[v],dist[j]+w);
         }
   }
   for(i=1; i<=nodes; i++)
      for(j=0; j<adj[i].size(); j++) {
         v=adj[i][j];
         w=cost[i][j];
         if(dist[v]>dist[i]+w) return true;
      }
   return false;
}
```

9. Directed MST

```
#define MAX_VERTEX 100100
#define INF 100100000
struct Edge {
   int u,v,w,ind;
   Edge(int u=0,int v=0,int w=0) {
      this->u = u;
      this->v = v;
      this->w = w;
   }
   bool operator < (const Edge &b)
   const {
      return w<b.w;
   }
};
int nV,nE; //nV -> Number of Vertex.
vector<Edge> Edges[MAX_VERTEX]; //Adjecency List.
//Edge u->v inserted into list of v.
vector<Edge> EdgeList; //All edges. Used
//if Path or Used Edges Required.
vector<int>adj[MAX_VERTEX]; // to check the
//graph connectivity.
int par[MAX_VERTEX],color[MAX_VERTEX];
int W[MAX_VERTEX],toUse[MAX_VERTEX];
bool used[MAX_VERTEX+100];
int vertexEdge[MAX_VERTEX];
vector<int>choosed;
int DMST(int nodes,int root,vector<Edge> Edges[]) {
   int i,j,t,u,v;
   Edges[root].clear();
```

```
for(i=0; i<nodes; i++) {
   par[i] = i;
   sort(Edges[i].begin(),Edges[i].end());
}
bool cycle_found = true;
while(cycle_found) {
   cycle_found = false;
   memset(color,0,sizeof color);
   color[root] = -1;
   for(i=0,t=1; i<nodes; i++,t++) {
      u = par[i];
      if(color[u]) continue;
      for(v=u; !color[v]; v=par[Edges[v][0].u]) {
         color[v] = t;
         choosed.push_back(Edges[v][0].ind);
      }
      if(color[v] != t) continue;
      cycle_found = true;
      int sum = 0, super = v;
      for( ; color[v]==t; v=par[Edges[v][0].u]) {
         color[v]++;
         sum+= Edges[v][0].w;
      }
      for(j=0; j<nodes; j++) W[j] = INF;
      for(; color[v]==t+1; v=par[Edges[v][0].u]) {
         color[v]--;
         for(j = 1; j<Edges[v].size(); j++) {
            int w = Edges[v][j].w+
                  sum-Edges[v][0].w;
            if(w<W[Edges[v][j].u]) {
               W[Edges[v][j].u] = w;
               toUse[Edges[v][j].u]=Edges[v][j].ind;
            }
         }
         par[v] = super;
      }
      Edges[super].clear();
      for(j=0; j<nodes; j++)
         if(par[j] != par[par[j]])
            par[j] = par[par[j]];
      for(j=0; j<nodes; j++)
         if(W[j]<INF && par[j]!= super) {
            Edge e = Edge(j,super,W[j]);
            e.ind = toUse[j];
            Edges[super].push_back(e);
         }
      sort(Edges[super].begin(),Edges[super].end());
      for(j=0; j<Edges[super].size(); j++) {
         Edge e=Edges[super][j];
      }
   }
}
//cout<<"In outside of Loop:"<<endl;
   int sum = 0;
   for(i=0; i<nodes; i++)
      if(i!=root && par[i]==i) {
         sum += Edges[i][0].w;
// i'th node's zero'th edge contains the
//minimum cost after DMST algo.
      }
   return sum;
}//End Of DMST Function....
```

```
int isPossible() {
   int i,j,u,v;
   for(i=0; i<nV; i++) {
      for(j=0; j<Edges[i].size(); j++) {
         adj[Edges[i][j].u].push_back(Edges[i][j].v);
      }
   }
   queue<int>Q;
   Q.push(0);
   memset(color,0,sizeof color);
   color[0] = 1;
   while(!Q.empty()) {
      //BFS to check graph Connectivity.
      u = Q.front();
      Q.pop();
      for(i=0; i<adj[u].size(); i++) {
         v = adj[u][i];
         if(color[v]) continue;
         color[v] = 1;
         Q.push(v);
      }
   }
   for(i=0; i<nV; i++) if(!color[i]) return -1;
   return DMST(nV,0,Edges);
}
int main() {
   int i,j,test,Case=1;
   Edge e;
   test = 1;
   while(test--) {
      scanf("%d %d",&nV,&nE);
      for(i=0; i<nE; i++) {
         scanf("%d %d %d",&e.u,&e.v,&e.w);
         e.u--;
         e.v--;
         e.ind = i;
         Edges[e.v].push_back(e);
         EdgeList.push_back(e);
      }
      memset(used,0,sizeof used);
      int res = isPossible();
      if(res == -1) printf("-1\n");
      else {
         memset(used,0,sizeof used);
         memset(color,0,sizeof color);
         for(i=choosed.size()-1; i>=0; i--) {
            Edge e = EdgeList[choosed[i]];
            if(color[e.v]) continue;
            color[e.v] = 1;
            used[choosed[i]] = true;
         }
         printf("%d\n",res);
         if(res) {
            for(i=0; i<nE; i++)
               if(used[i] && EdgeList[i].w)
                  printf("%d ",i+1);
            printf("\n");
         }
      }
   }
   return 0;
}
```

10. Dominator Tree

```
// note: Here root is 1
const int maxn = 200900;
vector<int> graph[maxn];
vector<int> tree[maxn], rg[maxn], bucket[maxn];
int sdom[maxn], par[maxn], dom[maxn], dsu[maxn], label[maxn];
int arr[maxn], rev[maxn], T ;
void dini(int node) {
   T = 0;
   for (int i = 0; i <= node ; i++) {
      sdom[i] = par[i] = dom[i] = dsu[i] = label[i] = 0;
      arr[i] = rev[i] = 0;
      graph[i].clear();
      tree[i].clear();
      rg[i].clear();
      bucket[i].clear();
   }
}
int Find(int u, int x = 0) {
   if (u == dsu[u])
      return x ? -1 : u;
   int v = Find(dsu[u], x + 1);
   if (v < 0)
      return u;
   if (sdom[label[dsu[u]]] < sdom[label[u]])
      label[u] = label[dsu[u]];
   dsu[u] = v;
   return x ? v : label[u];
}
void Union(int u, int v) { //Add an edge u-->v
   dsu[v] = u;
}
void dfs0(int u) {
   T++;
   arr[u] = T;
   rev[T] = u;
   label[T] = T;
   sdom[T] = T;
   dsu[T] = T;
   for (int i = 0; i < graph[u].size() ; i++) {
      int w = graph[u][i];
      if (!arr[w])
         dfs0(w), par[arr[w]] = arr[u];
      rg[arr[w]].push_back(arr[u]);
   }
}
void BuildTree() {
   //Build Dominator tree
   dfs0(1);
   int n = T;
   for (int i = n; i >= 1; i--) {
      for (int j = 0; j < rg[i].size() ; j++)
         sdom[i] = min(sdom[i], sdom[Find(rg[i][j])]);
      if (i > 1)
         bucket[sdom[i]].push_back(i);
      for (int j = 0; j < bucket[i].size() ; j++) {
         int w = bucket[i][j];
         int v = Find(w);
         if (sdom[v] == sdom[w])
            dom[w] = sdom[w];
         else
```

```
            dom[w] = v;
        }
        if (i > 1)
            Union(par[i], i);
    }
    for (int i = 2; i <= n; i++) {
        if (dom[i] != sdom[i])
            dom[i] = dom[dom[i]];
        tree[rev[i]].push_back(rev[dom[i]]);
        tree[rev[dom[i]]].push_back(rev[i]);
    }
}
```

11. Euler Path Print

```
int is[sz] = {0} ;
vector<int>path ;
void go(int u) {
    while(is[u]<adj[u].size())
        go(adj[u][is[u]++]);
    ans.push_back(u);
}
/// In ans vector path will be saved in reverse order
```

12. Articulation Points (or Cut Vertices)

```
const int sz = 1007 ;
vector<int>graph[sz] ;
int low[sz], disc[sz], tme, ans, compo, component[sz] ;
bool isPoint[sz] ;
void reset() {
    for(int i=0 ; i<sz ; i++) {
        graph[i].clear();
        low[i] = -1 ;
        disc[i] = -1 ;
        component[i] = -1 ;
        isPoint[i] = 0 ;
    }
    tme = 0 ;
    ans = 0 ;
    compo = 0 ;
}

void tarjan(int u,int p) {
    low[u] = disc[u] = ++tme ;
    int v, cont = 1, childern = 0 ;
    for(int i=0 ; i<graph[u].size() ; i++) {
        v = graph[u][i] ;
        if(v == p || v == u)
            continue;
        if(disc[v] == -1) {
            childern++;
            tarjan(v,u);
            low[u] = min(low[u],low[v]);
            if(p == -1 && childern>1)
                cont++, isPoint[u] = 1 ;
            if(p != -1 && low[v]>=disc[u])
                cont++, isPoint[u] = 1 ;
        } else
            low[u] = min(low[u],disc[v]);
    }
    component[u] = cont ;
```

```
}
```

13. Articulation Bridge

```
#define lim         1005
//in multiple edge bridge won't work
int tim[lim],low[lim];
int timer;
vector<int> adj[lim]; //only adj should be cleared
struct edge {
   int u, v;
};
vector<edge> bridge;//the ans(should be cleared)
void dfs(int u,int par) { //par=-1 dhore call dite hobe(root ar parent nai)
   tim[u] = low[u] = ++timer;

   for(int i = 0 ; i<adj[u].size() ; i++) {
      int v = adj[u][i];
      if(v==par) continue;
      if(!tim[v]) {
         dfs(v,u);
         low[u] = min(low[u],low[v]);
         if(low[v]>tim[u]) { //attention greater equals for bridge and
articulation point
            edge tem;
            tem.u=u;
            tem.v=v;
            bridge.push_back(tem);
         }
      } else { //determining back edge
         low[u] = min(low[u],tim[v]);
      }
   }
   return;
}

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

14. BCC

```
//1 Based
//no problem in multiple edge and self loop
int tim[lim],low[lim];
int timer;
vector<int> adj[lim]; //only adj should be cleared
stack<pair<int,int> >S;
pair<int,int> 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;
   pair<int,int> now;
   int tot=0;
   while(!S.empty()) {
      now = S.top();
      S.pop();
      uu = now.first, vv = now.second;
      ed[tot++] = make_pair(uu, vv);
```

```
      if(u==uu && v==vv) break;
      if(u==vv && v==uu) break;
   }
   if(tot<=1) return;
   puts("");
   for(int i=0; i<tot; i++) {
      cout<<ed[i].first<<" "<<ed[i].second<<" ";
   }
   cout<<endl;
   //doing according to problem
   return;
}
void bcc(int u,int par) {
// par=-1 dhore call dite hobe(root ar parent nai)
   tim[u] = low[u] = ++timer;
   for(int i = 0 ; i<adj[u].size() ; i++) {
      int v = adj[u][i];
      if(v==par) continue;
      if(tim[v]==0) {
         S.push(make_pair(u, v));
         bcc(v,u);
         low[u] = min(low[u],low[v]);
         if(low[v]>=tim[u]) {
            cout<<"cheak : "<<u<<' '<<v<<endl;
            calc_bcc(u, v);
         }
      }
      else if(tim[v] < tim[u]) {
         low[u] = min(low[u],tim[v]);
         S.push(make_pair(u, v));
      }
   }
   return;
}
void BCC(int n) {
   timer=0;
   memset(tim,0,sizeof tim);
   int i;
   for(i = 1; i <= n; i++)
      if(!tim[i])
         bcc(i,-1);
}
void add(int ina,int inb) {
   adj[ina].push_back(inb);
   adj[inb].push_back(ina);
}
int main() {
   int n,m,u,v;
   cin>>n>>m;
   while(m--) {
      cin>>u>>v;
      add(u,v);
   }
   BCC(n);
}
```

15. 2-SAT/ SCC

```
#define lim         1005  //number of nodes(yes/no nodes)
//0 based
vector<int> 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);
   for(i = 0; i < adj[u].size(); i++) {
      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;
   memset(col,0,sizeof col);
   while(!S.empty()) S.pop();
   for(i = 0; i < n; i++) 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]) //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].push_back(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();
}
```

16. Flow/BPM Notes

		Flow Algorithms	
	Name	**Complexity**	**Average Case**
1	Ford Fulkerson	VE^4	V^3
2	Dinic Maxflow	V^2E	V^3
3	Min cost using SPFA	VE^4	V^3
4	Hopcroft- carp	**E√V**	**E√V**
5	Hungarian	N^2M	N^2M
6	Non-weighted Blossoms	VE	VE
7	Weighted Blossom		
		N = number of rows, M = number of columns	
		Concepts	
	Name	**Description**	**Solution**
1	Vertex cover	Minimum number of vertex required to cover all edges	Equals to Matching for bipartite otherwise NP complete
2	Edge cover	Minimum number of edge required to cover all vertices	V-matching for all graphs
3	Minimum Independent path (IP)	Minimum number of disjoint paths to cover all vertices	V-matching for all graphs
4	Minimum path cover (MPC)	Minimum number of paths to cover all vertices	Convert it MIP problem by finding trasitive closure
5	Clique	A complete subgraph	
6	Maximal clique	A clique which cannot be expanded	
7	Maximum clique	A maximal clique with highest number of vertices	Make a reverse graph then answer = V - vertex cover
8	Closure	A directed subgraph with no outgoing edges outside the graph	

9	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)
10	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	
11	Perfect matching	Everynode can be matched	
12	Minimum Dominating set	Minimum number of vertex to cover all vertices	NP-complete
13	Set cover	Minimum number of set to cover all elements	NP-complete
14	Hitting set	Minimum number of element to cover all sets	NP-complete
15	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
16	Minimum weighted IP	Minimum IP with minimum weighted	Convert it to Minimum weighted matching
	Sometimes answer = matching/2		
`	Normally binary Search is better than iteration in flow		

Data Structure

17. Magic STL

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

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace __gnu_pbds;

typedef tree<
double,
int,
less<double>,
rb_tree_tag,
tree_order_statistics_node_update> map_t; //create map tree

typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update> set_t; //create set tree

int main()
{
   map_t s ;
   set_t ss ;
   s.insert(make_pair(12, 1012));
   s.insert(make_pair(505, 1505));
   s.insert(make_pair(30, 1030));
   s.insert(make_pair(12,580));

   cout<<s.find_by_order(0)->sc<<endl ; // find by indx
   cout<<s.order_of_key(20)<<endl; // count less than 20 by first element
   ss.insert(1);
   ss.insert(4);
   ss.insert(10);
   ss.insert(5);

   cout<<*ss.find_by_order(1)<<endl ;
   cout<<ss.order_of_key(0)<<endl ; //count less than 5
   ss.erase(4); // erase by element

   return 0;
}
```

18. BIT

```
int tree[Max] ;
void update(int idx,int x) {
   while(idx<=Max) {
      tree[idx] += x ;
      idx += (idx & (-idx));
   }
}
ll qury(int idx) {
   ll sum = 0 ;
```

```
    while(idx>0) {
        sum+=tree[idx];
        idx -= (idx&(-idx));
    }
    return sum ;
}
ll readsingle(int idx) {
    ll sum = tree[idx] ;
    if(idx>0) {
        int z = idx - (idx&(-idx)) ;
        idx--;
        while(z != idx) {
            sum -= tree[idx] ;
            idx -= (idx & (-idx));
        }
    }
    return sum ;
}
```

19. Build BST

```
struct Build_BST {
// Element of array must be a permutation of 1 to n
    set<int>st ;
    set<int>::iterator it ;
    vector<vector <int> >graph ;
    pair<int,int> isfree[sz] ;
    int arr[sz], ln ;
    Build_BST(int ln):ln(ln),graph(n+7) {
    for(int i=0 ; i<sz ; i++)
        isfree[i] = make_pair(-1,-1);
}
void add(int u) {
    int v ;
    it = st.lower_bound(u);
    if(it != st.end()) {
        v = *it ;
        if(isfree[v].first == -1) {
            isfree[v].first = 1 ;
            graph[v].push_back(u);
        }
    }
    it--;
    if(it != st.begin()) {
        v = *it ;
        if(isfree[v].second == -1) {
            isfree[v].second = 1 ;
            graph[v].push_back(u);
        }
    }
    st.insert(u);
}
void build() {
    st.insert(-1);
    st.insert(arr[0]);
```

```
    for(int i=1 ; i<ln ; i++)
        add(arr[i]);
  }
};
```

20. Persistent Segment Tree

```
struct data {
  int l, r, val;
  data() {
    l = r = val = 0;
  }
  data(int _l, int _r, int _val) {
    l = _l, r = _r, val = _val;
  }
} node[10*MXN+7];  /// node indexing from 1

int tree[MXN+7], cnt;
int build(int cur, int base, int top) {
  if( base==top ) {
    node[cur] = data(0, 0, 0);
    return 0;
  }
  int left, right, mid;
  node[cur].l = left = ++cnt;
  node[cur].r = right = ++cnt;
  mid = ( base+top )/2;
  node[cur].val = build(left, base, mid);
  node[cur].val += build(right, mid+1, top);
  return node[cur].val;
}
void upgrade(int pre, int cur, int base, int top, int pos, int v) {
  if( base==top ) {
    node[cur].val += v;
    return;
  }
  int left, right, mid;
  mid = ( base+top )/2;
  if( pos<=mid ) {
    node[cur].r = node[pre].r;
    node[cur].l = ++cnt;
    upgrade(node[pre].l, node[cur].l, base, mid, pos, v);
  } else {
    node[cur].l = node[pre].l;
    node[cur].r = ++cnt;
    upgrade(node[pre].r, node[cur].r, mid+1, top, pos, v);
  }
  node[cur].val = node[ node[cur].l ].val + node[ node[cur].r ].val;
}
int query(int pre, int cur, int base, int top, int pos) {
  if( base == top )   return base;
  int ele = node[ node[cur].l ].val - node[ node[pre].l ].val;
  int mid = ( base+top )/2;
  if( ele>=pos ) {
    return query(node[pre].l, node[cur].l, base, mid, pos);
  }
  return query(node[pre].r, node[cur].r, mid+1, top, pos-ele);
}
int arr[MXN+7];
/**   Problem: K-th Number in a Range. Assume all numbers between 1
to n and distinct    */
int main() {
```

```
//    freopen("E:\\00.txt", "r", stdin);
    int n, m, l, r, pos, res, i, j, k;
    cnt = 0;
    sf2(n, m);
    for( i = 1; i<=n; i++ ) {
        sf1(arr[i]);  /// arr[i] -> 1 to n
    }
    tree[0] = ++cnt;
    build(tree[1], 1, n);
    for( i = 1; i<=n; i++ ) {
        tree[i] = ++cnt;
        upgrade(tree[i-1], tree[i], 1, n, arr[i], 1);
    }
    for( i = 0; i<m; i++ ) {
        sf3(l, r, pos);
        res = query(tree[l-1], tree[r], 1, n, pos);
        pf("%d\n", res);
    }
    return 0;
}
```

21. Segment Tree (2D)

```
#include<bits/stdc++.h>
using namespace std;
#define D(x)    cout << #x " = " << (x) << endl
#define MAX     1005
#define xx      first
#define yy      second
typedef pair<int,int> pii;

const int inf = 1000000000;

struct segTree {
    int arr[MAX << 2];

    segTree() {
        for(int i = 0; i < (MAX << 2); i++) arr[i] = -inf;
    }

    void update(int idx, int st, int ed, int pos, int val,
                                        vector<int> &nodeList) {
        nodeList.push_back(idx);

        if(st == ed) {
            arr[idx] = max(arr[idx], val);
            return;
        }

        int mid = (st + ed)/2, l = idx << 1, r = l | 1;
        if(pos <= mid) update(l, st, mid, pos, val, nodeList);
        else update(r, mid+1, ed, pos, val, nodeList);

        arr[idx] = max(arr[l], arr[r]);
    }

    int query(int idx, int st, int ed, int i, int j) {
        if(st == i && ed == j) return arr[idx];

        int mid = (st + ed)/2, l = idx << 1, r = l | 1;
        if(j <= mid) return query(l, st, mid, i, j);
        if(i > mid) return query(r, mid+1, ed, i, j);
        else return max(query(l, st, mid, i, mid),
                                query(r, mid+1, ed, mid+1, j));
```

```
    }
};

struct _2DsegTree {
    segTree segArr[MAX << 2];
    vector<int> affected_nodes;

    void update(int idx, int st, int ed, int i, int j, int val) {
        if(st == ed) {
            affected_nodes.clear();
            segArr[idx].update(1, 1, MAX, j, val, affected_nodes);
            return;
        }

        int mid = (st + ed)/2, l = idx << 1, r = l | 1;
        if(i <= mid) update(l, st, mid, i, j, val);
        else update(r, mid+1, ed, i, j, val);

        for(int p = 0; p < (int) affected_nodes.size(); p++) {
            int q = affected_nodes[p];
            segArr[idx].arr[q] = max(segArr[l].arr[q], segArr[r].arr[q]);
        }
    }


    int query(int idx, int st, int ed, int st_r, int ed_r, int st_c, int ed_c) {
        if(st == st_r && ed == ed_r)
            return segArr[idx].query(1, 1, MAX, st_c, ed_c);

        int mid =(st + ed)/2, l =  idx << 1, r = l + 1;
        if(ed_r <= mid) return query(l, st, mid, st_r, ed_r, st_c, ed_c);
        if(st_r > mid) return query(r, mid+1, ed, st_r, ed_r, st_c, ed_c);
        return max(query(l, st, mid, st_r, mid, st_c, ed_c),
                   query(r, mid+1, ed, mid+1, ed_r, st_c, ed_c));
    }
};


_2DsegTree tree;
vector<pii> input;

int main() {
    int i, x, y, n, mx = 1;

    scanf("%d", &n);
    for(i = 1; i <= n; i++) {
        scanf("%d %d", &x, &y);
        input.push_back(pii(x,y));
    }

    for(i = n - 1; i >= 0; i--) {
        x = input[i].xx;
        y = input[i].yy;
        int q = 1 + max(0, tree.query(1, 1, MAX, x, MAX, y, MAX));
        tree.update(1, 1, MAX, x, y, q);

        mx = max(mx, q);
    }

    printf("%d\n", mx);
    return 0;
}
```

22. Sparse Table

```
#define Max 10000005
int ST[24][Max];
int A[Max];
void Compute_ST(int N) {
   for (int i=0; i<N; i++)ST[0][i] = i;
   for (int k = 1; (1 << k)<N; k++) {
      for (int i=0; i+(1<<k)<=N; i++) {
         int x = ST[k-1][i];
         int y = ST[k-1][i+(1<<k-1)];
         ST[k][i]=A[x]<=A[y]?x:y;
      }
   }
}
int RMQ(int i, int j) {
   int k = log2(j-i);
   int x = ST[k][i];
   int y = ST[k][j-(1<<k)+1];
   return A[x] <= A[y] ? x : y;
}
int main() {
   int N;
   cin>>N;
   for(int i=0; i<N; i++) {
      cin>>A[i];
   }
   Compute_ST(N);
   int Q;
   cin>>Q;
   while(Q--) {
      int x,y;
      cin>>x>>y;
      cout<<A[RMQ(x,y)]<<endl;
   }
   return 0;
}
```

23. Treap

```
struct item {
   int key, prior;
   int val, sum, lazy;
   int mx; /// mx value in this Treap Tree
   int repl;
   bool repl_flag;
   bool rev;
   item *l, *r;

   item() {}
   item(int _key, int _prior) {
      key = _key, prior = _prior;
      val = sum = 0;
      l = NULL,   r = NULL;
   }
};
typedef item* Treap;

/**
*   It'll return a new Treap node having value val
*/
Treap init(int val) {
   Treap node =(Treap)malloc(sizeof(item));
```

```
    node->key = 1;
    node->val = node->sum = val;
    node->mx = val;
    node->lazy = 0;
    node->repl = 0;
    node->repl_flag = false;
    node->rev = false;
    node->prior = rand();
    node->l = node->r = NULL;
    return node;
}
/**
*   It'll return the total size of current Treap node
*/
int cnt(Treap t) {
    if( t ) return t->key;
    return 0;
}
void upd_cnt(Treap &t) {
    if( t )
        t->key = cnt(t->l) + cnt(t->r) + 1;
}
void upd_lazy(Treap t) {
    if( !t or !t->lazy )    return;
    t->val += t->lazy;
    t->mx += t->lazy;
    t->sum = t->val*cnt(t);
    if( t->l )  t->l->lazy += t->lazy;
    if( t->r )  t->r->lazy += t->lazy;
    t->lazy = 0;
}
void upd_repl(Treap t) {
    if( !t or !t->repl_flag )   return;
    t->val = t->mx = t->repl;
    t->sum = t->val*cnt(t);
    if( t->l ) {
        t->l->repl = t->repl;
        t->l->repl_flag = true;
    }
    if( t->r ) {
        t->r->repl = t->repl;
        t->r->repl_flag = true;
    }
    t->repl_flag = false;
    t->repl = 0;
}
void upd_rev(Treap t) {
    if( !t or !t->rev ) return;
    t->rev = false;
    swap(t->l, t->r);
    if( t->l )  t->l->rev ^= true;
    if( t->r )  t->r->rev ^= true;
}
void reset(Treap t) {
    if( !t ) return;
    t->mx = t->val;
    t->sum = t->val;
}
void combine(Treap &t, Treap l, Treap r) {
    if( !l )    t = r;
    else if( !r )   t = l;
    else {
```

```
        t->mx = max(l->mx, r->mx);
        t->sum = l->sum + r->sum;
    }
}
void operation(Treap t) {
    if( !t )    return;
    reset(t);
    upd_rev(t->l);
    upd_rev(t->r);
    upd_repl(t->l);
    upd_repl(t->r);
    upd_lazy(t->l);
    upd_lazy(t->r);
    combine(t, t->l, t);
    combine(t, t, t->r);
}
void split(Treap t, Treap &l, Treap &r, int key, int add = 0) {
    upd_rev(t);
    upd_repl(t);
    upd_lazy(t);
    if( !t )
        return void (l = r = 0);
    int cur_key = add + cnt(t->l);
    if( key<=cur_key ) {
        split(t->l, l, t->l, key, add);
        r = t;
    } else {
        split(t->r, t->r, r, key, add+1+cnt(t->l));
        l = t;
    }
    upd_cnt(t);
    operation(t);
}
void Merge(Treap &t, Treap l, Treap r) {
    upd_rev(l);
    upd_rev(r);
    upd_repl(l);
    upd_repl(r);
    upd_lazy(l);
    upd_lazy(r);
    if( !l ) {
        t = r;
        return;
    }
    if( !r ) {
        t = l;
        return;
    }
    if( l->prior > r->prior ) {
        Merge(l->r, l->r, r);
        t = l;
    } else {
        Merge(r->l, l, r->l);
        t = r;
    }
    upd_cnt(t);
    operation(t);
}
/**
*   It'll add v to all the elements from l to r
*   1-indexing
*/
```

```
void range_update(Treap t, int l, int r, int v) {
   Treap lft, mid;
   split(t, lft, t, l-1);
   split(t, mid, t, r-l+1);
   upd_repl(mid);
   mid->lazy = v;
   Merge(t, mid, t);
   Merge(t, lft, t);
}
/**
*   It'll replace all the elements to v from l to r
*   1-indexing
*/
void range_replace(Treap t, int l, int r, int v) {
   Treap lft, mid;
   split(t, lft, t, l-1);
   split(t, mid, t, r-l+1);
   upd_lazy(mid);
   mid->repl_flag = true;
   mid->repl = v;
   Merge(t, mid, t);
   Merge(t, lft, t);
}
/**
*   It'll Reverse the elements from l to r
*   1-indexing
*/
void range_reverse(Treap t, int l, int r) {
   Treap lft, mid;
   split(t, lft, t, l-1);
   split(t, mid, t, r-l+1);
   mid->rev ^= true;
   Merge(t, mid, t);
   Merge(t, lft, t);
}
/**
*   It'll return the sum of all elements from l to r
*   1-indexing
*/
int range_query(Treap t, int l, int r) {
   Treap lft, mid;
   split(t, lft, t, l-1);
   split(t, mid, t, r-l+1);
   int ret = mid->sum;
   Merge(t, mid, t);
   Merge(t, lft, t);
   return ret;
}
Treap Root; /// Root of Treap
int main() {
   int v, n;
   scanf("%d", &n);
   for( int i = 0; i<n; i++ ) {
      /// All the elements will be inserted to Treap one by one
      /// automatically 1-indexing
      scanf("%d", &v);
      if( !i )    Root = init(v);
      else    Merge(Root, Root, init(v));
   }
   int q, typ, p, l, r;
   scanf("%d", &q);
   while(q--) {
```

```
      scanf("%d", &typ);
      if( typ == 0 ) { /// sum of all elements from l to r
         scanf("%d %d", &l, &r);
         /// range_query() can be modified to get other data of this Range
         int ans = range_query(Root, l, r);
         printf("%d\n", ans);
      } else if( typ == 1 ) { /// Add v to all elements from l to r
         scanf("%d %d %d", &l, &r, &v);
         range_update(Root, l, r, v);
      } else if( typ == 2 ) { /// Replace all elements from l to r by v
         scanf("%d %d %d", &l, &r, &v);
         range_replace(Root, l, r, v);
      } else if( typ == 3 ) { /// Reverse all elements from l to r
         scanf("%d %d", &l, &r);
         range_reverse(Root, l, r);
      } else if( typ == 4 ) { /// Replace p-th element by v
         scanf("%d %d", &p, &v);
         range_replace(Root, p, p, v);
      }
   }
   return 0;
}
```

24. LCA 1

```
#define MXN 100000
#define SZE 17
int n, q, dp[MXN+3][SZE+3], level[MXN+7];
vector<int> adj[MXN+7];
void dfs(int u, int pre) {
   for( int i = 1; i<=SZE; i++ ) {
      dp[u][i] = dp[dp[u][i-1]][i-1];
   }
   for( auto v:adj[u] ) {
      if( v == pre )  continue;
      dp[v][0] = u;
      level[v] = level[u]+1;
      dfs(v, u);
   }
}
int lca(int u, int v) {
   if( level[u]>level[v] ) swap(u, v);
   for( int i = SZE; i>=0; i-- ) {
      int par = dp[v][i];
      if( level[par]>=level[u] ) {
         v = par;
      }
   }
   if( u == v )    return u;
   for( int i = SZE; i>=0; i-- ) {
      if( dp[u][i] != dp[v][i] ) {
         u = dp[u][i];
         v = dp[v][i];
      }
   }
   return dp[u][0];
}

int main() {
//   freopen("H:\\00.txt", "r", stdin);
   int u, v, i, j, k;
   sf2(n, q);
   for( i = 1; i<n; i++ ) {
```

```
        sf2(u, v);
        adj[u].pb(v);
        adj[v].pb(u);
    }
    level[1] = 0;
    dp[1][0] = 1;
    dfs(1, 0);
    for( i = 0; i<q; i++ ) {
        sf2(u, v);
        printf("%d %d : %d\n", u, v, lca(u, v));
    }
    return 0;
}
```

25. LCA 2

```
///complexity build(V)
///per query complexity(strictly log(step))
///graph must be tree
int node, T;
int parent[MX][step+1];
int start[MX], finish[MX];
int distan[MX], level[MX];
vector<data> adj[MX];

void dfs(int u, int p, int dis, int lev) {
    start[u] = T++;
    level[u] = lev;
    distan[u] = dis;
    parent[u][0] = p;
    for(int i = 1; i <= step; i++)
        parent[u][i] = parent[parent[u][i-1]][i-1];
    for(int i = 0; i < adj[u].size(); i++) {
        v = adj[u][i].v;
        if(v != p)
            dfs(v,u,dis+adj[u][i].w,lev+1);
    }
    finish[u] = T++;
    return;
}
bool Is_Anchestor(int u, int v) {
    if(start[u] <= start[v] && finish[u] >= finish[v])
        return true;
    return false;
}
int LCA_Query(int u, int v) {
    if(Is_Anchestor(u,v)) return u;
    if(Is_Anchestor(v,u)) return v;
    int tem = u;
    for(int i = step; i>=0; i--)
        if(!Is_Anchestor(parent[tem][i],v))
            tem = parent[tem][i];
    return parent[tem][0];
}
int Kth_Query(int u, int k) {
    int tem = u;
    for(int i = step; i>=0; i--)
        if((k>>i)&1 == 1)
            tem = parent[tem][i];
    return tem;
}
void lca_cls(void) {
    for(int i = 0; i <= node; i++) {
```

```
        adj[i].clear();
        for(int j = 0; j <= step; j++)
            parent[i][j] = 1;
    }
}
void input() {
    scanf("%d", &node);
    lca_cls();
    for(int i = 1; i < node; i++) {
        scanf("%d%d%d", &u, &v, &w);
        adj[u].push_back(data(v,w));
        adj[v].push_back(data(u,w));
    }
}
int main() {
    input();
    T = 0;
    dfs(1,1,0,1);
    lca = LCA_Query(u,v);
    res = distan[u]+distan[v]-2*distan[lca];
    ///distance
    w--;///find w'th node
    if(level[u]-level[lca] == w) res = lca;
    else if(level[u]-level[lca] > w)
        res = Kth_Query(u,w);
    else
        res = Kth_Query(v,level[u]+level[v]-2*level[lca]-w);
}
```

26. HLD

```
int indx, tot_chain, chain_no[MXN+7]; /// for HLD
int chain_head[MXN+7], pos_in_base[MXN+7];
int base_arr[MXN+7];
int depth[MXN+7], cost[MXN+7];/// for Tree
int son[MXN+7], prnt[MXN+7];
int n, tree[4*MXN+7]; /// for segment tree

vector<int> adj[MXN+7]; /// for Graph

void reset(int num) {
    indx = -1;
    tot_chain = 0;
    for( int i = 0; i<=num; i++ ) {
        adj[i].clear();
        chain_head[i] = son[i] = -1;
    }
}
/// Start of Segment Tree
int build(int node, int base, int top) {
    if( base==top ) {
        tree[node] = cost[base_arr[base]];
        return tree[node];
    }
    int left, right, mid;
    left = node<<1;
    right = left+1;
    mid = ( base+top )>>1;
    tree[node] = build(left, base, mid);
    tree[node] += build(right, mid+1, top);
    return tree[node];
}
int update(int node, int base, int top, int i, int v) {
```

```
    if( base>i or top<i )   return tree[node];
    if( base==top ) {
        tree[node] = v;
        return tree[node];
    }
    int left, right, mid;
    left = node<<1;
    right = left+1;
    mid = ( base+top )>>1;
    tree[node] = update(left, base, mid, i, v);
    tree[node] += update(right, mid+1, top, i, v);
    return tree[node];
}
int query(int node, int base, int top, int i, int j) {
    if( base>j or top<i )   return 0;
    if( base>=i and top<=j )    return tree[node];
    int left, right, mid;
    left = node<<1;
    right = left+1;
    mid = ( base+top )>>1;
    int ret = query(left, base, mid, i, j);
    ret += query(right, mid+1, top, i, j);
    return ret;
}
/// End of Segment Tree

/**
*   dfs() used to set parent of a node,
*   depth of a node, special son of a node
*   it will return the subtree size of node - u
*/
int dfs(int u, int pre) {
    prnt[u] = pre;
    int mx = -1, ret = 1;
    for( int i = 0; i<adj[u].size(); i++ ) {
        int v = adj[u][i];
        if( v == pre )  continue;
        depth[v] = depth[u] + 1;
        int sz = dfs(v, u);
        ret += sz;
        if( sz>mx ) {
        /// For finding special son of node - u
            mx = sz;
            son[u] = v;
        }
    }
    return ret;
}

/**
*   Actual HLD Part
*/
void HLD(int u, int pre) {
    if( chain_head[tot_chain] == -1 )
        chain_head[tot_chain] = u;
    pos_in_base[u] = ++indx;
    base_arr[indx] = u;
    chain_no[u] = tot_chain;
    if( son[u] == -1 )  return;
    HLD(son[u], u);
    for( int i = 0; i<adj[u].size(); i++ ) {
        int v = adj[u][i];
```

```
        if( v == pre or v == son[u] )   continue;
        tot_chain++;
        HLD(v, u);
    }
}

/**
*   It'll return sum of cost
*   from node - u to node - v
*/
int solve(int u, int v) {
    int ch1 = chain_no[u];
    int ch2  = chain_no[v];
          /// chd_u = Chain Head of u_chain
    int chd_u = chain_head[ch1];
          /// chd_v = Chain Head of v_chain
    int chd_v = chain_head[ch2];
    int ret = 0;
          /// while two chains are in different chain
    while( chd_u != chd_v ) {
        if( depth[chd_u]<depth[chd_v] ) {
              /// So that u-head will always deeper than v-head
            swap(chd_u, chd_v);
            swap(u, v);
        }
        ret += query(1, 0, indx, pos_in_base[chd_u], pos_in_base[u]);
        u = prnt[chd_u];    /// Changing the Chain
        ch1 = chain_no[u];
        chd_u = chain_head[ch1];  /// updating chain head
    }
    if( depth[u]<depth[v] ) swap(u, v);
    ret += query(1, 0, indx, pos_in_base[v], pos_in_base[u]);
    return ret;
}

int main() {
    //freopen("H:\\00.txt", "r", stdin);
    int t, cas = 0, q, typ, u, v, pos, ans, i, j, k;
    scanf("%d", &t);
    while(t--) {
        scanf("%d", &n);
        reset(n);
        for( i = 0; i<n; i++ ) {
            scanf("%d", &cost[i]); /// i-th node, indexing 0
        }
        for( i = 1; i<n; i++ ) {
            scanf("%d %d", &u, &v);
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        depth[0] = 0; /// 0- is Root, depth = 0
        dfs(0, -1);
        HLD(0, -1);
        build(1, 0, indx);
        printf("Case %d:\n", ++cas);
        scanf("%d", &q);
        while(q--) {
            scanf("%d", &typ);
            if( typ == 1 ) {
                scanf("%d %d", &u, &v);
                cost[u] = v;
                update(1, 0, indx, pos_in_base[u], v);
```

```
        } else {
            scanf("%d %d", &u, &v);
            ans = solve(u, v);
            printf("%d\n", ans);
        }
      }
   }
   return 0;
}
```

27. MO's on Tree

```
vector<int> graph[sz] ;
int arr[sz], depth[sz], timer, par[sz][20], strt[sz], close[sz] ;

void dfs(int u,int p,int dep) {
   depth[u] = dep ;
   par[u][0] = p ;
   arr[++timer] = u ;
   strt[u] = timer ;

   for(int v : graph[u]) {
      if(v == p)
         continue;
      dfs(v,u,dep+1);
   }

   arr[++timer] = u ;
   close[u] = timer;
}

void build_lca(int n) {
   // build LCA Table
}

int lca(int u,int v) {
   return LCA ;
}

struct data {
   int l, r, box, id, ca ;

   data() {}

   data(int l,int r,int box,int id,int ca) {
      this->l = l ;
      this->r = r ;
      this->box = box ;
      this->id = id ;
      this->ca = ca ;
   }
};
bool cmp(data a,data b) {
   if(a.box == b.box)
      return a.r<b.r ;

   return a.box < b.box ;
}

data q_list[sz] ;
long sum, ans[sz] ;
bool flag[sz] ;
```

```
void add_list(int u) {
   // if flag[u] == false you should add this node
   // otherwise remove this node

   flag[u] = !flag[u] ;
}

void remove_list(int u) {
   // if flag[u] == false you should add this node
   // otherwise remove this node
}
void solve(int n) {
   memset(par,-1,sizeof par);
   dfs(1,-1,0);
   build_lca(n);

   int q, box = sqrt(timer) + 3, p ;

   scanf("%d",&q);

   for(int i=0 ; i<q ; i++) {
      scanf("%d %d",&u,&v);

      if(strt[v]<strt[u])
         swap(u,v);

      p = lca(u,v);

      q_list[i].ca = p ;
      q_list[i].id = i ;

      if(p == u) {
         q_list[i].l = strt[u] ;
         q_list[i].r = strt[v] ;
         q_list[i].box = q_list[i].l/box ;
      }

      else {
         q_list[i].l = close[u] ;
         q_list[i].r = strt[v] ;
         q_list[i].box = q_list[i].l/box ;
      }
   }

   sort(q_list,q_list+q,cmp);

   int l = 1, r = 1 ;

   for(int i=0 ; i<q ; i++) {
      while(r<=q_list[i].r) {
         add_list(arr[r]);
         r++;
      }

      while(r>q_list[i].r+1) {
         remove_list(arr[r-1]);
         r--;
      }

      while(l>q_list[i].l) {
         add_list(arr[l-1]);
         l--;
```

```
    }

    while(l<q_list[i].l) {
      remove_list(arr[l]);
      l++;
    }

    ans[q_list[i].id] = sum  ; // here sum is ans
  }
}
```

28. Centroid Decomposition

```
#define sz 100000

struct Centroid_Decomposition {
  int node, lgn ;
  vector <vector <ll>> weight ;
  vector <vector <int>> par, graph ;
  vector <int> depth, subtree, cenp ;
  ll cost[sz], ans[sz], total_node ;
  const ll inf = 1e16 ;
  bool complete[sz] ;
  Centroid_Decomposition(int node):graph(node+7), weight(node+7),
depth(node+7), subtree(node+7), cenp(node+7) {
    this->node = node ;
    lgn = 1 + log2(node) ;
    par = vector <vector <int>>(node+7, vector <int>(lgn, -1));
    for(int i=0 ; i<=node ; i++)
      ans[i] = inf, complete[i] = false ;
  }
  void add_edge(int u,int v,ll w = 1) {
    graph[u].push_back(v);
    weight[u].push_back(w);
  }
  void dfs0(int u,int p,int dep,ll cst) {
    par[u][0] = p ;
    depth[u] = dep ;
    cost[u] = cst ;
    int v ;
    for(int i=0 ; i<graph[u].size() ; i++) {
      v = graph[u][i] ;
      if(v == p) continue;
      dfs0(v,u,dep+1,cst+weight[u][i]);
    }
  }
  void build_lca() {
    for(int i=1 ; i<lgn ; i++) {
      for(int j=1 ; j<=node ; j++) {
        if(par[j][i-1] != -1)
          par[j][i] = par[par[j][i-1]][i-1] ;
      }
    }
  }
  int lca(int u,int v) {
    if(depth[u] < depth[v])
      swap(u,v);
    for(int i=lgn-1 ; i>=0 ; i--) {
      if(par[u][i] != -1 && depth[par[u][i]] >= depth[v])
        u = par[u][i] ;
    }
    if(u == v)
      return u ;
```

```
    for(int i=lgn-1 ; i>=0 ; i--) {
        if(par[u][i] != par[v][i]) {
            u = par[u][i] ;
            v = par[v][i] ;
        }
    }
    return par[u][0] ;
}
ll dist(int u, int v) {
    int ca = lca(u,v);
    ll ans = cost[u] + cost[v] - (ll)2*cost[ca] ;
    return ans ;
}
void cal_subtree(int u,int p) {
    int v ;
    total_node++;
    subtree[u] = 1 ;
    for(int i=0 ; i<graph[u].size() ; i++) {
        v = graph[u][i] ;
        if(v != p && complete[v] == false) {
            cal_subtree(v,u);
            subtree[u] += subtree[v] ;
        }
    }
}
int centroid(int u,int p) {
    int v ;
    for(int i=0 ; i<graph[u].size() ; i++) {
        v = graph[u][i] ;
        if(v != p && complete[v] == false &&
                    subtree[v] > total_node/2LL)
            return centroid(v,u);
    }
    return u ;
}
void decomposition(int u,int p) {
    total_node = 0 ;
    cal_subtree(u,p);
    int cen = centroid(u,p);
    if(p == -1)
        p = cen ;
    cenp[cen] = p ;
    complete[cen] = true ;
    int v ;
    for(int i=0 ; i<graph[cen].size() ; i++) {
        v = graph[cen][i] ;
        if(v != p && complete[v] == false)
            decomposition(v,cen);
    }
}
void update(int u) {
    int x = u ;
    while(true) {
        ans[x] = min(ans[x],dist(u,x));
        if(x == cenp[x])
            return ;
        x = cenp[x] ;
    }
}
ll query(int u) {
    int x = u ;
    ll ret = 1000000000000000LL ;
```

```
    while(true) {
      ret = min(ret,ans[x]+dist(u,x));
      if(x == cenp[x])
        return ret ;
      x = cenp[x] ;
    }
  }
};
```

29. DSU on Tree

```
ll ans[sz], sum, mxm ;
bool big[sz] ;
int color[sz], cont[sz], sub[sz] ;
vector<int> graph[sz] ;
void cal_subtree(int u,int p) {
  int v ;
  sub[u] = 1 ;
  for(int i=0 ; i<graph[u].size() ; i++) {
    v = graph[u][i] ;
    if(v == p)
      continue ;
    cal_subtree(v,u);
    sub[u] += sub[v] ;
  }
}
void add(int u,int p,int x) {
  cont[color[u]] += x ;
  if(cont[color[u]] > mxm) {
    sum = color[u] ;
    mxm = cont[color[u]] ;
  }
  else if(cont[color[u]] == mxm)
    sum += color[u] ;
  int v ;
  for(int i=0 ; i<graph[u].size() ; i++) {
    v = graph[u][i] ;
    if(v == p or big[v])
      continue;
    add(v,u,x);
  }
}
void dsu(int u,int p,bool keep) {
  int v, mx = -1, bc = -1 ;
  for(int i=0 ; i<graph[u].size() ; i++) {
    v = graph[u][i] ;
    if(v == p)
      continue ;
    if(sub[v] > mx) {
      mx = sub[v] ;
      bc = v ;
    }
  }
  for(int i=0 ; i<graph[u].size() ; i++) {
    v = graph[u][i] ;
    if(v == p or v == bc)
      continue;
    dsu(v,u,0);
  }
  if(bc != -1) {
    dsu(bc,u,1);
    big[bc] = 1 ;
  }
```

```
   add(u,p,1);
   ans[u] = sum ;
   if(bc != -1)
      big[bc] = 0 ;
   if(keep == false) {
      add(u,p,-1);
      mxm = 0 ;
      sum = 0 ;
   }
}
```

30. K-D tree+ KNN (K-nearest neighbour)

```
//it can be viewed as multidimensional binary search tree
//dimension 0 based
//all distance are euclidian distance
#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;
      //can be done in nlogn time by making optimizing here
   sort(arr+st,arr+end+1,comp);
      //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 k+logn complexity (not sure)
//kth nearest
void KNN(int bstnode,int bstindex,int depth,co query,int
k,priority_queue<LL> &Q)
{
  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;
}
```

```
//normally logn complexity
//number of points with sqrt(high) distance
void KNN2(int bstnode,int bstindex,int depth,co &query,LL
high,priority_queue<LL> &Q)
{
  if(!bstnode) return;
  //if(bstnode)
debug(bst[bstnode].now.x[0],bst[bstnode].now.x[1],bst[bstnode].now.x[
2]);
  if(bstnode>bstindex) return;
  Q.push(dist(bst[bstnode].now,query));
  if(Q.top()>=high) Q.pop();

  axis=depth%dimension;
  LL chc=bst[bstnode].now.x[axis]-query.x[axis];
  //print2(chc,query.x[0]);
  if(chc>=0) //go to left
  {
    KNN(bst[bstnode].left,bstindex,depth+1,query,high,Q);
    //special attention to > sign (sometimes >=)
    if(chc*chc<high)  //there is a chance of less
      KNN(bst[bstnode].right,bstindex,depth+1,query,high,Q);
    return;
  }

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

```
    return;
}
int main(){
    //freopen("A.in","r",stdin);
    //freopen("A.out","w",stdout);
    int n,k;
    while(cin>>n>>k &&(n||k))
    {
        int i;
        for(int i = 1; i <= n; i++)
            scanf("%lld %lld %lld",&arr[i].x[0],&arr[i].x[1],&arr[i].x[2]);
        int bstindex=0;
        kdtree(arr,1,n,0,bstindex);
        int ans=0;
        for(i=1;i<=n;i++)
        {
            priority_queue<LL>q;
            KNN(1,bstindex,0,arr[i],(LL)k*(LL)k,q);
            ans+=q.size()-1;
        }
        ans/=2;
        printf("%d\n",ans);
    }
    return 0;
}
/*
Input :-
7 2
0 0 0
1 0 0
1 2 0
1 2 3
1000 1000 1000
1001 1001 1000
1001 999 1001
7 3
0 0 0
1 0 0
1 2 0
1 2 3
-1000 1000 -1000
-1001 1001 -1000
-1001 999 -1001
7 4
0 0 0
1 0 0
1 2 0
1 2 3
1000 -1000 1000
1001 -1001 1000
1001 -999 1001
Output :
3
6
9
*/
```

31. Splay Tree

```
template< typename T, typename Comp = std::less< T > >
class splay_tree {
private:
    Comp comp;
```

```
unsigned long p_size;
struct node {
   node *left, *right;
   node *parent;
   T key;
   node( const T& init = T( ) ) : left( 0 ), right( 0 ), parent( 0 ), key( init ) { }
} *root;
void left_rotate( node *x ) {
   node *y = x->right;
   x->right = y->left;
   if( y->left ) y->left->parent = x;
   y->parent = x->parent;
   if( !x->parent ) root = y;
   else if( x == x->parent->left ) x->parent->left = y;
   else x->parent->right = y;
   y->left = x;
   x->parent = y;
}

void right_rotate( node *x ) {
   node *y = x->left;
   x->left = y->right;
   if( y->right ) y->right->parent = x;
   y->parent = x->parent;
   if( !x->parent ) root = y;
   else if( x == x->parent->left ) x->parent->left = y;
   else x->parent->right = y;
   y->right = x;
   x->parent = y;
}
void splay( node *x ) {
   while( x->parent ) {
      if( !x->parent->parent ) {
         if( x->parent->left == x ) right_rotate( x->parent );
         else left_rotate( x->parent );
      } else if( x->parent->left == x && x->parent->parent->left == x->parent ) {
         right_rotate( x->parent->parent );
         right_rotate( x->parent );
      } else if( x->parent->right == x && x->parent->parent->right == x->parent ) {
         left_rotate( x->parent->parent );
         left_rotate( x->parent );
      } else if( x->parent->left == x && x->parent->parent->right == x->parent ) {
         right_rotate( x->parent );
         left_rotate( x->parent );
      } else {
         left_rotate( x->parent );
         right_rotate( x->parent );
      }
   }
}
void replace( node *u, node *v ) {
   if( !u->parent ) root = v;
   else if( u == u->parent->left ) u->parent->left = v;
   else u->parent->right = v;
   if( v ) v->parent = u->parent;
}
node* subtree_minimum( node *u ) {
   while( u->left ) u = u->left;
   return u;
```

```
  }
  node* subtree_maximum( node *u ) {
    while( u->right ) u = u->right;
    return u;
  }
public:
  splay_tree( ) : root( 0 ), p_size( 0 ) { }
  void insert( const T &key ) {
    node *z = root;
    node *p = 0;
    while( z ) {
      p = z;
      if( comp( z->key, key ) ) z = z->right;
      else z = z->left;
    }
    z = new node( key );
    z->parent = p;
    if( !p ) root = z;
    else if( comp( p->key, z->key ) ) p->right = z;
    else p->left = z;
    splay( z );
    p_size++;
  }
  node* find( const T &key ) {
    node *z = root;
    while( z ) {
      if( comp( z->key, key ) ) z = z->right;
      else if( comp( key, z->key ) ) z = z->left;
      else return z;
    }
    return 0;
  }
  void erase( const T &key ) {
    node *z = find( key );
    if( !z ) return;
    splay( z );
    if( !z->left ) replace( z, z->right );
    else if( !z->right ) replace( z, z->left );
    else {
      node *y = subtree_minimum( z->right );
      if( y->parent != z ) {
        replace( y, y->right );
        y->right = z->right;
        y->right->parent = y;
      }
      replace( z, y );
      y->left = z->left;
      y->left->parent = y;
    }
    p_size--;
  }

  const T& minimum( ) {
    return subtree_minimum( root )->key;
  }
  const T& maximum( ) {
    return subtree_maximum( root )->key;
  }
  bool empty( ) const {
    return root == 0;
  }
  unsigned long size( ) const {
```

```
        return p_size;
    }
};
```

32. Rectangle Union Without Compress

```
/*    for initialize tree set :
      tree[1] = 0 ;
      prop[1] = -1 ;          */
long long tree[mx_coordinate*4], prop[mx_coordinate*4];

void relax(int nd,int l,int r) {
    if(l != r && prop[nd] == -1) {
      int lc = 2*nd, rc = lc + 1 ;
      tree[nd] = 0 ;
      tree[lc] = 0 ;
      tree[rc] = 0 ;
      prop[lc] = -1 ;
      prop[rc] = -1 ;
    }
}
void update(int nd,int l,int r,int ql,int qr,int v) {
    if(r<ql || l>qr)
      return ;
    relax(nd,l,r);
    int lc = 2*nd, rc = lc + 1 ;

    if(l>=ql && r<=qr) {
      prop[nd] = max(0LL,prop[nd]) + v ;
//printf("%d %d %lld\n",l,r,prop[nd]);
      if(prop[nd]>0)
        tree[nd] = r-l+1 ;
      else {
        if(l != r) {
          prop[nd] = (prop[lc] == -1 && prop[rc] == -1 ? -1 : 0) ;
          tree[nd] = (prop[nd] == -1 ? 0 : tree[lc] + tree[rc]);
        }
        else {
          prop[nd] = -1 ;
          tree[nd] = 0 ;
        }
      }
      return ;
    }
    int mid = (l+r)/2 ;
    update(lc,l,mid,ql,qr,v);
    update(rc,mid+1,r,ql,qr,v);
    if(prop[nd] <= 0) {
      if(prop[lc] == -1 && prop[rc] == -1)
        prop[nd] = -1 ;

      else
        prop[nd] = max(prop[nd],0LL);
    }
    tree[nd] = (prop[nd] == -1 ? 0 : (prop[nd] == 0 ?
                                      tree[lc] + tree[rc] : r-l+1));
}
```

33. Rectangle Union Compress

```
vector<int> xpoints, ypoints ;
pair<int,int> tree[100000+7] ; // fs sum , sc prop
int is_seg[100000+7] ;
```

```
void relax(int nd,int l,int r,int v) {
   tree[nd].sc += v ;
   tree[nd].fs = 0 ;

   if(tree[nd].sc>0)
      tree[nd].fs = ypoints[r-1] - ypoints[l-1] ;

   else if(l != r) {
      tree[nd].fs = tree[2*nd].fs + tree[2*nd + 1].fs ;

      if(is_seg[nd]) {
         int mid = (l+r)/2 ;
         tree[nd].fs += ypoints[mid] - ypoints[mid-1] ;
      }
   }
}
void update(int nd,int l,int r,int i,int j,int v) {
   if(r<i || l>j)
      return ;

   if(l>=i && r<=j) {
      relax(nd,l,r,v);
      return ;
   }

   int mid = (l+r)/2, lc = 2*nd, rc = lc + 1 ;

   update(lc,l,mid,i,j,v);
   update(rc,mid+1,r,i,j,v);

   tree[nd].fs = tree[lc].fs + tree[rc].fs ;

   if(tree[nd].sc>0)
      tree[nd].fs = ypoints[r-1] - ypoints[l-1] ;

   if(i<=mid && j>=mid+1)
      is_seg[nd] += v ;

   if(is_seg[nd]>0 && tree[nd].sc == 0)
      tree[nd].fs += ypoints[mid] - ypoints[mid-1] ;
}
```

34. Li chao (Convex Hull Trick With Segment Tree)

```
// initially root is -1
// l = minimum possible value of x
// r = maximum possible value of x
// this is calculate for minimum
// for maximum change < sign
struct data {
   ll m, c ;
   int l, r ;
   data() {
      m = 0 ;
      c = 0 ;
      l = -1 ;
      r = -1 ;
   }
   data(ll m,ll c) {
      this->m = m ;
      this->c = c ;
      l = -1 ;
      r = -1 ;
```

```
    }
    ll cal_y(ll x) {
        return m*x + c ;
    }
};
data tree[MAXN] ;
bool vis[MAXN] ;
int id ;
int update(int nd,int l,int r,data line) {
    if(nd == -1) {
        tree[id] = line ;
        return id++;
    }

    if(tree[nd].cal_y(l)<=line.cal_y(l) && tree[nd].cal_y(r)<=line.cal_y(r))
        return nd ;

if(line.cal_y(l)<=tree[nd].cal_y(l) && line.cal_y(r) <=      tree[nd].cal_y(r)) {
        tree[id] = tree[nd] ;
        tree[id].c = line.c ;
        tree[id].m = line.m ;
        return id++;
    }

    int mid = (l+r)/2, lc = tree[nd].l, rc = tree[nd].r ;
    int nnd = id++ ;

    tree[nnd] = tree[nd] ;

    if(tree[nnd].cal_y(l)>line.cal_y(l))
        swap(tree[nnd].c,line.c), swap(tree[nnd].m,line.m) ;

    if(tree[nnd].cal_y(mid) <= line.cal_y(mid))        // Be careful about this
condition
        tree[nnd].r = update(tree[nnd].r,mid+1,r,line) ;

    else {
        swap(tree[nnd].c,line.c), swap(tree[nnd].m,line.m) ;
        tree[nnd].l = update(tree[nnd].l,l,mid,line);
    }

    return nnd ;
}

ll query(int nd,int l,int r,int x) {
    if(nd == -1)
        return inf ;

    int mid = (l+r)/2 ;

    if(mid>=x) // Be careful about this condition
        return min(tree[nd].cal_y(x),query(tree[nd].l,l,mid,x));

    return min(tree[nd].cal_y(x),query(tree[nd].r,mid+1,r,x));
}
```

35. MO's with Update

```
/**
Given an array. Two types of operation are supported .
0 A B -> summation of unique numbers which are divisible by 3
1 A B -> change value of index A to B
**/
```

```
const int sz = 200000+7 ;
map<int,int> mmap ;
int inp[sz], uind, qind, block_size, updates[sz], tarr[sz], freq[sz] ;
long long ans, qans[sz] ;
pair<int,int> updates_value[sz] ;
// first means previous , second means now
int rev[sz] ;
struct data {
   int l, r, tym, ind ;
   data() {}
   data(int l,int r,int tym,int ind) : l(l), r(r), tym(tym), ind(ind) {}
};
data query[sz] ;
bool cmp(data a,data b) {
   int b1 = a.l/block_size ;
   int b2 = b.l/block_size ;
   if(b1 == b2) {
      b1 = a.r/block_size ;
      b2 = b.r/block_size ;
      if(b1 == b2)
         return a.tym < b.tym ;
      return a.r < b.r ;
   }
   return a.l < b.l ;
}
void PUpdate(int ind,int l,int r) {
   int a ;
   if(updates[ind]>=l && updates[ind]<=r) {
      a = updates_value[ind].first ;
      freq[a]--;
      a = updates_value[ind].second ;
      freq[a]++;
   }
   inp[updates[ind]] = updates_value[ind].second ;
}
void UUpdate(int ind,int l,int r) {
   int a ;
   if(updates[ind]>=l && updates[ind]<=r) {
      a = updates_value[ind].second ;
      freq[a]--;

      a = updates_value[ind].first ;
      freq[a]++;
   }
   inp[updates[ind]] = updates_value[ind].first ;
}

void add(int ind) {
   int a = inp[ind] ;
   freq[a]++;
}

void rmv(int ind) {
   int a = inp[ind] ;
   freq[a]--;
}
void solve(int n,int q) {
   int typ, l, r ;
   qind = 0 ;
   uind = 0 ;
   for(int i=0 ; i<q ; i++) {
```

```
    scanf("%d",&typ);
    scanf("%d %d",&l,&r);
    if(typ == 0) { // updates query
        updates[uind] = l ;
        updates_value[uind] = make_pair(tarr[l],r) ;
        tarr[l] = r ;
        uind++;
    }
    else
        query[qind] = data(l,r,uind,qind), qind++;
}
block_size = cbrt(n);
block_size = block_size*block_size ;
sort(query,query+qind,cmp);
int cur_time = 0 ;
l = 1 ;
r = 1 ;
for(int i=0 ; i<qind ; i++) {
    while(cur_time<query[i].tym) {
        PUpdate(cur_time,l,r-1);
        cur_time++;
    }

    while(cur_time>query[i].tym) {
        cur_time--;
        UUpdate(cur_time,l,r-1);
    }

    while(r<=query[i].r) {
        add(r);
        r++;
    }

    while(r>query[i].r+1) {
        r--;
        rmv(r);
    }

    while(l>query[i].l) {
        l--;
        add(l);
    }

    while(l<query[i].l) {
        rmv(l);
        l++;
    }
    qans[query[i].ind] = ans ;
  }
}
```

36. Kadane Algorithm of Maximum Sum (2-D)

```
// Maximum Sum Algo
int matrix[105][105], temp[110] ;
int finalleft, finalright, finaltop, finalbottom, n, start, finish ;
int main() {
  scanf("%d",&n);
  for(int i=0 ; i<n ; i++) {
    for(int j=0 ; j<n ; j++) {
      scanf("%d",&matrix[i][j]);
    }
  }
```

```
    find_maxsum();    return 0 ;
}
int kadane(int temp[]) {
    int sum = 0, maxsum = 0,local_start = 0 ;
    finish = -1 ;
    for(int i=0 ; i<n ; i++) {
        sum += temp[i];
        if(sum<0) {
            sum = 0 ;          local_start = i+1 ;
        }
        else if(sum>maxsum) {
            maxsum = sum ;          start = local_start ;
            finish = i ;
        }
    }
    if(finish != -1)       return maxsum;
    start = finish = 0 ;    sum = temp[0] ;
    for(int i=1 ; i<n ; i++) {
        if(temp[i]>sum) {
            sum = temp[i] ;          start = finish = i ;
        }
    }
    return sum ;
}
void find_maxsum() {
    int maxSum = 0, left, right, sum ;
    for(left = 0 ; left < n ; left++) {
        memset(temp,0,sizeof temp);
        for(right = left ; right<n ; right++) {
            for(int i=0 ; i<n ; i++)
                temp[i] += matrix[i][right];
            sum = kadane(temp);
            if(sum>maxSum) {
                maxSum = sum ;              finalleft = left ;
                finalright = right ;              finaltop = start ;
                finalbottom = finish ;
            }
        }
    }
    printf("%d\n",maxSum);
    printf("(Top, Left) (%d, %d)\n", finaltop, finalleft);
    printf("(Bottom, Right) (%d, %d)\n", finalbottom, finalright);
}
```

String Related Algorithm

37. Trie Tree

```
string str ;
int s_len, node[100000+5][53], id, cont[100000+5] ;
int new_node() {
   for(int i=0 ; i<52 ; i++)
      node[id][i] = 0 ;
   cont[id] = 0 ;
   return id++ ;
}
void add(int cur) {
   int a ;
   for(int i=0 ; i<s_len ; i++) {
      a = char_num(str[i]);
      if(node[cur][a] == 0)
         node[cur][a] = new_node();
      cur = node[cur][a] ;
   }
   cont[cur]++;
}
int query(int cur) {
   int a;
   for(int i=0 ; i<s_len ; i++) {
      a = char_num(str[i]);
      if(node[cur][a] == 0)
         return 0 ;
      cur = node[cur][a] ;
   }
   return cont[cur] ;
}
```

38. Trie XOR(Max/Min)

```
#define MAX 50001

struct trie {
   int cand[2];
   trie() {
      clrall(cand,-1);
   }
};

trie tree[MAX*32+7];
ll csum;
int tot_node;
void insert_trie(int root,ll val) {
   int i,j,k;
   int fbit;
   for(i = 31; i>=0; i--) {
      fbit=(int) ((val>>(ll) i)&1LL);
      if(tree[root].cand[fbit]==-1) {
         tree[root].cand[fbit] = ++tot_node;
      }
      root = tree[root].cand[fbit];
   }
   return ;
}

ll solve(int root,ll cval) {
   ll res=0;
   int fbit,cbit;
```

```
    int i,j,k;
    for(i = 31; i>=0; i--) {
        fbit=(int) ((cval>>(ll) i)&1LL);
        cbit=!(fbit);
        if(tree[root].cand[fbit]!=-1) {
            if(fbit) res|=(1LL << (ll) i);
            root=tree[root].cand[fbit];
        } else {
            if(cbit) res|=(1LL << (ll) i);
            root=tree[root].cand[cbit];
        }
    }
    return res;
}
ll max_val(ll val) {
    int i,j,k;
    ll ret=0;
    int gbit;
    for(i = 31; i>=0; i--) {
        gbit=(int) ((val>>(ll) i)&1LL);
        if(!gbit) ret|=(1LL << (ll) i);
    }
    return ret;
}
ll min_val(ll val) {
    int i,j,k;
    ll ret=0;
    int gbit;
    for(i = 31; i>=0; i--) {
        gbit=(int) ((val>>(ll) i)&1LL);
        if(gbit) ret|=(1LL << (ll) i);
    }
    return ret;
}
int main() {
    ll val;
    int test,cas=0,i,j,k,n;
    ll ansmx,ansmn;
    cin>>test;
    rep(i,0,MAX*32+7) tree[i]=trie();
    while(test--) {
        cin>>n;
        tot_node=0;
        csum=0LL;
        ansmx=0LL;
        ansmn=(1LL<<50LL);
        insert_trie(0,csum);
        rep(i,1,n+1) {
            cin>>val;
            csum=csum xor val;
            val=max_val(csum);
            ansmx=max(ansmx,solve(0,val) xor csum);
            val=min_val(csum);
            ansmn=min(ansmn,solve(0,val) xor csum);
            insert_trie(0,csum);
        }
        cas++;
        cout<<"Case "<<cas<<": "<<ansmx<<" "<<ansmn<<endl;
        rep(i,0,tot_node+4) tree[i]=trie();
    }
    return 0;
}
```

39. Suffix Array

```
#define MAX 100000

string text;
int revSA[MAX],SA[MAX];
int cnt[MAX], nxt[MAX];
bool bh[MAX],b2h[MAX];
int LCP[MAX];

bool cmp(const int &i,const int &j) {
   return text[i]<text[j];
}
void sortFirstChar(int n) {
   /// sort for the first char  ...
   for(int i =0 ; i<n ; i++)
      SA[i] = i;
   sort(SA,SA+n,cmp);

   ///indentify the bucket ........
   for(int i=0 ; i<n ; i++) {
      bh[i] = (i==0  || text[SA[i]]!=text[SA[i-1]]);
      b2h[i] = false;
   }
   return;
}
int CountBucket(int n) {
   int bucket = 0;
   for(int i =0,j; i<n ; i=j) {
      j = i+1;
      while(j<n && bh[j]==false) j++;
      nxt[i] = j;
      bucket++;
   }
   return bucket;
}
void SetRank(int n) {
   for(int i = 0 ; i<n ; i=nxt[i]) {
      cnt[i] = 0;
      for(int j =i ; j<nxt[i] ; j++) {
         revSA[SA[j]] = i;
      }
   }
   return;
}
void findNewRank(int l,int r,int step) {
   for(int j = l  ; j<r ; j++) {
      int pre = SA[j] - step;
      if(pre>=0) {
         int head = revSA[pre];
         revSA[pre] = head+cnt[head]++;
         b2h[revSA[pre]] = true;
      }
   }
   return;
}
void findNewBucket(int l,int r,int step) {
   for(int j = l  ; j<r ; j++) {
      int pre = SA[j] - step;
      if(pre>=0 && b2h[revSA[pre]]) {
         for(int k = revSA[pre]+1 ; b2h[k] && !bh[k] ; k++) b2h[k] = false;
      }
```

```
    }
    return;
}
void buildSA(int n) {
    ///start sorting in logn step ...
    sortFirstChar(n);
    for(int h =1 ; h<n ; h<<=1) {
        if(CountBucket(n)==n) break;
        SetRank(n);
        /// cause n-h suffix must be sorted
        b2h[revSA[n-h]] = true;
        cnt[revSA[n-h]]++;
        for(int i = 0 ; i<n ; i=nxt[i]) {
            findNewRank(i,nxt[i], h);
            findNewBucket(i, nxt[i], h);
        }
        ///set the new sorted suffix array ...
        for(int i =0 ; i<n ; i++) {
            SA[revSA[i]] = i;
            bh[i] |= b2h[i]; ///new bucket ....
        }
    }
    return;
}

void buildLCP(int n) {
    int len = 0;
    for(int i = 0 ; i<n ; i++)
        revSA[SA[i]] = i;
    for(int i =0 ; i< n ; i++) {
        int k = revSA[i];
        if(k==0) {
            LCP[k] = 0;
            continue;
        }
        int j = SA[k-1];
        while(text[i+len]==text[j+len]) len++;
        LCP[k] = len;
        if(len) len--;
    }
    return;
}

void printSA(int n) {
    for(int i=0; i<n; i++) printf("%2d ",SA[i]),cout<<text.substr(SA[i])<<endl;
    puts("");
    for(int i=1; i<n; i++) printf("%2d\n",LCP[i]);
    puts("");
    return ;
}
int main() {
    string a,b;
    int n,p,q;
    int tcase,cas=1;
    scanf("%d",&tcase);
    while(tcase--) {
//      cin>>a>>b;
//      text=a+"$"+b;
        cin>>text;
        buildSA(SZ(text));
        buildLCP(SZ(text));
        printSA(SZ(text));
```

```
    int r=0;
    int n=SZ(text);
    for(int i=0; i<n; i++) {
      r+=(n-i);
      r-=LCP[i];
    }
    deb(r);
  }
  return 0;
}
```

40. Suffix Automata

```
class Automata {
public:
  struct data {
    int link, len ;
    int next[27] ;

    data() {}
    data(int link,int len) : link(link), len(len) {}
  };
  data *node ;
  int num, last ;
  void reset() {
    num = 1 ;
    node[0].link = -1 ;
    node[0].len = 0 ;
    last = 0 ;
    memset(node[0].next,0,sizeof(node[0].next));
  }
  Automata() {}
  Automata(int mx_len) {
    mx_len += 7 ;
    mx_len = mx_len*2 ;
    node = new data[mx_len] ;
    reset();
  }
  void addLetter(char ch) {
    int cur = num++;
    int let = ch - 'a' ;
    int p = last ;
    node[cur].len = node[last].len + 1 ;
    memset(node[cur].next,0,sizeof(node[cur].next));
    for(p=last ; p != -1 && !node[p].next[let] ; p = node[p].link)
      node[p].next[let] = cur ;
    if(p == -1)
      node[cur].link = 0 ;
    else {
      int q = node[p].next[let] ;
      if(node[p].len+1 == node[q].len)
        node[cur].link = q ;
      else {
        int clone = num++;
        node[clone] = node[q] ;
        node[clone].len = node[p].len + 1 ;
        for(; p != -1 && node[p].next[let] == q ; p = node[p].link)
          node[p].next[let] = clone ;
        node[q].link = node[cur].link = clone ;
      }
    }
    last = cur ;
  }
```

```
};
```

41. Suffix Tree

```
int num[sz], match, node, n_d, graph[30*sz][30], link[30*sz] ;
jora_int edge[30*sz] ;
int add_edge(int st,int ln) {
   edge[n_d].fs = st ;
   edge[n_d].sc = ln ;
   for(int i=0 ; i<30 ; i++)
      graph[n_d][i] = 0 ;
   return n_d++;
}
void _jump(int pos) {
   while(match > edge[graph[node][num[pos+1-match]]].sc) {
      node = graph[node][num[pos+1-match]] ;
      match -= edge[node].sc ;
   }
}
void add_char(int pos) {
   int last = 0, a = num[pos], cur_ed, m_ed, u ;
   match++;
   while(match>0) {
      _jump(pos);
      cur_ed = num[pos - match + 1] ;
      int& v = graph[node][cur_ed] ;
      m_ed = num[edge[v].fs + match - 1] ;
      if(v == 0) {
         v = add_edge(pos-match+1,Max);//deb(pos,v);
         link[last] = node ;
         last = 0 ;
      } else if(a == m_ed) {
         link[last] = node ;
         return ;
      } else {
         u = add_edge(edge[v].fs,match-1);
         graph[u][a] = add_edge(pos,Max);
         graph[u][m_ed] = v ;
//if(edge[v].sc<match-1) deb(v,edge[v].sc,pos,match),wait;
         edge[v].fs += match - 1;
         edge[v].sc -= match - 1;
         v = u ;
         link[last] = u ;
         last = u ;
      }
      if(node == 0)
         match--;
      else
         node = link[node] ;
   }
}
void reset() {
   node = 0 ;
   n_d = 0 ;
   match = 0 ;
   add_edge(0,Max);
}
void print(int nd) {
   if(nd)
      pf("%d %d %d\n",nd,edge[nd].fs,edge[nd].sc);
   for(int i=0 ; i<29 ; i++) {
      if(graph[nd][i] > 0) {
         deb(nd,i);
```

```
            print(graph[nd][i]);
        }
    }
}
int main() {
    string str ;
    cin>>str ;
    reset();
    for(int i=0 ; i<str.length() ; i++) {
        num[i] = str[i] - 'a' ;
        add_char(i);
    }
    print(0);
    return 0 ;
}
```

42. KMP

```
vector<int> prefix_cal(char str[]) {
    int l = strlen(str+1) ;
    vector<int>prefix(l+1) ;
    prefix[1] = 0 ;
    int k = 0 ;
    for(int i=2 ; i<=l ; i++) {
        while(k>0 and str[i] != str[k+1])
            k = prefix[k];
        if(str[k+1] == str[i])
            k++;
        prefix[i] = k ;
    }
    return prefix;
}
vector<int> match_prefix(char par[],char str[]) {
    int l1 = strlen(str+1), l2 = strlen(par+1), k = 0 ;
    vector<int>prefix, match ;
    prefix = prefix_cal(par) ;
    for(int i = 1 ; i<=l1 ; i++) {
        while(k>0 and str[i] != par[k+1])
            k = prefix[k];
        if(str[i] == par[k+1])
            k++;
        if(k == l2) {
            match.pb(i-k);
            k = prefix[k] ;
        }
    }
    return match ;
}
```

43. Minimum Expression and ExKmp

```
int nxt[N],ex_a[N],exb[N];
void getnext(char *s) {
    int len = strlen(s),cur = 0;
    nxt[0] = len;
    while(cur < len&&s[cur]==s[cur+1])cur++;
    nxt[1] = cur;
    cur = 1;
    for(int k = 2; k<len; k++) {
        int p = cur + nxt[cur] - 1,L = nxt[k-cur];
        if(k + L -1 >= p) {
            int j = (p-k+1)>0?(p-k+1):0;
            while(k+j<len&&s[k+j]==s[j])j++;
            nxt[k] = j;
```

```
            cur = k;
        } else
            nxt[k] = L;
    }
}
/*  exkmp return match for each position between strings
suffix starts from this position and pattern  */
void exkmp(char *s1,char *s2,int *ex) { // s1 is main string, s2 pattern
    getnext(s2);
    int l1 = strlen(s1),l2 = strlen(s2),cur = 0;
    while(cur < min(l1,l2)&&s1[cur]==s2[cur])cur++;
    ex[0] = cur;
    cur = 0;
    for(int k = 1; k < l1; k++) {
        int p = cur + ex[cur] - 1,L = nxt[k-cur];
        if(k + L -1 >= p) {
            int j = (p-k+1)>0?(p-k+1):0;
            while(k+j<l1&&j<l2&&s1[k+j]==s2[j])j++;
            ex[k] = j;
            cur = k;
        } else
            ex[k] = L;
    }
}
int MinRep(char *s) { // return position from where this cyclic string is
Lexicographical minimum
    int i = 0,j = 1,k = 0,t,len = strlen(s);
    while(i<len&&j<len&&k<len) {
        t = s[(i+k)%len] - s[(j+k)%len];
        if(t==0)k++;
        else {
            if(t>0)
                i += k + 1;
            else
                j += k + 1;
            if(i==j)j++;
            k = 0;
        }
    }
    return min(i,j);
}
```

44. Aho Chorasic

```
const int MAXN = 404, MOD = 1e9 + 7, sigma = 26;
int term[MAXN], len[MAXN], to[MAXN][sigma], link[MAXN], sz = 1;
void add_str(string s) {
    int cur = 0;
    for(auto c: s) {
        if(!to[cur][c - 'a']) {
            to[cur][c - 'a'] = sz++;
            len[to[cur][c - 'a']] = len[cur] + 1;
        }
        cur = to[cur][c - 'a'];
    }
    term[cur] = cur ;
}
void push_links() {
    int que[sz];
    int st = 0, fi = 1;
    que[0] = 0;
    while(st < fi) {
        int V = que[st++];
```

```
      int U = link[V];
      if(!term[V]) term[V] = term[U];
      for(int c = 0; c < sigma; c++)
         if(to[V][c]) {
            link[to[V][c]] = V ? to[U][c] : 0;
            que[fi++] = to[V][c];
         } else {
            to[V][c] = to[U][c];
         }
   }
}
```

45. Manachers

```
char str[sz], fstr[2*sz] ;
int len, p[2*sz] ;
void reset() {
   len = strlen(str);
   fstr[0] = '^' ;
   p[0] = 0 ;
   int k = 1 ;
   for(int i=0 ; i<len ; i++) {
      p[k] = 0 ;
      fstr[k++] = '#';
      p[k] = 0 ;
      fstr[k++] = str[i] ;
   }
   fstr[k++] = '#' ;
   fstr[k++] = '$' ;
   len = len*2 + 2 ;
}
int manchers() {
   int r = 0, c = 0, miror ;
   int mx = 0 ;
   for(int i=1 ; i<len ; i++) {
      miror = 2*c - i ;
      p[i] = r > i ? min(r-i,p[miror]) : 0 ;
      while(fstr[i+1+p[i]] == fstr[i-1-p[i]])
         p[i]++;
      if(i+p[i] > r) {
         c = i ;
         r = i+p[i] ;
      }
      mx = max(mx,p[i]);
   }
   return mx;
}
```

46. Palindromic Tree

```
class palindromicTree {
public:
   struct node {
      int next[26] ;
      int len ;
      int suffixLink ;
      ll cont ;
   };
   int len ;
   char *str ;
   node *tree ;
   int num ;
   int suff ;
   palindromicTree() {}
```

```
palindromicTree(int MX_NODE) {
   str = new char[MX_NODE] ;
   tree = new node[MX_NODE] ;
}
bool addLetter(int pos) {
   int cur = suff, curlen = 0 ;
   int let = str[pos] - 'a' ;
   while(true) {
      curlen = tree[cur].len ;
      if(pos-1-curlen >=0 && str[pos-1-curlen] == str[pos])
         break;
      cur = tree[cur].suffixLink ;
   }
   if(tree[cur].next[let]) {
      suff = tree[cur].next[let] ;
      return false ;
   }
   num++;
   memset(tree[num].next,0,sizeof(tree[num].next));
   suff = num ;
   tree[num].len = tree[cur].len + 2 ;
   tree[cur].next[let] = num ;
   if(tree[num].len == 1) {
      tree[num].suffixLink = 1 ;
      tree[num].cont = 1 ;
      return true ;
   }
   while(true) {
      cur = tree[cur].suffixLink ;
      curlen = tree[cur].len ;
      if(pos-1-curlen >= 0 && str[pos-1-curlen] == str[pos]) {
         tree[num].suffixLink = tree[cur].next[let] ;
         break;
      }
   }

   tree[num].cont = tree[tree[num].suffixLink].cont+1 ;
   return true ;
}

void iniTree() {
   memset(tree[0].next,0,sizeof(tree[0].next));
   memset(tree[1].next,0,sizeof(tree[1].next));

   num = suff = 1 ;
   tree[0].len = -1 ;
   tree[0].suffixLink = 0 ;
   tree[1].len = 0 ;
   tree[1].suffixLink = 0 ;
}
void add() {
   for(int i=0 ; i<len ; i++)
      addLetter(i);
}
void cal_count() {
   for(int i=num ; i>0 ; i--)
      tree[tree[i].suffixLink].cont += tree[i].cont ;
}
};
```

Dynamic Programming Optimization

47. Notes

Convex Hull Optimization1	$dp[i] = min_{j<ij}\{ dp[k] + b[k] * a[j]\}$	$b[k] \geq b[k+1]$ ~~optionally~~ $a[j] \leq a[j+1]$	$O(n^2)$	$O(n)$
Convex Hull Optimization2	$dp[i][j] = min_{k<j}\{ dp[i-1][k] + b[k] * a[j]\}$		$O(kn^2)$	$O(kn)$
Divide and Conquer Optimization	$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j+1]$	$O(kn^2)$	$O(knlogn)$
Knuth Optimization	$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$	$A[i,j-1] \leq A[i,j] \leq A[i+1,j]$	$O(n^3)$	$O(n^2)$

Notes:

- $A[i][j]$ — the smallest k that gives optimal answer, for example in $dp[i][j] = dp[i-1][k] + C[k][j]$
- $C[i][j]$ — some given cost function
- We can generalize a bit in the following way: $dp[i] = min_{j<i}\{F[j] + b[j] * a[i]\}$, where $F[j]$ is computed from $dp[j]$ in constant time.
- It looks like **Convex Hull Optimization2** is a special case of **Divide and Conquer Optimization**.
- It is claimed (in the references) that **Knuth Optimization** is applicable if $C[i][j]$ satisfies the following 2 conditions:
  quadrangle inequality: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c],\ a \leq b \leq c \leq d$
  monotonicity: $C[b][c] \leq C[a][d],\ a \leq b \leq c \leq d$

48. CovexHull Trick 1D

```
struct cline {
    ll M, C;
    cline() {}
    cline(ll m, ll c): M(m), C(c) {}
```

```
};
//pointer=0,last=0 should be made initially
cline line[MX]; //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(const cline & l1,const cline & l2,const cline & 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*/
   //if the query x values is non-decreasing (reverse(> sign) for vice verse)
   return (l3.C-l1.C)*(l1.M-l2.M)<=(l2.C-l1.C)*(l1.M-l3.M);}
//Adding should be done serially
//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
void add(cline l,int &last) {
   //First, let's add it to the end
   line[last++]=l;
   //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(line[last-3],line[last-2],line[last-1])) {
      line[last-2]=line[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
   // Min Value wanted... (reverse(> sign) for max value)
   while (pointer<last-1 &&line[pointer+1].M*x+line[pointer+1].C
<=line[pointer].M*x+line[pointer].C)
      pointer++;
   return line[pointer].M*x+line[pointer].C;
}
//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;
// Min Value wanted... (reverse(> sign) for max value if(mid+1<last &&
line[mid+1].M*x+line[mid+1].C <line[mid].M*x+line[mid].C)
         return bs(mid+1,end,x,last);
   // Min Value wanted... (reverse(> sign) for max value)
   if(mid-1>=0 && line[mid-1].M*x+line[mid-
1].C<line[mid].M*x+line[mid].C)
return bs(st,mid-1,x,last);
   return line[mid].M*x+line[mid].C;
}
```

```
int main() {
   int last = 0, pointer = 0;
   return 0;
}
```

49. Covexhull Trick 2D

```
ll dp[MX][2];
ll func(int n, int p) {
   ll CostOfPartition;
   for(int i = 1; i <= n; i++)
      dp[i][1] = CostOfPartition;
   for(int k = 2; k <= p; k++) {
      int last = 0, pointer = 0;
      int cur = k&1;       int prv = (k-1)&1;
      for(int i = k; i <= n; i++) {
         ll M, C, CC, X;
          // M = slope of line    C = line constant factor
         add(cline(M,C),last);
          // X = value of query   CC = Extra cost for this partition
         dp[i][cur] = query(X,pointer,last)+CC;
      }
   }
   return dp[n][p&1];
}
```

50. Divide and Conquer

```
int arr[MX][MX];
int sum[MX][MX];
int cost[MX][MX];
void build(int n) {
   //cost calculation
}
int dp[MX][MX];
int calc(int divs,int pos, int searchL, int searchR) {
   dp[divs][pos] = inf;    int ret = searchL;
   for(int i = searchL; i <= searchR; i++) {
      int t = dp[divs-1][i]+cost[i+1][pos];
      if(t<dp[divs][pos]) {
         dp[divs][pos] = t;          ret = i;
      }
   }
   return ret;
}
void solve(int divs, int L, int R, int searchL, int searchR) {
   if(L>R) return;
   if(L == R) {
      calc(divs,L,searchL, searchR);       return;
   }
   searchR = min(searchR,R-1);
   if(searchL == searchR) {
      for(int i = L; i <= R; i++)
         calc(divs,i,searchL,searchR);
      return;
   }
   int M = (L+R)/2;
   int optM = calc(divs,M,searchL,searchR);
   solve(divs,L,M-1,searchL,optM);
   solve(divs,M+1,R,optM,searchR);
}
int main() {
   int n, d;    build(n);
```

```
   for(int i = 1; i <= n; i++)
      dp[1][i] = cost[1][i];
   for(int i = 2; i <= d; i++)
      solve(i,2,n,i-1,n);
   return 0;
}
```

51. Knuth Optimization 1

```
int sum[MAX][MAX];
int dp[MAX][MAX];
int opt[MAX][MAX];
inline int cost(int u, int v) {
   return sum[v][v] - sum[v][u] - sum[u][v] + sum[u][u];
}
int main() {
   for (int i = 1; i <= N; ++i)
      dp[1][i] = cost(0, i), opt[1][i] = 1;
   for (int i = 2; i <= K; ++i)
      for (int j = N; j; --j) {
         dp[i][j] = inf;          opt[i][N + 1] = N;
         for (int k = opt[i - 1][j]; k <= opt[i][j + 1]; ++k)
            if (dp[i][j] > dp[i - 1][k] + cost(k, j)) {
               dp[i][j] = dp[i - 1][k] + cost(k, j);
               opt[i][j] = k;
            }
      }
   return 0;
}
```

52. Knuth Optimization 2

Knuths optimization works for optimization over substrings for which optimal middle point depends monotonously on the end points. Let mid[l,r] be the first middle point for (l,r) substring which gives optimal result. It can be proven that mid[l,r-1] <= mid[l,r] <= mid[l+1,r] - this means monotonicity of mid by l and r.
Applying this optimization reduces time complexity from O(k^3) to O(k^2) because with fixed s (substring length) we have m_right(l) = mid[l+1][r] = m_left(l+1). That's why nested l and m loops require not more than 2k iterations overall.

```
for (int s = 0; s<=k; s++)                  //s - length(size) of substring
   for (int l = 0; l+s<=k; l++) {            //l - left point
     int r = l + s;                        //r - right point
     if (s < 2) {
       res[l][r] = 0;                      //DP base - nothing to break
       mid[l][r] = l;                      //mid is equal to left border
       continue;
     }
     int mleft = mid[l][r-1]; //Knuth's trick: getting bounds on m
     int mright = mid[l+1][r];
     res[l][r] = 1000000000000000000LL;
     for (int m = mleft; m<=mright; m++) {     //iterating for m in the
bounds only
       int64 tres = res[l][m] + res[m][r] + (x[r]-x[l]);
       if (res[l][r] > tres) {                //relax current solution
         res[l][r] = tres;          mid[l][r] = m;
       }
     }
   }
int64 answer = res[0][k];
```

53. SOS DP

```
const int LN = 20 ;
int dp[(1<<LN)+7] ;

void rec() {
   /** we must initialize dp array with value based on problem.
      Only one inner loop will be activated base on problem  **/
   /** actual dp state we write optimize version .
      if(mask & (1<<i))
                  dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
         else
                  dp[mask][i] = dp[mask][i-1];
   **/

   for(int i=0 ; i<LN ; i++) {
      for(int j=0 ; j<(1<<LN) ; j++) {
         /**
            this loop is used if we want j&i == i
            specifically mask&x == x ; here mask is j
         **/
         if(j&(1<<i))
            dp[j] += dp[j^(1<<i)] ;
      }

      for(int j=(1<<LN)-1 ; j>=0 ; j--) {
         /**
            this loop is used if we want mask&x == mask
         **/
         if((j&(1<<i)) == 0)
            dp[j] += dp[j^(1<<i)] ;
      }
   }
}
```

Matrix Related Algorithm

54. Guass Elimination

```
//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 ;
   }
   debug("::::::::::::::::::::::::::::");
   for(int i=0; i<n; i++) {
      for(int j=0; j<=m; j++) {
         printf("%5.2f ",a[i][j]);
      }
      printf("\n");
   }
   debug("::::::::::::::::::::::::::::");

   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 ) {
      debug("***",where[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
}
```

```
int main() {
   int n,m;
   while(scanf("%d",&n)==1) {
      vector<vector<double> > mat(n);
      vector<double> ans;
      double v;
      for(int i=0; i<n; i++) {
         for(int j=0; j<n+1; j++) {
            scanf("%lf",&v);
            mat[i].push_back(v);
         }
      }
      debug(gauss(mat,ans));
      for(int i=0; i<n; i++) debug(ans[i]);
   }
   return 0;
}

/**
3
1 2 1 3
2 0 1 1
0 1 1 2
*/
```

55. Guass Elimination(row order)

```
//a is the total matrix, last column is the constant matrix and other columns are coefficient matrix
//final ans is stored is ans matrix
//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 ;
      //print3(row,col,a[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 ) //no solution
        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 ;
}
```

56. Guass Elimination(Modular)

```
//a is the total matrix, last column is the constant matrix and other
columns are coefficient matrix
//final ans is stored in ans matrix
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 ( 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 ] ) == 0 )
        continue ;
      for ( int i = col ; i <= m ; ++ i )
        swap ( a [ sel ] [ i ], a [ row ] [ i ] ) ;
      where [ col ] = row ;
      //print3(row,col,a[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;
            //print3(i,j,a[i][j]);
          }
        }
      //cout<<endl;
      ++ row ;
    }
    ans. assign ( m, 0 ) ;
    for ( int i = 0 ; i < m ; ++ i )
      if ( where [ i ] != - 1 )
```

```
        ans [ i ] = (a [ where [ i ] ] [ m ] * bigmod( a [ where [ i ] ] [ 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 ) //no solution
         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 ;
}
```

57. Guass Elimination(Mod 2)

```
//a is the total matrix, last column is the constant matrix and other
columns are coefficient matrix
//final ans is stored in ans matrix
//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;
   //print2(n,m);
   vector < int > where ( m, - 1 ) ;
   for ( int col = 0, row = 0 ; col < m && row < n ; ++ col ) {
      int sel = row ;
      //print1(a[row][col]);
      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 ;
      //print3(row,col,a[row][col]);

      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;
}
int main() {
    int t,cas=0;
    cin>>t;
    while(t--) {
        int n,m;
        cin>>n>>m;
        mem(grid,0);
        int i,j;
        for(i=1; i<=m; i++) {
            int k;
            scanf("%d",&k);
            int light;
            while(k--) {
                scanf("%d",&light);
                grid[light][i]=1;
            }
        }
        int q;
        cin>>q;
        csprnt;
        while(q--) {
            vector<long long>ans;
            vector< vector<long long> > a;
            for(i=1; i<=n; i++) {
                int state;
                scanf("%d",&state);
                vector<long long>tem;
                long long temval=0;
                for(j=1; j<=m; j++)
                    temval+=((long long)grid[i][j]<<(j-1));
                temval+=(long long)state<<m;
                tem.pb(temval);
                a.pb(tem);
            }
            printf("%I64d\n",gauss(a,ans,m+1));
        }
    }
    return 0;
}
```

58. 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;
}
```

59. Determinant (modular)

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

```
void Egcd (int a, int b, int &x, int &y) { //extended gcd
   if (b == 0) {
      x = 1, y = 0;
      return ;
   }
   Egcd (b, a%b, x, y);
   int tp = x;
   x = y;
   y = tp - a/b*y;
}

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;
}
```

60. FFT(without modulo)

```
// nlogn complexity
// memory complexity 12n
/* application
  1. multiplying two arrays.
  2. multiplying two long(string) numbers.
*/
// i-th index mean coefficient of i-th power
typedef complex<long 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) {
    long 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;
  return ;
}
void multiply (vi &a, vi &b, vi &res) {
  vector<base> fa(all(a)),fb(all(b));
  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);
  return ;
}

void multiplyLongNum(vector<int> &a,vector<int> &b, vector < int > &
res ) { //multiplying two long(string) numbers.(normalizing)
```

```
    reverse(all(a));
    reverse(all(b));
    multiply(a,b,res);
    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 ;
    }
    for(int i=SZ(res)-1; i>0; i--) {
        if(res[i]==0) res.pop_back();
        else break;
    }
    reverse(all(res));
}
int main() {
    vector<int> a {1,2,9};
    vector<int> b{7,0,3,8};
    vector<int> r;
    multiplyLongNum(a,b,r);
    for(int i=0; i<SZ(r); i++) {
        printf("%d",r[i]);
    }
    printf("\n");
    return 0;
}
/**
998244353
70383776563201ll
7340033
*/
```

61. FFT(without modulo+complexStructure)

```
// nlogn complexity
// memory complexity 12n
/* application
   1. multiplying two arrays.
   2. multiplying two long(string) numbers.
*/
// i-th index mean coefficient of i-th power
struct cmplx {
    long double r,i;
    inline cmplx() {
        r=i=0.0;
    }
    inline cmplx(long double x) {
        r=x,i=0.0;
    }
    inline cmplx(long double x,long double y) {
        r=x,i=y;
    }
    inline void operator+= (const cmplx &q) {
        r+=q.r,i+=q.i;
    }
    inline void operator-= (const cmplx &q) {
        r-=q.r,i-=q.i;
    }
    inline cmplx operator+ (const cmplx &q) {
        return cmplx(r+q.r,i+q.i);
    }
    inline cmplx operator- (const cmplx &q) {
```

```
    return cmplx(r-q.r,i-q.i);
  }
  inline cmplx operator* (const cmplx &q) {
    return cmplx(r*q.r-i*q.i,r*q.i+i*q.r);
  }
};
typedef cmplx 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) {
    long 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=w*wlen;
      }
    }
  }
  if (invert)
    for (int i=0; i<n; ++i) a[i].r/=n;
  return ;
}
void multiply (vi &a, vi &b,vi &r) {
  vector<base> fa(all(a)),fb(all(b));
  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]=fa[i]*fb[i];
  fft(fa,true); //inverse fft
  r.clear();
  for (size_t i=0; i<n; ++i) {
    r.psb(int(fa[i].r+0.5));
  }
  return ;
}
void multiplyLongNum(vector<int> &a,vector<int> &b, vector < int > &
res ) { //multiplying two long(string) numbers.(normalizing)
  reverse(all(a));
  reverse(all(b));
  multiply(a,b,res);
  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 ;
  }
  for(int i=SZ(res)-1; i>0; i--) {
    if(res[i]==0) res.pop_back();
```

```
        else break;
    }
    reverse(all(res));
}
int main() {
    vector<int> a {1,2,9};
    vector<int> b {7,0,3,8};
    vector<int> r;
    multiplyLongNum(a,b,r);
    for(int i=0; i<SZ(r); i++) {
        printf("%d",r[i]);
    }
    printf("\n");
    return 0;
}
```

62. FFT(with modulo)

```
const int mod = 7340033;
//const int mod = 1000000007;
//const int mod = 761;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft (vector<int> & a, bool invert) {
    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) {
        int wlen = invert ? root_1 : root;
        for (int i=len; i<root_pw; i<<=1)
            wlen = int ((wlen * 1ll * wlen) % mod);
        for (int i=0; i<n; i+=len) {
            int w = 1;
            for (int j=0; j<len/2; ++j) {
                int u = a[i+j],  v = int ((a[i+j+len/2] * 1ll * w) % mod);
                a[i+j] = u+v < mod ? u+v : u+v-mod;
                a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
                w = int ((w * 1ll * wlen) % mod);
            }
        }
    }
    if (invert) {
        int nrev = BigMod(n,mod-2,mod);
        for (int i=0; i<n; ++i)
            a[i] = int ((a[i] * 1ll * nrev) % mod);
    }
}
/**
//previous find root
int find_root(ll n){
    ll d=n-1,s=0;
    while(!(d&1ll)) s++,d>>=1ll;
    for(int i=2;;i++) {
        if(BigMod(i,1ll<<s,n)==1) {
            return i;
```

```
      }
   }
   return -1;
}
*/
int find_root (int p) {
   vector<int> fact;
   int phi = p-1,  n = phi;
   for (int i=2; i*i<=n; ++i)
      if (n % i == 0) {
         fact.push_back (i);
         while (n % i == 0)
            n /= i;
      }
   if (n > 1)  fact.push_back (n);
   for (int res=2; res<=p; ++res) {
      bool ok = true;
      for (size_t i=0; i<fact.size() && ok; ++i)
         ok &= BigMod (res, phi / fact[i], p) != 1;
      if (ok) {
//          vector<int> vv;
//          for(int i=1;i<p;i++)
//          {
//             vv.psb(BigMod(res,i,p));
//          }
//          sort(all(vv));
//          for(int i=0;i<SZ(vv);i++)
//          {
//             if(i) assert(vv[i]!=vv[i-1]);
//          }
         return res;
      }
   }
   return -1;
}
int main() {
   deb(BigMod(2446678,7,mod));
   deb(BigMod(4665133,1<<20,mod));
//    int r1 = find_root(mod);
//    int r2 = BigMod(r1,mod-2,mod);
//    deb(r1,r2);
   return 0;
}
```

63. NTT

```
typedef long long ll;
const int N = 1 << 15;
const int mod = 1e9 + 7;
int p[20], n, e[20], a[2 * N + 10], b[2 * N + 10];
int tot, pnt;
const int p1 = 151 << 20 | 1, r1 = 3;
const int p2 = 141 << 20 | 1, r2 = 5;
const int p3 = 119 << 23 | 1, r3 = 3;
int p2b, p12b;

vector <int> v[3], w[3];

struct CP {
   ll x, y;
   CP (ll x = 0, ll y = 0) : x(x), y(y) {};
   inline CP operator + (CP &a) {
      return CP(x + a.x, y + a.y);
```

```
    }
    inline CP operator - (CP &a) {
        return CP(x - a.x, y - a.y);
    }
} cp[20];

int expmod(int k, int p, int q) {
    int res = 1;
    for ( ; p ; p >>= 1) {
        if (p & 1) res = 1ll * res * k % q;
        k = 1ll * k * k % q;
    }
    return res;
}

int sqrtmod(int k, int q) {
    if (expmod(k, (q - 1) / 2, q) != 1) return -1;
    if (q % 4 == 3) return expmod(k, (q + 1) / 4, q);
    int b, i = (q - 1) / 2, j = 0;
    for (b = 2; expmod(b, (q - 1) / 2, q) == 1; b ++) ;
    while (i % 2 == 0) {
        i /= 2;
        j /= 2;
        if ((1 + 1ll * expmod(k, i, q) * expmod(b, j, q)) % q == 0) j += (q - 1) / 2;
    }
    return 1ll * expmod(k, (i + 1) / 2, q) * expmod(b, j / 2, q) % q;
}



void func(int k, ll x, ll y) {
    if (n + 1 == k) {
        if (y == 0) {
            if (x > 0) {
                x %= mod;
                a[tot ++] = (mod - x * x % mod) % mod;
                a[tot ++] = 0, a[tot ++] = 1;
            } else if (!x) {
                if (!pnt) a[tot ++] = 1, a[tot ++] = 0, a[tot ++] = 0;
                pnt ^= 1;
            }
        } else if (y > 0) {
            x %= mod;
            y %= mod;
            a[tot ++] = (x * x % mod + y * y % mod) % mod;
            a[tot ++] = (2 * x + 2 * mod) % mod;
            a[tot ++] = 1;
        }
        return;
    }
    e[k] = 0;
    func(k + 1, x - cp[k].x, y - cp[k].y);
    func(k + 1, x + cp[k].x, y + cp[k].y);
    return;
}

void mul(int *a, int as, int *b, int bs, int *c) {
    int i, j;
    for (i = 0; i <= as + bs; i ++) c[i] = 0;
    for (i = 0; i <= as; i ++)
        for (j = 0; j <= bs; j ++)
            c[i + j] = (1ll * a[i] * b[j] + c[i + j]) % mod;
    return;
```

```
}

int inv(int k, int q) {
   return expmod(k, q - 2, q);
}

void ntt(vector <int> &aa, int d, int rt, int p) {
   int i, j, k, t, x, y, w;
   for (i = 1, j = 0; i < N; i ++) {
      k = N >> 1;
      for ( ; j >= k; k >>= 1) j ^= k;
      j ^= k;
      if (i < j) swap(aa[i], aa[j]);
   }
   for (int len = 2, half = 1; half < N; len <<= 1, half <<= 1) {
      t = expmod(d ? inv(rt, p) : rt, (p - 1) / len, p);
      for (i = 0; i < N; i += len) {
         for (j = 0, w = 1; j < half; j ++) {
            x = aa[i + j];
            y = 1ll * aa[i + j + half] * w % p;
            w = 1ll * w * t % p;
            aa[i + j] = x + y < p ? x + y : x + y - p;
            aa[i + j + half] = x - y < 0 ? x - y + p : x - y;
         }
      }
   }

   if (d) {
      t = inv(N, p);
      for (i = 0; i < N; i ++) aa[i] = 1ll * aa[i] * t % p;
   }
   return;
}


void multi(int *a, int as, int *b, int bs, int *c) {

   int i, j, k;

   for (i = 0; i < 3; i ++) v[i].clear(), w[i].clear();
   for (i = 0; i < 3; i ++) v[i].resize(N), w[i].resize(N);

   for (i = 0; i <= as; i ++) v[0][i] = a[i] % p1;
   for (i = 0; i <= as; i ++) v[1][i] = a[i] % p2;
   for (i = 0; i <= as; i ++) v[2][i] = a[i] % p3;
   for (i = 0; i <= bs; i ++) w[0][i] = b[i] % p1;
   for (i = 0; i <= bs; i ++) w[1][i] = b[i] % p2;
   for (i = 0; i <= bs; i ++) w[2][i] = b[i] % p3;

   ntt(v[0], 0, r1, p1);
   ntt(v[1], 0, r2, p2);
   ntt(v[2], 0, r3, p3);
   ntt(w[0], 0, r1, p1);
   ntt(w[1], 0, r2, p2);
   ntt(w[2], 0, r3, p3);

   for (i = 0; i < N; i ++) {
      v[0][i] = 1ll * v[0][i] * w[0][i] % p1;
      v[1][i] = 1ll * v[1][i] * w[1][i] % p2;
      v[2][i] = 1ll * v[2][i] * w[2][i] % p3;
   }
```

```
    ntt(v[0], 1, r1, p1);
    ntt(v[1], 1, r2, p2);
    ntt(v[2], 1, r3, p3);
    long long s;
    for (i = 0; i <= as + bs; i ++) {
      j = (1ll * p2b * (v[0][i] - v[1][i]) % p1 + p1) % p1;
      s = 1ll * p2 * j + v[1][i];
      c[i] = 1ll * p12b * ((v[2][i] - s) % p3 + p3) % p3;
      c[i] = (1ll * c[i] * p1 % mod * p2 + s) % mod;
    }
}

int main() {
    int i, j, k, T, l;
    int *x, *y;
    p2b = inv(p2, p1);
    j = 1ll * p1 * p2 % p3;
    p12b = inv(j, p3);
    for (scanf("%d", &T); T --; ) {
      scanf("%d", &n);
      tot = 0;
      pnt = 0;
      for (i = 1; i <= n; i ++) {
        scanf("%d", p + i);
        j = sqrtmod(p[i], mod);
        if (j == -1) cp[i] = CP(0, sqrtmod(mod - p[i], mod));
        else cp[i] = CP(j, 0);
      }
      func(1, 0, 0);
      x = a;
      y = b;
      for (i = 1; i <= n - 1; i ++) {
        l = 1 << i;
        for (j = 0, k = 0; j < tot; j += 2 * l + 2, k += 2 * l + 1) {
          if (i < n - 3) mul(x + j, l, x + j + l + 1, l, y + k);
          else multi(x + j, l, x + j + l + 1, l, y + k);
        }
        y = y == b ? a : b;
        x = x == b ? a : b;
        tot = k;
      }
      printf("%d\n", 1 << n);
      if (n == 15) x[0] --, x[tot - 1] = 1;
      for (i = 0; i < tot - 1; i ++) printf("%d ", x[i]);
      printf("%d\n", x[tot - 1]);
    }
    return 0;
}
```

Number Theory

64. Extended Euclid (ax+by=c)

```
//ax+by=1
pair<LL,LL> egcd ( LL a, LL b ) {
   if (b == 1)
      return make_pair(0, 1);
   pair<LL,LL> 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 make_pair(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;

   pair<LL,LL> 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);
}
```

65. Chinese Remainder Theorem(Garner's)

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

66. Burnside Lemma

```
//LJ 1419(Necklace)
//see emaxx for theory
//Bigmod Code Need
//Sieve Code Need
#define s 1010
bool col[s];
long long prime[s]; // Prime Keep here
int relPrime(int n) { //relative prime
  int i;
  int ans=n;
  for(i=1; prime[i]*prime[i]<=n; i++)
    if(n%prime[i]==0) {
      while(n%prime[i]==0) n/=prime[i];
      ans/=prime[i];
      ans*=(prime[i]-1);
    }
  if(n>1) {
    ans/=n;
```

```
        ans*=(n-1);
    }
    return ans;
}


//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;
}
int main() {
    seive();
    int mod=1000000007;
    int t,cas=0;
    cin>>t;
    while(t--) {
        int n,k;
        scanf("%d %d",&n,&k);
        csprnt;
        print1(burnside(n,k,mod));
    }
    return 0;
}
```

67. Inverse Module(E-GCD)

```
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;
   }
```

```
}
```

68. 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 ;
}
```

69. MillerRabin Primality Test

```
#define SZ1 10000100
#define SZ2 577145
char sieve[(SZ1>>4)+7];
int prime[SZ2];
int totP;
void bit_sieve() {
   int i,j,k,r;
   prime[0]=2;
   k=1;    totP=k;
   int lim=(int)sqrt(SZ1)+1;
   for(i=3; i<SZ1; i+=2) {
      if(!(sieve[i>>4]&(1<<((i>>1)&7)))) {
         prime[k++]=i;
         if(i<lim) {
            r=i<<1;
            for(j=i*i; j<SZ1; j+=r) {
               sieve[j>>4]|=(1<<((j>>1)&7));
            }
         }
      }
   }
   totP=k;
   return;
}
/**
1 means either n<=1
2 means prime
3 means composite square number
4 means composite non square number
*/
int miller_rabin(ll n,int it) {
   if(n<=1) return 1;
   else if(n==2) return 2;
   else {
      ll k=sqrt(n);
```

```
    for(ll i=max(0ll,k-2); i<=k+2; i++) {
      if(i*i==n) return 3;
    }
    if(n%2==0) return 4;
    else {
      ll s=0,d=n-1,a;
      while(d%2==0) {
        s++;
        d/=2;
      }
      bool f;        ll m1,m2;
      for(int i=0; i<it; i++) {
        a=prime[i];
        f=true;
        for(int j=0; j<s; j++) {
          m1=(BigModL(a,d,n)-1+n)%n;
          m2=(BigModL(a,(1ll<<j)*d,n)+1)%n;
          if(m1==0||m2==0) {
            f=false;
            break;
          }
        }
        if(f) return 4;
      }
    }
  }
  return 2;
}
ll get_div(ll n) {
  ll p=2,r=1,c;
  int i;
  for(i=0; i<totP && prime[i]*prime[i]<=n; i++) {
    p=prime[i];
    if(n%p==0) {
      c=1;
      while(n%p==0) {
        n/=p;
        c++;
      }
      r*=c;
    }
  }
  if(n>1) {
    if(n<=prime[totP-1]) r*=2ll;
    else r*=miller_rabin(n,12);
  }
  return r;
}
int main(void) {
  bit_sieve();   ll n;
  while(cin>>n) {
    cout<<get_div(n)<<"\n";
  }
}
```

70. Möbius function

μ(n) is defined for all positive integers n and has its values in {−1, 0, 1} depending on the factorization of n into prime factors. It is defined as follows:
μ(n) = 1 if n is a square-free positive integer with an even number of prime factors.

μ(n) = −1 if n is a square-free positive integer with an odd number of prime factors.
μ(n) = 0 if n has a squared prime factor.

```
//ray gun lightoj
int mob[MX];
int main(){
   mobius();
   scanf("%lld %lld", &a, &b);
   lli M = min(a,b);
   for(lli i = 1; i <= M; i++)
      res += mob[i]*(a/i)*(b/i); //res = 0 at first
   printf("%lld\n", res+2);
}
void mobius(void){
   for(lli i = 2; i < MX; i++) mob[i] = 4;
   mob[1] = 1;
   for(lli i = 2; i < MX; i++)
      if(mob[i] == 4){
         mob[i] = -1;
         for(lli j = i << 1; j < MX; j+=i)
         mob[j] = (mob[j] == 4)? -1:(mob[j]*(-1));
         lli ad = i*i;
         for(lli j = ad; j < MX; j += ad)
            mob[j] = 0;
      }
}
```

71. Phi Function

```
int phi[MX];
void funct(void){
   for(int i = 1; i < MX; i++)       phi[i] = i;
   for(int i = 2; i < MX; i++){
      if(phi[i] == i){
         for(int j = i ; j < MX; j += i)
            phi[j] -= phi[j] / i;
      }
   }
}
```

72. All pair GCD

```
/*uva extreme gcd
find all pair gcd
for(i=1;i<N;i++)
for(j=i+1;j<=N;j++)
   res+=gcd(i,j);   */
int phi[MX];  int sum[MX];
int main(){
   phi_function();
   sum_function();
   printf("%llu\n", sum[n]);
}
void sum_function(void){
   register int i, j;
   for(i = 2; i < MX; i++)
   {
      sum[i] += sum[i-1]+phi[i];
      for(j = i + i; j < MX; j+=i)
         sum[j] += i * phi[j/i];
   }
   return ;
}
```

73. 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))**

*****All division is integer division**

nCr(n,r) = nCr(n-1,r)+ nCr(n-1,r-1);

n = p1e1*p2e2*p3e3*.....

Euler's totient phi of n = number of integers k gcd(k,n) == 1(1 to n)

Phi(n) = n(1-1/p1)(1-1/p2)(1-1/p3).....

= p1(e1-1)(p1-1)*p2(e2-1)(p2-1)*p3(e3-1)(p3-1)*.....

= phi[n] – (phi[n]/p) ///p = 2 to n, p is prime factor of n

Sigma function

sigma zero = number of divisor
= (e1+1)(e2+1)(e3+1).....

sigma one = sum of divisor
= multiple of all p^(e+1)-1/p-1 p
///p = 2 to n, p is prime factor of n

sigma x = sum of dx
///d is the divisor of n (1 to n)
= multiple of all p(e+1)*x-1/px-1
///p = 2 to n, p is prime factor of n

Catalan numbers = $1/(n+1)*(^{2n}C_n)$
= $(^{2n}C_n)- (^{2n}C_{n+1})$
= $C_0 = 1$ and C_{n+1} = sum of C_iC_{n-i} (I = 0 to n)
=multiple of (n+k)/k (k = 2 to n) (not integer division)
= 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786

Application:

1.Cn is the number of Dyck words[2] of length 2n. A Dyck word is a string consisting of n X's and n Y's such that no initial segment of the string has more Y's than X's (see also Dyck language). For example, the following are the Dyck words of length 6:

XXXYYY XYXXYY XYXYXY XXYYXY XXYXYY.

2.Re-interpreting the symbol X as an open parenthesis and Y as a close parenthesis, Cn counts the number of expressions containing n pairs of parentheses which are correctly matched:

((())) ()(()) ()()() (())() (()())

3.Cn is the number of different ways n + 1 factors can be completely parenthesized (or the number of ways of associating n applications of a binary operator). For n = 3, for example, we have the following five different parenthesizations of four factors:

((ab)c)d (a(bc))d (ab)(cd) a((bc)d) a(b(cd))

4.The associahedron of order 4 with the C4=14 full binary trees with 5 leaves

Successive applications of a binary operator can be represented in terms of a full binary tree. (A rooted binary tree is full if every vertex has either two children or no children.) It follows that Cn is the number of full binary trees with n + 1 leaves:

5. Cn is the number of different ways a convex polygon with n + 2 sides can be cut into triangles by connecting vertices with straight lines (a form of Polygon triangulation)

6. Cn is the number of ways to tile a stairstep shape of height n with n rectangles

Miscellaneous

74. Big Integer

```
struct Bigint {
   string a; // to store the digits
   int sign; // sign = -1 for negative numbers, sign = 1 otherwise

   Bigint() {} // default constructor
   Bigint( string b ) {
      (*this) = b;    // constructor for string
   }
   Bigint( long long num ) {
      if(num<0) sign=-1;
      else sign=1;
      if(num==0)  a.push_back('0');
      while(num) {
         a.push_back( num%10 + '0');
         num/=10;
      }
   } // constructor for string

   int size() { // returns number of digits
      return a.size();
   }
   Bigint inverseSign() { // changes the sign
      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 {
// operator for equality
      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.a.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]);
```

```
      puts("");
   }
};
```

75. Stable Marriage Problem

```
// Number of Men or Women
// O based
#define lim  150
int prefer[2*lim][lim]; //preference for woman and man

// This function returns true if woman 'w' prefers man 'm1' over man 'm'
bool wPrefersM1OverM(int N, int w, int m, int m1) {
   // Check if w prefers m over her current engagment m1
   for (int i = 0; i < N; i++) {
      // If m1 comes before m in list of w, then w prefers her
      // current engagement, don't do anything
      if (prefer[w][i] == m1)
         return true;

      // If m cmes before m1 in w's list, then free her current
      // engagement and engage her with m
      if (prefer[w][i] == m)
         return false;
   }
}

// Prints stable matching for N boys and N girls. Boys are numbered as 0
to
// N-1. Girls are numbereed as N to 2N-1.
void stableMarriage(int N) {
   // Stores partner of women. This is our output array that
   // stores paing information.  The value of wPartner[i]
   // indicates the partner assigned to woman N+i.  Note that
   // the woman numbers between N and 2*N-1. The value -1
   // indicates that (N+i)'th woman is free
   int wPartner[N];

   // An array to store availability of men.  If mFree[i] is
   // false, then man 'i' is free, otherwise engaged.
   bool mFree[N];

   // Initialize all men and women as free
   memset(wPartner, -1, sizeof(wPartner));
   memset(mFree, false, sizeof(mFree));
   int freeCount = N;

   // While there are free men
   while (freeCount > 0) {
      // Pick the first free man (we could pick any)
      int m;
      for (m = 0; m < N; m++)
         if (mFree[m] == false)
            break;

      // One by one go to all women according to m's preferences.
      // Here m is the picked free man
      for (int i = 0; i < N && mFree[m] == false; i++) {
         int w = prefer[m][i];
         // The woman of preference is free, w and m become
         // partners (Note that the partnership maybe changed
         // later). So we can say they are engaged not married
         if (wPartner[w-N] == -1) {
```

```
            wPartner[w-N] = m;
            mFree[m] = true;
            freeCount--;
         }
         else { // If w is not free
            // Find current engagement of w
            int m1 = wPartner[w-N];
            // If w prefers m over her current engagement m1,
            // then break the engagement between w and m1 and
            // engage m with w.
            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

   // Print the solution
   for (int i = 0; i < N; i++)
      printf(" (%d %d)",wPartner[i]+1,i+1+N);
   printf("\n");
}
// Driver program to test above functions
int main() {
   int t,cas=0;
   cin>>t;
   while(t--) {
      int n;
      cin>>n;
      int i,j;
      for(i=0; i<2*n; i++) {
         for(j=0; j<n; j++) {
            cin>>prefer[i][j];
            prefer[i][j]--;
         }
      }
      printf("Case %d:",++cas);
      stableMarriage(n);
   }
   return 0;
}
/*
Sample Input
1
3
4 5 6
6 5 4
5 4 6
2 1 3
1 2 3
3 2 1
Sample Input
Case 1: (2 6) (1 4) (3 5)
*/
```

76. 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, n, A, B;
int C = ~(1<<31), M = (1<<16)-1;
int r() {
   A = 36969 * (A & M) + (A >> 16);
   B = 18000 * (B & M) + (B >> 16);
   return (C & ((A << 16) + B)) % 1000000;
}
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() {
   //freopen("pro.in", "r", stdin);
   while(scanf("%d%d%d%d", &m, &n, &A, &B)) {
      if(m == 0 && n == 0 && A == 0 && B == 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();
      for(int i = 1; i <= n; i++) {
         box[i + m].x = r();
```

```
        box[i + m].y = r();
        box[i + m].z = r();
      }
      n += m;
      int f_ans = 1;
      sort(box + 1, box + 1 + n, cmp);
      int mx = 0;
      for(int i = 1; i <= n; 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);
    }
}
```

77. Dates

```
string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
// converts Gregorian date to integer (Julian day number)
int dateToInt (int m, int d, int y) {
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}
// converts integer (Julian day number) to Gregorian date:
month/day/year
void intToDate (int jd, int &m, int &d, int &y) {
  int x, n, i, j;
  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}
// converts integer (Julian day number) to day of week
string intToDay (int jd) {
  return dayOfWeek[jd % 7];
}
int main (int argc, char **argv) {
  int jd = dateToInt (3, 24, 2004);
  int m, d, y;
  intToDate (jd, m, d, y);
  string day = intToDay (jd);
```

```
    // expected output:
    //   2453089
    //   3/24/2004
    //   Wed
    cout << jd << endl
         << m << "/" << d << "/" << y << endl
         << day << endl;
}
```

78. Latitude Longitude

```
/*Converts from rectangular coordinates to latitude/longitude and vice
versa. Uses degrees (not radians). */
struct ll {
    double r, lat, lon;
};
struct rect {
    double x, y, z;
};
ll convert(rect& P) {
    ll Q;
    Q.r = sqrt(P.x*P.x+P.y*P.y+P.z*P.z);
    Q.lat = 180/M_PI*asin(P.z/Q.r);
    Q.lon = 180/M_PI*acos(P.x/sqrt(P.x*P.x+P.y*P.y));

    return Q;
}
rect convert(ll& Q) {
    rect P;
    P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
    P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
    P.z = Q.r*sin(Q.lat*M_PI/180);
    return P;
}
int main() {
    rect A;
    ll B;
    A.x = -1.0;
    A.y = 2.0;
    A.z = -3.0;
    B = convert(A);
    cout << B.r << " " << B.lat << " " << B.lon << endl;
    A = convert(B);
    cout << A.x << " " << A.y << " " << A.z << endl;
}
```

79. Knights Move in infinity grid

```
ll distance(ll sx, ll sy, ll tx, ll ty) {
    ll x, y, t;
    double delta;
    // special corner cases
    if (test(1, 1, 2, 2) ||
        test(7, 7, 8, 8) ||
        test(7, 2, 8, 1) ||
        test(1, 8, 2, 7))
        return 4;
    // axes symmetry
    x = abs(sx - tx);
    y = abs(sy - ty);
    // diagonal symmetry
    if (x < y) {
        t = x;
        x = y;
```

```
    y = t;
  }
  // 2 corner cases
  if (x == 1 && y == 0)
    return 3;
  if (x == 2 && y == 2)
    return 4;
  // main
  delta = x - y;
  if (y > delta) {
    return (ll)(delta - 2 * floor((delta - y) / 3));
  }
  else {
    return (ll)(delta - 2 * floor((delta - y) / 4));
  }
}
```

80. Infix to Postfix

```
int prec[300];//precidence (it should be filled by user)
//make postfix notation with variables and numbers with proper
bracketing
void postfix(string &a,string &b) {
  b.clear();
  a.pb(')');
  stack<char>s;
  s.push('(');
  char tem;
  int i;
  for(i=0; i<SZ(a); i++) {
    if(a[i]==')') { //closing bracket
      while(s.size()) {
        tem=s.top();
        s.pop();
        if(tem=='(') break;
        b.push_back(' ');
        b.push_back(tem);
        b.push_back(' ');
      }
    }
    else if(prec[a[i]]) { //operators
      b.pb(' ');
      while(s.size()) {
        tem=s.top();
        if(prec[tem]<prec[a[i]]) break;
        s.pop();
        b.push_back(' ');
        b.push_back(tem);
        b.push_back(' ');
      }
      s.push(a[i]);
    }
    else if(a[i]=='(') {
      b.push_back(' ');   //opening bracket
      s.push(a[i]);
    }
    else if(isalpha(a[i])) { //variable (size 1)
      b.push_back(' ');
      b.push_back(a[i]);
      b.push_back(' ');
    }
    else b.push_back(a[i]); //number
}
```

```
}
/*
3*2*(2*x+2)+2*x*(3+2-1)=2*x
3*2*(2)*(x)+2*x*(3+2)=x*2*(2)+2*((x)*(2+3))+2+((2)+(3))
*/
```

81. SStream

```
string val;
stringstream ss (stringstream::in | stringstream::out);
ss << "120 ab 377 6 5 2000";
while(ss>>val)
{
  //ss >> val;
  cout << val << endl;
}
```

82. Maximum Disjoint Segment In an Interval

```
#define lson node*2,beg,mid
#define rson node*2+1,mid+1,end
int L[500005], points[500005],
   min_tree[300005 * 8], max_tree[300005 * 8],
   lazy1[300005 * 8], lazy2[300005 * 8],
   ans[500005];
struct line_data {
   int x, y;
};
line_data line[100005];
struct qryy_data {
   int l, r, id;
};
qryy_data qry[100006];
bool comp(qryy_data a, qryy_data b) {
   return a.r < b.r;
}
void refresh(int node, int beg, int end) {
   if(lazy1[node]) {
      if(beg != end) {
         min_tree[node * 2] = max_tree[node * 2] = min_tree[node * 2 + 1]
= max_tree[node * 2 + 1] = lazy1[node];
         lazy1[node * 2] = lazy1[node * 2 + 1]  = lazy1[node];
      }
      lazy1[node] = 0;
   }
   if(lazy2[node]) {
      if(beg != end) {
         lazy2[node * 2] += lazy2[node];
         lazy2[node * 2 + 1] += lazy2[node];
         lazy2[node] = 0;
      }
   }
}
void build(int node, int beg, int end) {
   lazy1[node] = lazy2[node] = 0;
   if(beg == end) {
      min_tree[node] = beg;
      max_tree[node] = beg;
      return;
   }
   int mid = (beg + end) / 2;
   build(lson);
   build(rson);
   min_tree[node] = min(min_tree[node * 2], min_tree[node * 2 + 1]);
```

```
    max_tree[node] = max(max_tree[node * 2], max_tree[node * 2 + 1]);
}

void update(int node, int beg, int end, int i, int j, int c, int d) {
    refresh(node, beg, end);
    if(beg > j || end < i) return;
    if(beg >= i && end <= j) {
        if(c >= max_tree[node]) {
            lazy2[node]++;
            max_tree[node] = min_tree[node] = d;
            lazy1[node] = d;
        } else if(c < min_tree[node]) return ;
        else {
            int mid = (beg + end) / 2;
            update(lson, i, j, c, d);
            update(rson, i, j, c, d);
            min_tree[node] = min(min_tree[node * 2], min_tree[node * 2 +
1]);
            max_tree[node] = max(max_tree[node * 2], max_tree[node * 2 +
1]);
        }
        return ;
    }
    int mid = (beg + end) / 2;
    update(lson, i, j, c, d);
    update(rson, i, j, c, d);
    min_tree[node] = min(min_tree[node * 2], min_tree[node * 2 + 1]);
    max_tree[node] = max(max_tree[node * 2], max_tree[node * 2 + 1]);
}

int query(int node, int beg, int end, int i) {
    if(beg >= i && end <= i) return lazy2[node];
    refresh(node, beg, end);
    int mid = (beg + end) / 2;
    if(i <= mid) return query(lson, i);
    else return query(rson, i);
}
int main() {
    int n, m;
    while(sf2(n, m) == 2) {
        int cnt = 0;
        for(int i = 1; i <= n; i++) {
            sf2(line[i].x, line[i].y);
            points[++cnt] = line[i].x;
            points[++cnt] = line[i].y;
        }
        for(int i = 1; i <= m; i++) {
            sf2(qry[i].l, qry[i].r);
            qry[i].id = i;
            points[++cnt] = qry[i].l;
            points[++cnt] = qry[i].r;
        }
        sort(points + 1, points + 1 + cnt);
        cnt = unique(points + 1, points + 1 + cnt) - points - 1;
        mem(L, -1);
        clr(ans);
        for(int i = 1; i <= n; i++) {
            line[i].x = lower_bound(points + 1, points + cnt + 1, line[i].x) -
points;
            line[i].y = lower_bound(points + 1, points + cnt + 1, line[i].y) -
points;
            L[ line[i].y ] = max(L[ line[i].y ], line[i].x);
```

```
        }
        for(int i = 1; i <= m; i++) {
            qry[i].l = lower_bound(points + 1, points + 1 + cnt, qry[i].l) - points;
            qry[i].r = lower_bound(points + 1, points + 1 + cnt, qry[i].r) - points;
        }
        build(1, 1, cnt);
        int p = 1;
        sort(qry + 1, qry + 1 + m, comp);
        for(int i = 1; i <= cnt; i++) {
            if(L[i] != -1) {
                update(1, 1, cnt, 1, i, L[i], i);
            }
            while(qry[p].r == i) {
                ans[qry[p].id] = query(1, 1, cnt, qry[p].l);
                p++;
            }
        }
        for(int i = 1; i <= m; i++) {
            pf("%d\n", ans[i]);
        }
    }
}
```

Geometry

83. Convex Hull

```
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);
}

//0 based
//res[0] and res[last] are same
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;
}
```

84. Line Intersection Integer

```
typedef long long ll;
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;
}
```

85. Closest Pair of Point

```
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);
}
```

86. Geometry 2D

```
#define PI acos(-1.0)

using namespace std;

const double INF = 1e100;
const double EPS = 1e-9;

int EQ(double x) {
  if(fabs(x)<EPS) return 0;
  else if(x>0) return 1;
  else return -1;
}

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  void input() {
    sf("%lf %lf",&x,&y);
  }
  void output() {
    pf("%f %f\n",x,y);
  }
  PT operator + (const PT &p)  const {
    return PT(x+p.x, y+p.y);
  }
  PT operator - (const PT &p)  const {
    return PT(x-p.x, y-p.y);
  }
  PT operator * (double c)     const {
    return PT(x*c,   y*c  );
  }
  PT operator / (double c)     const {
    return PT(x/c,   y/c  );
  }
};

double dot(PT p, PT q) {
  return p.x*q.x+p.y*q.y;
}
```

```
double dist2(PT p, PT q) {
   return dot(p-q,p-q);
}
double distPoint(PT p, PT q) {
   return sqrt(dot(p-q,p-q));
}
double cross(PT p, PT q) {
   return p.x*q.y-p.y*q.x;
}
//ostream &operator<<(ostream &os, const PT &p)
//{
//    os << "(" << p.x << "," << p.y << ")";
//}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) {
   return PT(-p.y,p.x);
}
PT RotateCW90(PT p) {
   return PT(p.y,-p.x);
}
PT RotateCCW(PT p, double t) {
   return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
PT RotateCW(PT p, double t) {
   return PT(p.x*cos(t)+p.y*sin(t), -p.x*sin(t)+p.y*cos(t));
}

// find a point from 'a' through 'b' with
// distance d
// use for better precision
PT PointAlongLine(PT a,PT b,double d) {
   return a + (((b-a) / sqrt(dot(b-a,b-a))) * d);
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
   return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
   double r = dot(b-a,b-a);
   if (fabs(r) < EPS) return a;
   r = dot(c-a, b-a)/r;
   if (r < 0) return a;
   if (r > 1) return b;
   return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
   return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

///return minimum distance from point p to line AB
double distToLine(PT p, PT A, PT B, PT &c) {
   double scale = (double)
            (dot(p-A,B-A)) /
            (dot(B-A,B-A));
   c.x = A.x + scale * (B.x - A.x);
```

```
   c.y = A.y + scale * (B.y - A.y);
   return distPoint(p, c);
}

///return minimum distance from point p to line segment AB
/**
dot product <= 0 means the angle is >= 90 and <=180
*/
double distToLineSegment(PT p, PT A, PT B, PT &c) {
   if (dot(B-A,p-A) < EPS) {
      c.x = A.x;
      c.y = A.y;
      return distPoint(p, A);
   }
   if (dot(A-B,p-B) < EPS) {
      c.x = B.x;
      c.y = B.y;
      return distPoint(p, B);
   }
   return distToLine(p, A, B, c);
}

bool isPointOnSegment(PT p,PT a,PT b) {
   if(fabs(cross(p-b,a-b))<EPS) {
      if(p.x<min(a.x,b.x)||p.x>max(a.x,b.x)) return false;
      if(p.y<min(a.y,b.y)||p.y>max(a.y,b.y)) return false;
      return true;
   }
   return false;
}
```

```
// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d) {
   return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
   return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
   return LinesParallel(a, b, c, d)
         && fabs(cross(a-b, a-c)) < EPS
         && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
   if (LinesCollinear(a, b, c, d)) {
      if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
            dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
      if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
         return false;
      return true;
   }
   if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
   if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
   return true;
}
```

```
// check if two lines are same
bool areLinesSame(PT a, PT b, PT c, PT d) {
   if(fabs(cross(a-c,c-d))<EPS && fabs(cross(b-c,c-d))<EPS) return true;
   return false;
}
// check if two lines are parallel
bool areLinesParallel(PT a, PT b, PT c, PT d) {
   if(fabs(cross(a-b,c-d))<EPS) return true;
   return false;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
/**
this sometimes does not work
//PT ComputeLineIntersection(PT a, PT b, PT c, PT d)
//{
//    b=b-a;
//    d=c-d;
//    c=c-a;
//    return a + b*cross(c, d)/cross(b, d);
//}
*/
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
   double a1,b1,c1,a2,b2,c2;
   a1 = a.y - b.y;
   b1 = b.x - a.x;
   c1 = cross(a, b);
   a2 = c.y - d.y;
   b2 = d.x - c.x;
   c2 = cross(c, d);
   double D = a1 * b2 - a2 * b1;
   return PT((b1 * c2 - b2 * c1) / D,(c1 * a2 - c2 * a1) / D);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
   b=(a+b)/2;
   c=(a+c)/2;
   return ComputeLineIntersection(b, b+RotateCW90(a-b), c,
c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
   bool c = 0;
   int s=p.size();
   for (int i = 0,j=s-1; i < s; j=i++) {
      if ( ( (p[i].y > q.y) != (p[j].y > q.y ) ) &&
            (q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y)))
         c = !c;
   }
   return c;
}
```

```
// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  int s=p.size();
  for (int i = 0,j=s-1; i < s; j=i++)
    if (isPointOnSegment(q,p[j],p[i]))
      return true;
  return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// This code computes the area or centroid of a (possibly non-convex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
```

```
  for (int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
/**
Return a parallel line of line ab in counterclockwise
direction with d distance from ab
*/
pair<PT,PT> getParallelLine(PT a,PT b,double d) {
  return mp(PointAlongLine(a,RotateCCW90(b-
a)+a,d),PointAlongLine(b,RotateCW90(a-b)+b,d));
}

/**
Return a tangent line of line ab which intersects
with it at point c in counterclockwise direction
*/
pair<PT,PT> getPerpendicularLine(PT a,PT b,PT c) {
  return mp(RotateCCW90(a-c)+c,RotateCCW90(b-c)+c);
}

vector<PT> halfPlaneIntersection(const vector<PT> &poly, pair<PT,PT> ln)
{
  vector<PT> ret;
  int s=SZ(poly);
  for(int i=0; i<s; i++) {
    double c1=cross(ln.sc-ln.fs,poly[i]-ln.fs);
    double c2=cross(ln.sc-ln.fs,poly[(i+1)%s]-ln.fs);
    if(EQ(c1)>=0) ret.psb(poly[i]);
    if(EQ(c1*c2)<0) {
      if(!areLinesParallel(poly[i],poly[(i+1)%s],ln.fs,ln.sc)) {

ret.psb(ComputeLineIntersection(poly[i],poly[(i+1)%s],ln.fs,ln.sc));
      }
    }
  }
  return ret;
}

/*
void Test()
{

  // expected: (-5,2)
  cerr << RotateCCW90(PT(2,5)) << endl;
```

```
// expected: (5,-2)
cerr << RotateCW90(PT(2,5)) << endl;

// expected: (-5,2)
cerr << RotateCCW(PT(2,5),PI/2) << endl;

// expected: (5,2)
cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

// expected: (5,2) (7.5,3) (2.5,1)
cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
    << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
    << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

// expected: 6.78903
cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

// expected: 1 0 1
cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
    << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
    << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 0 0 1
cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
    << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
    << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 1 1 1 0
cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

// expected: (1,2)
cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) <<
endl;

// expected: (1,1)
cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;


vector<PT> v;
v.push_back(PT(0,0));
v.push_back(PT(5,0));
v.push_back(PT(5,5));
v.push_back(PT(0,5));

// expected: 1 1 1 0 0
cerr << PointInPolygon(v, PT(2,2)) << " "
    << PointInPolygon(v, PT(2,0)) << " "
    << PointInPolygon(v, PT(0,2)) << " "
    << PointInPolygon(v, PT(5,2)) << " "
    << PointInPolygon(v, PT(2,5)) << endl;

// expected: 0 1 1 1 1
cerr << PointOnPolygon(v, PT(2,2)) << " "
    << PointOnPolygon(v, PT(2,0)) << " "
    << PointOnPolygon(v, PT(0,2)) << " "
    << PointOnPolygon(v, PT(5,2)) << " "
    << PointOnPolygon(v, PT(2,5)) << endl;

// expected: (1,6)
```

```
//        (5,4) (4,5)
//        blank line
//        (4,5) (5,4)
//        blank line
//        (4,5) (5,4)
vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;

// area should be 5.0
// centroid should be (1.1666666, 1.166666)
PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
vector<PT> p(pa, pa+4);
PT c = ComputeCentroid(p);
cerr << "Area: " << ComputeArea(p) << endl;
cerr << "Centroid: " << c << endl;

   PT a=PT(1.3,2.6), b=PT(8.1,13.7);
   double d=3.17096;
   PT r=PointAlongLine(a,b,d);
   deb(r.x,r.y);

   return ;
}
*/
```

87. Geometry 2D(Integer)

```
const double PI = acos(-1.0);
const double INF = 1e18;
typedef pair<double,double> pdd;
struct PT {
   ll x,y;
   PT() {}
   PT(ll x,ll y):x(x),y(y) {}
   void input() {
      sf("%lld %lld",&x,&y);
   }
   void output() {
      pf("%lld %lld\n",x,y);
   }
   bool operator < (const PT &p)  const {
      if(y==p.y) return x<p.x;
      return y<p.y;
   }
   bool operator == (const PT &p)  const {
      return mp(x,y)==mp(p.x,p.y);
```

```
    }
    bool operator != (const PT &p)  const {
        return mp(x,y)!=mp(p.x,p.y);
    }
    PT operator + (const PT &p)  const {
        return PT(x+p.x, y+p.y);
    }
    PT operator - (const PT &p)  const {
        return PT(x-p.x, y-p.y);
    }
    PT operator * (double c)     const {
        return PT(x*c,   y*c  );
    }
    PT operator / (double c)     const {
        return PT(x/c,   y/c  );
    }
};

//double Distance(PT a,PT b)
//{
//    PT p=a-b;
//    return sqrt(p.x*p.x+p.y*p.y);
//}

ll dot(PT p, PT q) {
    return p.x*q.x+p.y*q.y;
}
ll cross(PT p, PT q) {
    return p.x*q.y-p.y*q.x;
}
///***
bool areLinesSame(PT a, PT b, PT c, PT d) {
    if(cross(a-c,c-d)==0 && cross(b-c,c-d)==0) return true;
    return false;
}
double distPoint(PT p, PT q) {
    return sqrt(dot(p-q,p-q));
}
///***
bool areLinesParallel(PT a, PT b, PT c, PT d) {
    if(cross(a-b,c-d)==0) return true;
    return false;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
//void ComputeLineIntersection(PT a, PT b, PT c, PT d,pdd &ret)
//{
//    b=b-a;
//    d=c-d;
//    c=c-a;
//    double h=(double)cross(c, d)/(double)cross(b, d);
//    ret.xx=(double) a.x + (double) b.x * h;
//    ret.yy=(double) a.y + (double) b.y * h;
//    return ;
//}

void ComputeLineIntersection(PT a, PT b, PT c, PT d,pdd &ret) {
    double a1,b1,c1,a2,b2,c2;
    a1 = a.y - b.y;
```

```
   b1 = b.x - a.x;
   c1 = cross(a, b);
   a2 = c.y - d.y;
   b2 = d.x - c.x;
   c2 = cross(c, d);
   double D = a1 * b2 - a2 * b1;
   ret=mp((b1 * c2 - b2 * c1) / D,(c1 * a2 - c2 * a1) / D);
   return ;
}

bool onsegment(PT a,PT b,PT c) {
   ll cr=cross(a-c,b-c);
   return (!cr && ( 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 isSegmentIntersect(PT p1,PT p2,PT p3,PT p4) {
   ll d1,d2,d3,d4;

   d1 = cross(p4-p3,p1-p3);
   d2 = cross(p4-p3,p2-p3);
   d3 = cross(p2-p1,p3-p1);
   d4 = cross(p2-p1,p4-p1);

   int s1,s2,s3,s4;
   s1=d1==0?0:d1<0?-1:1;
   s2=d2==0?0:d2<0?-1:1;
   s3=d3==0?0:d3<0?-1:1;
   s4=d4==0?0:d4<0?-1:1;

   if(s1*s2<0 && s3*s4<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;
}

///return minimum distance from point p to line AB
double distToLine(PT p, PT A, PT B, pdd &c) {
   double scale = (double)
            (dot(p-A,B-A)) /
            (dot(B-A,B-A));
   c.fs = A.x + scale * (B.x - A.x);
   c.sc = A.y + scale * (B.y - A.y);
   double dx=(double)p.x-c.fs,dy=(double)p.y-c.sc;
   return sqrt(dx*dx+dy*dy);
}

///return minimum distance from point p to line segment AB
/**
dot product <= 0 means the angle is >= 90 and <=180
*/
double distToLineSegment(PT p, PT A, PT B, pdd &c) {
   if (dot(B-A,p-A) <= 0) {
      c.fs = A.x;
      c.sc = A.y;
      return distPoint(p, A);
   }
   if (dot(A-B,p-B) <= 0) {
      c.fs = B.x;
      c.sc = B.y;
```

```
      return distPoint(p, B);
   }
   return distToLine(p, A, B, c);
}

bool ComputeLineSegmentIntersection(PT a, PT b, PT c, PT d,pdd &ret) {
   if(!isSegmentIntersect(a,b,c,d)) return false;
   double a1,b1,c1,a2,b2,c2;
   a1 = a.y - b.y;
   b1 = b.x - a.x;
   c1 = cross(a, b);
   a2 = c.y - d.y;
   b2 = d.x - c.x;
   c2 = cross(c, d);
   double D = a1 * b2 - a2 * b1;
   ret=mp((b1 * c2 - b2 * c1) / D,(c1 * a2 - c2 * a1) / D);
   return true;
}

bool compAng(const PT& a,const PT& b) {
   ll c=cross(a,b);
   if(c!=0) return c>0;
   ll d1=dot(a,a),d2=dot(b,b);
   return d1>d2;
}


// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary

bool PointInPolygon(const vector<PT> &p, PT q) {
   bool c = 0;
   ll t,sgn;
   int s=p.size();
   for (int i = 0,j=s-1; i < s; j=i++) {
      t=(p[j].y - p[i].y);
      sgn=t<0?-1:t==0?0:1;
      t*=sgn;
      if ( ( (p[i].y > q.y) != (p[j].y > q.y ) ) &&
            ( (q.x-p[i].x)*t < (p[j].x - p[i].x) * (q.y - p[i].y)*sgn))
         c = !c;
   }
   return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
   int s=p.size();
   for (int i = 0,j=s-1; i < s; j=i++)
      if (onsegment(p[j],p[i],q))
         return true;
   return false;
}

/**
convex_hull : including collinear points
counterclockwise
```

```
*/
void ConvexHull(vector<PT>& poly,vector<PT>& ret) {
   int n=SZ(poly);
   if(n==0) return ;
   sort(all(poly));
   poly.resize(distance(poly.begin(),unique(all(poly))));
   n=SZ(poly);
   PT fpoint = poly[0];
   for(int i=0; i<n; i++) {
      poly[i]=poly[i]-fpoint;
   }
   stack<PT> S;
   PT f;
   PT p1,p2,p3;
   if(n>2) {
      sort(poly.begin()+1,poly.end(),compAng);
      bool ok;
      ll c;
      S.push(poly[0]);
      S.push(poly[1]);
      for(int i=2; i<=n; i++) {
         p3=poly[i%n];
         ok=(i!=n);
         do {
            p2=S.top();
            S.pop();
            p1=S.top();
            S.push(p2);
            c=cross(p2-p1,p3-p1);
            if(c<0) {
               if(SZ(S)>2) S.pop();
               else break;
            } else if(c==0) {
               ll d12=dot(p2-p1,p2-p1),d13=dot(p3-p1,p3-p1);
               if(d13<=d12) ok=false;
               else {
                  if(SZ(S)>=2) S.pop();
               }
               break;
            } else break;
         } while(SZ(S)>=2);
         if(ok) S.push(p3);
      }
      while(!S.empty()) {
         ret.psb(S.top());
         S.pop();
      }
      reverse(all(ret));
   } else {
      ret=poly;
   }
   n=SZ(ret);
   for(int i=0; i<n; i++) {
      ret[i]=ret[i]+fpoint;
   }
   return ;
}

// This code computes the area or centroid of a (possibly non-convex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
```

```
double ComputeSignedArea(const vector<PT> &p) {
   double area = 0;
   for(int i = 0; i < p.size(); i++) {
      int j = (i+1) % p.size();
      area += p[i].x*p[j].y - p[j].x*p[i].y;
   }
   return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
   return fabs(ComputeSignedArea(p));
}

double polygonDiameter(vector<PT> &p) {
   int n=SZ(p);
   if(n<2) return 0;
   else if(n==2) return distPoint(p[0],p[1]);
   else {
      int i=n-1,j=0,k=1;
      double res=0;
      while(abs(cross(p[j]-p[i],p[k+1]-p[i]) >
            abs(cross(p[j]-p[i],p[k]-p[i])))) {
         k++;
      }
      i=0,j=k;
      while(i<=k && j<n) {
         res=max(distPoint(p[i],p[j]),res);
         while(j<n-1 && abs(cross(p[i+1]-p[i],p[j+1]-p[i]) >
                     abs(cross(p[i+1]-p[i],p[j]-p[i])))) {
            j++;
            res=max(distPoint(p[i],p[j]),res);
         }
         i++;
      }
      return res;
   }
}

int main() {
   vector<PT> poly,cpoly;
   poly.psb(PT(5,7));
   poly.psb(PT(1,9));
   poly.psb(PT(3,6));
   poly.psb(PT(15,7));
   poly.psb(PT(27,8));
   poly.psb(PT(5,9));

//   poly.psb(PT(0,0));
//   poly.psb(PT(0,1));
//   poly.psb(PT(0,2));
//   poly.psb(PT(0,3));
//   poly.psb(PT(0,4));
//   poly.psb(PT(0,5));
   ConvexHull(poly,cpoly);
   for(int i=0; i<SZ(cpoly); i++) {
      cpoly[i].output();
   }
   return 0;
}
```

88. Geometry 3D

```
#define zero(x) (((x)>0?(x):-(x))<EPS)

const double INF = 1e100;
const double EPS = 1e-9;

int EQ(double x) {
   if(fabs(x)<EPS) return 0;
   else if(x>0) return 1;
   else return -1;
}

struct point3 {
   double x,y,z;
   point3() {}
   point3(double x,double y,double z):x(x),y(y),z(z) {}
   point3 operator + (const point3 &p)  const {
      return point3(x+p.x, y+p.y, z+p.z);
   }
   point3 operator - (const point3 &p)  const {
      return point3(x-p.x, y-p.y, z-p.z);
   }
   point3 operator * (double c)     const {
      return point3(x*c, y*c, z*c);
   }
   point3 operator / (double c)     const {
      return point3(x/c, y/c, z/c);
   }
};
struct line3 {
   point3 a,b;
   line3() {}
   line3(point3 a,point3 b):a(a),b(b) {}
};
struct plane3 {
   point3 a,b,c;
   plane3() {}
   plane3(point3 a,point3 b,point3 c):a(a),b(b),c(c) {}
};

//compute cross product U x V
point3 xmult(point3 u,point3 v) {
   point3 ret;
   ret.x=u.y*v.z-v.y*u.z;
   ret.y=u.z*v.x-u.x*v.z;
   ret.z=u.x*v.y-u.y*v.x;
   return ret;
}

//compute dot product U . V
double dmult(point3 u,point3 v) {
   return u.x*v.x+u.y*v.y+u.z*v.z;
}

// Vector difference U - V
point3 subt(point3 u,point3 v) {
   point3 ret;
   ret.x=u.x-v.x;
   ret.y=u.y-v.y;
   ret.z=u.z-v.z;
   return ret;
```

```
}

// Vector addition U + V
point3 addt(point3 u,point3 v) {
   point3 ret;
   ret.x=u.x+v.x;
   ret.y=u.y+v.y;
   ret.z=u.z+v.z;
   return ret;
}

// Take the plane normal vector
point3 pvec(plane3 s) {
   return xmult(subt(s.a,s.b),subt(s.b,s.c));
}
point3 pvec(point3 s1,point3 s2,point3 s3) {
   return xmult(subt(s1,s2),subt(s2,s3));
}

// Distance between two points, the size of a single parameter of the
alignment amount
double distance(point3 p1,point3 p2) {
   return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-
p2.z)*(p1.z-p2.z));
}

// Vector magnitude
double vlen(point3 p) {
   return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}

// Sentenced collinear
int dots_inline(point3 p1,point3 p2,point3 p3) {
   return vlen(xmult(subt(p1,p2),subt(p2,p3)))<EPS;
}

// Sentenced to four points are coplanar
int dots_onplane(point3 a,point3 b,point3 c,point3 d) {
   return zero(dmult(pvec(a,b,c),subt(d,a)));
}

// Sentenced point if the line segment, inclusive and collinear
int dot_online_in(point3 p,line3 l) {
   return zero(vlen(xmult(subt(p,l.a),subt(p,l.b))))&&(l.a.x-p.x)*(l.b.x-
p.x)<EPS&&
         (l.a.y-p.y)*(l.b.y-p.y)<EPS&&(l.a.z-p.z)*(l.b.z-p.z)<EPS;
}
int dot_online_in(point3 p,point3 l1,point3 l2) {
   return zero(vlen(xmult(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)*(l2.x-
p.x)<EPS&&
         (l1.y-p.y)*(l2.y-p.y)<EPS&&(l1.z-p.z)*(l2.z-p.z)<EPS;
}

// Sentenced point on whether the line segment, not inclusive
int dot_online_ex(point3 p,line3 l) {
   return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-
l.a.y)||!zero(p.z-l.a.z))&&
         (!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));
}
int dot_online_ex(point3 p,point3 l1,point3 l2) {
   return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-
l1.y)||!zero(p.z-l1.z))&&
```

```
        (!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
}

// Determines whether a point on a triangular space, including borders,
collinear meaningless
int dot_inplane_in(point3 p,plane3 s) {
   return zero(vlen(xmult(subt(s.a,s.b),subt(s.a,s.c)))-
vlen(xmult(subt(p,s.a),subt(p,s.b)))-
            vlen(xmult(subt(p,s.b),subt(p,s.c)))-
vlen(xmult(subt(p,s.c),subt(p,s.a))));
}
int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3) {
   return zero(vlen(xmult(subt(s1,s2),subt(s1,s3)))-
vlen(xmult(subt(p,s1),subt(p,s2)))-
            vlen(xmult(subt(p,s2),subt(p,s3)))-
vlen(xmult(subt(p,s3),subt(p,s1))));
}

// Determines whether a point on a triangular space, not including
borders, collinear meaningless
int dot_inplane_ex(point3 p,plane3 s) {
   return
dot_inplane_in(p,s)&&vlen(xmult(subt(p,s.a),subt(p,s.b)))>EPS&&

vlen(xmult(subt(p,s.b),subt(p,s.c)))>EPS&&vlen(xmult(subt(p,s.c),subt(p,s.
a)))>EPS;
}
int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3) {
   return
dot_inplane_in(p,s1,s2,s3)&&vlen(xmult(subt(p,s1),subt(p,s2)))>EPS&&
vlen(xmult(subt(p,s2),subt(p,s3)))>EPS&&vlen(xmult(subt(p,s3),subt(p,s1)
))>EPS;
}

// Sentenced to two line segments on the same side, returns 0 point line
segment, are not coplanar meaningless
int same_side(point3 p1,point3 p2,line3 l) {
   return
dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))>EP
S;
}
int same_side(point3 p1,point3 p2,point3 l1,point3 l2) {
   return
dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))>EPS;
}

// Sentenced to two different sides of the line segment, returns 0 point
line segment, are not coplanar meaningless
int opposite_side(point3 p1,point3 p2,line3 l) {
   return
dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))<-
EPS;
}
int opposite_side(point3 p1,point3 p2,point3 l1,point3 l2) {
   return
dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))<-EPS;
}

// Sentenced to two points in the plane on the same side, point in the
plane returns 0
```

```
int same_side(point3 p1,point3 p2,plane3 s) {
   return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))>EPS;
}
int same_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3) {
   return
dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))>EPS;
}

// Sentenced to two points in the plane of the opposite side, the point in
the plane returns 0
int opposite_side(point3 p1,point3 p2,plane3 s) {
   return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))<-EPS;
}
int opposite_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3) {
   return
dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))<-
EPS;
}

// Sentenced to two parallel lines
int parallel(line3 u,line3 v) {
   return vlen(xmult(subt(u.a,u.b),subt(v.a,v.b)))<EPS;
}
int parallel(point3 u1,point3 u2,point3 v1,point3 v2) {
   return vlen(xmult(subt(u1,u2),subt(v1,v2)))<EPS;
}

// Sentenced to two plane-parallel
int parallel(plane3 u,plane3 v) {
   return vlen(xmult(pvec(u),pvec(v)))<EPS;
}
int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3) {
   return vlen(xmult(pvec(u1,u2,u3),pvec(v1,v2,v3)))<EPS;
}

// Sentence straight and parallel to the plane
int parallel(line3 l,plane3 s) {
   return zero(dmult(subt(l.a,l.b),pvec(s)));
}
int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3) {
   return zero(dmult(subt(l1,l2),pvec(s1,s2,s3)));
}

// Sentenced to two straight lines perpendicular
int perpendicular(line3 u,line3 v) {
   return zero(dmult(subt(u.a,u.b),subt(v.a,v.b)));
}
int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2) {
   return zero(dmult(subt(u1,u2),subt(v1,v2)));
}

// Sentenced to two planes perpendicular
int perpendicular(plane3 u,plane3 v) {
   return zero(dmult(pvec(u),pvec(v)));
}
int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3) {
   return zero(dmult(pvec(u1,u2,u3),pvec(v1,v2,v3)));
}

// Sentence straight and parallel to the plane
int perpendicular(line3 l,plane3 s) {
```

```
    return vlen(xmult(subt(l.a,l.b),pvec(s)))<EPS;
}
int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3) {
    return vlen(xmult(subt(l1,l2),pvec(s1,s2,s3)))<EPS;
}

// Sentenced to two segments intersect, inclusive and partially overlap
int intersect_in(line3 u,line3 v) {
    if (!dots_onplane(u.a,u.b,v.a,v.b))
        return 0;
    if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
        return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
    return
dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_o
nline_in(v.b,u);
}
int intersect_in(point3 u1,point3 u2,point3 v1,point3 v2) {
    if (!dots_onplane(u1,u2,v1,v2))
        return 0;
    if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
        return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
    return
dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,
u2)||dot_online_in(v2,u1,u2);
}

// Sentenced to two line segments intersect, not inclusive and partially
overlap
int intersect_ex(line3 u,line3 v) {
    return
dots_onplane(u.a,u.b,v.a,v.b)&&opposite_side(u.a,u.b,v)&&opposite_sid
e(v.a,v.b,u);
}
int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2) {
    return
dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_sid
e(v1,v2,u1,u2);
}

// Sentenced triangle intersection and space segments, including cross
the boundary and (in part) that contains
int intersect_in(line3 l,plane3 s) {
    return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&
        !same_side(s.b,s.c,l.a,l.b,s.a)&&!same_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3) {
    return !same_side(l1,l2,s1,s2,s3)&&!same_side(s1,s2,l1,l2,s3)&&
        !same_side(s2,s3,l1,l2,s1)&&!same_side(s3,s1,l1,l2,s2);
}

// Sentenced triangle intersection and space segments, not including
delivery to the boundary and (in part) that contains
int intersect_ex(line3 l,plane3 s) {
    return opposite_side(l.a,l.b,s)&&opposite_side(s.a,s.b,l.a,l.b,s.c)&&
        opposite_side(s.b,s.c,l.a,l.b,s.a)&&opposite_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3) {
    return opposite_side(l1,l2,s1,s2,s3)&&opposite_side(s1,s2,l1,l2,s3)&&
        opposite_side(s2,s3,l1,l2,s1)&&opposite_side(s3,s1,l1,l2,s2);
}
```

```
// Calculate the intersection of two straight, pay attention to prejudge
whether coplanar and parallel to the straight line!
// Line intersects the intersection please also sentenced segment (and
still have to determine whether the parallel!)
point3 intersection(line3 u,line3 v) {
   point3 ret=u.a;
   double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
         /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
   ret.x+=(u.b.x-u.a.x)*t;
   ret.y+=(u.b.y-u.a.y)*t;
   ret.z+=(u.b.z-u.a.z)*t;
   return ret;
}
point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2) {
   point3 ret=u1;
   double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
         /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
   ret.x+=(u2.x-u1.x)*t;
   ret.y+=(u2.y-u1.y)*t;
   ret.z+=(u2.z-u1.z)*t;
   return ret;
}

// Calculate the intersection of the straight line and the plane, pay
attention to prejudge whether or not parallel, and to ensure that three
non-collinear!
// Line and space triangle intersection please also judge
point3 intersection(line3 l,plane3 s) {
   point3 ret=pvec(s);
   double t=(ret.x*(s.a.x-l.a.x)+ret.y*(s.a.y-l.a.y)+ret.z*(s.a.z-l.a.z))/
         (ret.x*(l.b.x-l.a.x)+ret.y*(l.b.y-l.a.y)+ret.z*(l.b.z-l.a.z));
   ret.x=l.a.x+(l.b.x-l.a.x)*t;
   ret.y=l.a.y+(l.b.y-l.a.y)*t;
   ret.z=l.a.z+(l.b.z-l.a.z)*t;
   return ret;
}
point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3) {
   point3 ret=pvec(s1,s2,s3);
   double t=(ret.x*(s1.x-l1.x)+ret.y*(s1.y-l1.y)+ret.z*(s1.z-l1.z))/
         (ret.x*(l2.x-l1.x)+ret.y*(l2.y-l1.y)+ret.z*(l2.z-l1.z));
   ret.x=l1.x+(l2.x-l1.x)*t;
   ret.y=l1.y+(l2.y-l1.y)*t;
   ret.z=l1.z+(l2.z-l1.z)*t;
   return ret;
}

// Calculate the two planes intersecting line, pay attention to prejudge
whether or not parallel, and to ensure that three non-collinear!
line3 intersection(plane3 u,plane3 v) {
   line3 ret;

ret.a=parallel(v.a,v.b,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersec
tion(v.a,v.b,u.a,u.b,u.c);

ret.b=parallel(v.c,v.a,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersect
ion(v.c,v.a,u.a,u.b,u.c);
   return ret;
}
line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3) {
   line3 ret;
```

```
ret.a=parallel(v1,v2,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v
1,v2,u1,u2,u3);

ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v
3,v1,u1,u2,u3);
   return ret;
}

// Point to the straight line distance
double ptoline(point3 p,line3 l) {
   return vlen(xmult(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
}
double ptoline(point3 p,point3 l1,point3 l2) {
   return vlen(xmult(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
}

// Point to plane distance
double ptoplane(point3 p,plane3 s) {
   return fabs(dmult(pvec(s),subt(p,s.a)))/vlen(pvec(s));
}
double ptoplane(point3 p,point3 s1,point3 s2,point3 s3) {
   return fabs(dmult(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
}

// Straight line to straight line distance
double linetoline(line3 u,line3 v) {
   point3 n=xmult(subt(u.a,u.b),subt(v.a,v.b));
   return fabs(dmult(subt(u.a,v.a),n))/vlen(n);
}
double linetoline(point3 u1,point3 u2,point3 v1,point3 v2) {
   point3 n=xmult(subt(u1,u2),subt(v1,v2));
   return fabs(dmult(subt(u1,v1),n))/vlen(n);
}

// The angle between two straight lines cos value
double angle_cos(line3 u,line3 v) {
   return
dmult(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.b));
}
double angle_cos(point3 u1,point3 u2,point3 v1,point3 v2) {
   return
dmult(subt(u1,u2),subt(v1,v2))/vlen(subt(u1,u2))/vlen(subt(v1,v2));
}

// The angle between two planes cos value
double angle_cos(plane3 u,plane3 v) {
   return dmult(pvec(u),pvec(v))/vlen(pvec(u))/vlen(pvec(v));
}
double angle_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3) {
   return
dmult(pvec(u1,u2,u3),pvec(v1,v2,v3))/vlen(pvec(u1,u2,u3))/vlen(pvec(v1,
v2,v3));
}

// Straight plane angle value sin
double angle_sin(line3 l,plane3 s) {
   return dmult(subt(l.a,l.b),pvec(s))/vlen(subt(l.a,l.b))/vlen(pvec(s));
}
double angle_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3) {
```

```
   return
dmult(subt(l1,l2),pvec(s1,s2,s3))/vlen(subt(l1,l2))/vlen(pvec(s1,s2,s3));
}
```

89. Vector Standard

```
#define pi        (2.0*acos(0.0))
#define ERR                    1e-9
#define PRE                    1e-8
#define EQ(a,b)    (fabs(a-b)<ERR)
#define CROSS(a,b,c,d) ((b.x-a.x)*(d.y-c.y)-(d.x-c.x)*(b.y-a.y))
//all angles in radian
//normally avoid asin
//vector3d or point3d
//atan2(y,x)
#define vectorVar3d double //change should be done here for different
datatype
struct vector3d {
   vectorVar3d x,y,z;
   vector3d(vectorVar3d  x1=0,vectorVar3d  y1=0,vectorVar3d  z1=0)  {
      x=x1;
      y=y1;
      z=z1;
   }
   int scan() {
      return scanf("%lf %lf %lf",&x,&y,&z);
   }
   int scanint() {
      return scanf("%d %d %d",&x,&y,&z);
   }
int scanLL() {
   return scanf("%lld %lld %lld",&x,&y,&z);
}

vector3d negate() {
   vector3d ret(-x,-y,-z);
   return ret;
}

vectorVar3d  length() {
   return sqrt(x*x+y*y+z*z);
}

vectorVar3d sqrLength() {
   return x*x+y*y+z*z;
}

vectorVar3d  length(vector3d b) { //length from a to b
   vector3d tem(x-b.x,y-b.y,z-b.z);
   return tem.length();
}

vector3d add(vector3d b) {
   vector3d ret(x+b.x,y+b.y,z+b.z);
   return ret;
}

vector3d substract(vector3d b) {
   vector3d ret(x-b.x,y-b.y,z-b.z);
   return ret;
}
```

```
vectorVar3d dot(vector3d b) {
   return x*b.x+y*b.y+z*b.z;
}

vector3d cross(vector3d b) {
   vector3d ret(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
   return ret;
}

vector3d lengthTransform(vectorVar3d  l) { //for unit vector l=1
   vectorVar3d  len=length();
   vector3d ret(x*l/len,y*l/len,z*l/len);
   return ret;
}

vectorVar3d angle(vector3d b) { //(0 to +pi)
   vectorVar3d  ret=dot(b)/(length()*b.length());
   if(ret<-1) ret=-1;
   if(ret>1) ret=1;
   return acos(ret);
}

//vectorVar3d angleWithSign(vector3d b) //(-pi to +pi)  (a to b)
//{
//if(cross(b)>0) return angle(b);
//  return -angle(b);
//}
//vector3d rotation(vectorVar3d  theta)  //under construction
//{
   //}
};

//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() {
      printf("%d %d", 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;
Vector operator + (Vector a, Vector b) {
  return Vector(a.x+b.x, a.y+b.y);
}
Vector operator - (Vector a, Vector b) {
  return Vector(a.x-b.x, a.y-b.y);
}
Vector operator * (Vector a, double p) {
  return Vector(a.x*p, a.y*p);
}
Vector operator / (Vector a, double p) {
  return Vector(a.x/p, a.y/p);
}
//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.push_back(p);
      if(l.inside(q)) ret.push_back(q);
      for(i = 0; i < ret.size(); i++)
        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.push_back(p);
    if(l.inside(q)) ret.push_back(q);
    for(i = 0; i < ret.size(); i++)
      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],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()+ar
eaOfArc(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])*chck>-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]).lineIntersectingPoint(line(
targetPoly[j],targetPoly[nextj])); //inside to outside
      }

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

```
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) //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]).shortestDistOfSegment(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]).shortestDistOfSegment(P[i]));
      j++;
      cntQ++;
    }

    else if(chck>pi&&cntP<n) { //P is near
      ans=min(ans,line(P[nexti],P[i]).shortestDistOfSegment(Q[j]));
      i++;
      cntP++;
    }

    else if(cntQ<m) { //only Q left
      ans=min(ans,line(Q[nextj],Q[j]).shortestDistOfSegment(P[i]));
      j++;
      cntQ++;
    }

    else { //only P left
      ans=min(ans,line(P[nexti],P[i]).shortestDistOfSegment(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
//Must required floating point values
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;
}

struct bspNode {
    line l;
    int left,right; //left for negative and right for positive
};
//1 based
//indBSP should be made 0
bspNode mainBSPNode[10000];
int indBSP;

//Binary Space Partitioning
void bsp(line arr[],int par,bool left,int n) {
    int now=rand()%n+1;
    //left or right child
    int temInd=(++indBSP);
    if(left) mainBSPNode[par].left=temInd;
    else mainBSPNode[par].right=temInd;

    //partioning the space into left and right parts
    line lft[n+2],right[n+2];
    int inl=0,inr=0;
    for(int i=1; i<=n; i++)
        if(i!=now) {
            if(arr[now].sideOfLine(arr[i].p)<0) {
                if(arr[now].sideOfLine(arr[i].q)<0) //both are left
                    lft[++inl]=arr[i];
                else { //left to right
                    Point intersectingPoint=arr[now].lineIntersectingPoint(arr[i]);
                    lft[++inl]=line(arr[i].p,intersectingPoint);
                    right[++inr]=line(intersectingPoint,arr[i].q);
                }
            }

            else if(arr[now].sideOfLine(arr[i].q)<0) { //right to left
```

```
            Point intersectingPoint=arr[now].lineIntersectingPoint(arr[i]);
            lft[++inl]=line(arr[i].q,intersectingPoint);
            right[++inr]=line(intersectingPoint,arr[i].p);
         }

         else right[++inr]=arr[i]; //both right
      }
   mainBSPNode[temInd].l=arr[now];
   mainBSPNode[temInd].left=0;
   mainBSPNode[temInd].right=0;
   if(inl) bsp(lft,temInd,true,inl);
   if(inr) bsp(right,temInd,false,inr);
}
int main() {
   line arr[200];
   int n;
   while(cin>>n) {
      for(int i=1; i<=n; i++) {
         arr[i].p.scan();
         arr[i].q.scan();
      }
      indBSP=0;
      bsp(arr,0,false,n);
      for(int i=1; i<=indBSP; i++) {
         print2("Now in index",i);
         mainBSPNode[i].l.print();
         print2("Left Child is ",mainBSPNode[i].left);
         print2("Right Child is ",mainBSPNode[i].right);
      }
   }
   return 0;
}
```

90. Circle Union

```
const double EPS = 1e-8;
const double PI  = acos(-1.0);
const double TAU = 2.0 * PI;
const double INF = 1e99;
int sig(double x) {
   return x < -EPS ? -1 : x > EPS ? 1 : 0;
}
template<class T> T pow2(T x) {
   return x * x;
}
class Vector {
public:
   double x, y;
   Vector() {}
   Vector(double x, double y): x(x), y(y) {}

   Vector operator -() const {
      return Vector(-x, -y);
   }
   Vector operator +(const Vector &v) const {
      return Vector(x+v.x, y+v.y);
   }
   Vector operator -(const Vector &v) const {
      return Vector(x-v.x, y-v.y);
   }
   Vector operator *(const double &s) const {
      return Vector(x * s, y * s);
   }
```

```
Vector operator /(const double &s) const {
   return Vector(x / s, y / s);
}
double operator *(const Vector &v) const {
   return x*v.x + y*v.y;
}
double operator ^(const Vector &v) const {
   return x*v.y - y*v.x;
}

// rotate vector (Right/Left hand)
Vector  R(double co, double si) {
   return Vector(x*co-y*si, y*co+x*si);
}
Vector  L(double co, double si) {
   return Vector(x*co+y*si, y*co-x*si);
}
Vector  R(double th) {
   return R(cos(th), sin(th));
}
Vector  L(double th) {
   return L(cos(th), sin(th));
}

double len2() {
   return x*x + y*y;
}
double len()  {
   return sqrt(len2());
}
double ang()  {
   return atan2(y, x);   // angle of vector
}
Vector e(double s = 1.0) {
   return *this / len() * s;
}
};
typedef Vector Point;


class Line {
public:
   Point a, b;
   Line() {}
   Line(Point a, Point b): a(a), b(b) {}
};

class Circle {
public:
   Point o;
   double r;
   Circle() {}
   Circle(Point o, double r): o(o), r(r) {}

   int posi(Circle c) {
      double d = (o - c.o).len();
      int in = sig(d - fabs(r - c.r)), ex = sig(d - (r + c.r));
      return in<0 ? -2 : in==0? -1 : ex==0 ? 1 : ex>0? 2 : 0;
   }
   Line chord(Circle c) {
      Vector v = c.o - o;
      double co = (pow2(r) + v.len2() - pow2(c.r)) / (2 * r * v.len());
```

```
    double si = sqrt(fabs(1.0 - pow2(co)));
    return Line(v.L(co, si).e(r) + o, v.R(co, si).e(r) + o);
  }
};
struct Range {
  double t;
  int evt;
  Point p;
  Range() {}
  Range(double t, int evt, Point p) : t(t), evt(evt), p(p) {}

  bool operator <(const Range &s) const {
    return sig(t - s.t) < 0 || (sig(t - s.t) == 0 && evt > s.evt);
  }
};
const int MAX_N = 1000 + 10;
Circle C[MAX_N];
Range R[MAX_N<<1];
// sort circle with desending of radii
bool cmp_r(const Circle &a, const Circle &b) {
  return a.r > b.r;
}

double segment_area(double r, double t) {
  return pow2(r) * (t - sin(t)) / 2;
}
double union_circle(Circle C[], int &n) {
  sort(C, C + n, cmp_r);
  int k = 0;
  for (int i = 0; i < n; i++) {
    if (sig(C[i].r) == 0) break;
  int j = 0;
  for (j = 0; j < k; j++)
    if (C[i].posi(C[j]) < 0 || !sig((C[i].o - C[j].o).len()))
      break;
  if (j == k)
    C[k++] = C[i];
}
n = k;

double ans = 0;
for (int i = 0; i < n; ++ i) {
  Point mpi = Point(- C[i].r, 0.0) + C[i].o;
  int nc = 0, rcnt = 0;
  R[rcnt++] = Range(-PI,  1, mpi);
  R[rcnt++] = Range( PI, -1, mpi);
  for (int j = 0; j < n; ++ j) {
    if (j == i || C[i].posi(C[j])) continue;

    Line l = C[i].chord(C[j]);
    double jR = (l.a - C[i].o).ang(), jL = (l.b - C[i].o).ang();

    if (sig(jR - jL) > 0) ++ nc;
    R[rcnt++] = Range(jR,  1, l.a);
    R[rcnt++] = Range(jL, -1, l.b);
  }
  sort(R, R + rcnt);
  double pj = - PI;
  Point  pp = mpi;
  for(int j = 0; j < rcnt; ++ j) {
    nc += R[j].evt;
    if((nc == 2 && R[j].evt > 0) || nc == 0)
```

```
                ans += segment_area(C[i].r, R[j].t - pj) + (pp ^ R[j].p) / 2;
            pj = R[j].t;
            pp = R[j].p;
        }
    }
    return ans;
}

int main() {
    int n;
    while (scanf("%d", &n) == 1) {
        if(n == 0) break;
        for (int i = 0; i < n; i++) scanf("%lf%lf%lf", &C[i].o.x, &C[i].o.y, &C[i].r);

        double ans = union_circle(C, n);
        printf("%.3f\n", ans);
    }
    return 0;
}
```