谁放肆了我们的眼泪

halfrost

谁放肆了我们的眼泪

Mr.Rock

halfrost

Standard Code Library

@halfrost
Department of Computer Science and Engineering
At Huazhong University of Science and Technology

July 25, 2012

前言

这是我第一次弄这么大的模板，做好了之后感到很有成就啊，这个模板收集了我从参加 ACM 以来总结的所有的经典问题的高效的代码，也算是为了今年 9，10 月份的区域赛做准备吧，希望能拿好成绩。

代码全部经过我调试验证过了。选取的都是时间最小的高效代码，也有一些是不知道其意的，先放进来，以后留着慢慢看。说好了是代码库，模板库，就只放一些模板型的了，至于灵活的变通与运用还是需要我去用的，这些无法描述出来，存在脑子了吧，这也许就叫做经验吧。这次代码库里面没有 DP 的内容，因为自己研究过几天，至于经验和心得写在本子上面了，在这里就不重复了。还有后缀数组的灵活变换，树状数组的奇妙，并查集的奇妙优化，等等这里不说了，这里只有单纯的代码。经验都存进大脑吧。。

在截稿之时，突然听闻华科的“精英班”的恐怖，差距确实很大，12 个小时的考试，不是一般人受得了的。我也要拼搏一把!!!

希望这是我的第一本，也是最后一本，以后代码全部争取存在脑子里面了，并且逐渐消化掉这本厚厚的书，为考研做点贡献吧，至于里面的错误，修订和一些理解，我会记下来，写在另外的本子上面的，要是多了，说不定还能再出一本外传，呵呵，开玩笑了，好了，好好欣赏和学习这里的代码库吧。

halfrost 在东湖畔

2012 年 7 月 25 日

Contents

1. Graph Theory and Network Algorithms

2. Number Theory

3. Geometry Theory

4. Advanced Data structures and Algorithms

5. Simulate Problem

目 录：

Chapter 1

Graph Theory and Network Algorithms

1.1 NP 搜索

```
/*==================================================*\
| 最大团问题 （当 n<64 时，快速的模板见浙大模板）
|       最大独立集+最小覆盖集=V
|       最大团=补图的最大独立集
|       最小覆盖集=最大匹配
\*==================================================*/
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;

int clique( int n,int *u,bool mat[][55],int size,int &max,int &bb,int *res,int *rr,int *c){
    int i,j,vn,v[55];
    if (n){
        if (size+c[u[0]]<=max) return 0;//要考虑的顶点数比已经有的最大团数还要少，不再计算下去
        for (i=0;i<n+size-max && i<n;i++){
            for (j=i+1,vn=0;j<n;j++){
                if (mat[u[i]][u[j]])
                    v[vn++]=u[j];
            }
            rr[size]=u[i];
            clique(vn,v,mat,size+1,max,bb,res,rr,c);
            if (bb) return 0;
        }
    }
    else if (size>max){
        max=size;
        for (i=0;i<size;i++){
            res[i]=rr[i];//res，用来存最大团的，也就是说 rr 存的是当前的最大团顶点
        }
        bb=1;//根据这个算法的性质，在搜索下去 max 只会变小，所以用 bb 标志，停止搜索
    }
    return 0;
}

int maxclique(int n,bool mat[][55],int *ret){
    int max=0,bb,c[55],i,j;
    int vn,v[55],rr[55];
    for (c[i=n-1]=0;i>=0;i--){
        for (vn=0,j=i+1;j<n;j++){
            if (mat[i][j])v[vn++]=j;
        }
        bb=0;//可能是标志作用
        rr[0]=i;
        clique(vn,v,mat,1,max,bb,ret,rr,c);
        c[i]=max;//这个记录的每次的最大个数
    }
    return max;
}
int main(){
    // freopen ("1.txt","r",stdin);
    int n,i,j;
    bool mat[55][55];
    int ret[55];
    while (scanf("%d",&n)==1&&n){
        for (i=0;i<n;i++){
            for (j=0;j<n;j++)
                scanf("%d",&mat[i][j]);
        }
        printf ("%d\n",maxclique(n,mat,ret));
    }
    return 0;
}
```

```
/*==================================================*\
| 竞赛图
|图中任意两点,至少存在一条有向边，构成一个哈密顿(好像和 哈密尔顿
|不同)路；
|对于一条以构建的哈密顿路的 n 个点(第 1 个点到第 n 个点,注意 第 i 个
|点 不一定是 点 i ,eg,1 - 4 - 2 - 3,第 2 个点为 4，第 3 个点为 2，第 4
|个点为 3),插入第 n+1 个点，肯定找到一条有向路与之相连 这里分 3
|种情况
|1> 第 n + 1 个点 与第一个点相连，使第 n + 1 个点成为第一个点
|2> 最后一个点与第 n+1 个点相连，使第 n + 1 个点成为最后一个点
|3> 能在原哈密顿路中找到一个相连的点 ，第 i 个点 和 第 i+1 个点 使
|得(第 i 个点与第 n+1 个点 且 第 n+1 个点与第 i+1 个点)相连，所以
|在第 i 个点与第 i+1 个点中插入第 n+1 个点
\*==================================================*/
#include <iostream>
#define M 101
#include <list>
using namespace std;
bool f[ M ][ M ];
bool flag;
list <int> L;
list <int>::iterator it , pre;

void init (int n){
    for (int i = 1;i <= n;++ i)
        for (int j = 1;j <= n;++ j)
            f[ i ][ j ] = false;
}
void solve (int n){
    int i;
    L.push_back(1);
    for (i = 2;i <= n;++ i){
        it = L.begin ();
        if (f[ i ][ *it ]){
            L.push_front(i);
            continue;
        }
        it = L.end();
        -- it;
        if (f[ *it ][ i ]){
            L.push_back(i);    continue;
        }
        it = pre = L.begin();
        ++ it;
        while (it != L.end()){
            if (f[ *pre ][ i ] && f[ i ][ *it ]){
                L.insert(it , i);
                break;
            }
            pre ++;
            it ++;
        }
    }
}
int main(){
    int t , n , i , k , g;
    int a , b;
    scanf ("%d" , &t);
    while (t --){
        L.clear();
        scanf ("%d" , &n);
        k = n * (n - 1) / 2;
```

```
        init (n);
        flag = true;
        for (i = 1;i <= k;++ i){
            scanf ("%d %d" , &a , &b);
            f[ a ][ b ] = true;
        }
        solve (n);
        if (L.size() != n){
            puts ("Impossible");
            continue;
        }
        for (it = L.begin(); it != L.end();++ it){
            if (it != L.begin()) printf (" ");
            printf ("%d" , * it);
        }
        printf ("\n");
    }
    return 0;
}
```

1.2 连通性

```
/*=====================================================*\
| 有向图强连通分量 Kosaraju 算法
| 步骤
| 1.DFS 有向图 G，并以后根序记录节点
| 2.把存在于记录集中且最后访问节点作为起点，DFS 反图 GT,并以先根
|    序把节点从记录中剔除；
| 3.若此次不能 DFS 反图 GT 所有节点，则重复步骤 2，直到所有节点都
|被剔除出记录；每次剔除掉的节点集即为原有向图 G 的一个强连通分量
|
| 本算法慢，但是它得到了一个连通分量的拓扑序，有时用的上
|\*=====================================================*/
#include <cstdio>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
#define NMAX 11000

vector< vector< int > > path;
vector< vector< int > > npath;
int n,m, scc;
int order[NMAX], order_pos, id[NMAX], id_total[NMAX];
bool vis[NMAX];
int out_degre[NMAX];

void dfs(int pos)
{
    int i,j,l;
    vis[pos] = true;
    l = path[pos].size();
    for (i=0;i<l;i++) {
        j = path[pos][i];
        if (!vis[j]) {
            dfs(j);
        }
    }
    order[ order_pos ++ ] = pos;//make order
}

void ndfs(int pos)
{
    int i,j,l;
    vis[pos] = true;
    id[pos] = scc;
    l = npath[pos].size();
    for (i=0;i<l;i++) {
        j = npath[pos][i];
        if (!vis[j]) {
            ndfs(j);
        }
    }
}

void Kosaraju()
{
    int i,j,l;
    //dfs in original graph
    memset(vis, 0, sizeof(vis));
    for (i=1; i<=n ;i++) {
        if (!vis[i]) {
            dfs(i);
        }
    }
    //dfs in inverse graph
    memset(vis, 0, sizeof(vis));
    memset(id, 0, sizeof(id));
    scc = 1;
    for (i=order_pos-1; i>=0 ;i--) {
        if (!vis[ order[i] ]) {
            ndfs(order[i]);
            scc ++;
        }
    }
    //statist
    scc --;
    memset(id_total, 0, sizeof(id_total));
    for (i=1;i<=n;i++) {
        id_total[ id[i] ] ++;
    }
    memset(out_degre, 0, sizeof(out_degre));
    for (i=1;i<=n;i++) {
        l = path[i].size();
        int id1 = id[i];
        for (j=0;j<l;j++) {
            int id2 = id[ path[i][j] ];
            if (id1 != id2) {//id1 -> id2
                out_degre[id1] ++;
            }
        }
    }
    int ans_id,zero_degre = 0;
    for (i=1;i<=scc;i++) {
        if (out_degre[i] == 0) {
            zero_degre ++;
            ans_id = i;
        }
    }
    if (zero_degre > 1) {
        printf("0\n");
    }
    else {
        printf("%d\n",id_total[ ans_id ]);
    }
}

int main()
{
    int i,j;
    while (scanf("%d %d",&n,&m)==2) {
        path.resize(n+10);
        npath.resize(n+10);
        for (i=0;i<=n;i++) {
            path[i].clear();
            npath[i].clear();
        }
        order_pos = 0;
        //set graph
        for (i=0;i<m;i++) {
            int x,y;
            scanf("%d %d",&x,&y);
            path[x].push_back(y);
            npath[y].push_back(x);
        }
        Kosaraju();
    }
}
/*=====================================================*\
| 有向图强连通分量 Tarjan 算法
| 扩展：求出强连通后把强连通缩点是一个 DAG，对 DAG 求传递闭包
| 复杂度是 O(E)的，可借此求有向图的连通性。
```

| (poj 2553 The Bottom of a Graph)
==/

```
#include <cstdio>
#include <algorithm>
#include <iostream>
#include <set>
using namespace std;

const int MAXN = 5001;
int V, E;//点数、边数
int pt, ptR;
struct node
{
    int v, next;//当前节点，下一节点位置
};
node adjlst[MAXN * 1000];//邻接表

bool sink[MAXN];
int prev[MAXN], low[MAXN], stk[MAXN], sc[MAXN];
int cnt[MAXN];
int cnt0, ptr, cnt1;
void dfs(int w)
{
        int Min(0);
        prev[w] = cnt0++;
        low[w] = prev[w];
        Min = low[w];
        stk[ptr++] = w;
        for(int i = adjlst[w].next; i != -1; i = adjlst[i].next)
        {
                int t = adjlst[i].v;
                if(prev[t] == -1)
                        dfs(t);
                if(low[t] < Min)
                        Min = low[t];
        }
        if(Min < low[w])
        {
                low[w] = Min;
                return;
        }
        do
        {
                int v = stk[--ptr];
                sc[v] = cnt1;
                low[v] = MAXN;
        }while(stk[ptr] != w);
        ++cnt1;
}
void Tarjan(int N)
{//传入 N 为点数，结果保存在 sc 数组中，同一标号的点在同一个强连通分量内，
//强连通分量数为 cnt1
        cnt0 = cnt1 = ptr = 0;
        int i;
        for(i = 0; i < N; ++i)
                prev[i] = low[i] = -1;
        for(i = 0; i < N; ++i)
                if(prev[i] == -1)
                        dfs(i);
};
int last[MAXN];
int main()
{
        int i, j;
      for(; ;)
      {
            scanf("%d", &V);
            if(!V) break;
            scanf("%d", &E);
            for(i = 0; i < V; i++)
            {
                  adjlst[i].next = -1;
                  last[i] = i;
            }
            pt = V;
            ptR = V;
            for(i = 0; i < E; i++)
            {
                  int a, b;
                  scanf("%d %d", &a, &b);
                  if(a == b) continue;
                  int pa = --a, pb = --b;
                  //while(adjlst[pa].next != -1)
                  pa = last[a]; //adjlst[pa].next;
                  adjlst[pa].next = pt;
                  adjlst[pt].v = b;
                  last[a] = pt;
                  adjlst[pt++].next = -1;
            }
            Tarjan(V);
            set<int> kn;
            for(i = 0; i < V; i++)
            {
                  for(int t = adjlst[i].next; t != -1; t = adjlst[t].next)
                        if(sc[i] != sc[adjlst[t].v])
                        {
                              kn.insert(sc[i]);
                              break;
                        }
            }
            for(i = 0; i < V; ++i)
               sink[i] = false;
            for(i = 0; i < V; i++)
                  if(kn.find(sc[i]) == kn.end() ) sink[i] = true;
            bool flag(false);
            for(i = 0; i < V; i++)
            {
                  if(sink[i])
                  {
                        if(flag) printf(" ");
                        flag = true;
                        printf("%d", i + 1);
                  }
            }
            printf("\n");
      }
      return 0;
}
```

/*==*\
| 有向图强连通分量 Tarjan 算法(Tarjan 算法+静态邻接表)
| ////////只判断 是否图上每两个点都连通，不输出路径
| (HDU 1269 迷宫城堡)
==/

```
#include <iostream>
#include <stack>
using namespace std;
typedef struct
{
      long v,next;
}Edge;
typedef struct
{
      long Num;
      long Used;
      long Alive;
      long Low;
      long belong;
      void init(long pos)
      {
            Num=Used=Alive=Low=0;
            belong=pos;
      }
}Node;
const long MAXN=100010;
long N,M;                  //点，边
Edge e[MAXN];              //边数组
long p[MAXN/10];           //辅助数组
Node vec[MAXN/10];         //点
stack<long> s;
long Permit;//时间戳
```

```
inline void init()
{
	long i;
	while (!s.empty())
	{
		s.pop();
	}
	Permit=0;
	memset(p,-1,sizeof(p));
	for (i=0;i<=N;++i)
	{
		vec[i].init(i);
	}
	for (i=0;i<M;++i)
	{
		long from,to;
		scanf("%ld %ld",&from,&to);
		e[i].next=p[from];
		e[i].v=to;
		p[from]=i;
	}
}
inline void update(long &a,long b)
{
	if(a>b) a=b;
}
inline void dfs(long pos)
{
	s.push(pos);
	vec[pos].Low=vec[pos].Num=++Permit;
	vec[pos].Used=vec[pos].Alive=true;
	long j;
	for (j=p[pos];j!=-1;j=e[j].next){
		long to=e[j].v;
		if (vec[to].Used){
			if (vec[to].Alive){
				update(vec[pos].Low,vec[to].Num);
			}
		}
		else
		{
			dfs(to);
			update(vec[pos].Low,vec[to].Low);
		}
	}
	if (vec[pos].Num==vec[pos].Low){
		long t;
		while ((t=s.top())!=pos){
			vec[t].belong=pos;
			vec[t].Alive=false;
			s.pop();
		}
		vec[pos].belong=pos;
		vec[pos].Alive=false;
		s.pop();
	}
}
inline void Tarjan(){
	long i;
	for (i=1;i<=N;++i){
		if (!vec[i].Used) {
			dfs(i);
		}
	}
}
int main(){
	while (scanf("%ld %ld",&N,&M)!=EOF){
		if (N==0&&M==0){
			break;
		}
		init();
		Tarjan();
		long i;
		long t=vec[1].belong;
		bool doit=true;
		for (i=2;i<=N;++i){
			if (vec[i].belong!=t){
				doit=false;
				break;
			}
		}
		if (doit){
			printf("Yes\n");
		}
		else{
			printf("No\n");
		}
	}
	return 0;
}
/*==================================================*\
| 有向图强连通分量 Gabow 算法
| 速度比 Tarjan 快一点，但是一般还是用 Tarjan，
| 当 Tarjan 也会超时时候，只有这种方法了
\*==================================================*/
#include<cstdio>
#include<cstring>
#define MAX 0x01010101
#define MIN(x,y) (x<y)?x:y
#define V 1005
#define E 5005
#define nStack 1005
struct node
{
	int u, v;
	node *nxt;
}*pp, pool[E], *adj[V];
int v, e;
int belong[V], nPart;
node * Add_Edge(int u, int v, node *Nxt)
{
	pp->u = u;
	pp->v = v;
	pp->nxt = Nxt;
	return pp++;
}
int stk1[nStack],top1;
int stk2[nStack],top2;
int low[V], num[V], times;
void DFS(int cur){
    low[cur] = ++ times;
    stk1[++ top1] = cur;
    stk2[++ top2] = cur;
    for (node * tp = adj[cur]; tp; tp = tp->nxt ){
        int idx = tp->v;
        if (low[idx] == MAX)//本点未曾访问
            DFS(idx);
        else if ( belong[idx] == 0 )//本点已经访问过，但是未有归属，即
在堆栈中
            while ( low[stk2[top2]] > low[idx])
                -- top2;
    }
    if ( stk2[top2] == cur )
    {
        -- top2; ++ nPart;
        do
        {
            belong[stk1[top1]] = nPart;
        }
        while ( stk1[top1 --]!=cur );
    }
}
void init(int Nv)
{
	/*Pool*/
	pp = pool;
	memset(adj, 0, sizeof(adj[0]) * (Nv + 2));
	/*DFS*/
	memset(low, 0x01, sizeof(low[0]) *(Nv + 2) );
	memset(belong, 0, sizeof(belong[0]) * (Nv + 2));
	times = 0;
	top1 = 0;
```

```
        top2 = 0;
        /*Main*/
        nPart = 0;
}

//FUNCTION:
//Number of SCC
void QueryNp()
{
        printf("%d\n", nPart);
}

//Rebuild the Graph with SCC
bool ava[V][V];
void Rebuild(int Nv)
{
        memset(ava, 0, sizeof(ava));
        for (node *tp = pool; tp != pp; tp ++)
                ava[belong[tp->u]][belong[tp->v]] = true;
        for (int i = 1; i <= Nv; i ++) ava[i][i] = true;
}

void Floyd()
{
        for (int k = 1; k <= nPart; k ++)
                for (int i = 1; i <= nPart; i ++)
                        if (ava[i][k])
                                for (int j = 1; j <= nPart; j ++)
                                        if (ava[k][j]) ava[i][j] = true;
}
int main()
{
        while (scanf("%d%d", &v, &e) == 2 )
        {
                init(v);
                while (e --)
                {
                        int a, b;
                        scanf("%d%d", &a, &b);
                        adj[a] = Add_Edge(a, b, adj[a]);
                }

                for (int i = 1; i <= v; i ++)
                        if (low[i] == MAX) DFS(i);

                Rebuild(v);
                Floyd();

                int Q;
                scanf("%d", &Q);        ////询问次数
                while (Q --)
                {
                        int a, b;
                        scanf("%d%d", &a, &b);   ////询问哪两个点是否可连通
                        if (ava[belong[a]][belong[b]]) printf("Yes\n");
                        else printf("No\n");
                }
        }
        return 0;
}
/*==================================================*\
| 无向图连通分支(dfs/bfs 邻接阵)
| DFS / BFS / 并查集
\*==================================================*/
/*==================================================*\
| 有向图强连通分支(dfs/bfs 邻接阵)O(n^2)
\*==================================================*/
//返回分支数,id 返回 1..分支数的值
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0

#define MAXN 100
void search(int n,int mat[][MAXN],int* dfn,int* low,int
now,int& cnt,int& tag,int* id,int* st,int& sp){
        int i,j;
         dfn[st[sp++]=now]=low[now]=++cnt;
      for (i=0;i<n;i++)
            if (mat[now][i]){
               if (!dfn[i]){
                    search(n,mat,dfn,low,i,cnt,tag,id,st,sp);
                      if (low[i]<low[now])
                          low[now]=low[i];
               }
               else if (dfn[i]<dfn[now]){
                      for (j=0;j<sp&&st[j]!=i;j++);
                          if (j<cnt&&dfn[i]<low[now])
                              low[now]=dfn[i];
                      }
            }
            if (low[now]==dfn[now])
               for (tag++;st[sp]!=now;id[st[--sp]]=tag);
}
int find_components(int n,int mat[][MAXN],int* id){
        int ret=0,i,cnt,sp,st[MAXN],dfn[MAXN],low[MAXN];
        for (i=0;i<n;dfn[i++]=0);
        for (sp=cnt=i=0;i<n;i++)
            if (!dfn[i])
            search(n,mat,dfn,low,i,cnt,ret,id,st,sp);
        return ret;
}

//有向图强连通分支,bfs 邻接阵形式,O(n^2)
//返回分支数,id 返回 1..分支数的值
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0

#define MAXN 100
int find_components(int n,int mat[][MAXN],int* id){
         int ret=0,a[MAXN],b[MAXN],c[MAXN],d[MAXN],i,j,k,t;
         for (k=0;k<n;id[k++]=0);
         for (k=0;k<n;k++)
              if (!id[k]){
                 for (i=0;i<n;i++)
                 a[i]=b[i]=c[i]=d[i]=0;
                 a[k]=b[k]=1;
                 for (t=1;t;)
                    for (t=i=0;i<n;i++){
                       if (a[i]&&!c[i])
                          for (c[i]=t=1,j=0;j<n;j++)
                             if (mat[i][j]&&!a[j])
                                 a[j]=1;
                       if (b[i]&&!d[i])
                          for (d[i]=t=1,j=0;j<n;j++)
                             if (mat[j][i]&&!b[j])
                                 b[j]=1;
                     }
               for (ret++,i=0;i<n;i++)
                  if (a[i]&b[i])
                    id[i]=ret;
        }
    return ret;
 }
/*==================================================*\
| 有向图最小点基(邻接阵)O(n^2)
| 点基 B 满足：对于任意一个顶点 Vj，一定存在 B 中的一个 Vi,使得 Vi
是 Vj 的前代。
|返回点基大小和点基
|传入图的大小 n 和邻接阵 mat,不相邻点边权 0
|需要调用强连通分支
\*==================================================*/
#define MAXN 100
int base_vertex(int n,int mat[][MAXN],int* sets){
      int ret=0,id[MAXN],v[MAXN],i,j;
      j=find_components(n,mat,id);
    for (i=0;i<j;v[i++]=1);
    for (i=0;i<n;i++)
      for (j=0;j<n;j++)
         if (id[i]!=id[j]&&mat[i][j])
            v[id[j]-1]=0;
    for (i=0;i<n;i++)
         if (v[id[i]-1])
           v[id[sets[ret++]=i]-1]=0;
return ret;
}
```

1.3 生成树和树形图

```
/*==================================================*\
| 最小生成树(Prime+邻接表)
| 使用优先队列+邻接表的 prime
\*==================================================*/
#include <iostream>
#include <queue>
using namespace std;
 typedef struct
  {
      long v;
      long next;
      long cost;
 }Edge;
 typedef struct
  {
      long v;
      long cost;
 }node;
 bool operator <(const node &a,const node &b)
  {
      return a.cost>b.cost;
 }
 priority_queue<node> q;
 const long MAXN=10000;
 Edge e[MAXN];
 long p[MAXN];
 bool vist[MAXN];
 long m,n;
 long from,to,cost;
 void init()
  {
      memset(p,-1,sizeof(p));
      memset(vist,0,sizeof(vist));
      while (!q.empty())
       {
           q.pop();
      }
      long i;
      long eid=0;
      for (i=0;i<n;++i)
       {
           scanf("%ld %ld %ld",&from,&to,&cost);
           e[eid].next=p[from];
           e[eid].v=to;
           e[eid].cost=cost;
           p[from]=eid++;
           //以下适用于无向图
           swap(from,to);
           e[eid].next=p[from];
           e[eid].v=to;
           e[eid].cost=cost;
           p[from]=eid++;
      }
 }
 void print(long cost)
  {
      printf("%ld\n",cost);
 }
 void Prime()
  {
      long cost=0;
      init();
      node t;
      t.v=from;//选择起点
      t.cost=0;
      q.push(t);
      long tt=0;
      while (!q.empty()&&tt<m)
       {
           t=q.top();
           q.pop();
           if (vist[t.v])
            {
                continue;
           }
           cost+=t.cost;
           ++tt;
           vist[t.v]=true;
           long j;
           for (j=p[t.v];j!=-1;j=e[j].next)
            {
                if (!vist[e[j].v])
                 {
                     node temp;
                     temp.v=e[j].v;
                     temp.cost=e[j].cost;
                     q.push(temp);
                }
           }
      }
      print(cost);
 }
 int main()
  {
      while (scanf("%ld %ld",&m,&n)!=EOF&& m && n)
       {
           Prime();
      }
      return 0;
 }
/*==================================================*\
| 最小生成树(Kruskal+邻接表)
| 使用优先队列存储边信息，用并查集判定是否已经连通
| 复杂度 O(eloge)
\*==================================================*/
#include <iostream>
#include <queue>
using namespace std;
const long MAXN=10000;
long hash[MAXN];
 long m,n;                //点数  边数
                          //m 标号从 1 开始
 typedef struct
  {
      long from;
      long to;
      long cost;
 }Edge;
 bool operator <(const Edge &a, const Edge &b)
  {
      return a.cost>b.cost;
 }
 priority_queue<Edge> q;
 Edge e;
 void MakeSet()
  {
      long i;
      for (i=0;i<=m;++i)
       {
           hash[i]=i;
      }
 }
 long Find(long i)
  {
      long r=i;
      while (hash[r]!=r)
       {
           r=hash[r];
      }
      while (hash[i]!=r)
       {
           long j=hash[i];
           hash[i]=r;
           i=j;
      }
      return r;
 }
 void Unition(long x,long y)
  {
```

```
        long fx=Find(x);
        long fy=Find(y);
        if (fx!=fy)
         {
            hash[fx]=fy;
        }
 }
 void Init()
  {
        while (!q.empty())
         {
            q.pop();
        }
        MakeSet();
        long i;
        for(i=0;i<n;++i)
         {
            scanf("%ld  %ld  %ld",&(e.from),&(e.to),&(e.cost));//得到
源点 终点
            q.push(e);
            //以下为无向图的处理
            swap(e.from,e.to);
            q.push(e);
        }
 }
 void print(long cost)
  {
        printf("%ld\n",cost);
 }
 void Kruskal()
  {
        Init();
        long t=0;//表示合并次数
        long cost=0;
        Edge e;
        while (!q.empty()&&t<m-1)
         {
            e=q.top();
            long v1=e.from;
            long v2=e.to;
            if (Find(v1)!=Find(v2))
             {
                Unition(v1,v2);
                cost+=e.cost;
                ++t;
            }
            q.pop();
        }
        print(cost);
 }
 int main()
  {
        while(scanf("%ld %ld",&m,&n)!=EOF)//输入点与边
         {
            If(m==0) break;
            Kruskal();
        }
        return 0;
 }
/*==================================================*\
| 最小树形图（朱永津刘振宏算法，朱刘算法）
| 即有向的最小生成树，
| 不定根的时候要虚拟一个根，这个自己去变通
\*==================================================*/
#define M 101
#define type int
#define inf INT_MAX
struct Node{
        int u , v;
        type cost;
}E[M*M];
int pre[M],ID[M],vis[M];
type In[M];
type Directed_MST(int root,int NV,int NE) {
        type ret = 0;
        while(true) {
                //1.找最小入边
                FF(i,NV) In[i] = inf;
                FF(i,NE) {
                        int u = E[i].u;
                        int v = E[i].v;
                        if(E[i].cost < In[v] && u != v) {
                                pre[v] = u;
                                In[v] = E[i].cost;
                        }
                }
                FF(i,NV) {
                        if(i == root) continue;
                        if(In[i] == inf)      return  -1;//除了跟以外有点没有入
边,则根无法到达它
                }
                //2.找环
                int cntnode = 0;
                CC(ID,-1);
                CC(vis,-1);
                In[root] = 0;
                FF(i,NV) {//标记每个环
                        ret += In[i];
                        int v = i;
                        while(vis[v] != i && ID[v] == -1 && v != root) {
                                vis[v] = i;
                                v = pre[v];
                        }
                        if(v != root && ID[v] == -1) {
                                for(int u = pre[v] ; u != v ; u = pre[u]) {
                                        ID[u] = cntnode;
                                }
                                ID[v] = cntnode ++;
                        }
                }
                if(cntnode == 0) break;//无环
                FF(i,NV) if(ID[i] == -1) {
                        ID[i] = cntnode ++;
                }
                //3.缩点,重新标记
                FF(i,NE) {
                        int v = E[i].v;
                        E[i].u = ID[E[i].u];
                        E[i].v = ID[E[i].v];
                        if(E[i].u != E[i].v) {
                                E[i].cost -= In[v];
                        }
                }
                NV = cntnode;
                root = ID[root];
        }
        return ret;
}
/*==================================================*\
| 最小树形图（朱永津刘振宏算法，朱刘算法）
| 即有向的最小生成树，
| 一个具体的例子   POJ   3164   (用 C++交,G++对 double 报错)
\*==================================================*/
#include<iostream>
#include<cmath>
#include<cstdio>
#include<string.h>
#define INF 1000000000
using namespace std;
double map[110][110];
bool visit[110],circle[110];
int pre[110];
int n,m;
struct PT
{
      double x,y;
}p[110];
double dist(int i,int j)
{
      return
sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x)+(p[i].y-p[j].y)*(p[i].y-p[j].y));
}
```

```
void dfs(int t)
{
	if(visit[t])
		return ;
	visit[t]=1;
	for(int i=1;i<=n;i++)
		if(map[t][i]<INF)
			dfs(i);
}
bool connect()//深搜，判断是否存在最小树形图
{
	dfs(1);
	for(int i=1;i<=n;i++)
		if(!visit[i])
			return 0;
	return 1;
}
double solve()
{
	double ans=0;
	int i,j,k;
	memset(circle,0,sizeof(circle));// 如果某点被删掉， 那么 circle[i]=1
	while(1)
	{
		for(i=2;i<=n;i++)//求出每个点的最小入边
		{
			if(circle[i])
				continue;
			map[i][i]=INF;
			pre[i]=i;
			for(j=1;j<=n;j++)
			{
				if(circle[j])
					continue;
				if(map[j][i]<map[pre[i]][i])
					pre[i]=j;
			}
		}
		for(i=2;i<=n;i++)//遍历找环
		{
			if(circle[i])
				continue;
			j=i;
			memset(visit,0,sizeof(visit));
			while(!visit[j]&&j!=1)
			{
				visit[j]=1;
				j=pre[j];
			}
			if(j==1)//j==1 说明 i 不在环上
				continue;
			i=j;//找到了环
			ans+=map[pre[i]][i];
			for(j=pre[i];j!=i;j=pre[j])
			{
				ans+=map[pre[j]][j];
				circle[j]=1;//用环上一点 i 代表此环，其他点删去，即 circle[j]=1
			}
			for(j=1;j<=n;j++)
			{
				if(circle[j])
					continue;
				if(map[j][i]<INF)
					map[j][i]-=map[pre[i]][i];//更新 j 的入边
			}
			for(j=pre[i];j!=i;j=pre[j])//环上所有点的最优边为人工顶点的边
			{
				for(k=1;k<=n;k++)
				{
					if(circle[k])
						continue;
					if(map[j][k]<INF)
						map[i][k]=min(map[i][k],map[j][k]);
					if(map[k][j]<INF)
						map[k][i]=min(map[k][i],map[k][j]-map[pre[j]][j]);
				}
			}
			break;
		}
		if(i>n)
		{
			for(j=2;j<=n;j++)
			{
				if(circle[j])
					continue;
				ans+=map[pre[j]][j];
			}
			break;
		}
	}
	return ans;
}
int main()
{
	int i,j;
	int a,b;
	while(scanf("%d%d",&n,&m)!=EOF)
	{
		for(i=1;i<=n;i++)
			scanf("%lf%lf",&p[i].x,&p[i].y);
		for(i=1;i<=n;i++)
			for(j=1;j<=n;j++)
				map[i][j]=INF;
		for(i=1;i<=m;i++)
		{
			scanf("%d%d",&a,&b);
			map[a][b]=dist(a,b);
		}
		memset(visit,0,sizeof(visit));
		if(!connect())
			printf("poor snoopy\n");   /////这就是不能生成树的情况
		else printf("%.2lf\n",solve());
	}
		return 0;
}
/*==================================================*\
|最优比率生成树
|
|代价/距离为 r.
|r = sigma(Ci*Xi)/sigma(Di*Xi);Xi 为 0 或 1
| Z(r) = sigma(Ci*Xi) - sigma(Di*Xi) * r;
|则 Z(r)=0 时，r 为解.
|我们将边权变为 Ci-Di*r 求最小生成树
\*==================================================*/
#include <iostream>
#include <queue>
#include <cmath>
using namespace std;
#define EPS 1E-5
#define MAXV 1005
#define DINF 100000000.00

struct Point {

	int x;
	int y;
	int z;
}pt[MAXV];

double distance(int i, int j) {

	return sqrt(1.0*(pt[ i ].x - pt[ j ].x)*(pt[ i ].x - pt[ j ].x) + 1.0*(pt[ i ].y - pt[ j ].y)*(pt[ i ].y - pt[ j ].y));
}

double higher(int i, int j) {

	if(pt[ i ].z > pt[ j ].z) return pt[ i ].z - pt[ j ].z;
```

```
        return pt[ j ].z - pt[ i ].z;
}

double dist[MAXV][MAXV];
double high[MAXV][MAXV];
bool vist[MAXV];
int pre[MAXV];
double dis[MAXV];
int n;

void init() {
    memset(vist, false, sizeof(vist));
}

double prim(double r) {
    int i, j, u;
    init();
    for(i = 1; i <= n; ++ i) {
        dis[ i ] = high[ 1 ][ i ] - dist[ 1 ][ i ] * r;
            pre[ i ] = 1;
     }
     dis[ 1 ] = 0; vist[ 1 ] = true;
     double hig = 0, di = 0;
     for(i = 1; i < n; ++ i) {
        double Min = DINF;
        for(j = 2; j <= n; ++ j) {
                if(!vist[ j ] && Min > dis[ j ]) {
                        Min = dis[ j ];
                        u = j;
                }
            }
        vist[ u ] = true;
            hig += high[pre[ u ]][ u ];
             di += dist[pre[ u ]][ u ];
        for(j = 2; j <= n; ++ j) {
             double val = high[ u ][ j ] - dist[ u ][ j ] * r;
             if(!vist[ j ] && dis[ j ] > val) {
                   dis[ j ] = val;
                   pre[ j ] = u;
             }
          }
     }
    return hig / di;
}

int main() {

        int i, j;
        double a, b;
        while(~scanf("%d", &n) && n) {
            for(i = 1; i <= n; ++ i) {
                scanf("%d %d %d", &pt[ i ].x, &pt[ i ].y, &pt[ i ].z);
            }
            for(i = 1; i <= n; ++ i) {
                for(j = i + 1; j <= n; ++ j) {
                        dist[ j ][ i ] = dist[ i ][ j ] = distance(i, j);
                        high[ j ][ i ] = high[ i ][ j ] = higher(i, j);
                }
            }
            a = 0;
            while(true) {
                b = prim(a);
                if(fabs(a-b)<EPS) break;
                a = b;
            }
            printf("%.3lf\n", a);
        }
        return 0;
}

/*==================================================*\
| Prim 求 MST（最小生成树）
| INIT: cost[][]耗费矩阵(inf 为无穷大);
| CALL: prim(cost, n); 返回-1 代表原图不连通;
\*==================================================*/
#define typec int // type of cost
const typec inf = 0x3f3f3f3f;              // max of cost
int vis[V]; typec lowc[V];
typec prim(typec cost[][V], int n)          // vertex: 0 ~ n-1
{
        int i, j, p;
        typec minc, res = 0;
        memset(vis, 0, sizeof(vis));
        vis[0] = 1;
        for (i=1; i<n; i++) lowc[i] = cost[0][i];
        for (i=1; i<n; i++) {
            minc = inf; p = -1;
            for (j=0; j<n; j++)
              if (0 == vis[j] && minc > lowc[j]) {
                  minc = lowc[j]; p = j;
              }
             if (inf == minc) return -1; // 原图不连通
             res += minc; vis[p] = 1;
          for (j=0; j<n; j++)
               if (0 == vis[j] && lowc[j] > cost[p][j])
                lowc[j] = cost[p][j];
        }
        return res;
}
/*==================================================*\
| 次小生成树 O(V^2)
\*==================================================*/
```

结论 次小生成树可由最小生成树换一条边得到.
证明: 可以证明下面一个强一些的结论:
T 是某一棵最小生成树,T0 是任一棵异于 T 的树,通过变换 T0 --> T1 --> T2 --> ... --> Tn (T) 变成最小生成树.所谓的变换是，每次把 T_i 中的某条边换成 T 中的一条边，而且树 T_(i+1)的权小于等于 T_i 的权.
具体操作是：
step 1. 在 T_i 中任取一条不在 T 中的边 u_v.
step 2. 把边 u_v 去掉，就剩下两个连通分量 A 和 B，在 T 中，必有唯一的边 u'_v' 连结 A 和 B.
step 3. 显然 u'_v'的权比 u_v 小 (否则,u_v 就应该在 T 中).把 u'_v' 替换 u_v 即得树 T_(i+1).
特别地：取 Tn 为任一棵次小生成树，T_(n-1) 也就是次小生成树且跟 T 差一条边. 结论得证.
算法：只要充分利用以上结论，即得 V^2 的算法. 具体如下：

step 1. 先用 prim 求出最小生成树 T. 在 prim 的同时，用一个矩阵 max[u][v] 记录在 T 中连结任意两点 u,v 的唯一的路中权值最大的那条边的权值. (注意这里). 这是很容易做到的，因为 prim 是每次增加一个结点 s，而设已经标号了的结点集合为 W，则 W 中所有的结点到 s 的路中的最大权值的边就是当前加入的这条边. step 1 用时 O(V^2).

step 2. 枚举所有不在 T 中的边 u_v，加入边 u_v 替换权为 max[u][v] 的边.不断更新求最小值，即次小生成树. step 2 用时 O(E).故总时间为 O(V^2).

```
/*==================================================*\
#include<stdio.h>
#include<string.h>
const int N=110;
const int inf=0x3f3f3f3f;
int edge[N][N];
int max_val[N][N];
int vis[N],lowc[N];
int pre[N];                     //记录到 i 的最短距离的点
bool visited[N][N];
int max(int a,int b)
{
    return a>b?a:b;
}
int prim(int n)
{
    int i,j,k,p;
    int minc,res=0;
    memset(vis,0,sizeof(vis));
    memset(pre,0,sizeof(pre));
    memset(max_val,0,sizeof(max_val));
    vis[1]=1;
    pre[1]=1;
    for(i=2;i<=n;i++)//初始化
    {
        lowc[i]=edge[1][i];
```

```
        pre[i]=1;
    }
    for(i=2;i<=n;i++)
    {
        minc=inf;
        p=-1;
        for(j=1;j<=n;j++)
            if(vis[j]==0&&minc>lowc[j])
            {
                minc=lowc[j];
                p=j;
            }
        max_val[pre[p]][p]=minc;
        visited[pre[p]][p]=0;
        visited[p][pre[p]]=0;
        for(k=1;k<=n;k++)
      max_val[k][p]=max(max_val[pre[p]][p],max_val[k][pre[p]]);
        res+=minc;
        vis[p]=1;
        for(j=1;j<=n;j++)//更新
            if(vis[j]==0&&lowc[j]>edge[p][j])
            {
                lowc[j]=edge[p][j];
                pre[j]=p;
            }
    }
    return res;
}
int main()
{
    int casee,n,m,i,j;
    int start,end,w;
    int min1,min2;
    scanf("%d",&casee);
    while(casee--)
    {
        scanf("%d%d",&n,&m);
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                edge[i][j]=inf;
        memset(max_val,0,sizeof(max_val));
        memset(visited,0,sizeof(visited));
        for(i=1;i<=m;i++)
        {
            scanf("%d%d%d",&start,&end,&w);
            edge[start][end]=w;
            edge[end][start]=w;
            visited[start][end]=1;
            visited[end][start]=1;
        }
        min1=prim(n);
      // printf("%d\n",min1);
        bool flag=true;
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                if(edge[i][j]==inf||visited[i][j]==0)
                    continue;
                min2=min1+edge[i][j]-max_val[i][j];
                if(min1==min2)
                {
                    flag=false;
                    break;
                }
            }
            if(flag==false)
                break;
        }
        if(flag)      printf("%d\n",min1);
        else          printf("Not Unique!\n");
    }
    return 0;
}
/*==================================================*\
|次小生成树,LEN 是输入边数量。MAX 是最大权值
|最小生成树调用 Kruskal1() 返回最小生成树，次小生成树调用 Find()
|返回次小生成树。当调用 Kruskal1()时，如 num=e 或返回值 s1>=MAX
|说明不存在最小生成树。当 s2=Find()，s2>=MAX 时，说明没有次小生
|成树。
|根据题意的不同，改变相应的参数。
\*==================================================*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define LEN 100100
#define MAX 100000000
typedef struct{
    int vx , vy , w;
}TGraph;
TGraph gh[LEN];
int v , e , num;
int f[1005] , path[LEN];
int cmp(const void *a , const void *b){
    TGraph *A = (TGraph *)a;
    TGraph *B = (TGraph *)b;
    return A->w - B->w;
}
int getfather(int p){
    if(f[p] == p)    return p;
    else {
        f[p] = getfather(f[p]);
        return f[p];
      }
}
bool uion(int x   , int y){
    int a = getfather(x);
    int b = getfather(y);
    if(a == b) return false;
    f[y] = f[b] = a;
    return true;
}
int Kruskal1( ){
    qsort(gh+1 , e , sizeof(TGraph) , cmp);
    int i = 1 , k = v-1 , mincost = 0;
    bool ok , right;
    num = 0;    right = true;
    while( k ){
        ok = false;
        do{
            if(i > e){right = false; break;}
            if(uion(gh[i].vx , gh[i].vy)) ok = true;
            i++;
          }while(!ok);
        mincost += gh[i-1].w;
        path[num++] = i - 1;
        k--;
      }
      if(!right)mincost = MAX;
      return mincost;
}
int Kruskal2( ){
    int i = 1 , k = v-1 , mincost = 0;
    bool ok , right;
    right = true;
    while( k ){
        ok = false;
        do{
            if(i > e){right = false; break;}
            if(gh[i].w == MAX){i++; continue;}
            if(uion(gh[i].vx , gh[i].vy)) ok = true;
            i++;
          }while(!ok);
        mincost += gh[i-1].w;
        k--;
      }
      if(!right) mincost = MAX;
      return mincost;
}
int Find(){
    int min = MAX;
    for( int i = 0; i < num; i++){
```

```
            int tt = gh[path[i]].w;
            gh[path[i]].w = MAX;
            for( int j = 1; j <= v; j++) f[j] = j;
            int t = Kruskal2( );
            gh[path[i]].w = tt;
            if( t < min ) min = t;
        }
        return min;
}
int main(){
        int i,res;
    while( scanf("%d%d" , &v , &e)!=EOF ){
        for( i = 1; i <= v; i++ ) f[i] = i;
        for(    i = 1; i <= e; i++ )
            scanf("%d%d%d" , &gh[i].vx , &gh[i].vy , &gh[i].w);
        int i = Kruskal1() ;
            if(num == e || i == MAX) {
            printf("-1\n");
            continue;
            }
        res = Find();
        if(res >= MAX) printf("-1\n");
        else printf("%d\n" , res);
    }
    return 0;
}
```

/*==*\

| 最小生成森林问题(k 颗树)O(mlogm).

==/

数据结构：并查集 算法：改进 Kruskal

根据 Kruskal 算法思想，图中的生成树在连完第 n-1 条边前，都是一个最小生成森林，每次贪心的选择两个不属于同一连通分量的树（如果连接一个连通分量，因为不会减少块数，那么就是不合算的）且用最“便宜”的边连起来，连

接 n-1 次后就形成了一棵 MST，n-2 次就形成了一个两棵树的最小生成森林，n-3，……，n-k 此后就形成了 k 颗树的最小生成森林，就是题目要求求解的。

/*==*\

| 有向图最小树形图

| INIT: eg 置为边表; res 置为 0; cp[i]置为 i;

| CALL: dirtree(root, nv, ne); res 是结果;

==/

```
#define typec int                                    // type of res
const typec inf = 0x3f3f3f3f;                        // max of res
typec res, dis[V];
int to[V], cp[V], tag[V];
struct Edge { int u, v; typec c; } eg[E];
    int iroot(int i){
        if (cp[i] == i) return i;
        return cp[i] = iroot(cp[i]);
}
int dirtree(int root, int nv, int ne)                // root: 树根
{                                                    // vertex: 0 ~ n-1
    int i, j, k, circle = 0;
    memset(tag, -1, sizeof(tag));
    memset(to, -1, sizeof(to));
    for (i = 0; i < nv; ++i) dis[i] = inf;
    for (j = 0; j < ne; ++j) {
        i = iroot(eg[j].u); k = iroot(eg[j].v);
        if (k != i && dis[k] > eg[j].c) {
        dis[k] = eg[j].c;
        to[k] = i;
    }
}
to[root] = -1; dis[root] = 0; tag[root] = root;
   for (i = 0; i < nv; ++i) if (cp[i] == i && -1 == tag[i]){
            j = i;
        for ( ; j != -1 && tag[j] == -1; j = to[j])
        tag[j] = i;
        if (j == -1) return 0;
      if (tag[j] == i) {
        circle = 1; tag[j] = -2;
        for (k = to[j]; k != j; k = to[k]) tag[k] = -2;
       }
  }
      if (circle) {
            for (j = 0; j < ne; ++j) {
            i = iroot(eg[j].u); k = iroot(eg[j].v);
            if (k != i && tag[k] == -2) eg[j].c -= dis[k];
        }
      for (i = 0; i < nv; ++i) if (tag[i] == -2) {
          res += dis[i]; tag[i] = 0;
          for (j = to[i]; j != i; j = to[j]) {
             res += dis[j]; cp[j] = i; tag[j] = 0;
          }
      }
      if (0 == dirtree(root, nv, ne)) return 0;
  } else {
      for (i = 0; i < nv; ++i) if (cp[i] == i) res += dis[i];
  }
return 1; //  若返回 0 代表原图不连通
}
```

/*==*\

| Minimal Steiner Tree

| G(V, E), A 是 V 的一个子集，求至少包含 A 中所有点的最小子树.

| 时间复杂度: O(N^3 + N * 2^A * (2^A + N))

| INIT: d[][]距离矩阵; id[]置为集合 A 中点的标号;

| CALL: steiner(int n, int a);

| main()函数解决的题目: Ticket to Ride, NWERC 2006/2007

| 给 4 个点对(a1, b1) ... (a4, b4),

| 求 min(sigma(dist[ai][bi])),其中重复的路段只能算一次.

| 这题要找出一个 steiner 森林，最后要对森林中树的个数进行枚举

==/

```
#define typec int                            // type of cost
const typec inf = 0x3f3f3f3f;                // max of cost
int vis[V], id[A];                           //id[]: A 中点的标号
typec d[V][V], dp[1<<A][V];          //dp[i][v]:  点 v 到点集 i 的最短距离
void steiner(int n, int a){
     int i, j, k, mx, mk, top = (1 << a);
 for (k = 0; k < n; k++)
     for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
             if (d[i][j] > d[i][k] + d[k][j])
                 d[i][j] = d[i][k] + d[k][j];
 for (i = 0; i < a; i++) {                    // vertex: 0 ~ n-1
     for (j = 0; j < n; j++)
         dp[1 << i][j] = d[j][ id[i] ];
 }
 for (i = 1; i < top; i++) {
     if ( 0 == (i & (i - 1)) ) continue;
     memset(vis, 0, sizeof(vis));
     for (k = 0; k < n; k++) {                       // init
         for (dp[i][k] = inf, j = 1; j < i; j++)
             if ((i | j) == i &&
             dp[i][k] > dp[j][k] + dp[i - j][k])
             dp[i][k] = dp[j][k] + dp[i - j][k];
     }
    for (j = 0; mx = inf, j < n; j++) {                  // update
        for (k = 0; k < n; k++)
            if (dp[i][k] <= mx && 0 == vis[k])
                mx = dp[i][mk = k];
        for (k = 0, vis[mk] = 1; k < n; k++)
            if (dp[i][mk] > dp[i][k] + d[k][mk])
                dp[i][mk] = dp[i][k] + d[k][mk];
      }
    }
}
int main(void){
int n, a = 8;
                // TODO: read data;
steiner(n, a);
               // enum to find the result
for (i = 0, b = inf; z = 0, i < 256; b>z ? b=z : b, i++)
    for (j = 0; y = 0, j < 4; z += !!y * dp[y][x], j++)
        for (k = 0; k < 8; k += 2)
            if ((i >> k & 3) == j)
                y += 3 << k, x = id[k];
            // TODO: cout << b << endl;
    return 0;
}
```

/*==*\

|Tips K 度最小生成树

```
|求 K 度限制生成树，要注意重边的情况，这个程序没考虑
|
|步骤是先求最小的 m 限制生成树，对去除顶点的余图的 m 割联通分量
|分别求最小生成树，然后在每个联通分量里找一条与顶点相连的最小边，
|这样就可以构成 m 度限制的最小生成树，当我们要从 m 度求出 m+1 度
|的最小生成树，方法是这样的，枚举没在 mst 中的与顶点相连的边加入
|到 mst 中，然后我们删除环中除了与顶点直接相连的边以外的边权最大
|的边，这样我们的 m+1 度最小生成树就好了。
|    这里需要一个优化，就是当我们要知道环中最大的边的时候，我们
|不用遍历这个环，我们可以预处理该点到根的最大边权，当我们算好
|m+1 度生成树的时候，我们可以用 o（n）的时候去维护这个 maxlen
|（i），表示 i 点到根的路径上除了与根直接相连的边外，边权最大的边
|的边权。
\*==================================================*/
#include <iostream>
#include <algorithm>
#include <string>
#include <map>
#define M 23
#define INF 1000000000
using namespace std;
int N,K;
map <string , int> PP;
int linked[ M ][ M ] , v[ M ] , MaxL[ M ] , ans , father[ M ] ,
used[ M ][ M ];
int SearchT(int node , int MaxLen){
     int i;
    MaxL[ node ] = MaxLen;
    for (i = 0;i < N;++ i){
        if (used[ node ][ i ] == 1 && father[ node ] != i){
            MaxLen = max(MaxLen ,linked[ node ][ i ]);
            SearchT (i , MaxLen);
        }
    }
    return 0;
}//搜索树，每个点到根节点路径上的最大边；
int MST(){
    int minn[ M ] , ret = 0 , j , k , i;
    for (i = 0;i < N;++ i){
        v[ i ] = 0 , father[ i ] = -1;
    }
    for (i = 0;i < N;++ i)
          for (j = 0;j < N;++ j)
               used[ i ][ j ] = 0;
    while ( 1 ){
        int node = -1 ,MM = INF;
        for (i = 1;i < N;++ i){
           if (!v[ i ] && linked[ 0 ][ i ] < MM){
               node = i,
               MM = linked[ 0 ][ i ];
           }
           minn[ i ] = INF;
        }//找出与根节点最近的点开始做最小生成树；
        if (node == -1) break;
        MaxL[ node ] = -INF;
        for (minn[ node ] = 0 , j = 1;j < N;++ j){
            for (k = -1 , i = 1;i < N;++ i){
                if (!v[ i ] && (k == -1 || minn[ i ] < minn[ k ]))
                           k = i;
            }
            if (k == -1) break;
            if (minn[ k ] == INF) break;
            for (v[ k ] = 1 , ret += minn[ k ] ,i = 1;i < N;++ i){
                if (!v[ i ] && linked[ k ][ i ] < minn[ i ]){
                    minn[ i ] = linked[ k ][ i ];
                    father[ i ] = k;
                }
            }
        }
    }
    for (i = 1;i < N;++ i){
        if (father[ i ] == -1){
            father[ i ] = 0;
            ret += linked[ i ][ 0 ];
            linked[ i ][ 0 ] = linked[ 0 ][ i ] = 0;
            used[ i ][ 0 ] = used[ 0 ][ i ] = 1;
            K --;
            continue;
        }
        used[ i ][father[ i ]] = 1;
        used[father[ i ]][ i ] = 1;
    }
    SearchT (0 , 0);
    return ret;
}
int solve(){
    int out = ans;
     int i;
    while (K --){                          /*K 为限制数*/
        int MIN = INF , p , q;
        for (i = 1;i < N;++ i){
            if (linked[ i ][ 0 ] != INF && used[ i ][ 0 ] == 0 &&
linked[ i ][ 0 ] - MaxL[ i ] < MIN){
                p = i;
                MIN = linked[ i ][ 0 ] - MaxL[ i ];
            }
        }
        if (MIN == INF) break;       //若根节点的出边小于 K；
        int len = MaxL[ p ];
        linked[ p ][ 0 ] = linked[ 0 ][ p ] = 0;
        used[ p ][ 0 ] = used[ 0 ][ p ] = 1;
        ans += MIN;
        out = min(out , ans);
        q = father[ p ];
        father[ p ] = 0;          //赋给所找出来的点父亲节点为根节点；
        while (linked[ p ][ q ] != len){
            int cnt = p;
            p = q;
            q = father[ p ];
            father[ p ] = cnt;
         }             //不断更新新的父子关系；
        used[ p ][ q ] = used[ q ][ p ] = 0;
        SearchT(0 , 0);
    }
    cout << "Total miles driven: " << out << endl;
    return 0;
}
int main(){
    int t , dist;
     int i , j;
    string a , b;
    while(cin >> t){
        N = 1;
        PP.clear();
        for (i = 0;i <= 20;++ i)
              for (j = 0;j <= 20;++ j)
                   linked[ i ][ j ] = INF;
        while (t --){
            cin >> a >> b >> dist;
               //注意有时候要考虑重边
            if (PP[ a ] == 0 && a != "Park") PP[ a ] = N ++;
            if (PP[ b ] == 0 && b != "Park") PP[ b ] = N ++;
            linked[PP[ a ]][PP[ b ]] = dist;
            linked[PP[ b ]][PP[ a ]] = dist;
        }
        cin >> K;
        ans = MST ();
        solve ();
    }
    return 0;
}
```

1.4 最短路径

```
/*==================================================*\
| 最短路径模版(Dijkstra) 邻接表+优先队列
| 有路径输出,但是有局限性，不能判断负环路
| Dijkstra 对于标记过的点就不再进行更新了，所以即使有负权导致最短
距离的改变也不会重新计算已经计算过的结果，所以判断负权环路出错
| O(nlogn)
|单源最短路径
\*==================================================*/
```

```
#include <iostream>
#include <queue>
#define MAXV 1000
#define MAXE 1000000
#define INTF 1000000000
using namespace std;

struct Edge {
        int next;
        int v;
        int val;
}e[MAXE];

struct Node {
        int v;
        int val;
        friend bool operator < (Node a, Node b) {
                return a.val > b.val;
        }
};

int p[MAXV], vist[MAXV], dist[MAXV], eid;
//record a shortest path
//pos 存弧 pre 存点
int pos[MAXV], pre[MAXV];

void add(int u, int v, int val) {
        e[eid].next = p[ u ];
        e[eid].v = v;
        e[eid].val = val;
        p[ u ] = eid ++;
}

void dijkstra(int n, int m) {
     memset(p, -1, sizeof(p));
     memset(vist, false, sizeof(vist));
     memset(pre, -1, sizeof(pre));
     eid = 0;
     int u, v, val;
     priority_queue<Node> Q;
     while(m --) {
             scanf("%d %d %d", &u, &v, &val);
             add(u, v, val);
             add(v, u, val);//无向
     }
     int source = 1, sink = n;
                                        //scanf("%d %d", &source, &sink);
     pre[source] = source;
     Node t;
     t.v = source;//选择起点
     t.val = 0;
     Q.push(t);
     int value;
     while(!Q.empty()) {
        t = Q.top();
        Q.pop();
        if(t.v == sink) break;
            if(vist[t.v]) continue;
                vist[t.v] = true;
                for(int j = p[t.v]; j != -1; j = e[ j ].next) {
                        if(vist[e[ j ].v]) continue;
                        Node tmp;
                        tmp.v = e[ j ].v;
                        tmp.val = e[ j ].val + t.val;
                        pos[ tmp.v ] = j;
                        pre[ tmp.v ] = t.v;
                        Q.push(tmp);
                }
     }
     if(t.v == sink) value = t.val;
     else value = -1;
     printf("%d\n", value);
//     for(int i = sink; i != source; i = pre[ i ]) {
                //e[pos[ i ]].val = INTF;
                //e[pos[ i ] ^ 1].val = INTF;
//             printf("%d-->", i);
//     }
//     printf("%d\n", source);
}
int main() {
        int V, E;
        while(scanf("%d %d", &V, &E) != EOF && V && E) {
                dijkstra(V, E);
        }
        return 0;
}
/*==================================================*\
| 最短路径模版(Bellman-Ford) (改进：SPFA 算法) 邻接表+优先队列
| SPFA 即 shotest path faster algorithm，由
|意思就可以看出该算法效率比较高。主要是用于负权图
|若要判负环路,则记录一个点的入队次数,若超
|过边数(?点数)，则有负权环。
|无路径输出
| O(kE)
|单源最短路径
\*==================================================*/
#include <iostream>
#include <queue>
#define MAXV 1000
#define MAXE 1000000
#define INTF 1000000000
using namespace std;

struct Edge {
        int next;
        int v;
        int val;
}e[MAXE];

int p[MAXV], vist[MAXV], dist[MAXV],   eid;
void add(int u, int v, int val) {
        e[eid].next = p[ u ];
        e[eid].v = v;
        e[eid].val = val;
        p[ u ] = eid ++;
}

void SPFA(int n, int m) {
     memset(p, -1, sizeof(p));
     memset(vist, false, sizeof(vist));
     fill(dist, dist+n+1, INTF);
     eid = 0;
     int u, v, val;
     queue<int> Q;
     while(m --) {
          scanf("%d %d %d", &u, &v, &val);
             add(u, v, val);
             add(v, u, val);//无向
     }
     int source = 1, sink = n, t;
                                        //scanf("%d %d", &source, &sink);
     dist[source] = 0;
     vist[source] = true;
     Q.push(source);
     while(!Q.empty()) {
               t = Q.front();
               Q.pop();
               vist[ t ] = false;
               for(int j = p[ t ]; j != -1; j = e[ j ].next) {
                       if(e[ j ].val + dist[ t ] < dist[e[ j ].v]) {
                               dist[e[ j ].v] = e[ j ].val + dist[ t ];
                               if(! vist[e[ j ].v]) {
               vist[e[ j ].v] = true;
               Q.push(e[ j ].v);
               }
          }
     }
}
     printf("%d\n", dist[sink]);
}
int main() {
```

```
    int V, E;
    while(scanf("%d %d", &V, &E) != EOF && V && E) {
        SPFA(V, E);
    }
    return 0;
}
```

```
/*==================================================*\
|实例  （要灵活变通）
/*==================================================*\
// POJ 3159 Candies
const int INF = 0x3F3F3F3F;
const int V = 30001;
const int E = 150001;
int pnt[E], cost[E], nxt[E];
int e, head[V]; int dist[V]; bool vis[V];
int main(void){
    int n, m;
    while( scanf("%d%d", &n, &m) != EOF ){
    int i, a, b, c;
    e = 0;
    memset(head, -1, sizeof(head));
      for( i=0; i < m; ++i )
      {             // b-a <= c, 有向边(a, b):c ，边的方向！！！
       scanf("%d%d%d", &a, &b, &c);
       addedge(a, b, c);
      }
     printf("%d\n", SPFA(1, n));
    }
  return 0;
}
int relax(int u, int v, int c){
     if( dist[v] > dist[u] + c ) {
         dist[v] = dist[u] + c; return 1;
      }
return 0;
}
inline void addedge(int u, int v, int c){
     pnt[e] = v; cost[e] = c; nxt[e] = head[u]; head[u] = e++;
}
int SPFA(int src, int n)
{                       // 此处用堆栈实现，有些时候比队列要快
     int i;
     for( i=1; i <= n; ++i ){ // 顶点 1...n
         vis[i] = 0; dist[i] = INF;
      }
     dist[src] = 0;
     int Q[E], top = 1;
     Q[0] = src; vis[src] = true;
     while( top ){
         int u, v;
         u = Q[--top]; vis[u] = false;
         for( i=head[u]; i != -1; i=nxt[i] ){
             v = pnt[i];
             if( 1 == relax(u, v, cost[i]) && !vis[v] ) {
                   Q[top++] = v; vis[v] = true;
              }
         }
      }
  return dist[n];
}
/*==================================================*\
/*==================================================*\
// 队列实现，而且有负权回路判断—POJ 3169 Layout
#define swap(t, a, b) (t=a, a=b, b=t)
const int INF = 0x3F3F3F3F;
const int V = 1001;
const int E = 20001;
int pnt[E], cost[E], nxt[E];
int e, head[V], dist[V];
bool vis[V];
int cnt[V];                              // 入队列次数
int main(void){
         int n, ml, md;
         while( scanf("%d%d%d", &n, &ml, &md) != EOF ){
         int i, a, b, c, t;
         e = 0;
         memset(head, -1, sizeof(head));
         for( i=0; i < ml; ++i )          // 边方向！！！
          {          // 大-小<=c, 有向边(小, 大):c
            scanf("%d%d%d", &a, &b, &c);
            if( a > b) swap(t, a, b);
            addedge(a, b, c);
          }
         for( i=0; i < md; ++i )
         {        // 大-小>=c ==> 小-大<=-c, 有向边(大, 小):-c
            scanf("%d%d%d", &a, &b, &c);
            if( a < b ) swap(t, a, b);
            addedge(a, b, -c);
         }
              //for( i=1; i <= n; ++i ) printf("%d\n", dist[i]);
        printf("%d\n", SPFA(1, n));
       }
    return 0;
}
int relax(int u, int v, int c){
         if( dist[v] > dist[u] + c ) {
             dist[v] = dist[u] + c; return 1;
          }
   return 0;
}
inline void addedge(int u, int v, int c){
    pnt[e] = v; cost[e] = c; nxt[e] = head[u]; head[u] = e++;
}
int SPFA(int src, int n){                 // 此处用队列实现
         int i;
         memset(cnt, 0, sizeof(cnt));      // 入队次数
         memset(vis, false, sizeof(vis));
         for( i=1; i <= n; ++i ) dist[i] = INF;
         dist[src] = 0;
         queue<int> Q;
         Q.push(src); vis[src] = true; ++cnt[src];
         while( !Q.empty() ){
              int u, v;
              u = Q.front(); Q.pop(); vis[u] = false;
              for( i=head[u]; i != -1; i=nxt[i] ){
                  v = pnt[i];
                  if( 1 == relax(u, v, cost[i]) && !vis[v] ) {
                      Q.push(v); vis[v] = true;
                      if( (++cnt[v]) > n ) return -1;
                   // cnt[i]为入队列次数，用来判断是否存在负权回路
                  }
              }
        }
      if( dist[n] == INF ) return -2; // src 与 n 不可达，有些题目可省！！！
    return dist[n]; // 返回 src 到 n 的最短距离，根据题意不同而改变
}
/*==================================================*\
|差分约束系统
|
|标准模型  a - b <= val; a, b 都没有系数
|将 a-b<=val 转换为 a<=b+val,看成 a 指向 b 的有向边权值为 val,这样
|就相当于求各个顶点到源点的最短距离,并且只有图中存在负权环时，方
|程组才无解，否则一定有解。
|
|判断有无负回路,当所有点入队次数大于 T*(n+m)(n 行 m 列)时,T 一般
|取 2 或者某个点的入队次数>sqrt(n);
|
\*==================================================*/
#include <iostream>
#include <queue>
#include <cmath>
using namespace std;
const long MAXN=400000;
const long lmax=0x7FFFFFFF;

typedef struct
{
    long v;
    long next;
    double cost;
```

```
}Edge;
Edge e[MAXN];
long p[MAXN], eid;
double Dis[MAXN];
bool vist[MAXN];
queue<long> q;
long m,n,L,U;//点,边

void insert(int from, int to, double cost) {

        e[eid].next=p[from];
        e[eid].v=to;
        e[eid].cost=cost;
        p[from]=eid++;
}

void init()
{
        eid = 0;
        memset(p,-1,sizeof(p));
        memset(vist,0,sizeof(vist));
        fill(Dis,Dis+MAXN,lmax);
}

bool SPF() {
        int s = 0;//源点
        int cnt = 0;
        while(!q.empty()) q.pop();
        Dis[s]=0;
        vist[s]=true;
        q.push(s);
        while (!q.empty())
        {
            long t=q.front();
            q.pop();
            vist[t]=false;
            long j;
            for (j=p[t];j!=-1;j=e[j].next)
            {
                double w=e[j].cost;
                if (w+Dis[t]<Dis[e[j].v])
                {
                    Dis[e[j].v]=w+Dis[t];
                    if (!vist[e[j].v])
                    {
                        vist[e[j].v]=true;
                        q.push(e[j].v);
                        cnt ++;
                        if(cnt > 2*(n+m)) return false;
                    }
                }
            }
        }
        return true;
}
/*==================================================*\
| 最短路径模版(Floyd-Warshall)
| 最朴实的版本
| 多源最短路径
\*==================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int a[101][101];
int d[101][101];
int n;

void Floyd()
{
    int u,v,w,min,max,pi;
    for(v=1; v<=n; v++)
        for(w=1; w<=n; w++)
            d[v][w]=a[v][w];
    for(u=1; u<=n; u++)
        for(v=1; v<=n; v++)
            for(w=1; w<=n; w++)
                if(d[v][u]+d[u][w]<d[v][w])
                    d[v][w]=d[v][u]+d[u][w];
}

int main()
{
    int i,j,k,p,q;
    while(scanf("%d",&n)==1 && n!=0)
    {
        for(i=1; i<=n; i++)
            for(j=1; j<=n; j++)
                a[i][j]=100000;
        for(i=1; i<=n; i++)
        {
            scanf("%d",&k);
            for(j=1; j<=k; j++)
            {
                scanf("%d %d",&p,&q);
                a[i][p]=q;
            }
        }
        Floyd();
    }
    return 0;
}
/*==================================================*\
| 最短路径模版(Floyd-Warshall)
| 稍稍优化的版本
| in[a]存的是以 a 为终点的起点
| out[a]存的是以 a 为起点的终点
| 多源最短路径
\*==================================================*/
  for(int k = 1; k <= n; k++)
      for(int i = 0; i < in[k].size(); i++)
          for(int j = 0; j < out[k].size(); j++)
              if(!g[in[k][i]][out[k][j]]&&g[in[k][i]][k]&& g[k][out[k][j]])
              {
                  g[in[k][i]][out[k][j]] = 1;
                  in[out[k][j]].push_back(in[k][i]);
                  out[in[k][i]].push_back(out[k][j]);
              }
/*==================================================*\
| 最短路径模版(Johnson)
| Johnson 算法适用于求 All Pairs Shortest Path.
|Johnson 算法应用了重标号技术，先进行一次 Bellman-Ford 算法，然
|后对原图进行重标号，w'(i,j)=h[i]-h[j]+w(i,j)。 然后对每个点进行一次
|Dijkstra，每次 Dijkstra 的复杂度为 O(nlogn+m)，于是算法复杂度为
|O(n^2logn+m)。
|
|   Johson 算法是目前最高效的在无负环可带负权重的网络中求所有点
|对最短路径的算,Johson 算法是 Bellman-Ford 算法,Reweighting(重
|赋权重)和 Dijkstra 算法的大综合. 对每个顶点运用 Dijkstra 算法的时
|间开销决定了 Johnson 算法的时间开销. 每次 Dijkstra 算法(d 堆 PFS
|实现)的时间开销是 O( E * lgd(V) ). 其中 E 为边数, V 为顶点数, d 为采
|用 d 路堆实现优先队列 ADT. 所以，此种情况下 Johnson 算法的时间
|复杂度是 O( V * E * lgd(V) ).
|
| 多源最短路径
\*==================================================*/
网上代码很乱，估计也用不到这么高深的，代码略
/*==================================================*\
| 最短路径 BFS+DP 模版  （启示作用）
\*==================================================*/
#include <iostream>
#include <cstring>
#include <cstdio>
#include <queue>
using namespace std;
#define MAXV 101
#define MAXE 10000
#define MAXK 11
typedef struct {

      int v;
      int k;
      int val;
```

```
	int next;
}Edge;

typedef struct {

	int v;
	int k;
}Node;

Edge e[MAXE];
int eid, K, p[MAXV];
int dp[MAXV][MAXK];
void add(int u, int v, int val, int k) {

	e[eid].next = p[ u ];
	e[eid].v = v;
	e[eid].k = k;
	e[eid].val = val;
	p[ u ] = eid ++;
}

void init() {
	eid = 0;
	memset(p, -1, sizeof(p));
}

void print() {
	for(int i = 1; i <= 1; ++ i) {
		for(int j = 0; j <= K; ++ j) {
			printf("%10d", dp[ i ][ j ]);
		}
		cout << endl;
	}
}

void BFS() {
	memset(dp, -1, sizeof(dp));
	int s, t;
	cin >> s >> t;
	//dp[ i ][ j ] 表示从源点 S 出发到达标号为 i 的点时 经过 j 条指定
边的最短距离
	queue<Node> Q;
	Node q, qe;
	q.k = 0; q.v = s;
	dp[ s ][ 0 ] = 0;
	Q.push(q);
	while(!Q.empty()) {
		q = Q.front();
		//print();
		int u = q.v;
		int j = q.k;
		Q.pop();
		for(int i = p[q.v]; i != -1; i = e[ i ].next) {
			int v = e[ i ].v;
			int k = e[ i ].k;
			int val = e[ i ].val;
			qe.v = v;
			if(k == 0) {
			if(j <= K && (dp[ v ][ j ] == -1 || dp[ v ][ j ] >
dp[ u ][ j ] + val)) {
					dp[ v ][ j ] = dp[ u ][ j ] + val;
					qe.k = j;
					if(qe.k <= K) Q.push(qe);
				}
			}
			else {
				if(j + 1 <= K && (dp[ v ][j + 1] == -1 || dp[ v ][j
+ 1] > dp[ u ][ j ] + val)) {
					dp[ v ][j + 1] = dp[ u ][ j ] + val;
					qe.k = j + 1;
					if(qe.k <= K) Q.push(qe);
				}
			}
		}
	}
	printf("%d\n", dp[ t ][ K ]);
}
int main() {
	int n, m, test;
	cin >> test;
	while(test --) {
		cin >> n >> m >> K;
		init();
		while(m --) {
			int u, v, val, k;
			cin >> u >> v >> val >> k;
			add(u, v, val, k);
			add(v, u, val, k);
		}
		BFS();
	}
	return 0;
}
/*==================================================*\
| 第 K 短路（Dijkstra   +   A*）（经典）
|n 个点，m 个路径，从点 u 到点 v 花费 w 小时（权值），起点 s 终点 t，
|在花费小时数最短的情况下的第 k 短路径
| POJ   2449
\*==================================================*/
#include<iostream>
#include<cstdio>
#include<string.h>
#include<queue>
#include<vector>
using namespace std;
const int maxn=5001;
const int INF=999999999;
int N,M,S,T,K,dis[maxn];
struct node
{
	int v,l;
	node(int v,int l):v(v),l(l){}
	inline bool operator<(const node &b) const
	{
		return l+dis[v]>b.l+dis[b.v];
	   //这里是对估价函数的使用，使得路径最短的先出队
	}
};
vector<node> map[maxn];
vector<node> g[maxn];
int getint()
//这个 getchar 的输入对大数据量输入非常有用，甚至可以挽救效率不高
的算法
{
       int ret=0;
	char tmp;
       while(!isdigit(tmp=getchar()));
       do{
		ret=(ret<<3)+(ret<<1)+tmp-'0';
       }while(isdigit(tmp=getchar()));
       return ret;
}
void dij()                          //求所有点到终点的最短路径
{
	int i,u,mark[maxn];
	for(i=1;i<=N;i++) dis[i]=INF;
	memset(mark,0,sizeof mark);
	priority_queue<node> heap;
	dis[T]=0;
	heap.push(node(T,0));
	while(!heap.empty())
	{
		u=heap.top().v;
		heap.pop();
		if(mark[u])continue;
		mark[u]=1;
		for(i=0;i<map[u].size();i++)
			if(!mark[map[u][i].v]                                  &&
dis[u]+map[u][i].l<dis[map[u][i].v])
			{
				dis[map[u][i].v]=dis[u]+map[u][i].l;
			heap.push(node(map[u][i].v,dis[map[u][i].v]));
```

```
                }
        }
        while (!heap.empty ())
                heap.pop ();
}
int A_star()              //A*搜索
{
        int u,i,len,c[maxn];
        if(dis[S]==INF)return -1;
        priority_queue<node> heap;
        memset(c,0,sizeof c);
        heap.push(node(S,0));
        while(!heap.empty())
        {
                u=heap.top().v;
                len=heap.top().l;
                heap.pop();
                c[u]++;
                if(c[T]==K)return len;
                if(c[u]>K)continue;
                for(i=0;i<g[u].size();i++)
                        heap.push(node(g[u][i].v,len+g[u][i].l));
        }
        while (!heap.empty ())
           heap.pop ();
        return -1;
}
int main()
{
        //freopen("D:\\text.in","r",stdin);
        int i,u,v,w;
        while (scanf("%d %d",&N,&M) == 2)
        {
                for (i = 0 ; i < maxn ; ++ i)
                {
                   g[ i ].clear ();
                   map[ i ].clear ();
                }

                for(i=0;i<M;i++)
                {
                        u=getint();
                        v=getint();
                        w=getint();
                        g[u].push_back(node(v,w));
                        map[v].push_back(node(u,w));
                }
                S=getint();
                T=getint();
                K=getint();//第 K 最短路径
                if(S==T)K++;
                dij();
                printf("%d\n",A_star());
        }
        return 0;
}
/*==================================================*\
| 第 K 短路（Dijkstra）
| dij 变形，可以证明每个点经过的次数为小于等于 K，所有把 dij 的数组 dist
| 由一维变成 2 维，记录经过该点 1 次，2 次。。。k 次的最小值。
| 输出 dist[n-1][k]即可
\*==================================================*/
//WHU1603
int g[1010][1010];
int n,m,x;
const int INF=1000000000;
int v[1010];
int dist[1010][20];
int main(){
        while (scanf("%d%d%d",&n,&m,&x)!=EOF){
                for (int i=1;i<=n;i++)
                   for (int j=1;j<=n;j++)
                         g[i][j]=INF;
                for (int i=0;i<m;i++){
                         int p,q,r;
                         scanf("%d%d%d",&p,&q,&r);
                         if (r<g[p][q]) g[p][q]=r;
                }
                for (int i=1;i<=n;i++){
                     v[i]=0;
                     for (int j=0;j<=x;j++)
                     dist[i][j]=INF;
                }
                  dist[1][0]=0;
                  dist[0][0]=INF;
                while (1){
                   int k=0;
                   for (int i=1;i<=n;i++)
                  if (v[i]<x && dist[i][v[i]]<dist[k][0])
                        k=i;
                  if (k==0) break;
                  if (k==n && v[n]==x-1) break;
                 for (int i=1;i<=n;i++){
                      if (v[i]<x &&dist[k][v[k]]+g[k][i]<dist[i][x]){
                      dist[i][x]=dist[k][v[k]]+g[k][i];
                      for (int j=x;j>0;j--)
                      if (dist[i][j]<dist[i][j-1])
                     swap(dist[i][j],dist[i][j-1]);
                   }
}
         v[k]++;
    }
    if (dist[n][x-1]<INF) printf("%d\n",dist[n][x-1]);
    else printf("-1\n");
  }
return 0;
}
/*==================================================*\
| 第 K 短路（A*）
| A* 估价函数 fi 为到当前点走过的路经长度， hi 为该点到终点的长度
| gi=hi+fi；
\*==================================================*/
//WHU1603
int n,m,x,ct;
int g[1010][1010],gr[1010][1010];
int dist[1010],v[1010];
const int INF=1000000000;
struct node{
int id,fi,gi;
   friend bool operator <(node a,node b){
      if (a.gi==b.gi) return a.fi>b.fi;
      return a.gi>b.gi;
   }
}s[2000010];
int init(){
      for (int i=0;i<=n;i++){
      dist[i]=INF;
      v[i]=1;
      }
dist[n-1]=0;
 for (int i=0;i<n;i++){
        int k=n;
        for (int j=0;j<n;j++)
        if (v[j] && dist[j]<dist[k]) k=j;
        if (k==n) break;
        v[k]=0;
        for (int j=0;j<n;j++)
           if (v[j] && dist[k]+gr[k][j]<dist[j])
          dist[j]=dist[k]+gr[k][j];
   }
   return 1;
}
int solve(){
        if (dist[0]==INF) return -1;
        ct=0;
        s[ct].id=0;
        s[ct].fi=0;
        s[ct++].gi=dist[0];
        int cnt=0;
        while (ct){
             int id=s[0].id,fi=s[0].fi,gi=s[0].gi;
```

```
            if (id==n-1) cnt++;
            if (cnt==x) return fi;
            pop_heap(s,s+ct);
            ct--;
        for (int j=0;j<n;j++)
          if (g[id][j]<INF){
            s[ct].id=j;
            s[ct].fi=fi+g[id][j];
            s[ct++].gi=s[ct].fi+dist[j];
            push_heap(s,s+ct);
          }
      }
    return -1;
}
int main(){
    while (scanf("%d%d%d",&n,&m,&x)!=EOF){
        for (int i=0;i<n;i++)
            for (int j=0;j<n;j++)
                gr[i][j]=g[i][j]=INF;
        for (int i=0;i<m;i++){
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            x--,y--;
            g[x][y]<?=z;
            gr[y][x]<?=z;
        }
        init();
        printf("%d\n",solve());
    }
    return 0;
}
```

1.5 二分图匹配

```
/*==================================================*\
|最大匹配： 图中包含边数最多的匹配称为图的最大匹配。
|
|完美匹配： 如果所有点都在匹配边上，称这个最大匹配是完美匹配。
|
|最小覆盖： pku(1325)
|最小覆盖要求用最少的点（X集合或Y集合的都行）让每条边都至少和
|其中一个点关联。可以证明：最少的点（即覆盖数）＝最大匹配数
|
|最小路径覆盖： (pku1422)
|用尽量少的不相交简单路径覆盖有向无环图G的所有结点。解决此类问
|题可以建立一个二分图模型。把所有顶点 i 拆成两个：X结点集中的 i
|和 Y 结点集中的 i',如果有边 i->j，则在二分图中引入边 i->j'，设二分图
|最大匹配为 m,则结果就是 n-m。
|
|最大独立集问题： (pku1466)
|在N个点的图 G 中选出 m 个点，使这 m 个点两两之间没有边．求 m 最
|大值，如果图G满足二分图条件，则可以用二分图匹配来做．最大独立
|集点数 = N - 最大匹配数
\*==================================================*/
1、左右端点数相同且等于匹配数称完备匹配
2、有向图的最小路径覆盖数是点数-最大匹配数
/*==================================================*\
| 二分图匹配（匈牙利算法 DFS 实现）
| INIT: g[][]邻接矩阵;
| CALL: res = MaxMatch();
| 优点：实现简洁容易理解，适用于稠密图，DFS 找增广路快。
| 找一条增广路的复杂度为 O（E），最多找 V 条增广路，故时间复杂度
为 O（VE）
\*==================================================*/
#include  < iostream >
#include  < fstream >
using  namespace  std;
const  int  MAXN  =  1000 ;
int  uN, vN;   // u, v 数目
bool  g[MAXN][MAXN]; // g[i][j] 表示 xi 与 yj 相连
int  xM[MAXN], yM[MAXN];  //  输出量
bool  chk[MAXN];  // 辅助量 检查某轮 y[v]是否被 check
bool  SearchPath( int  u){
      int  v;
      for  (v = 0 ; v < vN; v ++ ){
          if  (g[u][v]  &&  ! chk[v]) {
              chk[v]  =  true ;
              if  (yM[v]  ==  - 1  ||  SearchPath(yM[v])) {
                  yM[v]  =  u;
                  xM[u]  =  v;
                  return  true ;
              }
          }
      }
      return  false ;
}
int  MaxMatch(){
      int  u;
      int  ret  =  0 ;
      memset(xM,  - 1 ,  sizeof (xM));
      memset(yM,  - 1 ,  sizeof (yM));
      for  (u = 0 ; u < uN; u ++ ){
          if  (xM[u]  ==  - 1 ){
              memset(chk,  false ,  sizeof (chk));
              if (SearchPath(u)) ret ++ ;
          }
      }
      return  ret;
}
int  main(){
      int  i, k;
      int  tU, tV;
      ifstream cin( " test.txt " );
      cin  >>  uN  >>  vN  >>  k;
      memset(g,  false ,  sizeof (g));
      for  (i = 0 ; i < k; i ++ ){
         cin  >>  tU  >>  tV;
         g[tU][tV]  =  true ;
      }
      int  M  =  MaxMatch();
      cout  <<  " Total Match: "  <<  M  <<  endl;
       for  (i = 0 ; i < MAXN; i ++ )
          if  (xM[i]  !=  - 1 )
              cout  <<  i  <<  '  '  <<  xM[i]  <<  endl;
      system( " pause " );
      return  0 ;
}
/* ********** test data: *********
    3 3 3
    1 1
    1 0
    2 2
********** ***********************/
/*==================================================*\
| 二分图匹配（匈牙利算法 BFS 实现）
| INIT: g[][]邻接矩阵;
| CALL: res = MaxMatch();Nx, Ny 初始化!!!
| 优点：适用于稀疏二分图，边较少，增广路较短。
| 匈牙利算法的理论复杂度是 O(VE)
\*==================================================*/
const int MAXN = 1000;
int g[MAXN][MAXN], Mx[MAXN], My[MAXN], Nx, Ny;
int chk[MAXN], Q[MAXN], prev[MAXN];
int MaxMatch(void) {
    int res = 0;
    int qs, qe;
    memset(Mx, -1, sizeof(Mx));
    memset(My, -1, sizeof(My));
    memset(chk, -1, sizeof(chk));
    for (int i = 0; i < Nx; i++){
        if (Mx[i] == -1){
            qs = qe = 0;
            Q[qe++] = i;
            prev[i] = -1;
            bool flag = 0;
            while (qs < qe && !flag){
                int u = Q[qs];
                for (int v = 0; v < Ny && !flag; v++)
                    if (g[u][v] && chk[v] != i) {
                        chk[v] = i; Q[qe++] = My[v];
                        if (My[v] >= 0) prev[My[v]] = u;
                    else {
```

```
            flag = 1;
            int d = u, e = v;
            while (d != -1) {
                int t = Mx[d];
                Mx[d] = e; My[e] = d;
                d = prev[d]; e = t;
            }
          }
        }
      qs++;
    }
    if (Mx[i] != -1) res++;
  }
 }
 return res;
}
/*==================================================*\
| 二分图匹配（Hopcroft－Karp 的算法）
| INIT: g[][]邻接矩阵;
| CALL: res = MaxMatch(); Nx, Ny 要初始化!!!
| 时间复杂度为 O（sqrt(V)* E）
\*==================================================*/
const int MAXN = 3001;
const int INF = 1 << 28;
int g[MAXN][MAXN], Mx[MAXN], My[MAXN], Nx, Ny;
int dx[MAXN], dy[MAXN], dis;
bool vst[MAXN];
bool searchP(void){
    queue<int> Q;
    dis = INF;
    memset(dx, -1, sizeof(dx));
    memset(dy, -1, sizeof(dy));
    for (int i = 0; i < Nx; i++)
      if (Mx[i] == -1){
        Q.push(i); dx[i] = 0;
      }
  while (!Q.empty()) {
     int u = Q.front(); Q.pop();
     if (dx[u] > dis) break;
       for (int v = 0; v < Ny; v++)
           if (g[u][v] && dy[v] == -1) {
                 dy[v] = dx[u]+1;
             if (My[v] == -1) dis = dy[v];
             else{
                   dx[My[v]] = dy[v]+1;
                   Q.push(My[v]);
             }
         }
    }
   return dis != INF;
}
bool DFS(int u){
        for (int v = 0; v < Ny; v++)
           if (!vst[v] && g[u][v] && dy[v] == dx[u]+1) {
                vst[v] = 1;
             if (My[v] != -1 && dy[v] == dis) continue;
             if (My[v] == -1 || DFS(My[v])) {
                   My[v] = u; Mx[u] = v;
                 return 1;
             }
          }
  return 0;
}
int MaxMatch(void){
      int res = 0;
      memset(Mx, -1, sizeof(Mx));
      memset(My, -1, sizeof(My));
      while (searchP()) {
          memset(vst, 0, sizeof(vst));
         for (int i = 0; i < Nx; i++)
             if (Mx[i] == -1 && DFS(i)) res++;
      }
   return res;
}
/*==================================================*\
| 二分图最佳匹配（kuhn munkras 算法 O(m*m*n)）（带边权）
| 邻接距阵形式,复杂度 O(m*m*n) 返回最佳匹配值
|KM algotirhm,用来解决最大权匹配问题: 在一个二分图内,左顶点为 X,
|右顶点为 Y,现对于每组左右连接 Xi、Yj 有权 wij,求一种匹配使得所有
|wij 的和最大.(把最大权匹配转化成完备匹配)
\*==================================================*/
#include <cstdio>
#include <memory.h>
#include <algorithm>      // 使用其中的min 函数
using namespace std;
const int MAX = 1024;
int n;                            // X 的大小
int weight [MAX] [MAX];           // X 到Y 的映射（权重）
int lx [MAX], ly [MAX];           // 标号
bool sx [MAX], sy [MAX];          // 是否被搜索过
int match [MAX];                  // Y(i) 与X(match [i]) 匹配
void init (int size);             // 初始化权重
bool path (int u);          // 从X(u) 寻找增广道路，找到则返回true
int   KM_match (bool maxsum = true);
// 参数maxsum 为true ，返回最大权匹配，否则最小权匹配

void init (int size)
{
    // 根据实际情况，添加代码以初始化
    n = size;
    for (int i = 0; i < n; i ++)
        for (int j = 0; j < n; j ++)
            scanf ("%d", &weight [i] [j]);
}
bool path (int u){
    sx [u] = true;
    for (int v = 0; v < n; v ++)
        if (!sy [v] && lx[u] + ly [v] == weight [u] [v]){
            sy [v] = true;
            if (match [v] == -1 || path (match [v])){
                match [v] = u;
                return true;
            }
        }
   return false;
}
int KM_match (bool maxsum){
    int i, j;
    if (!maxsum){
        for (i = 0; i < n; i ++)
            for (j = 0; j < n; j ++)
                weight [i] [j] = -weight [i] [j];
    }
    // 初始化标号
    for (i = 0; i < n; i ++){
        lx [i] = -0x1FFFFFFF;
        ly [i] = 0;
        for (j = 0; j < n; j ++)
            if (lx [i] < weight [i] [j])
                lx [i] = weight [i] [j];
        }
    memset (match, -1, sizeof (match));
    for (int u = 0; u < n; u ++)
        while (1){
            memset (sx, 0, sizeof (sx));
            memset (sy, 0, sizeof (sy));
            if (path (u))break;
            // 修改标号
            int dx = 0x7FFFFFFF;
            for (i = 0; i < n; i ++)
                if (sx [i])
                   for (j = 0; j < n; j ++)
                      if(!sy [j])
                         dx = min (lx[i] + ly [j] - weight [i] [j], dx);
            for (i = 0; i < n; i ++){
                if (sx [i])lx [i] -= dx;
                if (sy [i])ly [i] += dx;
             }
    }
    int sum = 0;
    for (i = 0; i < n; i ++)
        sum += weight [match [i]] [i];
```

```
    if (!maxsum){
        sum = -sum;
        for (i = 0; i < n; i ++)
            for (j = 0; j < n; j ++)
                weight [i] [j] = -weight [i] [j];
//  如果需要保持weight [ ] [ ]  原来的值，这里需要将其还原
     }
    return sum;
}
int main(){
    int n;
    scanf ("%d", &n);
    init (n);
    int cost = KM_match (true);
    printf ("%d \n", cost);
    for (int i = 0; i < n; i ++){
        printf ("Y %d -> X %d \n", i, match [i]);
    }
    return 0;
}
/*
5
3 4 6 4 9
6 4 5 3 8
7 5 3 4 2
6 3 2 2 5
8 4 5 4 7
//执行 bestmatch (true) ， 结果为  29
5
7 6 4 6 1
4 6 5 7 2
3 5 7 6 8
4 7 8 8 5
2 6 5 6 3
//执行  bestmatch (false) ， 结果为  2*/
/*==================================================*\
| 一般图匹配（Edmonds' Blossom-Contraction 算法）
|若已经知道二分图匹配则一般图匹配很好理解，就是在可能有奇圈的图上做一个匹配
|    ZOJ   3316
\*==================================================*/
#include <algorithm>
#include <cstdio>
#include <deque>
#include <vector>
#include <iostream>
using namespace std;
const int N = 362;
bool adj[N][N], flag[N], in[N];
int n, block[N], mate[N], path[N];
deque<int> q;
void Modify(int u, int lca){
    while (block[u] != lca){
        int v = mate[u];
        flag[block[u]] = true;
        flag[block[v]] = true;
        u = path[v];
        if (block[u] != lca) path[u] = v;
    }
}
void Contract(int u, int v, int s){
    int lca;
    fill_n(flag, n, false);
    for (lca = u; ; ){
        lca = block[lca];
        flag[lca] = true;
        if (lca == s) break;
        lca = path[mate[lca]];
    }
    for (lca = v; ; lca = path[mate[lca]]){
        lca = block[lca];
        if (flag[lca]) break;
    }
    fill_n(flag, n, false);
    Modify(u, lca);
    Modify(v, lca);
    if (block[u] != lca) path[u] = v;
    if (block[v] != lca) path[v] = u;
    for (int i = 0; i < n; ++i)
        if (flag[block[i]]){
            block[i] = lca;
            if (!in[i]) in[i] = true, q.push_back(i);
        }
}
int Find(int s){
    fill_n(in, n, false);
    fill_n(path, n, -1);
    for (int i = 0; i < n; ++i)
        block[i] = i;
    for (q.clear(), in[s] = true, q.push_back(s); !q.empty(); ){
        int x = q.front();
        q.pop_front();
        for (int i = 0; i < n; ++i)
            if (adj[x][i] && block[x] != block[i] && mate[x] != i){
                if (i == s || mate[i] != -1 && path[mate[i]] != -1)
                    Contract(x, i, s);
                else if (path[i] == -1){     //找到交错路径
                    path[i] = x;
                    if (mate[i] == -1){
                        while (i != -1){
                            x = path[i];
                            int ii = mate[x];
                            mate[x] = i;
                            mate[i] = x;
                            i = ii;
                        }
                        return 1;
                    }
                    in[mate[i]] = true;
                    q.push_back(mate[i]);
                }
            }
    }
    return 0;
}
vector<pair<int,int> > ston;
int dist(pair<int,int> ps1, pair<int,int> ps2){
    return abs(ps1.first-ps2.first)+abs(ps1.second-ps2.second);
}
int main()
{
    int u, v, res;
    int L;
    while(scanf("%d", &n) != EOF){
        ston.clear();
        res = 0;
        for(int i = 0; i < n; ++i){
            scanf("%d %d", &u, &v);
            ston.push_back(make_pair(u, v));
        }
        scanf("%d", &L);
        memset(adj, 0, sizeof(adj));
        for(int i = 0; i < n; ++i)
            for(int j = i+1; j < n; ++j)
                if(dist(ston[i], ston[j]) <= L){
                    adj[i][j] = adj[j][i] = 1;
                }
        memset(mate, -1, sizeof(mate));
        for(int i = 0; i < n; ++i)
            if(mate[i] == -1 && Find(i))++res;
        res <<= 1;
        if(res == n)printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

1.6 网络流

1.6.1 最大网络流

```
/*==================================================*\
| 最大网络流（Ford-Fulkerson 算法）
|最大流问题实际上是求一可行流{fij}，使得 v(f 达到最大。若给了一个可
|行流 f，只要判断 N 中有无关于 f 的增广路径，如果有增广路径，改进 f，
|得到一个流量增大的新的可行流；如果没有增广路径，则得到最大流。
|  POJ   1273     有环的情况存在
\*==================================================*/
#include <iostream>
#include <queue>
#include <string.h>
using namespace std;
const int maxN=201;
static int edge[maxN][maxN];
bool visited[maxN];
int father[maxN];
int N, M;                       //边数,顶点数
int ans;                        //结果
void Ford_Fulkerson( ) {
    while(1)    {
        //一次大循环,找到一条可能的增广路径
        queue <int> q;
        memset(visited, 0, sizeof(visited));
        memset(father, -1, sizeof(father));
        int now;
        visited[0] = true;
        q.push(0);
        while(!q.empty())                    //广度优先
        {
            now = q.front();
            q.pop();
            if(now == M-1) break;
            for(int i = 0; i < M; i++)     {
                //每次父亲节点都要更新,权值减为 0 的边就不算了.
                if(edge[now][i] && !visited[i])
                {
                    father[i] = now;
                    visited[i] = true;
                    q.push(i);
                }
            }
        }
        //可能的增广路不存在了
        if(!visited[M-1]) break;
        int u, min = 0xFFFF;
        for(u = M-1; u; u = father[u]) {          //找出权值最小的边
            if(edge[father[u]][u] < min)
                min = edge[father[u]][u];
        }
        //减去最小权值
        for(u = M-1; u; u = father[u])     {
            //前向弧减去
            edge[father[u]][u] -= min;
            //后向弧加上
            //存在圆环,这句话关键
            edge[u][father[u]] += min;
        }
        //当前增广路径增加的流
        ans += min;
    }
}
int main()    {
    int s, e, w;
    while(cin >> N >> M)
    {
        ans = 0;
        memset(edge, 0, sizeof(edge));
        for(int i = 0; i<N; i++)
        {
            cin >> s >> e >> w;
            edge[s-1][e-1] += w;
        }
        Ford_Fulkerson();
        cout << ans << endl;
    }
    return 0;
}
```

```
/*==================================================*\
|最大网络流（Edmonds-Karp 算法）
|先 BFS 再求 maxflow。路径上的每个点都需要两个标记：一个标记从|
源点到当前节点实际还剩多少流量可用；另一个标记在这条路径上当前|
节点的前驱。
|O(VE^2)
| POJ   1273     有环的情况存在
\*==================================================*/
#include <iostream>
#include <queue>
#include<cstdio>
#include<string.h>
#define msize 205
using namespace std;

int r[msize][msize]; //残留网络，初始为原图
bool visited[msize];
int pre[msize];
int n,m; //n 条边， m 个顶点

bool bfs(int s,int t) //寻找一条从 s 到 t 的增广路，若找到，返回 true
{
    int p;
    queue<int> Q;
    memset(pre,-1,sizeof(pre));
    memset(visited,false,sizeof(visited));
    pre[s]=s;
    visited[s]=true;
    Q.push(s);
    while (!Q.empty())
    {
        p=Q.front(),Q.pop();
        for (int i=1;i<=m;i++)
        {
            if (r[p][i]>0&&!visited[i])
            {
                pre[i]=p;
                visited[i]=true;
                if (i==t) return true;
                Q.push(i);
            }
        }
    }
    return false;
}

int Edmonds_Karp(int s,int t)
{
    int flow=0,d;
    while (bfs(s,t))
    {
        d=INT_MAX;
        for (int i=t;i!=s;i=pre[i]) d=min(d,r[pre[i]][i]);
        for (int i=t;i!=s;i=pre[i]) r[pre[i]][i] -= d, r[i][pre[i]] += d;
        flow += d;
    }
    return flow;
}

int main()
{
    int s,e,c;

    while (scanf("%d%d",&n,&m)==2)
    {
        memset(r,0,sizeof(r));
        for (int i=0;i<n;i++)
        {
            scanf("%d%d%d",&s,&e,&c);
            r[s][e] += c;
        }
        printf("%d\n",Edmonds_Karp(1,m));
    }
    return 0;
}
/*==================================================*\
```

|最大网络流(Dinic 算法) O(V^2 * E)
|Dinic 算法的思想是为了减少增广次数，建立一个辅助网络 L，L 与原网
|络 G 具有相同的节点数，但边上的容量有所不同，在 L 上进行增广，将
|增广后的流值回写到原网络上，再建立当前网络的辅助网络，如此反复，
|达到最大流。分层的目的是降低寻找增广路的代价（层次图思想，层次
|图，就是把原图中的点按照到到源的距离分"层"，只保留不同层之间的|
边的图。）
==/

```
#include<iostream>
using namespace std;
const long maxn=300;
const long maxm=300000;
const long inf=0x7fffffff;
struct node
{
	long v,next;
	long val;
}s[maxm*2];
long level[maxn],p[maxn],que[maxn],out[maxn],ind;
void init()
{
	ind=0;
	memset(p,-1,sizeof(p));
}
inline void insert(long x,long y,long z)
{
	s[ind].v=y;
	s[ind].val=z;
	s[ind].next=p[x];
	p[x]=ind++;
	s[ind].v=x;
	s[ind].val=0;
	s[ind].next=p[y];
	p[y]=ind++;
}
inline void insert2(long x,long y,long z)	/////无向图
{
	s[ind].v=y;
	s[ind].val=z;
	s[ind].next=p[x];
	p[x]=ind++;
	s[ind].v=x;
	s[ind].val=z;
	s[ind].next=p[y];
	p[y]=ind++;
}
long max_flow(long n,long source,long sink)
{
	long ret=0;
	long h=0,r=0;
	while(1)
	{
		long i;
		for(i=0;i<n;++i)
			level[i]=0;
		h=0,r=0;
		level[source]=1;
		que[0]=source;
		while(h<=r)
		{
			long t=que[h++];
			for(i=p[t];i!=-1;i=s[i].next)
			{
				if(s[i].val&&level[s[i].v]==0)
				{
					level[s[i].v]=level[t]+1;
					que[++r]=s[i].v;
				}
			}
		}
		if(level[sink]==0)break;
		for(i=0;i<n;++i)out[i]=p[i];
		long q=-1;
		while(1)
		{
			if(q<0)
			{
				long cur=out[source];
				for(;cur!=-1;cur=s[cur].next)
				{
if(s[cur].val&&out[s[cur].v]!=-1&&level[s[cur].v]==2)
					{
						break;
					}
				}
				if(cur>=0)
				{
					que[++q]=cur;
					out[source]=s[cur].next;
				}
				else
				{
					break;
				}
			}
			long u=s[que[q]].v;
			if(u==sink)
			{
				long dd=inf;
				long index=-1;
				for(i=0;i<=q;i++)
				{
					if(dd>s[que[i]].val)
					{
						dd=s[que[i]].val;
						index=i;
					}
				}
				ret+=dd;
				//cout<<ret<<endl;
				for(i=0;i<=q;i++)
				{
					s[que[i]].val-=dd;
					s[que[i]^1].val+=dd;
				}
				for(i=0;i<=q;i++)
				{
					if(s[que[i]].val==0)
					{
						q=index-1;
						break;
					}
				}
			}
			else
			{
				long cur=out[u];
				for(;cur!=-1;cur=s[cur].next)
				{
if(s[cur].val&&out[s[cur].v]!=-1&&level[u]+1==level[s[cur].v])
					{
						break;
					}
				}
				if(cur!=-1)
				{
					que[++q]=cur;
					out[u]=s[cur].next;
				}
				else
				{
					out[u]=-1;
					q--;
				}
			}
		}
	}
	return ret;
}
```

```
long m,n;

int main()
{

	while(scanf("%ld %ld",&m,&n)!=EOF)
	{
		init();
		for(int i=0;i<n;i++)
		{
			long from,to,cost;
			scanf("%ld %ld %ld",&from,&to,&cost);
			insert(--from,--to,cost);
		}
		long Start,End;
		scanf("%ld %ld",&Start,&End);
		printf("%ld\n",max_flow(n,--Start,--End));
	}
	return 0;
}
/*==================================================*\
|最大网络流（最短路径增殖 EK-2（MPLA）算法）(ISAP)(V*E^2)
|比 Dinic 快一些，灵活运用 GAP 优化就很好了
|所谓 gap 优化就是计算出层次图后，层次出现断层，这是可以确定残
|余网络中不存在增广路径了，算法就可以提前结束。这个优化看似微小，
|实际作用确不小。做法就是保存某一个标号在残余网络中出现的次数，
|如果是 0 ，就断层了。
\*==================================================*/
#include <iostream>
using namespace std;
#define MAXV 210
#define MAXE 2000
#define INF 1 << 30
typedef struct {
	int v;
	int val;
	int next;
}Edge;
Edge e[MAXE];
int cnt, head[MAXV], ans;
void add(int u, int v, int val) {
	e[cnt].next = head[ u ];
	e[cnt].v = v;
	e[cnt].val = val;
	head[ u ] = cnt ++;
}
void sap(int s, int t, int n, int &res) {
	int i, j, k, aug = INF;
	int pre[MAXV], gap[MAXV];
	int cur[MAXV], dis[MAXV];
	int u, v, val, mindis, GAP;
	bool flag;
	for(i = 0;i <= n; ++ i) {
		cur[ i ] = head[ i ];
		gap[ i ] = dis[ i ] = 0;
	}
	gap[ 0 ] = n;
	res = 0;
	pre[ s ] = u = s;
	while(dis[ s ] < n) {
		flag = false;
		for(j = cur[ u ]; j != -1; j = e[ j ].next) {
			v = e[ j ].v;    cur[ u ] = j;
			val = e[ j ].val;
			if(val > 0 && dis[ u ] == dis[ v ] + 1) {
				if(aug > val) aug = val;
				flag = true;
				pre[ v ] = u; u = v;
				if(u == t) {
					res += aug;
					while(u != s) {
						u = pre[ u ];
						e[cur[ u ]  ].val -= aug;
						e[cur[ u ]^1].val += aug;
					}
					aug = INF;
				}
				break;
			}
		}
		if(flag) continue;
		mindis = n;
		for(j = head[ u ]; j != -1; j = e[ j ].next) {
			v = e[ j ].v;    val = e[ j ].val;
			if(val > 0 && mindis > dis[ v ]) {
				mindis = dis[ v ];
				cur[ u ] = j;
			}
		}
		GAP = --gap[dis[ u ]];
		if(GAP <= 0) break;
		dis[ u ] = mindis + 1;
		gap[dis[ u ]] ++;
		u = pre[ u ];
	}
}
int main() {
	int m, n, ans;			//边，点，最大流
	int s, t, len;			//起点,终点,点的个数
	int a, b, c;			//a-->b  流量 c
	while(cin >> m >> n) {
		s = 0; t = n - 1; len = n;
		ans = cnt = 0;
		memset(head, -1, sizeof(head));
		while(m --) {
			cin >> a >> b >> c;
			add(a-1, b-1, c);
			add(b-1, a-1, 0);		//有向图
			//add(b-1, a-1, c);		//无向图
		}
		sap(s, t, len, ans);
		cout << ans << endl;
	}
	return 0;
}
/*==================================================*\
|最大网络流(预流推进算法)O(V^2 * sqrt(E))
| 加强版 HLPP 高标预流推进的时间复杂度更加优化，不过代码更加复
|杂，在这里就不列出来了
|预流推进的算法思想是以边为单元进行推流操作
\*==================================================*/
#include <cstdio>
#include <vector>
#include <queue>
#include <algorithm>
#include<iostream>
using namespace std;
#define MAX_SIZE 1000
int p[MAX_SIZE][MAX_SIZE];
int dis[MAX_SIZE];
int N;
class point{
	public:   int n, h, w;
};
bool operator<(const point &a, const point &b)
{     return a.h < b.h;     }
int min(int a, int b)
{    return a<b?a:b;      }
void work_dis(int s, int t) {		//求出距离标号
	queue<int>w;
	w.push(t);
	dis[t]=0;
	int i, flag[MAX_SIZE]={0}, a;
	flag[t]=1;
	while(!w.empty()){
		a=w.front();
		w.pop();
		for(i=0; i<N; i++){
			if(flag[i] == 0 && p[i][a] >=0){
				flag[i]=1;
				dis[i]=dis[a]+1;
				w.push(i);
```

```
            }
        }
    }
    dis[s]=N;
}
//预流推进算法
int max_flow(int s, int t){
    int flow=0;
    int i;
    point poi, poi1;
    priority_queue<point>v;
    poi.n=s;
    poi.h=dis[s];
    poi.w=0;
    for(i=0; i<N; i++){
        if(p[s][i] > 0) poi.w+=p[s][i];
    }
    v.push(poi);
    while(!v.empty()){
        poi=v.top();
        v.pop();
        for(i=0; i<N; i++){
          if(p[poi.n][i] > 0 && (poi.n==s || dis[poi.n] == dis[i]+1)){
                poi1.n=i;
                poi1.h=dis[i];
                if(i == t) flow+=min(poi.w, p[poi.n][i]);
                if(poi.w <= p[poi.n][i]){
                    p[poi.n][i]-=poi.w;
                    p[i][poi.n]+=poi.w;
                    poi1.w=poi.w;
                    poi.w=0;
                    if(i!=s && i!=t) v.push(poi1);
                    break;
                }
                else{
                    poi.w-=p[poi.n][i];
                    poi1.w=p[poi.n][i];
                    p[i][poi.n]+=p[poi.n][i];
                    p[poi.n][i]=0;
                    if(i!=s && i!=t) v.push(poi1);
                }
            }
        }
        if(poi.w > 0 && poi.n!=s){
            poi.h=relable(poi.n);
            dis[poi.n]=poi.h;
            v.push(poi);
        }
    }
    return flow;
}
int main(){
    int bu, bubu, i, n, m, s, t, a, b, c;
    cin >> bu;
    string str;
    while(bu--){
        cin >> str >> bubu;
        cin >> n >> m >> s >> t;
        memset(p, -1, sizeof(p));
        while(m--){
            scanf("%d%d%d", &a, &b, &c);
            if(p[a][b] == -1) p[a][b]=c;
            else p[a][b]+=c;
            if(p[b][a] == -1) p[b][a]=0;
        }
        N=n;
        work_dis(s, t);
        cout << max_flow(s, t) << endl;
    }
}
```

1.6.2 最小费用流

```
/*==================================================*\
|最小费用流(Primal-Dual 原始对偶算法)
|最小费用流的直接 SPFA 算法和前面说的 KM 重标号算法，各自都有
|一些情况非常慢。这里要写的就是一个所谓的"新算法"(其实非常经典)，
|融合两者优势。
|POJ   3680
\*==================================================*/
#include <cstdio>
#include <cstring>
#include <deque>
#include <algorithm>
using namespace std;
const int V=440, E=V*2, maxint=0x3F3F3F3F;
struct etype
{
    int t, c, u;
    etype *next, *pair;
    etype() {}
    etype(int T, int C, int U, etype* N): t(T), c(C), u(U), next(N) {}
    void* operator new(unsigned, void* p){return p;}
    } *e[V], Te[E+E], *Pe;

    int S, T, n, piS, cost;
    bool v[V];
    void addedge(int s, int t, int c, int u)
    {
        e[s] = new(Pe++) etype(t, +c, u, e[s]);
        e[t] = new(Pe++) etype(s, -c, 0, e[t]);
        e[s]->pair = e[t];
        e[t]->pair = e[s];
    }
    int aug(int no, int m)
    {
        if (no == T) return cost += piS * m, m;
        v[no] = true;
        int l = m;
        for (etype *i = e[no]; i; i = i->next)
            if (i->u && !i->c && !v[i->t])
            {
                int d = aug(i->t, l < i->u ? l : i->u);
                i->u -= d, i->pair->u += d, l -= d;
                if (!l) return m;
            }
        return m - l;
    }
    bool modlabel()
    {
        static int d[V]; memset(d, 0x3F, sizeof(d)); d[T] = 0;
        static deque<int> Q; Q.push_back(T);
        while(Q.size())
        {
            int dt, no = Q.front(); Q.pop_front();
            for(etype *i = e[no]; i; i = i->next)
                if(i->pair->u && (dt = d[no] - i->c) < d[i->t])
                 (d[i->t] = dt) <= d[Q.size() ? Q.front() : 0]
                        ?Q.push_front(i->t): Q.push_back(i->t);
        }
        for(int i = 0; i < n; ++i)
            for(etype *j = e[i]; j; j = j->next)
                j->c += d[j->t] - d[i];
        piS += d[S];
        return d[S] < maxint;
    }
    int ab[V], *pab[V], w[V];
    struct lt
    {   bool operator()(int* p1,int* p2) {return *p1 < *p2;}   };
    int main(){
        int t;
        scanf("%d",&t);
        while(t--){
            memset(e,0,sizeof(e));
            Pe = Te;
            static int m, k;
            scanf("%d %d", &m, &k);
            int abz = 0;
            for(int i = 0; i < m; ++i){
                scanf("%d", pab[abz] = &ab[abz]), abz++;
                scanf("%d", pab[abz] = &ab[abz]), abz++;
                scanf("%d", &w[i]);
```

```
            }
            sort(&pab[0], &pab[abz], lt());
            int c=0xDEADBEEF; n=0;
            for(int i = 0; i < abz; ++i){
                if(c != *pab[i]) c = *pab[i], ++n;
                *pab[i] = n;
            }
            ++n, S = 0, T = n++;
            for(int i = 0; i < T; ++i) addedge(i, i+1, 0, k);
        for(int i = 0; i < m; ++i) addedge(ab[i+i], ab[i+i+1], -w[i], 1);
            piS = cost = 0;
            while(modlabel())
                do memset(v, 0, sizeof(v));
                while(aug(S, maxint));
            printf("%d\n", -cost);
        }
        return 0;
    }
```

```
/*==================================================*\
| 最小费用流 O(V * E * f) (SPFA 求增广)
| 注意: SPFA 增广，实际复杂度远远小于 O(V * E);
\*==================================================*/
#include <iostream>
#include <queue>
using namespace std;
#define size 110
struct edge {
    int from, to, next;
    int flow, cost;
} e[size * 1000];
int v[size], cnt, pre[size], pos[size], dis[size], vst[size];
//v1 --> v2, flow 流量, cost 费用
//无向||有向都这个加
void insert(int from, int to, int flow, int cost) {
    e[cnt].from = from;
    e[cnt].to = to;
    e[cnt].flow = flow;
    e[cnt].cost = cost;
    e[cnt].next = v[from];
    v[from] = cnt++;
    swap(from, to);
    e[cnt].from = from;
    e[cnt].to = to;
    e[cnt].flow = 0;
    e[cnt].cost = -cost;
    e[cnt].next = v[from];
    v[from] = cnt++;
}
//判断有无最小代价路(就是求最短路)，如果存在负回路，说明结果已求出
bool spfa(int s, int t, int n) {
    memset(pre, -1, sizeof(pre));
    memset(vst, 0, sizeof(vst));
    queue<int> Q;
    for(int i = 1; i <= n; ++ i) dis[i] = INT_MAX;
    Q.push(s); pre[s] = s; dis[s] = 0; vst[s] = 1;
    while(!Q.empty()) {
        int id = Q.front(); Q.pop(); vst[id] = 0;
        id = v[id];
        while(id != -1) {
            if (e[id].flow > 0 && dis[e[ id ].from] + e[ id ].cost < dis[e[ id ].to]) {
                dis[e[ id ].to] = dis[e[ id ].from] + e[ id ].cost;
                pre[e[ id ].to] = e[ id ].from; pos[e[ id ].to] = id;
                if (!vst[e[ id ].to]) {
                    vst[e[ id ].to] = 1;
                Q.push(e[id].to);
                }
            }
            id = e[ id ].next;
        }
    }
    return pre[ t ] != -1 && dis[t] < INT_MAX;
}
int MinCostFlow(int s, int t, int n, int k) {
    int flow = 0, cost = 0, i;
    while(spfa(s, t, n)) {
        int fl = INT_MAX;
        for(i = t; i != s; i = pre[ i ])
            if (e[pos[ i ]].flow < fl) fl = e[pos[ i ]].flow;
                flow += fl; cost += dis[ t ] * fl;
                for(i = t; i != s; i = pre[ i ]) {
                    e[pos[ i ]].flow -= fl;
                    e[pos[ i ] ^ 1].flow += fl;
                }
    }
     if(flow < k) cost = -1;
     return cost; //flow 是最大流值
}
int n, m, k, a, b, c, d;
int main() {
    while(scanf("%d %d %d", &n, &m, &k) == 3) {
        cnt = 0;
        memset(v, -1, sizeof(v));
        for(int i = 0; i < m; i++) {
            scanf("%d %d %d %d", &a, &b, &c, &d);
            for(int j = 0; j < d; ++ j) {
                insert(a, b, 1, (j*2+1)*c);
            }
        }
        insert(0, 1, k, 0);
        insert(n, n+1, k, 0);
        printf("%d\n", MinCostFlow(0, n + 1, n + 2, k));
     }
    return 0;
}
```

```
/*==================================================*\
|上下界最大流(表)
\*==================================================*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
const long maxn = 210;
const long maxm = 120000;
const long inf = 0x7fffffff;
struct node {
    long v, next;
    long val;
}s[maxm];
long level[maxn], p[maxn], que[maxn], out[maxn], ind;
long b[maxm];
void init() {
    ind = 0;
    memset(p, -1, sizeof(p));
}
inline void insert(long x, long y, long z) {     //////有向
    s[ind].v = y;
    s[ind].val = z;
    s[ind].next = p[ x ];
    p[x] = ind ++;
    s[ind].v = x;
    s[ind].val = 0;
    s[ind].next = p[ y ];
    p[y] = ind ++;
}
inline void insert2(long x,long y,long z) {     //////无向
    s[ind].v=y;
    s[ind].val=z;
    s[ind].next=p[x];
    p[x]=ind++;
    s[ind].v=x;
    s[ind].val=z;
    s[ind].next=p[y];
    p[y]=ind++;
}
long max_flow(long n, long source, long sink) {
    long ret = 0;
    long h = 0, r = 0;
    while(true) {
```

```
        long i;
        for(i = 0; i < n; ++ i) level[ i ] = 0;
        h = 0, r = 0;
        level[source]=1;
        que[ 0 ] = source;
        while(h <= r) {
            long t = que[h ++];
            for(i = p[ t ]; i != -1; i = s[ i ].next) {
                if(s[ i ].val && level[s[ i ].v] == 0) {
                    level[s[ i ].v] = level[ t ] + 1;
                    que[++ r] = s[ i ].v;
                }
            }
        }
        if(level[sink] == 0) break;
        for(i = 0; i < n; ++ i) out[ i ] = p[ i ];
          long q = -1;
        while(true) {
            if(q < 0) {
                long cur = out[source];
                for(;cur != -1; cur =s[cur].next) {
                    if(s[cur].val   &&   out[s[cur].v]   !=   -1   &&
level[s[cur].v] == 2) break;
                }
                if(cur < 0) break;
                     que[++ q] = cur;
                     out[source] = s[cur].next;
            }
            long u = s[que[q]].v;
            if(u == sink) {
                long dd = inf;
                long index = -1;
                for(i = 0; i<= q; ++ i) {
                    if(dd > s[que[ i ]].val) {
                        dd = s[que[ i ]].val;
                        index = i;
                    }
                }
                ret += dd;
                //cout<<ret<<endl;
                for(i = 0;i <= q;++ i) {
                    s[que[ i ]].val -= dd;
                    s[que[ i ]^1].val += dd;
                }
                for(i = 0;i <= q; ++ i) {
                    if(s[que[ i ]].val == 0) {
                        q = index - 1;
                        break;
                    }
                }
            }
            else {
                long cur = out[ u ];
                for(;cur != -1; cur = s[cur].next) {
                    if(s[  cur  ].val  &&  out[s[  cur  ].v]  !=  -1  &&
level[ u ] + 1 == level[s[cur].v]) break;
                }
                if(cur != -1) {
                    que[++ q] = cur;
                    out[ u ] = s[cur].next;
                }
                else {
                    out[ u ] = -1;
                    q --;
                }
            }
        }
    }
    return ret;
}
bool hasSolution() {
        bool ff = true;
        for(int k = 2; k < ind; k += 6) {
            if(s[ k ].val || s[k + 2].val) {
                ff = false;
                break;
            }
        }
        return ff;
}
long m, n;
int main() {
        int z = 0;
        int test;
        cin >> test;
      while(test -- && scanf("%ld %ld", &n, &m) != EOF) {
          init();
            long S = 0, E = n + 1;
          for(int i = 0; i < m; ++ i) {
              long from, to, c1, c2;
              scanf("%ld %ld %ld %ld", &from, &to, &c1, &c2);
              insert(from, to, c2 - c1);
                  insert(S, to, c1);
              insert(from, E, c1);
                  b[ i ] = c1;
          }
          max_flow(n+2, S, E);
            if(!hasSolution()) {
                  puts("NO");
            }
            else {
                  puts("YES");
                  int cnt = 0;
                  for(int j = 1; j < ind; j += 6) {
                      printf("%ld\n", s[ j ].val + b[cnt ++]);
                  }
            }
            if(test) cout << endl;
      }
      return 0;
}
/*==================================================*\
|上下界最小流(表)
|增加一条弧(T, S)，使原网络变成一个无源汇的网络。如果求最大流，则
|B(T, S) = a，C(T, S) = inf;
|如果求解最小流，则 B(T, S) = 0，C(T, S) = a。可以通过二分法枚举 a，
|按上文提到的方法构造附加网络，
|判断附加网络中是否有可行流即可。最终，a 即为所求。
\*==================================================*/
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int MaxN=210;
const int INF=999999999;
int gap[MaxN],dis[MaxN],pre[MaxN],cur[MaxN];
int s, t;
int
NE,NV,head[MaxN],in[210],out[210],connetT[MaxN],len,N,M,the
ad[MaxN],tne,bound[100010],ans[100010],flow;
struct node {
  int c,pos,next;
}E[100010], tE[100010];
void checkmin(int &a,int b) {
  if(b < a) a = b;
}
int sap(int s, int t) {
   int v, i;
         memset(dis,0,sizeof dis);
         memset(gap,0,sizeof gap);
         for(i=0;i<=NV;i++)
             cur[i]=head[i];
         int u=pre[s]=s,maxflow=0,aug=INF;
         gap[s]=NV;
        while(dis[s]<NV){
loop: for(int &i=cur[u];i!=-1;i=E[i].next){
      v=E[i].pos;
      if(E[i].c && dis[u]==dis[v]+1){
        checkmin(aug,E[i].c);
        pre[v]=u;
        u=v;
        if(v==t){
```

```
        maxflow+=aug;
        for(u=pre[u];v!=s;v=u,u=pre[u]){
         E[cur[u]].c-=aug;
         E[cur[u]^1].c+=aug;
        }
        aug=INF;
       }
       goto loop;
      }
     }
     int mindis=NV;
     for(i=head[u];i!=-1;i=E[i].next){
      v=E[i].pos;
      if(E[i].c && mindis>dis[v]){
       cur[u]=i;
       mindis=dis[v];
      }
     }
     if((--gap[dis[u]])==0)
      break;
     gap[dis[u]=mindis+1]++;
     u=pre[u];
    }
    return maxflow;
}
void Insert(int u,int v,int c,int cc=0){
    E[NE].c=c;E[NE].pos=v;
    E[NE].next=head[u];head[u]=NE++;
    E[NE].c=cc;E[NE].pos=u;
    E[NE].next=head[v];head[v]=NE++;
}
void init(){
    memset(head,-1,sizeof head);
    memset(in,0,sizeof in);
    memset(out,0,sizeof out);
    NE=0;
    len=0;
}
void repeat(){
      int i;
      NE=tne;
      for(i=0;i<tne;i++)
        E[i]=tE[i];
      for(i=1;i<=NV;i++)
        head[i]=thead[i];
}
bool check(int mid){
        int i,j;
        repeat();
        Insert(N,1,mid);
        sap(s,t);
        for(i=head[s];i!=-1;i=E[i].next)
            if(E[i].c!=0)
               return false;
        for(i=1,j=0;j<M;i+=2,j++)
            ans[j]=E[i].c+bound[j];
            flow=mid;
            return true;
}
void solve(){
        int l,r,mid,j;
        l=-1;r=1000000;flow=-1;
        if(check(r)){
            while(l+1<r)
            {
               mid=(l+r)/2;
               if(check(mid))
                r=mid;
          else
               l=mid;
     }
    }
    if(flow!=-1){
        printf("%d\n",flow);
        for(j=0;j<M-1;j++)
        printf("%d ",ans[j]);
           printf("%d\n",ans[j]);
    }
    else
           printf("Impossible\n");
}
int main(){
    int i,c,u,v,b,w;
    init();
    scanf("%d %d",&N,&M);
    for(i=0;i<M;i++){
     scanf("%d %d %d %d",&u,&v,&w,&b);
     if(b){
      in[v]+=w;
      out[u]+=w;
      Insert(u,v,0);
      bound[i]=w;
     }
     else{
      Insert(u,v,w);
      bound[i]=0;
     }
    }
         s=N+1;t=s+1;NV=t;
         for(i=1;i<=N;i++)
         if(in[i]-out[i]>0)
         Insert(s,i,in[i]-out[i]);
     else {
           Insert(i,t,out[i]-in[i]);
           connetT[len++]=i;
     }
         tne=NE;
         for(i=1;i<=NV;i++)
         thead[i]=head[i];
         for(i=0;i<NE;i++)
           tE[i]=E[i];
         solve();
    return 0;
}
```

1.7 图的一些割集

```
/*==================================================*\
| 无向图连通度(割)
| INIT: edge[][]邻接矩阵;vis[],pre[],anc[],deg[]置为 0;
| CALL: dfs(0, -1, 1, n);
| k=deg[0], deg[i]+1(i=1…n-1)为删除该节点后得到的连通图个数
| 注意:0 作为根比较特殊!
\*==================================================*/
int edge[V][V], anc[V], pre[V], vis[V], deg[V];
void dfs(int cur, int father, int dep, int n)
{                                          // vertex: 0 ~ n-1
    int cnt = 0;
    vis[cur] = 1; pre[cur] = anc[cur] = dep;
    for (int i=0; i<n; ++i) if (edge[cur][i]) {
        if (i != father && 1 == vis[i]) {
          if (pre[i] < anc[cur])
            anc[cur] = pre[i];              //back edge
         }
        if (0 == vis[i]) {                  //tree edge
          dfs(i, cur, dep+1, n);
              ++cnt;                        // 分支个数
        if (anc[i] < anc[cur]) anc[cur] = anc[i];
 if ((cur==0 && cnt>1) ||(cnt!=0 && anc[i]>=pre[cur]))
             ++deg[cur];                    // link degree of a vertex
       }
    }
    vis[cur] = 2;
}
/*==================================================*\
| 无向图最小割 O(N^3)
| INIT: 初始化邻接矩阵 g[][];
| CALL: res = mincut(n);
| 注: Stoer-Wagner Minimum Cut;
| 找边的最小集合，若其被删去则图变得不连通（我们把这种形式称为
最小割问题）
\*==================================================*/
```

```
/***********************************************************
*Stoer-Wagner Minimum Cut 求无源无汇的无向网络最小割
* poj 2914 Minimum Cut
***********************************************************/
#include <iostream>
#include <cstdio>
using namespace std;
int const MAXN = 512;
int const INF = 123456789;
int graph[MAXN][MAXN], vertex[MAXN], weight[MAXN];
//容量矩阵和节点数组
bool a[MAXN];
int minCut( int n ){                    //传入 n 为节点数
        for( int i = 0; i < n; i++ )
            vertex[i] = i;
            int best = INF;
           while( n > 1 ){
               a[vertex[0]] = true;
               for( int i = 1; i < n; i++ ){
                   a[vertex[i]] = false;
                   weight[i] = graph[vertex[0]][vertex[i]];
               }
                int prev = vertex[0];
                for( int i = 1; i < n; i++ ){
                     int t = -1;
                     for( int j = 1; j < n; j++ ){
                if( !a[vertex[j]] && ( t < 0 || weight[j] > weight[t] ) ){
                               t = j;
                 }
           }
                a[vertex[t]] = true;
                if( i == n - 1 ){
                best = (best< weight[t])?best:weight[t];
                  for( int i = 0; i < n; i++ )
                     graph[vertex[i]][prev] =
           graph[prev][vertex[i]] += graph[vertex[t]][vertex[i]];
                    vertex[t] = vertex[--n];
                     break;
               }
           prev = vertex[t];
        for( int j = 1; j < n; j++ ){
           if( !a[vertex[j]] ){
             weight[j] += graph[vertex[t]][vertex[j]];
            }
         }
      }
 }
  return best;
}
int N, M;
bool read_data(){
    if(scanf("%d %d", &N, &M) != EOF) {
        for (int i = 0; i < N; ++i){
          for (int j = 0; j < N; ++j){
              graph[i][j] = 0;
          }
        }
      int u, v, w;
      while (M--){
         scanf("%d %d %d", &u, &v, &w);
             graph[u][v] = graph[v][u] += w;
       }
     return true;
   }
return false;
}
int main(){
    while (read_data() ){
      int ans = minCut(N);
         printf("%d\n", ans);
    }
  return 0;
}
```

/*==*\
| 最佳边割集
==/

```
#define MAXN 100
#define inf 1000000000
int max_flow(int n,int mat[][MAXN],int source,int sink){
     int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;
     for (;;){
       for (i=0;i<n;i++)
         v[i]=c[i]=0;
       for (c[source]=inf;;){
          for (j=-1,i=0;i<n;i++)
              if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))
       j=i;
      if (j<0) return ret;
      if (j==sink) break;
      for (v[j]=1,i=0;i<n;i++)
        if (mat[j][i]>c[i]&&c[j]>c[i])
          c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;
     }
     for (ret+=j=c[i=sink];i!=source;i=p[i])
        mat[p[i]][i]-=j, mat[i][p[i]]+=j;
    }
 }
int best_edge_cut(int n,int mat[][MAXN],int source,int
sink,int set[][2],int& mincost){
int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,l,ret=0,last;
            if (source==sink)
               return -1;
          for (i=0;i<n;i++)
             for (j=0;j<n;j++)
                m0[i][j]=mat[i][j];
         for (i=0;i<n;i++)
             for (j=0;j<n;j++)
                m[i][j]=m0[i][j];
             mincost=last=max_flow(n,m,source,sink);
         for (k=0;k<n&&last;k++)
             for (l=0;l<n&&last;l++)
                 if (m0[k][l]){
                    for (i=0;i<n+n;i++)
                       for (j=0;j<n+n;j++)
                          m[i][j]=m0[i][j];
                     m[k][l]=0;
           if(max_flow(n,m,source,sink)==last-mat[k][l]){
                  set[ret][0]=k;
                  set[ret++][1]=l;
                  m0[k][l]=0;
                  last-=mat[k][l];
             }
     }
 return ret;
}
```

/*==*\
| 最佳点割集
==/

```
#define MAXN 100
#define inf 1000000000
int max_flow(int n,int mat[][MAXN],int source,int sink){
int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;
        for (;;){
           for (i=0;i<n;i++)
             v[i]=c[i]=0;
          for (c[source]=inf;;){
              for (j=-1,i=0;i<n;i++)
                 if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))
                 j=i;
             if (j<0) return ret;
             if (j==sink) break;
             for (v[j]=1,i=0;i<n;i++)
               if (mat[j][i]>c[i]&&c[j]>c[i])
                 c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;
           }
      for (ret+=j=c[i=sink];i!=source;i=p[i])
          mat[p[i]][i]-=j,mat[i][p[i]]+=j;
   }
}
int best_vertex_cut(int n,int mat[][MAXN],int* cost,int
source,int sink,int* set,int& mincost){
int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,ret=0,last;
```

```
        if (source==sink||mat[source][sink])
                return -1;
        for (i=0;i<n+n;i++)
            for (j=0;j<n+n;j++)
                m0[i][j]=0;
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                if (mat[i][j])
                    m0[i][n+j]=inf;
        for (i=0;i<n;i++)
            m0[n+i][i]=cost[i];
        for (i=0;i<n+n;i++)
            for (j=0;j<n+n;j++)
                m[i][j]=m0[i][j];
            mincost=last=max_flow(n+n,m,source,n+sink);
        for (k=0;k<n&&last;k++)
            if (k!=source&&k!=sink){
        for (i=0;i<n+n;i++)
            for (j=0;j<n+n;j++)
                m[i][j]=m0[i][j];
         m[n+k][k]=0;
        if(max_flow(n+n,m,source,n+sink)==last-cost[k]){
             set[ret++]=k;
             m0[n+k][k]=0;
             last-=cost[k];
        }
    }
return ret;
}
```

/*==*\
| 最小边割集
==/

```
#define MAXN 100
#define inf 1000000000
int max_flow(int n,int mat[][MAXN],int source,int sink){
int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;
            for (;;){
                for (i=0;i<n;i++)
                    v[i]=c[i]=0;
                for (c[source]=inf;;){
                    for (j=-1,i=0;i<n;i++)
                        if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))
                                j=i;
                        if (j<0) return ret;
                        if (j==sink) break;
                    for (v[j]=1,i=0;i<n;i++)
                        if (mat[j][i]>c[i]&&c[j]>c[i])
                       c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;
                }
        for (ret+=j=c[i=sink];i!=source;i=p[i])
            mat[p[i]][i]-=j,mat[i][p[i]]+=j;
            }
}
int min_edge_cut(int n,int mat[][MAXN],int source,int
sink,int set[][2]){
int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,l,ret=0,last;
            if (source==sink)
                return -1;
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                m0[i][j]=(mat[i][j]!=0);
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                m[i][j]=m0[i][j];
         last=max_flow(n,m,source,sink);
        for (k=0;k<n&&last;k++)
            for (l=0;l<n&&last;l++)
                if (m0[k][l]){
                    for (i=0;i<n+n;i++)
                        for (j=0;j<n+n;j++)
                            m[i][j]=m0[i][j];
                            m[k][l]=0;
        if (max_flow(n,m,source,sink)<last){
                set[ret][0]=k;
                set[ret++][1]=l;
                m0[k][l]=0;
                    last--;
            }
        }
return ret;
}
```

/*==*\
| 最小点割集（点连通度）
==/

```
#define MAXN 100
#define inf 1000000000
int max_flow(int n,int mat[][MAXN],int source,int sink){
int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;
            for (;;){
                for (i=0;i<n;i++)
                    v[i]=c[i]=0;
                for (c[source]=inf;;){
                    for (j=-1,i=0;i<n;i++)
                        if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))
                                j=i;
                    if (j<0) return ret;
                    if (j==sink) break;
                for (v[j]=1,i=0;i<n;i++)
                    if (mat[j][i]>c[i]&&c[j]>c[i])
                    c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;
                }
            for (ret+=j=c[i=sink];i!=source;i=p[i])
                mat[p[i]][i]-=j,mat[i][p[i]]+=j;
            }
}
int min_vertex_cut(int n,int mat[][MAXN],int source,int
sink, int* set){
int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,ret=0,last;
            if (source==sink||mat[source][sink])
                return -1;
            for (i=0;i<n+n;i++)
                for (j=0;j<n+n;j++)
                    m0[i][j]=0;
            for (i=0;i<n;i++)
                for (j=0;j<n;j++)
                    if (mat[i][j])
                        m0[i][n+j]=inf;
            for (i=0;i<n;i++)
                m0[n+i][i]=1;
            for (i=0;i<n+n;i++)
                for (j=0;j<n+n;j++)
                    m[i][j]=m0[i][j];
            last=max_flow(n+n,m,source,n+sink);
            for (k=0;k<n&&last;k++)
                if (k!=source&&k!=sink){
                    for (i=0;i<n+n;i++)
                        for (j=0;j<n+n;j++)
                            m[i][j]=m0[i][j];
                    m[n+k][k]=0;
                if (max_flow(n+n,m,source,n+sink)<last){
                        set[ret++]=k;
                        m0[n+k][k]=0;
                        last--;
                }
            }
return ret;
}
```

1.8 图的一些综合应用

/*==*\
|最小路径覆盖 O(n^3)

路径覆盖：就是在图中找一些路经，使之覆盖了图中的所有顶点，且任何一个顶点有且只有一条路径与之关联。
最小路径覆盖：就是找出最少的路径条数，使之成为 P 的一个路径覆盖.
路径覆盖与二分图匹配的关系：最小路径覆盖＝｜P｜－最大匹配数；
其中最大匹配数的求法是把 P 中的每个顶点 pi 分成两个顶点 pi'与 pi"，如果在 p 中存在一条 pi 到 pj 的边，那么在二分图 P′ 中就有一条连接 pi'与 pj"的有向边（求二分图匹配时必须是单向边）；这里 pi' 就是 p 中 pi 的出边，pj"就是 p 中 pj 的一条入边；
有向图：最小路径覆盖＝｜P｜－最大匹配数;
无向图：最小路径覆盖＝｜P｜－最大匹配数/2;

```
\*==================================================*/
/*==================================================*\
| 最小点集覆盖
| 结论：一个二分图中的最大匹配数等于这个图中的最小点覆盖数。
\*==================================================*/
//最小路径覆盖,O(n^3)
//求解最小的路径覆盖图中所有点,有向图无向图均适用
//注意此问题等价二分图最大匹配,可以用邻接表或正向表减小复杂度
//返回最小路径条数,pre 返回前指针(起点-1),next 返回后指针(终点-1)
#include <string.h>
#define MAXN 310
#define _clr(x) memset(x,0xff,sizeof(int)*n)

int hungary(int n,int mat[][MAXN],int* match1,int* match2){
    int s[MAXN],t[MAXN],p,q,ret=0,i,j,k;
    for
(_clr(match1),_clr(match2),i=0;i<n;ret+=(match1[i++]>=0))
        for (_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)
            for (k=s[p],j=0;j<n&&match1[i]<0;j++)
                if (mat[k][j]&&t[j]<0){
                    s[++q]=match2[j],t[j]=k;
                    if (s[q]<0)
                        for (p=j;p>=0;j=p)

    match2[j]=k=t[j],p=match1[k],match1[k]=j;
                }
    return ret;
}

inline int path_cover(int n,int mat[][MAXN],int* pre,int* next){
    return n-hungary(n,mat,next,pre);
}
/*==================================================*\
| DAG 的深度优先搜索标记
| INIT: edge[][]邻接矩阵; pre[], post[], tag 全置 0;
| CALL: dfstag(i, n); pre/post:开始/结束时间
\*==================================================*/
int edge[V][V], pre[V], post[V], tag;
void dfstag(int cur, int n){                    // vertex: 0 ~ n-1
    pre[cur] = ++tag;
    for (int i=0; i<n; ++i) if (edge[cur][i]) {
        if (0 == pre[i]) {
            printf("Tree Edge!\n");
            dfstag(i, n);
        }
        else {
            if (0 == post[i]) printf("Back Edge!\n");
            else if (pre[i] > pre[cur])
                printf("Down Edge!\n");
            else printf("Cross Edge!\n");
        }
    }
    post[cur] = ++tag;
}
/*==================================================*\
|求割顶
|连通图 G 的一个顶点子集 V,如果删除这个顶点子集和它所附带的边后,
|图便不再连通。则称 V 是 G 的割顶集。
|最小割顶集中顶点的个数，称为 G 的连通度。连通度等于 1 时，割顶集
|中的那个顶点叫做割顶。
|注意：完全图的连通度为总顶点数－1；
|牵一发而动全身的点称为割点
|ZOJ   1311
\*==================================================*/
#include<cstdio>
#include<cstring>
#define MAXN 105
int Edge[MAXN][MAXN];//邻接矩阵
int visited[MAXN];//表示顶点访问状态
int n;//顶点数目
int tmpdfn;//在 dfs 过程中记录当前的深度优先搜索序数
int dfn[MAXN];//每个顶点的 dfn 值
int low[MAXN];//每个顶点的 low 值，根据该值来判断是否是关节点
int son;//根结点的子女结点个数
int critical;//该图关节点个数
int subnet[MAXN];//记录关节点 u 是否已经被计数的标志数组
int min(int a,int b){
    if(a>b) return b;else return a;
}
void dfs(int u){
    for(int v=1;v<=n;v++){
        //v 和 u 邻接，则有：1.v 是 u 的祖先结点，（v,u）是一条回边；2.v 是 u 的儿子结点
        if(Edge[u][v])
        {
            if(!visited[v])//v 是 u 的儿子结点
            {
                visited[v]=1;
                tmpdfn++;dfn[v]=low[v]=tmpdfn;
                dfs(v);//自底向上求出每个 low[v]
                low[u]=min(low[u],low[v]);//回退计算 u 的 low 值
                if(low[v]>=dfn[u])//u 为关节点
                {
    if(u!=1&&!subnet[u]){subnet[u]=1;  critical++;}//找到一个关节点 u
                    if(u==1) son++;//根结点特殊处理
                }
            }
            //此前 v 已经被访问过了，则 v 是 u 的祖先结点，为回边情况
            else low[u]=min(low[u],dfn[v]);
        }
    }
}
void init()//初始化函数{
    low[1]=dfn[1]=1;
    tmpdfn=1;
    son=0;
    memset(visited,0,sizeof(visited));
    visited[1]=1;
    critical=0;
    memset(subnet,0,sizeof(subnet));
}
int main(){
    int i;
    int u,v;
    while( scanf("%d",&n),n ){
        getchar( );             //跳过上一行的回车换行符
        memset( Edge,0,sizeof(Edge));
        char buf[128]; //用来读取每一行信息的字符数组
        int u;        //读入的第一个数字，如果第一个数字是 0，则该 block 结束
        int v;        //用来存放从 buf 字符串中读入的整数
        while( 1 ){
            int k = 0; //在 buf 字符串，读数据的起始位置
            int first = 1;    //判断读入的是不是第一个数字
            gets( buf );
            //读入一个 space 通往其他 spaces 的信息至 buf，通过 buf，对 j 进行赋值
            while( sscanf(buf+k,"%d",&v) ==1){
                if(first){
                    first = 0;u=v;
                }
                else Edge[u][v]=Edge[v][u]=1;
                while(buf[k]&&buf[k]==' ')      k++;  //移动 k,跳过空格
                while(buf[k]&&buf[k]!=' ')      k++;  //移动 k,跳过前面的数，准备读入下一个数
            }
            if(u==0)   break;     //该测试数据的输入结束
        }
        init();
        dfs(1);//从深度优先树根结点 1 开始（默认 1 为根）
        if(son>1) critical++;
        printf("%d\n",critical);
    }
    return 0;
}
/*==================================================*\
| 无向图找桥
```

```
| INIT: edge[][]邻接矩阵;vis[],pre[],anc[],bridge 置 0;
| CALL: dfs(0, -1, 1, n);
\*==================================================*/
int bridge, edge[V][V], anc[V], pre[V], vis[V];
void dfs(int cur, int father, int dep, int n){        // vertex: 0 ~ n-1
        if (bridge) return;
        vis[cur] = 1; pre[cur] = anc[cur] = dep;
        for (int i=0; i<n; ++i) if (edge[cur][i]) {
              if (i != father && 1 == vis[i]) {
                  if (pre[i] < anc[cur])
                anc[cur] = pre[i];//back edge
             }
        if (0 == vis[i]) {                          //tree edge
           dfs(i, cur, dep+1, n);
           if (bridge) return;
           if (anc[i] < anc[cur]) anc[cur] = anc[i];
           if (anc[i] > pre[cur]) { bridge = 1; return; }
       }
  }
vis[cur] = 2;
}
```

```
/*==================================================*\
|求桥的算法 (zoj 2588 Burning Bridges)
\*==================================================*/
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <set>
using namespace std;
int N, M;
int const MAXN = 204800;
struct node{//边节点
        int u, v, num;//num 是桥的标号
        node(){}
        node(int u_, int v_){
                u = u_;
                v = v_;
        }
};
node edge[MAXN];
struct cmp{
        bool operator() (node a, node b){
                if(a.u != b.u){
                        return a.u < b.u;
                }
                return a.v < b.v;
        }
};
void read_data(){
        scanf("%d %d", &N, &M);
        for (int i = 0; i < M; ++i){
                scanf("%d %d", &edge[i].u, &edge[i].v);
                --edge[i].u;
                --edge[i].v;
                edge[i].num = i + 1;
                edge[i + M] = edge[i];
                swap(edge[i + M].u, edge[i + M].v);
        }
        M *= 2;
        sort (edge, edge + M, cmp() );
}
int T, root;
int Anc[MAXN];//记录祖先节点
int D[MAXN], C[MAXN], A[MAXN];//辅助数组
set <int> bridge;//记录桥的 set
void init(){//注意初始化
      for(int   i = 0; i < N; i++)
            C[i] = 0;
      T = 0;
        root = 0;
        bridge.clear();
}
void DFS(int k, int father, int deep){
        //调用与求割顶算法类似，初次调用的时候，取 k=0(若点从 0 开始
标记）作为当前点，
        //father 为-1，depp 为 0。
        int i, tot(0);
        C[k] = 1;
        D[k] = deep;
        Anc[k] = deep;
        int op = lower_bound(edge, edge + M, node(k, -1), cmp() ) -
edge;
        for (int j = op; j < M && edge[j].u == k; ++j){
                i = edge[j].v;
                if(i != father && C[i] == 1){
                        Anc[k] = min(Anc[k], D[i]);
                }
                if(C[i] == 0){
                        DFS(i, k, deep + 1);
                        ++tot;
                        Anc[k] = min(Anc[k], Anc[i]);
                        if(Anc[i] > D[k]){//i 点在 k 之前，找到桥，插入集合中
                                if(j < M - 1 && edge[j].u == edge[j + 1].u &&
                                      edge[j].v == edge[j + 1].v)
                                      {//原题中有重边
                                       }
                                       else{
                                               bridge.insert(edge[j].num);
                                       }
                        }
                }
        }
        C[k] = 2;
        A[i] = ++T;
}
int main(){
        int T;
        scanf("%d", &T);
        bool flag(false);
        while (T--){
                read_data();
                init();
                DFS(0, -1, 0);
                if(flag){
                        printf("\n");
                }
                flag = true;
                printf("%d\n", bridge.size() );
                bool f(false);
                for (set <int>::iterator iter = bridge.begin();
                      iter != bridge.end(); ++iter){
                          if(f){
                                  printf(" ");
                          }
                          f = true;
                          printf("%d", *iter);
                }
                if(bridge.size() ){
                        printf("\n");
                }
        }
        return 0;
}
```

```
/*==================================================*\
|欧拉回路(判断是否存在)
|1 对于无向图的欧拉回路
|2 任意一点的度数都为偶数，且在同一个连通图中
\*==================================================*/
#include <iostream>
using namespace std;
#define MAXN 1001
int pre[MAXN], n, m, ans;
int Find(int x) {
      int b , _x = x;
      while (pre[ _x ] != _x) _x = pre[ _x ];
      while (pre[ x ] != x) {
            b = pre[ x ];
            pre[ x ] = _x;
            x = b;
      }
      return _x;
}
```

```
int main() {
	int a , b, i;
	int deg[MAXN];
	bool flag;
	while(~scanf("%d", &n) && n) {
		memset(deg, 0, sizeof (deg));
		scanf ("%d" , &m);
		for (i = 1; i <= n; ++ i) pre[ i ] = i;
		while(m --) {
			scanf("%d %d", &a, &b);
			deg[ a ] ++; deg[ b ] ++;
			a = Find (a);
			b = Find (b);
			if (a != b) pre[ a ] = b;
		}
		ans = 0;
		for (i = 1; i <= n; ++ i) if (pre[ i ] == i) ans ++;
		if (ans != 1) {
			//有 2 个连通分量以上，不存在欧拉回路
			puts ("0");
			continue;
		}
		flag = true;
		for (i = 1; i <= n; ++ i) {
			if (deg[ i ] & 1) {
				flag = false;
				break;
			}
		}
		if (flag) puts("1");
		else	puts("0");
	}
	return 0;
}
/*==================================================*\
| 欧拉回路(输出回路的弧)
|输出欧拉回路经过的弧
\*==================================================*/
#include <iostream>
#include <stack>
using namespace std;
#define MAXV 1001
#define MAXE 100000
#define MIN(a, b) a<b?a:b
#define MAX(a, b) a>b?a:b
typedef struct {
	int v;
	bool vist;
	int next;
	int id;
}Edge;
Edge e[MAXE];
int eid, p[MAXV];
int deg[MAXV];
stack<int> S;
void add(int u, int v, int id) {
	e[ eid ].next = p[ u ];
	e[ eid ].v = v;
	e[ eid ].vist = false;
	e[ eid ].id = id;
	p[ u ] = eid ++;
}
void init() {
	eid = 0;
	memset(p, -1, sizeof(p));
	memset(deg, 0, sizeof(deg));
}
void dfs(int u) {
	for(int i = p[ u ]; i != -1; i = e[ i ].next) {
		if(e[ i ].vist) continue;
		e[ i ].vist = e[i^1].vist = true;
		dfs(e[ i ].v);
		S.push(e[ i ].id);
	}
}
int main() {
	int u, v, id, s, n;
	bool flag;
	while(cin >> u >> v) {
		if(u == 0 && v == 0) break;
		flag = true;
		init();
		deg[ u ] ++;
		deg[ v ] ++;
		s = MIN(u, v);
		n = MAX(u, v);
		cin >> id;
		add(u, v, id);
		add(v, u, id);
		while(cin >> u >> v) {
			if(u == 0 && v == 0) break;
			cin >> id;
			add(u, v, id);
			add(v, u, id);
			deg[ u ] ++;
			deg[ v ] ++;
			n = MAX(u, n);
			n = MAX(v, n);
		}
		for(int i = 1; i <= n; ++ i) {
			if(deg[ i ] & 1) {
				flag = false;
				break;
			}
		}
		if(!flag) {
			puts("Round trip does not exist.");
			continue;
		}
		dfs(s);
		if(!S.empty()) {
			u = S.top();
			S.pop();
		}
		printf("%d", u);
		while(!S.empty()) {
			u = S.top();
			S.pop();
			printf(" %d", u);
		}
		cout << endl;
	}
	return 0;
}
/*==================================================*\
|弦图的判定
|弦图的定义：无向图中,如果任意边数大于 3 的环,至少存在一条边连接
环中不相邻的某两个点,则称此图为弦图(Chordal Graph)
| ZOJ   1015
\*==================================================*/
#include <iostream>
#include <cstring>
#include <cstdio>
#define maxn 1111
using namespace std;
bool g[maxn + 1][maxn + 1];
int weight[maxn + 1];
int label[maxn + 1];
int op_label[maxn + 1];
int n, m;
int main() {
	int case_count = 0, i, j;
	while (scanf("%d %d", &n, &m) == 2 && n) {
//		if (case_count) printf("\n");
		memset(g, 0, sizeof(g));
		int a, b;
		for(i = 1; i <= m; ++ i) {
			scanf("%d %d", &a, &b);
			g[ a ][ b ] = 1; g[ b ][ a ] = 1;
		}
		memset(label, 0, sizeof(label));
		memset(weight, 0, sizeof(weight));
```

```
            for(i = n; i >= 1; -- i) {
                int ans = -1, ansj;
                for(j = 1; j <= n; ++ j) {
                    if (!label[j] && weight[ j ] > ans) {
                   ans = weight[j];
                   ansj = j;
                    }
                }
            for(j = 1; j <= n; ++ j) {
                    if (g[ ansj ][ j ]) {
                         weight[ j ] ++;
                    }
                }
                label[ ansj ] = i;
            op_label[ i ] = ansj;
            }
        bool flag = true;
        for(i = 1; i <= n; ++ i) {
            int x = op_label[ i ];
            int st = i + 1;
                for(j = i + 1; j <= n; ++ j) {
                    if (g[ x ][op_label[ j ]]) {
                 st = j;
                 break;
                }
        }
            for(j = st + 1;j <= n; ++ j) {
    if(g[ x ][op_label[ j ]] && !g[op_label[ st ]][op_label[ j ]])
                  flag = false;
            }
                if (!flag) break;
    }
        if (flag) printf("Perfect\n");
        else printf("Imperfect\n");
        printf("\n");
    }
    return 0;
}
```

/*==*\
| 稳定婚姻问题 O(n^2)
==/

```
const int N = 1001;
struct People{
        bool state;
        int opp, tag;
        int list[N];                    // man 使用
        int priority[N]; // woman 使用, 有必要的话可以和 list 合并,
以节省空间
        void Init(){ state = tag = 0; }
}man[N], woman[N];
struct R{
         int opp; int own;
}requst[N];
int n;
void Input(void);
void Output(void);
void stableMatching(void);
int main(void){
//...
    Input();
    stableMatching();
    Output();
//...
   return 0;
}
void Input(void){
    scanf("%d\n", &n);
    int i, j, ch;
    for( i=0; i < n; ++i ) {
        man[i].Init();
        for( j=0; j < n; ++j ){         //按照 man 的意愿递减排序
            scanf("%d", &ch); man[i].list[j] = ch-1;
        }
    }
   for( i=0; i < n; ++i ) {
            woman[i].Init();
            for( j=0; j < n; ++j ){       //按照 woman 的意愿递减排序,
但是, 存储方法与 man 不同! ! ! !
    scanf("%d", &ch); woman[i].priority[ch-1] = j;
            }
    }
}
void stableMatching(void){
    int k;
    for( k=0; k < n; +k ){
        int i, id = 0;
    for( i=0; i < n; ++i )
        if( man[i].state == 0 ){
          requst[id].opp =man[i].list[ man[i].tag ];
          requst[id].own = i;
          man[i].tag += 1; ++id;
        }
      if( id == 0 ) break;
for( i=0; i < id; ++i ){
    if( woman[requst[i].opp].state == 0 ){
       woman[requst[i].opp].opp =requst[i].own;
       woman[requst[i].opp].state = 1;
       man[requst[i].own].state = 1;
       man[requst[i].own].opp =requst[i].opp;
    }
   else{
if( woman[requst[i].opp].priority[ woman[requst[i].opp].opp ] >
woman[requst[i].opp].priority[requst[i].own] ){ //
        man[ woman[requst[i].opp].opp ].state = 0;
        woman[ requst[i].opp ].opp =
        requst[i].own;
        man[requst[i].own].state = 1;
        man[requst[i].own].opp =requst[i].opp;
        }
     }
   }
  }
}
void Output(void){
for( int i=0; i < n; ++i ) printf("%d\n", man[i].opp+1);
}
```

/*==*\
| 拓扑排序
| INIT:edge[][]置为图的邻接矩阵;count[0…i…n-1]:顶点 i 的入度.
==/

```
void TopoOrder(int n){
    int i, top = -1;
    for( i=0; i < n; ++i )
    if( count[i] == 0 ){              // 下标模拟堆栈
        count[i] = top; top = i;
     }
    for( i=0; i < n; ++i )
       if( top == -1 ) { printf("存在回路\n"); return ; }
       else{
          int j = top; top = count[top];
             printf("%d", j);
    for( int k=0; k < n; ++k )
      if( edge[j][k] && (--count[k]) == 0 ){
         count[k] = top; top = k;
       }
    }
}
```

/*==*\
| Floyd 求最小环
==/

朴素算法

令 e(u,v)表示 u 和 v 之间的连边, 令 min(u,v)表示删除 u 和 v 之间的连边之后 u 和 v 之间的最短路, 最小环则是 min(u, v) + e(u, v). 时间复杂度是 O(EV^2).

改进算法

在 floyd 的同时, 顺便算出最小环

```
g[i][j]=i, j 之间的边长
dist:=g;
for k:=1 to n do
begin
    for i:=1 to k-1 do
        for j:=i+1 to k-1 do
```

```
            answer:=min(answer, dist[i][j]+g[i][k]+g[k][j]);
      for i:=1 to n do
          for j:=1 to n do
              dist[i][j]:=min(dist[i][j],dist[i][k]+dist[k][j]);
end;
```

最小环改进算法的证明

一个环中的最大结点为 k(编号最大)，与他相连的两个点为 i, j，这个环的最短长度为 g[i][k]+g[k][j]+i 到 j 的路径中所有结点编号都小于 k 的最短路径长度．根据 floyd 的原理，在最外层循环做了 k-1 次之后，dist[i][j] 则代表了 i 到 j 的路径中所有结点编号都小于 k 的最短路径

综上所述,该算法一定能找到图中最小环.

```
const int INF = 1000000000;
const int N = 110;
int n, m; // n:节点个数, m:边的个数
int g[N][N]; // 无向图
int dist[N][N]; // 最短路径
int r[N][N]; // r[i][j]: i 到 j 的最短路径的第一步
int out[N], ct; // 记录最小环
int solve(int i, int j, int k){// 记录最小环
   ct = 0;
   while ( j != i ){
      out[ct++] = j;
      j = r[i][j];
    }
  out[ct++] = i; out[ct++] = k;
  return 0;
}
int main(void){
    while( scanf("%d%d", &n, &m) != EOF ){
        int i, j, k;
        for ( i=0; i < n; i++ )
            for ( j=0; j < n; j++ ){
                g[i][j] = INF; r[i][j] = i;
            }
        for ( i=0; i < m; i++ ){
             int x, y, l;
             scanf("%d%d%d", &x, &y, &l);
            --x; --y;
            if ( l < g[x][y] ) g[x][y] = g[y][x] = l;
        }
   memmove(dist, g, sizeof(dist));
   int Min = INF;                      // 最小环
   for ( k=0; k < n; k++ ) {           //Floyd
        for ( i=0; i < k; i++ )        // 一个环中的最大结点为 k(编号最大)
          if ( g[k][i] < INF )
            for ( j=i+1; j < k; j++ )
               if ( dist[i][j] < INF && g[k][j]
                  < INF && Min > dist[i][j]+g[k][i]+g[k][j] ){
                      Min =dist[i][j]+g[k][i]+g[k][j];
          solve(i, j, k);              // 记录最小环
      }
  for ( i=0; i < n; i++ )
      if ( dist[i][k] < INF )
     for ( j=0; j < n; j++ )
        if ( dist[k][j] < INF &&dist[i][j] > dist[i][k]+dist[k][j] ){
            dist[i][j] =dist[i][k]+dist[k][j];
        r[i][j] = r[k][j];
     }
}
  if ( Min < INF ){
     for ( ct--; ct >= 0; ct-- ){
        printf("%d", out[ct]+1);
          if ( ct ) printf(" ");
    }
}
    else printf("No solution.");
    printf("\n");
  }
  return 0;
}
/*==================================================*\
| 2-sat 问题
* N 个集团，每个集团 2 个人，现在要想选出尽量多的人，
* 且每个集团只能选出一个人。如果两人有矛盾，他们不能同时被选中
* 问最多能选出多少人
\*==================================================*/
const int MAXN=3010;
int n,m;
int g[3010][3010],ct[3010],f[3010];
int x[3010],y[3010];
int prev[MAXN], low[MAXN], stk[MAXN], sc[MAXN];
int cnt[MAXN];
int cnt0, ptr, cnt1;
void dfs(int w){
     int min(0);
     prev[w] = cnt0++;
     low[w] = prev[w];
     min = low[w];
     stk[ptr++] = w;
     for(int i = 0; i < ct[w]; ++i){
        int t = g[w][i];
        if(prev[t] == -1)
        dfs(t);
        if(low[t] < min)
            min = low[t];
      }
        if(min < low[w]){
           low[w] = min;
           return;
          }
 do{
     int v = stk[--ptr];
     sc[v] = cnt1;
     low[v] = MAXN;
 }while(stk[ptr] != w);
  ++cnt1;
}
void Tarjan(int N){
//传入 N 为点数，结果保存在 sc 数组中，同一标号的点在同一个强连通分量内，
//强连通分量数为 cnt1
     cnt0 = cnt1 = ptr = 0;
     int i;
    for(i = 0; i < N; ++i)
        prev[i] = low[i] = -1;
    for(i = 0; i < N; ++i)
        if(prev[i] == -1)
      dfs(i);
  }
 int solve(){
 Tarjan(n);
  for (int i=0;i<n;i++){
     if (sc[i]==sc[f[i]]) return 0;
 }
 return 1;
}
int check(int Mid){
     for (int i=0;i<n;i++)
        ct[i]=0;
     for (int i=0;i<Mid;i++){
          g[f[x[i]]][ct[f[x[i]]]++]=y[i];
          g[f[y[i]]][ct[f[y[i]]]++]=x[i];
      }
return solve();
}
int main(){
         while (scanf("%d%d",&n,&m)!=EOF && n+m){
            for (int i=0;i<n;i++){
                int p,q;
                scanf("%d%d",&p,&q);
                  f[p]=q,f[q]=p;
            }
          for (int i=0;i<m;i++) scanf("%d%d",&x[i],&y[i]);
                n*=2;
               int Min=0,Max=m+1;
               while (Min+1<Max){
                   int Mid=(Min+Max)/2;
                   if (check(Mid)) Min=Mid;
                   else Max=Mid;
               }
     printf("%d\n",Min);
 }
```

```
return 0;
}
/*==================================================*\
| 最少染色数
\*==================================================*/
#include <iostream>
using namespace std;
#define MAXV 10001
#define MAXE 1000000
#define INF 1000000000
typedef struct {
        int v;
        int next;
}Edge;
Edge e[MAXE];
int eid, p[MAXV];
int a[MAXV], c[MAXV], l[MAXV], ans;
int n, m;
bool f[MAXV];
void add(int u, int v) {
        e[ eid ].next = p[ u ];
        e[ eid ].v = v;
        p[ u ] = eid ++;
}
void init() {
        memset(p, -1, sizeof(p));
        ans = eid = 0;
}
void color() {
         int i, j, k;
        for(i = 1; i <= n; ++ i) l[ i ] = 0;
        for(i = 0; i < n; ++ i) {
                k = 1;
            for(j = 2; j <= n; ++ j) if (l[ j ] > l[ k ]) k = j;
                for(j = p[ k ]; j != -1; j = e[ j ].next) l[e[ j ].v] ++;
            a[n-i-1] = k; l[ k ] = -INF; c[i + 1] = 0;
        }
        for(i = n - 1; i >= 0; -- i) {
                for(j = 1; j <= n; ++ j) f[ j ] = false;
            int x = a[ i ];
                for(j = p[ x ]; j != -1; j = e[ j ].next) {
                        int v = e[ j ].v;
                        if(c[ v ]) f[c[ v ]] = true;
                }
            for(k = 1; f[ k ]; ++ k);
                c[ x ] = k;
                if (k > ans) ans = k;
        }
}
int main() {
        int u, v;
        while(~scanf("%d %d", &n, &m)) {
                init();
                while (m --) {
                        scanf("%d %d", &u, &v);
                  add(u, v);
                        add(v, u);
                }
                color();
                printf("%d\n", ans);
        }
      return 0;
}
/*==================================================*\
| 仙人掌图的判定(未验证)
|仙人掌图的判定
|如果一个有向图:
|1. 它是一个强连通图。
|2. 它的任意一条边都属于且仅属于一个环。
|性质 1 仙人掌图的 DFS 树没有横向边
|性质 2 low[v] <= dfs[u](v 是 u 的儿子)
|性质 3 设某点 u 有 a(u)个儿子的 low 值小于 dfs(u),
|          同时 u 自己有 b(u)条逆向边
\*==================================================*/
#include <iostream>
#include <cstring>
#include <cstdio>
#include <stack>
using namespace std;
#define MAXV 100009
#define MAXE 300001
#define MIN(a, b) a<b?a:b
typedef struct {
        int v;
        int next;
        bool vist;
}Edge;
Edge e[MAXE];
stack<int> S;
int dfn[MAXV], low[MAXV];
int p[MAXV], n, a[MAXV], b[MAXV];
int cnt, eid, index;
bool flag, instack[MAXV];
void add(int u, int v) {

        e[ eid ].next = p[ u ];
        e[ eid ].vist = false;
        e[ eid ].v = v;
        p[ u ] = eid ++;
}
void init() {
        eid = cnt = index = 0;
        memset(dfn, -1, sizeof(dfn));
        memset(low, -1, sizeof(low));
        memset(a, 0, sizeof(a));
        memset(b, 0, sizeof(b));
        memset(instack, false, sizeof(instack));
}
void tarjan(int x) {
      dfn[ x ] = low[ x ] = index ++;
      S.push(x);
      instack[ x ] = true;
      for(int i = p[ x ]; i != -1; i = e[ i ].next) {
                int v = e[ i ].v;
                if(dfn[ v ] == -1) {
                  tarjan(v);
                  low[ x ] = MIN(low[ x ], low[ v ]);
            }
            else if(instack[ v ])
                  low[ x ] = MIN(low[ x ], dfn[ v ]);
      }
      if(low[ x ] == dfn[ x ]) {
            while(true) {
                  int tmp = S.top();
                  S.pop();
                  instack[ tmp ] = false;
                        if(tmp == x) break;
            }
            cnt ++;
                if(cnt > 1) flag = false;
      }
}
void dfscute(int u) {
      dfn[ u ] = low[ u ] = ++index;
      for(int i = p[ u ]; i != -1; i = e[ i ].next) {
                if(e[ i ].vist) continue;
                e[ i ].vist = true;
                int v = e[ i ].v;
            if(dfn[ v ] == -1) {
                  dfscute(v);
                        if(low[ u ] > low[ v ]) {
                              low[ u ] = low[ v ];
                              a[ u ] ++;
                        }
                  if(low[ v ] > dfn[ u ] || a[ u ] + b[ u ] >= 2) {
                              flag = false;
                              return ;
                  }
                }
                else {

                        low[ u ] = MIN(low[ u ], dfn[ v ]);
```

```
                b[ u ] ++;
                if(a[ u ] + b[ u ] >= 2) {
                    flag = false;
                    return ;
                }
            }
        }
}
int main() {
    int n, m;
    int u, v;
    int i;
    while(cin >> n >> m) {
        flag = true;
        memset(p, -1, sizeof(p));
        init();
        while(m --) {

            cin >> u >> v;
            add(u, v);
        }
        for(i = 1; i <= n; ++ i) if(dfn[ i ] == -1) tarjan(i);
        init();
        for(i = 1; i <= n; ++ i) if(dfn[ i ] == -1)    dfscute(i);
        if(flag) puts("YES");
        else        puts("NO");
    }
    return 0;
}
/*==================================================*\
|求树中 2 点的距离
\*==================================================*/
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
#define MAXV 40001
#define INF 1000000000
typedef struct {
    int v;
    int val;
    int next;
}Edge;
Edge e[MAXV*2], q[MAXV*2];
int eid, qid;
int pe[MAXV], pq[MAXV], ans[MAXV*2];
bool vist[MAXV];
int dis[MAXV];
int fa[MAXV];
void init() {
    memset(pe, -1, sizeof(pe));
    memset(pq, -1, sizeof(pq));
    memset(vist, false, sizeof(vist));
    memset(fa, -1, sizeof(fa));
    eid = qid = 0;
}
void add(int u,int v,int val) {

    e[ eid ].next = pe[ u ];
    e[ eid ].v = v;
    e[ eid ].val = val;
    pe[ u ] = eid ++;
}
void addQ(int u,int v,int val) {
    q[ qid ].next = pq[ u ];
    q[ qid ].v = v;
    q[ qid ].val = val;
    pq[ u ] = qid ++;
}
int find(int x) {
    if(fa[ x ] == -1) return x;
    return fa[ x ] = find(fa[ x ]);
}
void tarjan(int u,int curval) {
    int i, v, val;
    vist[ u ] = true;
    dis[ u ] = curval;
    for(i = pq[ u ]; i != -1; i = q[ i ].next) {
        v = q[ i ].v;
            val = q[ i ].val;
            if(vist[ v ]) ans[ val ] = dis[ u ] + dis[ v ] - dis[find(v)] * 2;
    }
    for(i = pe[ u ]; i != -1; i = e[ i ].next) {
            v = e[ i ].v;
            val = e[ i ].val;
        if(!vist[ v ]) {
            tarjan(v, curval + val);
            fa[ v ] = u;
        }
    }
}
int main() {
    int n, m, i, k;
    int u, v, val;
    char dir[10];
    while(~scanf("%d %d", &n, &m)) {
            init();
        while(m --) {
            scanf("%d %d %d %s", &u, &v, &val, dir);
            add(u, v, val);
            add(v, u, val);
        }
            scanf("%d", &k);
        for(i = 1; i <= k; ++ i) {
            scanf("%d %d", &u, &v);
            addQ(u, v, i);
            addQ(v, u, i);
        }
        tarjan(1, 0);
        for(i = 1; i <= k; ++ i) {
            printf("%d\n", ans[ i ]);
        }
        printf("\n");
    }
    return 0;
}
/*==================================================*\
|精确覆盖问题
|可以适合 N*N 的任意大小数独问题，其中 nil 表示未填写部分的 ascii，
|flag 表示最小数字 ascii，所有输入必须转化到 grid[i][j]中。
\*==================================================*/
#include <iostream>
using namespace std;
 const int N=9;//规模大小
 const int Per=3;//sqrt(N);
 char grid[N+2][N+2];//存储格
 char res[N+2][N+2];//答案
 const int MAXR=N*N*N;
 const int MAXC=N*N*4;
 const int MAXLEN=4*N*MAXC+MAXC+MAXR;
  const int st[4]={0,N*N,2*N*N,3*N*N};
 int L[MAXLEN];
 int R[MAXLEN];
 int D[MAXLEN];
 int U[MAXLEN];
 int nRow[MAXLEN];
 int nCol[MAXLEN];
 int Col[MAXC];                  //判定列集是否已插入
 int Row[MAXR];                  //判定行集是否已插入
 //int RC[MAXR][MAXC];           //判定元素是否已插入
 int RS[MAXR],CS[MAXC];          //行长与列长
 int eid;                        //内存标识
 int head;
 int Cn;
 int ans[MAXR];
 int alen;
 char flag;
 char nil;
 //DLX 算法  进行精确覆盖  判定前请先判断是否各列中都有 1 存在
 //对于行列唯一的情况  可以考虑将 RC 数组取消以加速
 inline void init(){
     memset(Row,-1,sizeof(Row));
```

```
	memset(Col,-1,sizeof(Col));
//	memset(RC,-1,sizeof(RC));
	memset(nCol,-1,sizeof(nCol));
	memset(nRow,-1,sizeof(nRow));
	head=0;
	L[head]=R[head]=D[head]=U[head]=head;
	eid=1;
	Cn=0;
}
				//插入行
inline void insRow(int r){
	U[D[head]]=eid;
	U[eid]=head;
	D[eid]=D[head];
	D[head]=eid;
	L[eid]=R[eid]=eid;
	RS[r]=1;
	Row[r]=eid++;
}
				//插入列
inline void insColumn(int c)
 {
	L[R[head]]=eid;
	L[eid]=head;
	R[eid]=R[head];
	R[head]=eid;
	U[eid]=D[eid]=eid;
	CS[c]=1;
	Col[c]=eid++;
}
//插入元素
inline void insElement(int r,int c) {
	int rid=Row[r];
	int cid=Col[c];
	L[R[rid]]=eid;
	L[eid]=rid;
	R[eid]=R[rid];
	R[rid]=eid;
	U[D[cid]]=eid;
	U[eid]=cid;
	D[eid]=D[cid];
	D[cid]=eid;
	nRow[eid]=r;
	nCol[eid]=c;
	++CS[c];
	++RS[r];
//	RC[r][c]=eid;
	++eid;
}
//插入操作
inline void insert(int r, int c) {
	if (Col[c]==-1){
		++Cn;
		insColumn(c);
	}
	if(Row[r]==-1){
		insRow(r);
	}
//	if(RC[r][c]==-1){
		insElement(r,c);
	}
}
//删除列(使用 cid)
inline void RemoveCol(int c){
	//c=Col[c];
	L[R[c]]=L[c];
	R[L[c]]=R[c];
	int i,j;
	for (i=D[c];i!=c;i=D[i]){
		for (j=R[i];j!=i;j=R[j]){
			if(nCol[j]==-1) continue;
			U[D[j]]=U[j];
			D[U[j]]=D[j];
			--CS[nCol[j]];
		}
	}
```

```
}
//恢复列(使用 cid)
inline void ResumeCol(int c){
	//c=Col[c];
	int i,j;
	for (i=U[c];i!=c;i=U[i]) {
		for (j=L[i];j!=i;j=L[j]) {
			if(nCol[j]==-1) continue;
			++CS[nCol[j]];
			U[D[j]]=j;
			D[U[j]]=j;
		}
	}
	L[R[c]]=c;
	R[L[c]]=c;
}
//精确覆盖
inline bool dfs(int k){
	if (R[head]==head){
		alen=k;
		return true;
	}
	int i,j;
	int s=INT_MAX;
	int c;
	for (i=R[head];i!=head;i=R[i]){
		 if(nCol[D[i]]==-1) {c=i;continue;}
		if (CS[nCol[D[i]]]<s){
			s=CS[nCol[D[i]]];
			c=i;
		}
	}
	RemoveCol(c);
	for (i=U[c];i!=c;i=U[i]){
		ans[k]=nRow[i];
		for (j=L[i];j!=i;j=L[j]){
			if (nCol[j]==-1){
				continue;
			}
			RemoveCol(Col[nCol[j]]);
		}
		if(dfs(k+1)){
			return true;
		}
		for (j=R[i];j!=i;j=R[j]){
			if (nCol[j]==-1){
				continue;
			}
			ResumeCol(Col[nCol[j]]);
		}
	}
	ResumeCol(c);
	return false;
}
inline int BoxNum(int i,int j){
	return (i/Per)*Per+j/Per;
}
inline void Sudoku(){
	init();
	int i,j,k;
	int R,C;
	for (i=0;i<N;++i){
		for (j=0;j<N;++j){
			int ind=N*i+j;
			int BN=BoxNum(i,j);
			if (grid[i][j]==nil){
				for (k=0;k<N;++k){
					R=N*N*k+N*i+j;
					C=st[0]+ind;
					insert(R,C);
					C=st[1]+N*i+k;
					insert(R,C);
					C=st[2]+N*j+k;
					insert(R,C);
					C=st[3]+N*BN+k;
					insert(R,C);
```

```
                }
            }
            else{
                R=N*N*(grid[i][j]-flag)+N*i+j;
                C=st[0]+ind;
                insert(R,C);
                C=st[1]+N*i+(grid[i][j]-flag);
                insert(R,C);
                C=st[2]+N*j+(grid[i][j]-flag);
                insert(R,C);
                C=st[3]+N*BN+(grid[i][j]-flag);
                insert(R,C);
            }
        }
    }
    if (Cn==MAXC&&dfs(0)){
        int r,c,k;
        for(i=0;i<alen;++i){
            c=ans[i]%N;
            ans[i]/=N;
            r=ans[i]%N;
            ans[i]/=N;
            k=ans[i];
            res[r][c]=k+flag;
        }
        for (i=0;i<N;++i){
            for (j=0;j<N;++j){
                putchar(res[i][j]);
            }
            puts("");
        }
    }
    else{
        puts("Could not complete this grid.");
    }
}
```

目　录:

Chapter 2

Number Theory

2.1 公式

/*==*\
|打表:
|当打表时，数组都存不下来的时候，也许有循环，找循环看看，若有，
|就打出一个循环就好了
|要是数组存的下来，就一次性打好，记的要放在循环外面打好，不要因
|为这个小错误而超时
==/

/*==*\
|划分问题:
==/

1.n 个点最多把直线分成 C(n,0)+C(n,1)份;
2.n 条直线最多把平面分成 C(n,0)+C(n,1)+C(n,2)份;
3.n 个平面最多把空间分成 C(n,0)+C(n,1)+C(n,2)+C(n,3)=(n³+5n+6)/6 份;
4.n 个空间最多把"时空"分成 C(n,0)+C(n,1)+C(n,2)+C(n,3)+C(n,4)份.
5. 一个平面上有一个圆和 n 条直线，这些直线中每一条在圆内同其他直线相交，假设没有 3 条直线相交于一点，这些直线将圆分成 F(n) = n(n+1)/2 +1 区域。
6. 平面上有 n 条折线，这些折线最多能将平面分割成 F(n)=2 n^2 – n + 1 块
7. 平面上有 n 个"Z"，平面最多可以分割为 9*n*(n-1)/2+n+1 部分
8. 有 n 条封闭曲线画在平面上，而任何两条封闭曲线恰好相交于两点，且任何三条封闭曲线不相交于同一点，这些封闭曲线把平面分割成的区域个数 F(1)=2，F(n)=F(n-1)+2(n-1)

/*==*\
| Stirling 公式
==/

lim(n→∞) √(2πn) * (n/e)^n = n!

也就是说当 n 很大的时候,n!与 √(2πn) * (n/e) ^ n 的值十分接近

这就是 Stirling 公式. (2πn) ^0.5× n^ n × e^(-n) =n!;

lnN!=NlnN-N +0.5*ln(2*N*pi)　　log10(N!)=lnN!/ln(10)

/*==*\
| 皮克 pick 定理
==/

一个多边形的顶点如果全是格点，这多边形就叫做格点多边形。有趣的是，这种格点多边形的面积计算起来很方便，只要数一下图形边线上的点的数目及图内的点的数目，就可用公式算出。
给定顶点坐标均是整点（或正方形格点）的简单多边形，皮克定理说明了其面积 S 和内部格点数目 a、边上格点数目 b 的关系：
S=a+ b/2 - 1。
(其中 a 表示多边形内部的点数,b 表示多边形边界上的点数,S 表示多边形的面积)

/*==*\
| catalan 卡特兰数
==/

原理:
令 h(1)＝1,h(0)=1，
数满足递归式：
h(n)= h(0)*h(n-1)+h(1)*h(n-2) + ... + h(n-1)h(0) (其中 n>=2)
另类递归式：
h(n) = h(n-1)*(4*n-2)/(n+1);
该递推关系的解为：
h(n)=C(2n,n)/(n+1) (n=1,2,3,...)
卡特兰数的应用
（实质上都是递归等式的应用）

/*==*\
| 错排公式
==/

当 n 个编号元素放在 n 个编号位置,元素编号与位置编号各不对应的方法数用 M(n)表示,那么 M(n-1)就表示 n-1 个编号元素放在 n-1 个编号位置,各不对应的方法数,其它类推.
第一步,把第 n 个元素放在一个位置,比如位置 k,一共有 n-1 种方法;
第二步,放编号为 k 的元素,这时有两种情况.1,把它放到位置 n,那么,对于剩下的 n-2 个元素,就有 M(n-2)种方法;2,不把它放到位置 n,这时,对于这 n-1 个元素,有 M(n-1)种方法;

综上得到递推公式：
M(n)=(n-1)[M(n-2)+M(n-1)]
特殊地，M(1)=0,M(2)=1
通项公式：
M(n)=n![(-1)^2/2!+…+(-1)^(n-1)/(n-1)!+(-1)^n/n!]
优美的式子：
Dn=[n!/e+0.5],[x]为取整函数.
公式证明较简单. 观察一般书上的公式，可以发现 e-1 的前项与之相同，然后作比较可得/Dn-n!e-1/<1/(n+1)<0.5,于是就得到这个简单而优美的公式（此仅供参考）

/*==*\
==/

1、全错排：
a[n]=(a[n-1]+a[n-2])*(n-1);
特殊：a[0]=a[1]=0;a[2]=1;a[3]=2;
注意：排列的数很大，用 double 或_int64;

2、部分错排：
N 个数 M 个错排；
X[N][M]=C(N,M)*a[M];
解释：先从 N 个数中取 M 个数，然后对 M 个数全错排；
例题：HUTCOJ 1249；

```
#include<stdio.h>
long zuheshu(long n,long m)
{
    long i,j,k;
    k=1;
    for(i=n;i>n-m;i--){
        k*=i;
    }
    for(i=m;i>1;i--){
        k/=i;
    }
    return k;
}
int main(){
    __int64 a[60];
    long i,j,n,m,k,l;
    a[0]=a[1]=0;
    a[2]=1;
    a[3]=2;
    for(i=4;i<60;i++){
        a[i]=(a[i-2]+a[i-1])*(i-1);
    }
    while(scanf("%d%d",&n,&m)!=EOF){
        printf("%I64d\n",zuheshu(n,m)*a[m]);
    }
}
```

/*==*\
| 等比数列
==/

(1) 等比数列：
a (n+1)/an=q　(n∈N)。
(2) 通项公式：
an=a1×q^(n-1)；
推广式：
an=am×q^(n-m)；
(3) 求和公式：
Sn=n*a1 (q=1)
Sn=a1(1-q^n)/(1-q) =(a1-an*q)/(1-q) (q≠1) (q 为比值,n 为项数)
(4)性质：
①若 m、n、p、q∈N，且 m＋n=p＋q，则 am*an=ap*aq；
②在等比数列中，依次每 k 项之和仍成等比数列.
③若 m、n、q∈N，且 m+n=2q，则 am*an=aq^2
(5)"G 是 a、b 的等比中项""G^2=ab（G ≠ 0）".
(6)在等比数列中，首项 a1 与公比 q 都不为零.

注意：上述公式中 an 表示等比数列的第 n 项。

/*==*\
等差数列
==/

1.Sn=n(a1+an)/2
2.Sn=a1*n+n(n-1)d/2
an=a1+(n-1)d
Sn＝（a1＋an）*n/2
项数＝（末项-首项）÷公差＋1
A1=2*S/n-an
an=2*S/n-a1
an=a1+（n-1）*d;
性质：
若 m、n、p、q∈N
① m＋n=p＋q，则 am+an=ap+aq
②若 m+n=2q，则 am+an=2aq
注意：上述公式中 an 表示等差数列的第 n 项。

/*==*\
二次函数
==/

定义与定义表达式
1：一般式：
y=ax^2;+bx+c　(a≠0，a、b、c 为常数)
对称轴为直线 x = -b/2a，顶点坐标(-b/2a,(4ac-b^2)/4a)。

2：顶点式：
y=a(x-h）^2+k　或　y=a(x+m)^2+k
(两个式子实质一样,但初中课本上都是第一个式子)（若给出抛物线的顶点坐标或对称轴与最值，通常可设顶点式）

3：交点式（与 x 轴）：
y=a(x-x1)(x-x2)
（若给出抛物线与 x 轴的交点及对称轴与 x 轴的交点距离或其他一的条件，通常可设交点式）
重要概念：
（a，b，c 为常数，a≠0，且 a 决定函数的开口方向，a>0 时，开口方向向上，a<0 时，开口方向向下。a 的绝对值还可以决定开口大小,a 的绝对值越大开口就越小,a 的绝对值越小开口就越大。）

$$y=ax^2+bx+c=a\left(x^2+\frac{b}{a}x+\frac{c}{a}\right)$$

$$=a\left[x^2+2\times\frac{b}{2a}x+\left(\frac{b}{2a}\right)^2-\left(\frac{b}{2a}\right)^2+\frac{c}{a}\right]$$

$$=a\left[\left(x+\frac{b}{2a}\right)^2+\frac{4ac-b^2}{4a^2}\right]$$

$$=a\left(x+\frac{b}{2a}\right)^2+\frac{4ac-b^2}{4a}$$

/*==*\
二次函数上任意三点坐标求面积　HDU 1071
==/

```
#include<iostream>
#include<cstdio>
using namespace std;
int main()
{
    int n;
    double x1,x2,x3,y1,y2,y3,a,b,c,k,m,area;
    cin>>n;
    while(n--){
        cin>>x1>>y1;
        cin>>x2>>y2;
        cin>>x3>>y3;
        a=(y2-y1)/((x1-x2)*(x1-x2));
        b=-2*x1*(y2-y1)/((x1-x2)*(x1-x2));

c=y1-x1*x1*(y2-y1)/((x1-x2)*(x1-x2))+2*x1*x1*(y2-y1)/((x1-x2)*(x1-x2));
        k=(y3-y2)/(x3-x2);
        m=y2-(y3-y2)/(x3-x2)*x2;

area=(a*x3*x3*x3/3+(b-k)*x3*x3/2+(c-m)*x3)-(a*x2*x2*x2/3+(b-k)*x2*x2/2+(c-m)*x2);
        printf("%.2lf\n",area);
    }
}
```

/*==*\
二次方程
==/

a*x+b*y+c=0;
当△<0 ，方程无解;
当△=0 ，x1=x2= -b/(2*a);
当△>0.
x1=[-b-sqrt(b*b-4*a*c)]/(2*a),　x2=[-b+ sqrt(b*b-4*a*c)]/(2*a)。

/*==*\
均值不等式
==/

概念：
1、调和平均数：
Hn=n/(1/a1+1/a2+...+1/an)
2、几何平均数：
Gn=(a1a2...an)^(1/n)=n 次√(a1*a2*a3*...*an)
3、算术平均数：
An=(a1+a2+...+an)/n
4、平方平均数：
Qn=√ [(a1^2+a2^2+...+an^2)/n]
5、这四种平均数满足:
Hn≤Gn≤An≤Qn
a1、a2、… 、an∈R +，当且仅当 a1=a2= … =an 时取“=”号
均值不等式的一般形式：
设函数 D(r)=[（a1^r+a2^r+...an^r）/n]^(1/r)　(当 r 不等于 0 时);
(a1a2...an)^(1/n)(当 r=0 时）(即 D(0)=(a1a2...an)^(1/n)）
则有：当 r<s 时，D(r)≤D(s)
注意到 Hn≤Gn≤An≤Qn 仅是上述不等式的特殊情形，即 D(-1)≤D(0)≤D(1)≤D(2)

/*==*\
均值不等式变形
==/

(1)对实数 a,b，有 a^2+b^2≥2ab (当且仅当 a=b 时取“=”号)，a^2+b^2>0>-2ab
(2)对非负实数 a,b，有 a+b≥2√(a*b)≥0，即(a+b)/2≥√(a*b)≥0
(3)对负实数 a,b，有 a+b<0<2√(a*b)
(4)对实数 a,b，有 a(a-b)≥b(a-b)
(5)对非负数 a,b，有 a^2+b^2≥2ab≥0
(6)对非负数 a,b，有 a^2+b^2 ≥1/2*(a+b)^2≥ab
(7)对非负数 a,b,c，有 a^2+b^2+c^2≥1/3*(a+b+c)^2
(8)对非负数 a,b,c，有 a^2+b^2+c^2≥ab+bc+ac
(9)对非负数 a,b，有 a^2+ab+b^2≥3/4*(a+b)^2
(10)对实数 a,b,c，有(a+b+c)/3>=(abc)^(1/3)
(11) a^3+b^3+c^3>=3abc,a、b、c 都是正数。
扩展：若有 y=x1*x2*x3.....Xn 且 x1+x2+x3+...+Xn=常数 P,则 Y 的最大值为((x1+x2+x3+.....+Xn)/n)^n
1、| |a|-|b| |≤|a-b|≤|a|+|b|
2、| |a|-|b| |≤|a+b|≤|a|+|b|
设 a1,a2,…an；b1,b2…bn 均是实数，且 a1≥a2≥a3≥…≥an,b1≥b2≥b3≥…≥bn;则有
a1b1+a2b2+…+anbn（顺序和）≥a1b2+a2b1+a3b3+…+aibj+…+anbm（乱序和）≥a1bn+a2bn-1+a3bn-2+…+anb1（逆序和）,仅当 a1=a2=a3=…an,b1=b2=b3=…=bn 时等号成立。

/*==*\
蚂蚁爬绳
==/

一绳长 L 米，一蚂蚁从绳的一端爬向另一端，速度为每秒 v m/s，同时，绳子以每秒 u 米的速度均匀伸长，问:蚂蚁能否达到绳的另一端？如能，需多长时间？如不能，请说明理由。（假设绳子质量无限好，蚂蚁寿命无限长）**T=[e^(u/v)-1]*L/u;**

/*==*\
一些公式
==/

(1)1 + 3 + 5 + ... + (2n - 1) = n^2
(2)1*2 + 2*3 + 3*4 + ... + n*(n + 1) = n*(n + 1)*(n + 2) / 3
(3)1*1! + 2*2! + 3*3! + ... + n*n! = (n + 1)! - 1
(4)1^2 + 2^2 + 3^2 + ... + n^2 = n*(n + 1)*(2n + 1) / 6
(5)1^2 - 2^2 + 3^2 -... + (-1)^n * n^2 = (-1)^(n + 1) * n * (n + 1) / 2

(6)2^2 + 4^2 + ... + (2n)^2 = 2n*(n+1)*(2n+1) / 3
(7)1/2! + 2/3! + ... + n/(n+1)! = 1 - 1/(n+1)!
(8)2^(n + 1) < 1 + (n + 1)2^n
(9)1^3 + 2^3 + 3^3 + ... + n^3 = (n*(n + 1) / 2)^2
(10)1/(2*4)+1*3/(2*4*6)+1*3*5/(2*4*6*8)+...+(1*3*5*...*(2n-1))/(2*4*6*...*(2n+2)) = 1/2 - (1*3*5*...*(2n+1))/(2*4*6*...*(2n+2))
(11)1/(2^2-1) + 1/(3^2-1) + .. + 1 / ((n+1)^2 - 1) = 3/4 - 1/(2*(n+1)) - 1/(2*(n+2))
(12)1/2n <= 1*3*5*...*(2n-1) / (2*4*6*...*2n) <= 1 / sqrt(n+1) n=1,2...
(13)2^n >= n^2 , n=4, 5,...
(14)2^n >= 2n + 1, n=3,4,...
(15)r^0 + r^1 + ... + r^n < 1 / (1 - r), n>=0, 0<r<1
(16)1*r^1 + 2*r^2 + ... + n*r^n < r / (1-r)^2, n>=1, 0<r<1
(17)1/2^1 + 2/2^2 + 3/2^3 + ... + n /2^n < 2, n>=1
(18) (a(1)*a(2)*...*a(2^n))^(1/2^n) <= (a(1) + a(2) + ... + a(2^n)) / 2^n, n = 1, 2, ... a(i)是正数 注:()用来标记下标
(19)cos(x) + cos(2x) + ... + cos(nx) = cos((x/2)*(n+1))*sin(nx/2) / sin(x/2), 其中 sin(x/2) != 0
(20)1*sin(x) + 2*sin(2x) + ... + n*sin(nx) = sin((n+1)*x) / (4*sin(x/2)^2) - (n+1)cos((2n + 1)/2 * x) / (2 * sin(x/2))
其中 sin(x/2) != 0

2.2 常见经典问题

```
/*==================================================*\
|N !
\*==================================================*/
/*==================================================*\
|求 1/N! = 1/X + 1/Y 解的个数
\*==================================================*/
#include <iostream>
#define MAX 10001
using namespace std;
bool prime[ MAX ] = {false};
int r[ MAX ];
int ff[ MAX ] , kf;
bool v[ MAX ];
int rank[ MAX ];
__int64 ans[MAX];
int k;
void factor (int n){
    int i;
    for (i = 2 ; i <= n ; ++ i){
        while (n % i == 0){
            n /= i;
            r[ i ] ++;
        }
        if (n == 1)
            break;
        if (prime[ n ]){
            r[ n ] ++;
            break;
        }
    }
}
int main(){
    int i , j;
    int n , g;
    __int64    tt;
    kf = 0;
    for (i = 2 ; i <= MAX ; ++ i)
        prime[ i ] = true;
    for (i = 2 ; i <= MAX ; ++ i)
        if (prime[ i ]){
            ff[++ kf] = i;
            rank[ i ] = kf;
            for (j = i + i ; j <= MAX ; j += i)
                prime[ j ] = false;
        }
        while (scanf ("%d" , &n) , n){
            g = 2;
            memset (r , 0 , sizeof (r));
            for (i = 2 ; i <= n ; ++ i){
                if (prime[ i ]){
                    r[ i ] ++;
                    if (i > g)
                        g = i;
                }
                else
                    factor (i);
            }
            k = 1;
            ans[ 1 ] = 1;
            int tmp = 0;
            for (i = 1 ; i <= rank[ g ] ; ++ i){
                tt = (r[ff[ i ]] << 1) + 1;
                for (j = 1 ; j <= k ; ++ j){
                    ans[ j ] = ans[ j ] * tt + tmp;
                    tmp = ans[ j ] / 10000 , ans[ j ] %= 10000;
                }
                while (tmp)
                    ans[++ k] = tmp % 10000 , tmp /= 10000;
            }
            printf ("%I64d" , ans[ k ]);
            for (i = k - 1 ; i > 0 ; -- i)
                printf ("%04I64d" , ans[ i ]);
            printf ("\n");
        }
        return 0;
}
/*==================================================*\
|n!的非 0 末位
\*==================================================*/
#include <iostream>
#include <cstring>
#define max 10000
using namespace std;
int lastdigit (char * str) {
    const int mod[ 20 ] = {1 , 1 , 2 , 6 , 4 , 2 , 2 , 4 , 2 , 8 , 4 , 4 , 8 ,
4 , 6 , 8 , 8 , 6 , 8 , 2};
    int len = strlen (str) , a[ max ] , i , c , ans = 1;
    if (len == 1)
        return mod[str[ 0 ] - '0'];
    for (i = 0;i < len; ++ i)
        a[ i ] = str[len - 1 - i] - '0';
    for (; len ; len -= !a[len - 1]) {
        ans = ans * mod[a[ 1 ] % 2 * 10 + a[ 0 ]] % 5;
        for (c = 0 , i = len - 1 ; i >= 0; -- i)
            c = c * 10 + a[ i ] , a[ i ] = c / 5 , c %= 5;
    }
    return ans + ans % 2 * 5;
}
int main() {
    char n[ max ];
    while (scanf ("%s",n) == 1)
        printf ("%d\n" , lastdigit(n));
    return 0;
}
/*==================================================*\
|n!的首位
\*==================================================*/
#include <iostream>
#include <cmath>
#define PI acos(-1.0)
#define e 2.7182818284590452354
#define EPS 0.0000000001
using namespace std;
int main(){
        int n;
        int ans , i;
        double tmp;
        while (scanf ("%d" , &n) == 1){
            if (n < 50){
                tmp = 1;
                for (i = 1;i <= n; ++ i)
                    tmp *= i;
                while (tmp >= 1)
                {
                    if (!(tmp/10 >= 1))
                        ans = tmp;
                    tmp /= 10;
```

```
		}
	}
	else{
	tmp = n * log10(n * 1.0 / e) + 0.5 * log10(2 * PI * n);
		ans = pow (10.0 , tmp - int(tmp)) + EPS;
	}
	printf ("%d\n" , ans);
	}
	return 0;
}
```

/*==*\
|n!的位数
==/

```
#include <stdio.h>
#include <math.h>
const double PI = acos(-1.0);
const double ln_10 = log(10.0);
//lnN!=NlnN－N+0.5ln(2N*pi)
double f(int N){
	return ceil((N*log(double(N))-N+0.5*log(2.0*N*PI))/ln_10);
}
int main(){
	int N;
	while(scanf("%d", &N) == 1) {
		if(N<=1)
			printf("1\n");
		else
			printf("%.0lf\n",f(N));
	}
	return 0;
}
```

/*==*\
|n!末尾 0 的个数
==/

```
#include <iostream>
using namespace std;
int solve (int n){
  if (n < 5)
	return 0;
  return n / 5 + solve (n / 5);
}
int main(){
  int t , n , ans;
  scanf ("%d" , &t);
  while (t --){
    scanf ("%d" , &n);
    ans = solve (n);
      printf ("%d\n" , ans);
  }
  return 0;
}
```

/*==*\
|素数
==/

/*==*\
|素数标记
|将 0-MAX_SIZE 的数标记在 prime[MAX_SIZE]数组中，当 prime[k]=0,
|表示 k 为素数，否则不是素数函数调用在输入前之前调用，节约时间
==/

```
#include <iostream>
using namespace std;
#define MAX_SIZE 100000
int prime[MAX_SIZE + 1];
void calcPrime(){
	memset(prime,0,sizeof(prime));
	int i,j;
	for(i = 0; i <= MAX_SIZE; i = i + 2)
		prime[i] = 1;
	prime[2]=0;
	prime[1]=1;
	for (i = 3; i * i <= MAX_SIZE ; i++)
		if (!prime[i]){
			for (j = i * i; j <= MAX_SIZE; j += 2 * i)
				prime[j] = 1 ;
		}
}
int main(){
calcPrime();
int m;
while(cin>>m){
	if(prime[m])
		printf("no\n");
	else printf("yes\n");
}
return 0;
}
```

/*==*\
|素数打表
| calcPrime(),用于记录从 0-MAX_SIZE 范围内的所有素数素数存放在
|prime[MAX]数组中，其中在调用函数时要估计 MAX 的范围。bj[]数组|
是用来标记 0-MAX_SIZE 范围内的素数，bj[k]=0，表示 k 是素数
==/

```
#include <iostream>
using namespace std;
#define MAX_SIZE 10010
#define MAX 2000
int bj[MAX_SIZE + 1];
int prime[MAX];
void calcPrime(){
	memset(bj,0,sizeof(bj));
	int i,j,k=0;
	for(i = 0; i <= MAX_SIZE; i = i + 2)
		bj[i] = 1;
	bj[2]=0;
	bj[1]=1;
	for (i = 3; i * i <= MAX_SIZE ; i++)
		if (!bj[i]){
			for (j = i * i; j <= MAX_SIZE; j += 2 * i)
				bj[j] = 1 ;
		}
	prime[k++]=2;
	for(i = 3; i <= MAX_SIZE; i += 2)
		if(!bj[i])
			prime[k++] = i;
}
int main(){
calcPrime();
int k;
while(cin>>k){
	cout<<bj[k]<<endl;
  }
  return 0;
}
```

/*==*\
|判断大素数
==/

```
#include <iostream>
#include <stdlib.h>
#define max 10
using namespace std;
typedef unsigned __int64 u64;
u64 prime;

u64 gcd (u64 a , u64 b){
	if (!b) return a;
	return gcd (b , a % b);
}
u64 random (){
	u64 a = rand();
	a *= rand();
	a *= rand();
	a *= rand();
	return a;
}
// a * b % n ( a, b, n < 2^54 )
u64 mod (u64 a, u64 b, u64 n){
	int i,j;
	u64 a1 = (a & 0x7ffffff);//????????27??
	u64 a2 = (a >> 27);////?????????27??
	u64 b1 = (b & 0x7ffffff);
	u64 b2 = (b >> 27);
	u64 r1 = a2 * b2;
```

```
        u64 r2 = a2 * b1;
        u64 r3 = a1 * b2;
        u64 r4 = a1 * b1;
        for (i = 0;i < 54;++ i){
                r1 <<= 1;
                if(r1 >= n)
                        r1 -= n;
        }
        for (j = 0;j < 27;++ j){
                r2 <<= 1;
                if(r2 >= n)
                        r2 -= n;
                r3 <<= 1;
                if(r3 >= n)
                        r3 -= n;
        }
        return (r1 + r2 + r3 + r4) % n;
}
u64 powmod (u64 a , u64 b , u64 n){
        u64 r=1;
        while (b){
                if ((b & 1) == 1)
                        r = mod(a , r , n);
                a = mod(a , a , n);
                b >>= 1;
        }
        return r;
}
bool Miller_Rabin (u64 n){
        int i;
        if (n == 2) return true;
        if (n < 2 || !(n & 1)) return false;
        u64 m = n - 1 , s = 0 , a , j;
        while (!(m & 1)){
                m >>= 1;
                s ++;
        }
        for (i = 0;i < max;++ i){
                a = powmod(random() % (n - 1) + 1, m , n);
                if (a == 1)
                        continue;
                for (j = 0;j < s;++ j){
                        if (a == n - 1) break;
                        a = mod(a , a , n);
                }
                if (j == s) return false;
        }
        return true;
}
u64 f(u64 x , u64 n){
        return (mod(x, x, n) + 1) % n;
}
u64 Pollard (u64 n){
        int i;
        if (n <= 2)
                return 0;//??????????
        if(!(n & 1))
                return 2;
        u64 x , y , p;
        for (i = 0;i < max;++ i){
                x = random() % n;
                y = f(x , n);
                p = gcd((y - x + n) % n , n);
                while (p == 1){
                        x = f(x , n);
                        y = f(f(y , n) , n);
                        p = gcd((y - x + n) % n , n) % n;
                }
                if(p)
                        return p;
        }
        return 0;
}
void divide (u64 n){
        if (Miller_Rabin (n)){
                if (n > prime) prime = n; //??????????????.
                        return;
        }
        u64 t = Pollard(n);
        divide (t);
        divide (n/t);
}
//2 ^ 54 = 18014398509481984,???? < 2 ^ 54;
int main(){
        u64 n;
        //0x7fffffff=2^31-1;
        //0x7ffffff 134217727=2^27-1
        while (scanf("%I64u" , &n) == 1){
                if (!n) break;

                if (Miller_Rabin(n))
                        printf ("prime\n");
                else{
                        prime = 2;
                        divide (n);
                        printf ("%I64u\n" , prime); //找出 n 的最小因子
                }
        }
        return 0;
}
```

/*==*\

|区间里的素数个数[a,b]间的 prime

==/

```
#include <stdio.h>
#include <math.h>
#include <memory.h>
#include <time.h>
typedef unsigned int uint;
#ifdef S6
#define FACT 15015 * 8 // 3 * 5 * 7 * 11 * 13 = 15015
#define ST 6
#else
#define FACT 255255      // 3 * 5 * 7 * 11 * 13 * 17 = 255255
#define ST 7
#endif
#define BLOCKSIZE (FACT * 4)
#define MOVE 4
#define MASK7 7
#define MAXN (1u << 31)
#define MASKN(n) (1 << ((n >> 1) & MASK7))
#define MAXM ((BLOCKSIZE >> MOVE) + 1)
uint prime[4800];
unsigned char BaseTpl[MAXM];
uint sieve( ){
        uint i, pnums = 1;
        uint QMAXN = (uint)sqrt(MAXN) + 20;
        prime[0] = 2;
        for (i = 3; i < QMAXN; i += 2){
                if ( !(BaseTpl[i >> MOVE] & MASKN(i)) ){
                        prime[pnums++] = i;
                        for (uint p = i * i; p < QMAXN; p += i << 1)
                                BaseTpl[p >> MOVE] |= MASKN(p);
                }
        }
        memset(BaseTpl, 0, sizeof(BaseTpl));
        for (i = 1; i < ST; i++){
        for (uint p = prime[i]; p < BLOCKSIZE; p += prime[i] << 1)
                BaseTpl[p >> MOVE] |= MASKN(p);
        }
        return pnums;
}
uint getBlocks(uint start, uint len = BLOCKSIZE){
        uint next, pnums = 0;
        uint maxp = (uint)sqrt((float)start + len) + 1;
        uint pmax = (uint)sqrt((float)start) + 1;
        unsigned char bits[1 << 8] = {0};
        for (uint j = 1; j < sizeof(bits); j++)
                bits[j] = bits[j >> 1] + (j & 1);
        unsigned char block[MAXM];
        memcpy(block, BaseTpl, (len >> MOVE) + 1);
        block[len >> MOVE] |= ~(MASKN(len) - 1);
        for (uint i = ST, p = prime[ST]; p < maxp; p = prime[++i]){
```

```
            if (p < pmax){
                next = (p - (start - 1) % p) - 1;
                if ((next & 1) == 0)
                    next += p;
            }
            else
                next = p * p - start;
            for (p <<= 1; next < len; next += p)
                block[next >> MOVE] |= MASKN(next);
        }
        pnums = (len >>= MOVE) * 8 + 8;
        for (uint k = 0; k <= len; k++)
            pnums -= bits[block[k]];
        return pnums;
}
uint getPrimes(uint beg, uint end){
        uint static table[MAXN / BLOCKSIZE + 1] = {0};
        int pnums = 0;
        if (beg >= end)
            return 0;
        if (beg < 2)
            beg = 2;
        for (int j = ST - 1; j >= 0 && beg <= prime[j]; j--){
            if (end > prime[j])
                pnums++;
        }
        pnums += getBlocks(end - end % BLOCKSIZE, end % BLOCKSIZE);
        int bsize = getBlocks(beg - beg % BLOCKSIZE, beg % BLOCKSIZE);
        beg /= BLOCKSIZE, end /= BLOCKSIZE;
        for (uint i = beg; i < end; i++){
            if (table[i] == 0)
                table[i] = getBlocks(i * BLOCKSIZE);
            pnums += table[i];
        }
        return pnums - bsize;
}
int main( ){
        uint beg, end;
        clock_t tstart = clock();
        sieve();
        printf("time use %u ms to cal primes %u\n", clock() - tstart, getPrimes(0, 30000u));
        while (scanf("%u %u", &beg, &end) == 2 && end >= 0){
            tstart = clock();
            uint primeCnt = getPrimes(beg, end + 1);
            printf("[%u, %u] : primes = %u, time use %u ms\n", beg, end, primeCnt, clock() - tstart);
        }
        return 0;
}
/*==================================================*\
|区间里的素数个数
|区间素数(两个大数之间的区间,区间长度小于 10^6)
\*==================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
bool flag[40004];
int top, prime[5005];
bool ans[1001*1001];
void Prime(){
        int i;
      memset(flag,false,sizeof(flag));
      for(i=2; i<=200;i++ ){
          if(!flag[i]){
              int j=i+i;
              while(j<=40000)
                {
                flag[j]=true;
                j+=i;
              }
          }
      }
      top=0;
      for(i=2;i<=40000;i++){
          if(!flag[i])
                prime[top++]=i;
      }
}
int main (){
      int kcase,l,u , i;
      bool ok=false;
      Prime();
      scanf("%d",&kcase);
      while(kcase--){
          scanf("%d%d",&l,&u);
          if(l==1)l++;
          if(ok)printf("\n");
          for(i=0;i<=u-l;i++){
              ans[i]=false;
          }
          int j;
          for(i=0;i<top && prime[i]*prime[i]<=u; i++){
              for( j=0;j<=u-l;j++)
                  if((j+l)>prime[i]&&(j+l)%prime[i]==0)
                        break;
                  if(j>u-l)continue;
                  while(j<=u-l){
                        ans[j]=true;
                        j+=prime[i];
                  }
          }
          for(i=0;i<=u-l;i++)
            {
              if(!ans[i])printf("%d\n",i+l);
          }
          ok=true;
      }
      return 0;
}
/*==================================================*\
|广义斐波那契数列
|f(n) = p * f(n - 1) + q * f(n - 2);求 f(n) % m 的值
\*==================================================*/
#include <iostream>
#include <stdio.h>
#include <string>
using namespace std;
int main()
{
   __int64 p , q , f[ 3 ] , n , m , y , yi;
   __int64 a[ 64 ][ 4 ], g[ 4 ] , h[ 4 ] , ans;
   int i , j , k , ff[ 64 ];
   while (scanf ("%I64d %I64d %I64d %I64d %I64d %I64d" , &p , &q , &f[ 1 ] , &f[ 2 ] , &n , &m) != EOF)
   {
      if(p==0&&q==0&&f[1]==0&&f[2]==0&&n==0&&m==0) break;
       if (n < 3)
       {
         printf ("%I64d\n" , f[ n ] % m);
         continue;
       }
      a[ 0 ][ 0 ] = p % m;
      a[ 0 ][ 1 ] = q % m;
      a[ 0 ][ 2 ] = 1 % m;
      a[ 0 ][ 3 ] = 0 % m;
      y = n;yi = 0;
       while (y)
            y /= 2 , yi ++;
       for (i = 1 ; i < yi ; ++ i)
       {
         a[ i ][ 0 ] = (a[i - 1][ 0 ] * a[i - 1][ 0 ] + a[i - 1][ 1 ] * a[i - 1][ 2 ]) % m;
         a[ i ][ 1 ] = (a[i - 1][ 0 ] * a[i - 1][ 1 ] + a[i - 1][ 1 ] * a[i - 1][ 3 ]) % m;
         a[ i ][ 2 ] = (a[i - 1][ 2 ] * a[i - 1][ 0 ] + a[i - 1][ 3 ] * a[i - 1][ 2 ]) % m;
         a[ i ][ 3 ] = (a[i - 1][ 2 ] * a[i - 1][ 1 ] + a[i - 1][ 3 ] * a[i - 1][ 3 ]) % m;
       }
```

```
        memset (ff , 0 , sizeof (ff));
        n = n - 2;
        k = 0;
      while (n)
        {
          ff[ k ] = n % 2;
          n = n / 2;
          k ++;
        }
      g[ 0 ] = g[ 3 ] = 1;
        g[ 1 ] = g[ 2 ] = 0;
        for (i = 0 ; i < k ; ++ i)
              if (ff[ i ])
              {
                h[ 0 ] = (g[ 0 ] * a[ i ][ 0 ] + g[ 1 ] * a[ i ][ 2 ]) % m;
             h[ 1 ] = (g[ 0 ] * a[ i ][ 1 ] + g[ 1 ] * a[ i ][ 3 ]) % m;
                h[ 2 ] = (g[ 2 ] * a[ i ][ 0 ] + g[ 3 ] * a[ i ][ 2 ]) % m;
                h[ 3 ] = (g[ 2 ] * a[ i ][ 1 ] + g[ 3 ] * a[ i ][ 3 ]) % m;
                for (j = 0 ; j < 4 ; ++ j)
                      g[ j ] = h[ j ];
              }
      ans   = (g[ 0 ] * f[ 2 ] + g[ 1 ] * f[ 1 ]) % m;
        printf ("%I64d\n" , ans);
    }
    return 0;
}
/*==================================================*\
| 大数平方根（字符串数组表示）
\*==================================================*/
void Sqrt(char *str){
        double i, r, n;
        int j, l, size, num, x[1000];
        size = strlen(str);
        if( size == 1 && str[0] == '0' )
            printf("0\n"); return;
        if( size%2 == 1 ){
            n = str[0]-48; l = -1;
         }
        else
            n = (str[0]-48)*10+str[1]-48; l = 0;
        r = 0; num = 0;
while(true){
      i = 0;
      while( i*(i+20*r) <= n ) ++i;
       --i;
      n -= i*(i+20*r);
      r = r*10+i;
      x[num] = (int)i;
      ++num;
      l += 2;
      if( l >= size ) break;
      n = n*100+(double)(str[l]-48)*10+(double)(str[l+1]-48);
}
        for(j = 0;j < num; j++) printf("%d", x[j]);
printf("\n");
}
/*==================================================*\
|大数计算 a*b % c
|(1)计算 a * b mod n, 思路: 利用 b 的二进制表示进行
|拆分计算
|(2)例如: b = 1011101 那么 a * b mod n = (a *
|1000000 mod n + a * 10000 mod n + a * 1000 mod
|n + a * 100 mod n + a * 1 mod n) mod n
|(3)思路就是上面描述的那样，那么可以用从低位往高
|位遍历 b, 并用 a 来记录当前位为 1 的值，每次遇到 b
|当前位为 1 就将结果值加上 a 并  mod n, 然后 a  要乘
|以 2
\*==================================================*/
__int64 multAndMod(__int64 a, __int64 b, __int64 n)
{       a = a % n;
      __int64 res = 0;
while(b)
      {     //当前位为 1
            if(b & 1)
            {     //加上当前权位值
                  res += a;
                  //相当于 mod n
                  if(res >= n) res -= n;
            }  //乘以 2，提高一位
            a = a<<1;
            //mod n
            if(a >= n) a -= n;
            b = b >> 1;
      }
 return res;
}
/*==================================================*\
|大数计算 a^b % c
|(1)计算 a ^ b mod n, 思路: 和上面类似，也是利用 b
|的二进制表示进行拆分计算
|(2)例如: b = 1011101 那么 a ^ b mod n = [(a ^
|1000000 mod n) * (a ^ 10000 mod n) * (a ^ 1000
|mod n) * (a ^ 100 mod n) * (a ^ 1 mod n)] mod n
|(3)思路就是上面描述的那样，那么可以用从低位往高
|位遍历 b, 并用 a 来记录当前位为 1 的值，每次遇到 b
|当前位为 1 就将结果乘上 a 并  mod n, 然后 a  要乘以
|a 以提升一位
\*==================================================*/
#include <iostream>
using namespace std;
#include<stdio.h>
__int64 FPM (__int64 a , __int64 b , __int64 c)//a ^ b % c{
        __int64 ans = 1 , tmp = a % c;
        while (b){
                if (b & 1)
                        ans = ans * tmp % c;
                tmp = tmp * tmp % c;
                b = b >> 1;
        }
        return ans % c;
}
int main(){
        __int64 a , b , c;
        while (scanf ("%I64d %I64d %I64d" , &a , &b , &c) == 3)
                printf ("%I64d\n" , FPM (a , b , c));
        return 0;
}
/*==================================================*\
|大数计算 a%c ,a 为大整数,c 为 int 型整数
\*==================================================*/
#include <iostream>
using namespace std;
int main(){
   char str[ 100001 ];
   int m , k , ans;
   while (scanf ("%s %d" , str , &m) != EOF){
      ans = 0;
      for (k = 0 ; str[ k ] ; ++ k)
            ans = (ans * 10 + str[ k ] - '0') % m;
      printf ("%d\n" , ans);
   }
   return 0;
}
/*==================================================*\
|大数计算 a 模 b 的逆
\*==================================================*/
#include <iostream>
using namespace std;
int exgcd (int a , int b , int &x , int &y){
   if (b == 0){
      x = 1 , y = 0;
        return a;
   }
   int r = exgcd (b , a % b , x , y);
   int t = x;
   x = y , y = t - a / b * y;
   return r;
}
int main(){
   int a , b , m , n , r;
   while (scanf ("%d %d" , &a , &b) == 2){
      r = exgcd (a , b , m , n);     //当 r == 1 时，m 即为 a 模 b 的逆；
```

```
        if (r == 1){
            while (m < 0)
                m += b , n -= a;
            printf ("%d %d\n" , m , n);
        }
        else
            puts("sorry");
    }
    return 0;
}
```

```
/*==================================================*\
|大数计算 C（m,n）*F(n)(斐波拉契数列)
\*==================================================*/
```

```
#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;
int t[3][3] = {
    {0, 0, 0},
    {0, 1, 1},
    {0, 1, 0}
};
int MOD, N;
int a[3][3];
void Dfs(int P) {
    if (P == 1) {
        for (int i = 1; i <= 2; i++)
            for (int j = 1; j <= 2; j++)
              a[i][j] = t[i][j];
              return;
    }
    Dfs(P / 2);
    int b[3][3];
     memset(b, 0, sizeof (b));
      for (int i = 1; i <= 2; i++) {
         for (int j = 1; j <= 2; j++) {
             for (int k = 1; k <= 2; k++) {
                 b[i][j] = (b[i][j] + a[i][k] * a[k][j]) % MOD;
             }
         }
    }
    for (int i = 1; i <= 2; i++) for (int j = 1; j <= 2; j++) a[i][j] = b[i][j];
    if (P % 2 == 0) return;
    memset(b, 0, sizeof (b));
    for (int i = 1; i <= 2; i++) {
        for (int j = 1; j <= 2; j++) {
            for (int k = 1; k <= 2; k++) {
                b[i][j] = (b[i][j] + a[i][k] * t[k][j]) % MOD;
            }
        }
    }
    for (int i = 1; i <= 2; i++)
      for (int j = 1; j <= 2; j++)
        a[i][j] = b[i][j];
}
int main() {
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        scanf("%d%d", &N, &MOD);
        if (N == 0) {
            printf("0\n");
            continue;
        }
        Dfs(2 * N - 1);
        printf("%d\n", a[1][1]);
    }
    return 0;
}
```

```
/*==================================================*\
|大数计算 C（m,n）% P (P 为素数)
|组合数 C(n,m)%mod   其中 mod 为素数，恰 mod 的|范围有限
|制，不能太大。利用 Lucas 定理，和求解(a/b)%m （其中
|a%b==0,gcd(b,m)=1）方法求解。
\*==================================================*/
```

```
#include <iostream>
using namespace std;
#define mod 10009
#define INT int
INT fid[mod];
void init(){                    //  对 n!%mod 初始化
        INT i;
        fid[0]=1;
        for(i=1;i<mod;i++)
                fid[i]=i*fid[i-1]%mod;
}
INT gcd(INT a,INT b) {              //求最大公约数
        if(b==0)
                return a;
        return gcd(b,a%b);
}
void exGCD(INT a,INT b,INT &x,INT &y) {         //解线性方程
        if(b==0){
                x=1;
                y=0;
                return ;
        }
         exGCD(b,a%b,x,y);
        INT t=x;
        x=y;
        y=t-a/b*y;
        return ;
}
INT chose(INT n, INT m) {           //求 C(n,m)%mod
        if(n < m)return 0;          //如个 n<m 直接返回 0
        if(m==0||n==m)return 1;     //m=0 或 m=n 返回 1
INT nn=fid[n],mm=fid[m]*fid[n-m]%mod;//C(n,m)=n!/(m!*(n-m)!)
        INT d=gcd(nn,mm);
        nn/=d;
        mm/=d;
        INT x,y;
        exGCD(mm,mod,x,y);          //求 mm 的逆元
        x=(x+mod)%mod;
        return x*nn%mod;
}
INT ANS(INT n,INT m) {
//求 C(n,m)    (这里的 n，m 较大，未经处理，和上面的不同)
        INT cn[50],cm[50];
        memset(cm,0,sizeof(cm));
        memset(cn,0,sizeof(cn));
        INT   i,s,nt=0;
        while(n) {                  //将 n 转变成 mod 进制
                cn[nt++]=n%mod;
                n/=mod;
        }
        nt=0;
        while(m) {                  //将 m 转换成 mod 进制
                cm[nt++]=m%mod;
                m/=mod;
        }
        s=1;
        for(i=0;i<nt;i++){
               s=s*chose(cn[i],cm[i])%mod;
        }
        return s;
}
int main(){
        int n,m,s;
        init();
        while(cin>>n>>m){
                s=ANS(n,m);
                printf("%d\n",s);
        }
        return 0;
}
```

```
/*==================================================*\
|高精度求解组合数 C(n,m) （m<=n,0<=m<=5000），
|效率为 O(m*m)
\*==================================================*/
```

```
#include<iostream>
using namespace std;
#define INT int
```

```
#define MAX_SIZE 50000
#define MOD 100000
struct CnCm{
        INT c[MAX_SIZE];
        INT len;
}cs;
INT gcd(INT x,INT y){
        if(y==0)
                return x;
        return gcd(y,x%y);
}
void Cn_Cm(INT n,INT m){
        INT md,i,j,cn[MAX_SIZE],cm[MAX_SIZE];
        if(m==n||m==0){
                cs.c[0]=1;
                cs.len=1;
                return ;
        }
        if(m>n-m)
                m=n-m;
        for(i=0;i<m;i++)
                cm[i]=i+1;
        //cm[0]=m+1;   //如求卡特兰数时调用 Cn_Cm(2*n,n)，并在此处加入 cm[0]=m=1。
        j=0;
        for(i=n-m+1;i<=n;i++)
                cn[j++]=i;
        for(i=0;i<m;i++){
                j=0;
                while(cm[i]!=1){
                        md=gcd(cm[i],cn[j]);
                        cm[i]/=md;
                        cn[j]/=md;
                        j++;
                }
        }
        cs.c[0]=cn[0];
        cs.len=1;
        for(i=1;i<m;i++){
                md=0;
                if(cn[i]!=1)
                for(j=0;j<cs.len;j++){
                        cs.c[j]=cs.c[j]*cn[i]+md;
                        md=cs.c[j]/MOD;
                        cs.c[j]%=MOD;
                }
                while(md){
                        cs.c[cs.len++]=md%MOD;
                        md/=MOD;
                }
        }
}
//输出函数，可以再主函数里输出，也可以在 Cn_Cm()函数里输出。根据不同情况选择
void out_ans() {
        int i;
        printf("%d",cs.c[cs.len-1]);
        for(i=cs.len-2;i>=0;i--)
                printf("%0.5d",cs.c[i]);
        printf("\n");
}
int main(){
        int n,m;
      Cn_Cm(n,m);
      out_ans();
}
/*==================================================*\
| 已知圆上的一个坐标  求另外 2 个能与给定点构成三角
| 形的坐标点
\*==================================================*/
#include<iostream>
#include<cmath>
using namespace std;
int main(){
   double a,b;
   double x1,x2,y1,y2;
   int t;
   double g;
   double temp;
   g=sqrt(3.0);
   cin>>t;
   while(t--){
      scanf("%lf%lf",&a,&b);
      x1=(g*b-a)/2.0;     y1=(-b-g*a)/2.0;
      x2=-(a+g*b)/2.0;    y2=(g*a-b)/2.0;
      if(y1>y2)
      {
         temp=x1;x1=x2;x2=temp;
         temp=y1;y1=y2;y2=temp;
      }
      else
            if(y1==y2){
               if(x1>x2 && x1-x2>=0.0005)
               {
                  temp=x1;x1=x2;x2=temp;
               }
            }
      printf("%.3f %.3f",x1,y1);
      printf(" %.3f %.3f\n",x2,y2);
   }
   return 0;
}
/*==================================================*\
| 计算 n 的 n 次方最左面的数字,对 a 的 b 次方这种情况
| 适不适用还不知道
\*==================================================*/
#include<stdio.h>
#include<math.h>
int main(){
      int m,n,i,ans;
      double a,b;
      scanf("%d",&n);
      for(i=0;i<n;i++){
            scanf("%d",&m);
            a=m*log10(m*1.0)-(__int64)(m*log10(m*1.0));
            b=pow(10,a);
            ans=(int)b;
            printf("%d\n",ans);
      }
      return 0;
}
/*==================================================*\
| 计算 n 的 n 次方最右面的数字,对 a 的 b 次方这种情况
| 适不适用还不知道
\*==================================================*/
#include<stdio.h>
int main(){
       long int ri,n,m,t;
       int a[4];
      scanf("%ld",&m);
      for(ri=0;ri<m;ri++){
            scanf("%ld",&n);
            t=n;
            n=n%10;
            if(n==0||n==5||n==6||n==1||n==9)
            printf("%ld\n",n);
            else {
                  a[0]=n;
                  a[1]=(n*n)%10;
                  a[2]=(a[1]*n)%10;
                  a[3]=(a[2]*n)%10;
                  t=t%4;
                  if(t==0)t=4;
                  printf("%ld\n",a[t-1]);
            }
      }
}

/*==================================================*\
| 求 A^B 最后一位
\*==================================================*/
#include<iostream>
```

```
#include<math.h>
using namespace std;
//int
a[10][5]={{0},{1,1,1,1,1},{1,2,4,8,0},{1,3,7,9,0},{1,4,6,4},{1,5,5,5,5
},{1,6,6,6,6},{1,7,9,3,0},{1,8,4,2,6},{1,9,1,9,1}}
int main(){
		int n,m;
		while(cin>>n>>m){
			n=n%10;
			//cout<<n<<"haha"<<endl;
			if(n==0) cout<<'0'<<endl;
			if(n==1) cout<<'1'<<endl;
			if(n==2){
				if(m==0) cout<<'1'<<endl;
				else{
					int temp=(m-1)%4;
					if(temp==0) cout<<'2'<<endl;
					if(temp==1) cout<<4<<endl;
					if(temp==2) cout<<8<<endl;
					if(temp==3) cout<<6<<endl;
				}
			}
			if(n==3){
					int temp=m%4;
					if(temp==0) cout<<'1'<<endl;
					if(temp==1) cout<<'3'<<endl;
					if(temp==2) cout<<'9'<<endl;
					if(temp==3) cout<<'7'<<endl;
			}
			if(n==4){
				if(m==0) cout<<'1'<<endl;
				else{
					int temp=(m-1)%2;
					if(temp==0) cout<<'4'<<endl;
					else cout<<'6'<<endl;
				}
			}
			if(n==5){
				if(m==0) cout<<'1'<<endl;
				cout<<'5'<<endl;

			}
			if(n==6){
				if(m==0) cout<<'1'<<endl;
				cout<<'6'<<endl;
			}
			if(n==7){
				//cout<<"hello"<<endl;
					int temp=m%4;
					if(temp==0) cout<<'1'<<endl;
					if(temp==1) cout<<'7'<<endl;
					if(temp==2) cout<<'9'<<endl;
					if(temp==3) cout<<'3'<<endl;
			}
			if(n==8){
				if(m==0) cout<<'1'<<endl;
				else{
					int temp=(m-1)%4;
					if(temp==0) cout<<'8'<<endl;
					if(temp==1) cout<<'4'<<endl;
					if(temp==2) cout<<'2'<<endl;
					if(temp==3) cout<<'6'<<endl;
				}
			}
			if(n==9){
				if(m%2==0) cout<<'1'<<endl;
				else cout<<'9'<<endl;
			}
		}
	}
```

/*==*\
|圆桌问题
|如果他们在每一分钟内,一对相邻的两个 ACM 队员交换一下位子,那么要多少时间才能得到与原始状态相反的座位顺序呢?
==/

```
#include<stdio.h>
int main(){
	long i,j,n,m,a[32768];
	a[2]=1;
	for(i=3;i<=32768;i++)
		a[i]=a[i-1]+i-1;
	while(scanf("%d",&n)==1){
		if(n%2==0)
		m=a[n/2]*2;
		else m=a[n/2]+a[n/2+1];
		printf("%d\n",m);
	}
}
```

/*==*\
|计算大数的卡特兰数，不过答案是 2 倍，更加高精度的见 JAVA 程序
==/

```
#include<iostream>
#include<stdio.h>
using namespace std;
int a[1000][1000]={0};				//a[i][]表示第 i 个卡特兰数
int s[1000];
int main(){
	int n,i,j,len,r,temp,t;
	int b[1000];				//数字长度
	b[1] = len = a[1][0] = 1;
	for(i=2;i<=300;i++){
		t = i-1;
		for(j=0;j<len;j++)			//乘法
			a[i][j] = a[i-1][j]*(4*t+2);
		for(r=j=0;j<len;j++)			//处理相乘结果{
			temp = a[i][j] + r;
			a[i][j] = temp % 10;
			r = temp / 10;
		}
		while(r) {				//进位处理
			a[i][len++] = r % 10;
			r /= 10;
		}
		for(j=len-1,r=0;j>=0;j--){		//除法
		  temp = r*10 + a[i][j];
			a[i][j] = temp/(t+2);
			r = temp%(t+2);
		}
		while(!a[i][len-1])		//高位零处理
			len --;
		b[i] = len;
	}
	int h,k;
	while(cin>>n&&n){
		k=0;
		int flag=0;
		for(j=0;j<=b[n]-1;j++){
			h=a[n][j]*2;
			if(flag)
			  h++;
			if(h>9)flag=1;
			else flag=0;
			   s[k++]=h%10;
		}
		if(flag)
		s[k++]=1;
		for(i=k-1;i>=0;i--)
		printf("%d",s[i]);
		printf("\n");
	}
	return 0;
}
```

/*==*\
|计算 A^B (大数)
==/

```
#include <iostream>
#include<stdio.h>
using namespace std;
#define MAX 25000
#define MAX_NUM 100000
```

```
#define INT long long
void   BIG_NUM( INT *num , INT n){
        INT i,k,mod,j;
        INT big[MAX];
        INT mt[MAX];
        big[0]=1;
        k=1;
        for(i=0;i<n;i++){
                mod=0;
                for(j=0;j<k;j++){
                        big[j]=big[j]*num[i]+mod;
                        mod=big[j]/MAX_NUM;
                        big[j]=big[j]%MAX_NUM;
                }
                while(mod){
                        big[k++]=mod%MAX_NUM;
                        mod/=MAX_NUM;
                }
        }
        j=0;
        for(i=0;i<k-1;i++){
                int t=5;
                while(t--){
                        mt[j++]=big[i]%10;
                        big[i]/=10;
                }
        }
        while(big[k-1]){
                mt[j++]=big[k-1]%10;
                big[k-1]/=10;
        }
        k=0;
        for(i=j-1;i>=0;i--){
                k++;
                printf("%lld",mt[i]);
                if(k%70==0)
                        printf("\n");
        }
        if(k%70)
        printf("\n");
}
int main(){
        INT a[MAX];
        INT n,i,m;
        INT big[MAX];
        while(scanf("%lld%lld",&n,&m)!=EOF){
                for(i=0;i<m;i++)
                        a[i]=n;
                        //cin>>a[i];
                BIG_NUM(a,m);
        }
        return 0;
}
/*==================================================*\
|求 2^k 的值的后 R 位数是否为连续的'1','2',并求出 K
|值（比如，when k = 1 ,2^k 的后 1 位就是 2 ,k =
|2 ,2^k 的后 2 位就是 12 , k = 89,
|2^k 的后 R 位就是 2112 ， 他代表 R = 3 , R = 4)
|Sample Input
|6
|1
|2
|4
|5
|7
|15
|Sample Output
|1 1 1
|2 2 9
|3 4 89
|4 5 589
|5 7 3089
|6 15 11687815589
\*==================================================*/
#include<iostream>
using namespace std;
int main(){
        int m,n,i;
        char
ant[23][30]={"0","1","9","89","89","589","3089","3089","3089","315589","315589","8128089","164378089","945628089","1922190589","11687815589","109344065589","231414378089","1452117503089","4503875315589","65539031565589"};
        scanf("%d",&m);
        for(i=1;i<=m;i++)
        {
                scanf("%d",&n);
                printf("%d %d %s\n",i,n,ant[n]);
        }
return 0;
}
/*==================================================*\
|统计从 N 到 M 之间  0-9 的个数
\*==================================================*/
#include<stdio.h>
void statisticsnumber(int *sn,int n){
        int i,cur;
        int k=0,s=0;
        int pow=1;

        for(i=0;i<10;i++)
        sn[i]=0;
        for(;n>0;k++,n/=10,pow*=10){
                cur=n%10;
                for(i=0;i<10;i++)
                sn[i]+=cur*k*pow/10;
                for(i=0;i<cur;i++)
                sn[i]+=pow;
                sn[cur]+=1+s;
                sn[0]-=pow;
                s+=cur*pow;
        }
}
int main(){
        int sn[10],a[10],i,n,x,y,m;
        while(scanf("%d%d",&n,&m)!=EOF){
            if(n==0&&m==0)break;
            if(n>m){
                x=n;y=m;
              }
              else {
                x=m;y=n;
                }
                statisticsnumber(sn,x);
                statisticsnumber(a,y-1);
                for(i=0;i<9;i++)
                printf("%d ",sn[i]-a[i]);
                printf("%d\n",sn[9]-a[9]);
        }
}
/*==================================================*\
|约瑟夫问题
|f[1]=0; f[i]=(f[i-1]+m)%i;  (i>1)
|令 f[i]表示 i 个人玩游戏报 m 退出最后胜利者的编号，最后的结果是 f[n]。
|这个算法的时间复杂度为 O(n)。
\*==================================================*/
/*==================================================*\
|类型一：已知开始人数 m，报数 n,求最后出列的编号
\*==================================================*/
/*==================================================*\
|方法一：用数组存每组数据
\*==================================================*/
#include<stdio.h>
#define MAXN 10001
int main(){
        long c,f,m,n,s[10001]={0};
        scanf("%d",&c);
        while(c--&&scanf("%d %d",&m,&n)){
                for(f=2;f<=m;f++) s[f]=(s[f-1]+n)%f;
                printf("%d\n",s[f-1]+1);
        }
        return 0;
```

```
}
/*==================================================*\
|方法二：不用数组
\*==================================================*/
#include<stdio.h>
int main(){
	long c,f,m,n,s;
	scanf("%d",&c);
	while(c--&&scanf("%d %d",&m,&n)){
		for(f=2,s=0;f<=m;f++) s=(s+n)%f;
		printf("%d\n",s+1);
	}
	return 0;
}
/*==================================================*\
|类型二：已知最后的元素 为 2（为 k 也可以，稍微改动一下下面的程
|序即可），n 从输入中给出，求出最小的 m
\*==================================================*/
#include<stdio.h>
void main(){
	int n,s,i,m;
	while(scanf("%d",&n)!=EOF&&n){
		for(m=1; ; m++){
		for(s=1,i=2; i<=n-1; i++)		//n－1 个人编号从 1 开始
			s=(s+m-1)%i+1;
		if(s==1)	//题中的 2 即约瑟夫问题的 1  或将其改成 k-1
			break;
		}
		printf("%d\n",m);
	}
}
/*==================================================*\
|类型三：有 n 个人，站成一个圆圈。给你两个数 k,m。。首先请第 m 个
|人出列，然后从 m+1 开始数数 1-k,数到 k 的人出列，，最后会留下只剩
|下一个人。让你求出这个人的编号是什么。
\*==================================================*/
#include<stdio.h>
int main(){
	int n,m,s,k,i,j;
	while(scanf("%d%d%d",&n,&k,&m)!=EOF&&(n+k+m)){
		if(n==0&&m==0&&k==0)break;
		s=0;
		for(i=2;i<n;i++)
		s=(s+k)%i;
		printf("%d\n",(s+m)%i+1);
	}
	return 0;
}
/*==================================================*\
|矩阵 A^n
|求矩阵 A*B，其中矩阵 A,B 为 n 阶矩阵
|相乘的结果 C=A*B,其中对 C 中里的每个元取于 mod
\*==================================================*/
#include<iostream>
using namespace std;
#define MAX 100
#define INT int
INT mod;
struct mat		//矩阵定义
{
	INT arr[MAX][MAX];		//存放元素的数组
	INT len;			//矩阵的大小
};
mat ride(mat m1,mat m2) {	//矩阵 m1 乘 m2，记录到 m 里
	mat m;
	INT s;
	m.len=m1.len;
	int i,j,k;
	for(i=0;i<m1.len;i++)
		for(j=0;j<m1.len;j++)
		{
			s=0;
			for(k=0;k<m1.len;k++)
			{
				s=(s+m1.arr[i][k]*m2.arr[k][j]%mod)%mod;
			}
				m.arr[i][j]=s%mod;
		}
	return m;
}
mat MAT_MOD(mat m,INT n) {	//求矩阵 m^n（n>0），返回矩阵 s
	mat s;
	s=m;
	n--;
	while(n)
	{
		if(n & 1)s=ride(s,m);
		m=ride(m,m);
		n/=2;
	}
	return s;
}
int main(){
	INT n;
	mat m;
	while(scanf("%d",&n)!=EOF&&n>=0){
		mod=10000;
		if(n==0)
		{
			printf("0\n");
			continue;
		}
		m.len=2;
		m.arr[0][0]=m.arr[0][1]=m.arr[1][0]=1;
		m.arr[1][1]=0;
		m=MAT_MOD(m,n);
		printf("%d\n",m.arr[0][1]);
	}
	return 0;
}
/*==================================================*\
|最基础的矩阵的乘积,矩阵的幂求法(二分法)
\*==================================================*/
#include <iostream>
#define MAX 11	//注意这里太大的话,不能传要用全局变量计算
using namespace std;
typedef struct node
{
	int matrix[MAX][MAX];
//	int nn,mm;	//nn 代表行,mm 代表列
}Matrix;
Matrix init,unit;	//分别定义 init 为初始的输入矩阵,unit 为单位矩阵
int n,kk;
void Init()
{
	int i,j;
	scanf("%d%d",&n,&kk);
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
		{
			scanf("%d",&init.matrix[i][j]);	//输入初始矩阵
			unit.matrix[i][j]=(i==j);	//初始化初始矩阵
		}
}
Matrix Mul(Matrix a,Matrix b)//据说传结构体比传数组快
{
	int i,j,k;
	Matrix c;
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
		{
			c.matrix[i][j] = 0;
			for(k=0;k<n;k++)
				c.matrix[i][j] += a.matrix[i][k]*b.matrix[k][j];
			c.matrix[i][j]%=9973;
		}
	return c;
}
Matrix Cal(int k)//k 代表幂,这里是利用二分法求矩阵的幂
{
	Matrix p,q;
	p = unit;	//p 为单位矩阵
```

```
    q = init;     //q 为初始矩阵
    while(k!=1)
    {
        if(k&1)   //k 是奇数,实际这里只进行一次运算
        {
            k--;
            p = Mul(p,q);   //如果 k 是奇数,那么就不能进行平均的二分,所以让 p 乘以一个单位矩阵,保证其不变,然后 k--就可以进行二分了
        }
        else   //k 是偶数
        {
            k>>=1;   //k 除 2
            q = Mul(q,q);
        }
    }
    p = Mul(p,q);//如果 k 是偶数的话这一步是没有必要的
    return p;
}
int main()
{
    Matrix r;
    int t;
    cin>>t;
    while(t--)
    {
        Init();
        r=Cal(kk);
        int i,j,sum;
        i=0;
        sum=0;
        while(i<n)
        {
            sum+=r.matrix[i][i];
            sum%=9973;
            i++;
        }
        printf("%d\n",sum);
    }
    return 0;
}
/*==================================================*\
| 给一个 n×n 矩阵 A 和正数 k, 求和 S = A + A2 + A3 + … + Ak.
\*==================================================*/
#include <iostream>
#define MAX 33
using namespace std;
typedef struct node
{
    int matirx[MAX][MAX];}Matrix;
Matrix a,sa,unit;
int n,k,m;
void Init()
{
    int i,j;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            scanf("%d",&a.matirx[i][j]);
            a.matirx[i][j]%=m;    //初始化要先%m
            unit.matirx[i][j]=(i==j);   //如果 i==j 那么矩阵中此值就是 1,否则为 0,就是主对角线是 1 的单位矩阵
        }
}
Matrix Add(Matrix a,Matrix b)   //矩阵加法
{
    Matrix c;
    int i,j;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            c.matirx[i][j]=a.matirx[i][j]+b.matirx[i][j];
            c.matirx[i][j]%=m;    //加的时候也要%m
        }
    return c;
}
Matrix Mul(Matrix a,Matrix b)   //矩阵乘法
{
    Matrix c;
    int i,j,k;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            c.matirx[i][j]=0;   //初始化矩阵 c
            for(k=0;k<n;k++)
                c.matirx[i][j]+=a.matirx[i][k]*b.matirx[k][j];
            c.matirx[i][j]%=m;   //计算乘法的时候也要%m
        }
    return c;
}
Matrix Cal(int exp)    //矩阵幂
{
    Matrix p,q;
    p=a;   //p 是初始矩阵
    q=unit;   //q 是单位矩阵
    while(exp!=1)
    {
        if(exp&1)   //要求得幂是奇数
        {
            exp--;
            q=Mul(p,q);
        }
        else      //要求的幂是偶数
        {
            exp>>=1;   //相当于除 2
            p=Mul(p,p);
        }
    }
    p=Mul(p,q);
    return p;
}
Matrix MatrixSum(int k)
{
    if(k==1)   //做到最底层就将矩阵 a 返回就好
        return a;
    Matrix temp,tnow;
    temp=MatrixSum(k/2);
    if(k&1)   //如果 k 是奇数
    {
        tnow=Cal(k/2+1);
        temp=Add(temp,Mul(temp,tnow));
        temp=Add(tnow,temp);
    }
    else      //如果 k 是偶数
    {
        tnow=Cal(k/2);
        temp=Add(temp,Mul(temp,tnow));
    }
    return temp;
}
int main(){
    int i,j;
    while(scanf("%d%d%d",&n,&k,&m)!=EOF)
    {
        Init();
        sa=MatrixSum(k);
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-1;j++)
            {
                printf("%d ",sa.matirx[i][j]%m);
            }
            printf("%d\n",sa.matirx[i][n-1]%m);
        }
    }
    return 0;
}

/*==================================================*\
| 归并排序求逆序数
| (也可以用树状数组做)
| a[0...n-1] cnt=0; call: MergeSort(0, n)
\*==================================================*/
```

```
void MergeSort(int l, int r){
        int mid, i, j, tmp;
    if( r > l+1 ){
        mid = (l+r)/2;
        MergeSort(l, mid);
        MergeSort(mid, r);
        tmp = l;
        for( i=l, j=mid; i < mid && j < r; ){
            if( a[i] > a[j] ){
                c[tmp++] = a[j++];
                cnt += mid-i; //
            }
          else c[tmp++] = a[i++];
        }
        if( j < r ) for( ; j < r; ++j ) c[tmp++] = a[j];
        else for( ; i < mid; ++i ) c[tmp++] = a[i];
        for ( i=l; i < r; ++i ) a[i] = c[i];
    }
}
/*==================================================*\
| 逆序数推排列数
| 动态规划：f(n,m)表示逆序数为 m 的 n 元排列的个数，则
| f(n+1,m)=f(n,m)+f(n,m-1)+...+f(n,m-n)(当 b<0 时，f(a,b)=0)
|优化 又考虑到如果直接利用上式计算时间复杂度为 O(n^3)，我们分析
|上式不难发现 f(n+1,m)=f(n,m)+f(n+1,m-1)
| if( m-n-1 >= 0 ) f(n+1, m) -= f(n, m-n-1).
| JOJ 2443
\*==================================================*/
const int N = 1001;
const int C = 10001;
const long MOD = 1000000007;
long arr[N][C];
long long temp;
int main(void){
        int i, j;
        arr[1][0] = arr[2][0] = arr[2][1] = 1;
        for( i=3; i < N; ++i ){
            arr[i][0] = 1;
            long h = i*(i+1)/2+1;
            if( h > C ) h = C;
            for( j=1; j < h; ++j ){
                temp = arr[i-1][j] + arr[i][j-1];
                arr[i][j] = temp%MOD;
              if( j-i >= 0 ) {
                  arr[i][j] -= arr[i-1][j-i];
                    if( arr[i][j] < 0 ){
//注意：由于 arr[i][j]和 arr[i-1][j-i]都是模过的，所以可能会得到负数
                        arr[i][j] += MOD;
                    }
              }
            }
        }
    while( scanf("%d %d", &i, &j) != EOF )
            printf("%ld\n", arr[i][j]);
    return 0;
}
/*==================================================*\
| 所有数位相加
| dig(x) := x if 0 <= x <= 9
| dig(x) := dig(sum of digits of x) if x >= 10
\*==================================================*/
方法一：模拟
int dig(int x){
        if( x < 10 ) return x;
        int sum = 0;
        while( x ) { sum += x%10; x /= 10; }
        return dig(sum);
}
方法二：公式 【不太明白...】
        int dig(int x){ return (x+8)%9+1; }
```

2.3 数论

```
/*==================================================*\
|欧几里德算法求最大公约数,最小公倍数
\*==================================================*/
int gcd(int a,int b)
{ int temp;
    while (b!=0)
    {
        temp=a%b;
        a=b;
        b=temp;
     }
     return(a);
}
int lcm(int a,int b)
{
    int c;
    c=a;
    while(a%b!=0)
        a=a+c;
    return a;
}
/*==================================================*\
| 快速 GCD
\*==================================================*/
int kgcd(int a, int b){
     if (a == 0) return b;
     if (b == 0) return a;
     if (!(a & 1) && !(b & 1)) return kgcd(a>>1, b>>1) << 1;
     else if (!(b & 1)) return kgcd(a, b>>1);
     else if (!(a & 1)) return kgcd(a>>1, b);
     else return kgcd(abs(a - b), min(a, b));
}
/*==================================================*\
| 扩展 GCD
| 求 x, y 使得 gcd(a, b) = a * x + b * y;
\*==================================================*/
int extgcd(int a, int b, int & x, int & y){
        if (b == 0) { x=1; y=0; return a; }
        int d = extgcd(b, a % b, x, y);
        int t = x; x = y; y = t - a / b * y;
  return d;
}
/*==================================================*\
| 模线性方程 a * x = b (% n)
\*==================================================*/
void modeq(int a, int b, int n) {              // ! n > 0
        int e, i, d, x, y;
        d = extgcd(a, n, x, y);
        if (b % d > 0) printf("No answer!\n");
        else {
            e = (x * (b / d)) % n;
          for (i = 0; i < d; i++)             // !!! here x maybe < 0
         printf("%d-th ans: %d\n", i+1, (e+i*(n/d))%n);
        }
}
/*==================================================*\
|欧几里德算法求大于 N 的个数
\*==================================================*/
#include <iostream>
#define MAX 64
using namespace std;
int k , ans , N , M;
int p[ MAX ] , r[ MAX ];
void factor (int n) {                  //核心部分
      int i;
      bool v;
      for (i = 2 ; i * i <= n ; ++ i){
           v = false;
           r[ k ] = 0;
           while (n % i == 0){
                p[ k ] = i;
                r[ k ] ++;
                n /= i;
                v = true;
           }
           if (v)
                k ++;
      }
      if (n > 1)
```

```
            p[ k ] = n , r[k ++] = 1;
}
void solve (int h , int g , int n){
        int j;
        if (h == k){
            if (g >= M)
                ans += n / g;
            return ;
        }
        int next = n   / p[ h ] * (p[ h ] - 1);
        for (j = 0 ; j < r[ h ] ; ++ j , g *= p[ h ])
            solve (h + 1 , g , next);
        solve (h + 1 , g , n);
}
int main(){
        int t;
        cin >> t;
        while (t --){
            cin >> N >> M;
            k = 0;
            factor (N);
            ans = 0;
            solve (0 , 1 , N);
            printf ("%d\n" , ans);
        }
        return 0;
}
```

/*==*\
|欧几里德的扩展求解线性方程 **a*x+b*y=c** 的整数解方程（扩展 **GCD**）
|这里按两种情况：
|**1**、求 **x** 的最小非负整数解，**2**、求 **x** 的正整数解。具体如下
|不同情况，要不同的考虑。
==/

```
#include<iostream>
using namespace std;
#define INT __int64
INT x,y;
INT mood(INT a,INT b){
//    if(a>0)              //当求最小正整数解时
        if(a>=0)              //当求最小非负解时用此 if 语句
            return a%b;
        return a%b+b;
}
INT exGCD(INT a,INT b){
        if(b==0){
            x=1;
            y=0;
            return a;
        }
        INT r=exGCD(b,a%b);
        INT t=x;
        x=y;
        y=t-a/b*y;
        return r;
}
int main(){
        INT a,b,c,d;
        while(scanf("%I64d%I64d%I64d",&a,&b,&c)!=EOF){
            if(a==0&&b==0&&c==0){
                //printf("1\n");           //求最小整数解
                printf("0\n");            //求最小非负整数解
                continue;
            }
            if(b==0&&a==0){
                printf("NO\n");
                continue;
            }
            if(b==0&&c%a==0){
            //      printf("");
            }
            if(a==0&&c%b==0){
                //printf("1\n");    最小正整数解
                printf("0\n");     //最下非负整数解
                continue;
            }
            else if(a==0)
                printf("NO\n");
            d=exGCD(a,b);
            if(c%d){
                printf("NO\n");
                continue;
            }
            b=b/d;
            printf("x = %I64d\n",x);
            x=mood(x*c/d,b);
            b=b*d;
            y=(c-a*x)/b;

        printf("%I64d*%I64d+%I64d*%I64d=%I64d\n",a,x,b,y,c);
        }
        return 0;
}
```

/*==*\
|高效的欧拉函数算法（可任意替换成__**int64**）
==/

```
unsigned euler(unsigned x){                // 就是公式
        unsigned i, res=x;
        for (i = 2; i < (int)sqrt(x * 1.0) + 1; i++)
            if(x%i==0) {
                res = res / i * (i - 1);
            while (x % i == 0) x /= i; // 保证 i 一定是素数
            }
        if (x > 1) res = res / x * (x - 1);
return res;
}
```

/*==*\
|欧拉函数打表
|欧拉函数 **phi(n)**表示比 **n** 小且与 **n** 互质的数的个数，包括 **1**，
|**pr[]**存放的是素数，**phi[n]**为 **n** 的欧拉函数值
==/

```
#include <iostream>
#define MAX 3000005
using namespace std;
int phi[MAX],pr[1000005],pn;
void init(){
        int i,j;
        memset(phi,0,sizeof(0));
        pn=0;
        for(i=2;i<MAX;++i){
            if(!phi[i]){
                pr[pn++]=i;
                phi[i]=i-1;
            }
            for(j=0;j<pn&&pr[j]*i<MAX;j++){
                if(i%pr[j]==0){
                    phi[pr[j]*i]=phi[i]*pr[j];
                    break;
                }
                else
                    phi[pr[j]*i]=phi[i]*(pr[j]-1);
            }
        }
}
int main(){
        return 0;
}
```

/*==*\
|欧拉函数求<=**n** 中与 **n** 互素的个数
|对正整数 **n**，欧拉函数是少于或等于 **n** 的数中与 **n** 互质的数的数目
==/

```
int phi (int n){
    int r[ 64 ] , p[ 64 ];
    int k = 0 , i , ans = 1;
    bool v;
    for (i = 2 ; i * i <= n ; ++ i){
            v = false;
            r[ k ] = 0;
            while (n % i == 0){
                p[ k ] = i;
              r[ k ] ++;
                n /= i;
                v = true;
```

```
        }
        if (v)
                k ++;
    }
    if (n != 1)
        p[ k ] = n , r[k ++] = 1;
    for (i = 0 ; i < k ; ++ i)
        ans *= (p[ i ] - 1) * (int)pow (p[ i ] * 1.0 , r[ i ] - 1);
    return ans;
}
/*==================================================*\
|欧拉函数求Σgcd(i, N) 1<=i<=N.求所有公约数的和
\*==================================================*/
#include <iostream>
#define MAX 50001
using namespace std;
int r[ MAX ] , p[ MAX ];
void solve (long long n) {                //核心部分
    int k = 0;
    long long i;
    double ans = n;
    bool v;
    for (i = 2 ; i * i <= n ; ++ i){
        v = false;
        r[ k ] = 0;
        while (n % i == 0){
            p[ k ] = i;
            r[ k ] ++;
            n /= i;
            v = true;
        }
        if (v)
                k ++;
    }
    if (n != 1)
        p[ k ] = n , r[k ++] = 1;
    for (i = 0 ; i < k ; ++ i)
        ans *= (1 + r[ i ] * (1 - 1.0 / p[ i ]));
    printf ("%.f\n" , ans);
}
int main(){
    long long n;
    while (scanf ("%lld" , &n) != EOF)
        solve(n);
    return 0;
}
/*==================================================*\
|数 n 约数的个数
\*==================================================*/
#include <iostream>
using namespace std;
typedef unsigned int uint;
uint divnum (uint n){
    int r[ 64 ] , p[ 64 ];
    int k = 0 , i;
    uint ans = 1;
    bool v;
    for (i = 2 ; i * i <= n ; ++ i){
        v = false;
        r[ k ] = 0;
        while (n % i == 0){
            p[ k ] = i;
            r[ k ] ++;
            n /= i;
            v = true;
        }
        if (v)
                k ++;
    }
    if (n != 1)
        p[ k ] = n , r[k ++] = 1;
    for (i = 0 ; i < k ; ++ i)
        ans *= (r[ i ] + 1);
    return ans;
}
int main(){
    int n;
    while (cin >> n)
        cout << divnum (n) << endl;
    return 0;
}
/*==================================================*\
|数 n 约数之和
\*==================================================*/
#include <iostream>
#include <cmath>
using namespace std;
typedef unsigned int uint;
uint rou (uint n)      {                  //包括本身
    int r[ 64 ] , p[ 64 ];
    int k = 0 , i;
    uint ans = 1;
    bool v;
    for (i = 2 ; i * i <= n ; ++ i){
        v = false;
        r[ k ] = 0;
        while (n % i == 0){
            p[ k ] = i;
            r[ k ] ++;
            n /= i;
            v = true;
        }
        if (v)
                k ++;
    }
    if (n != 1)
        p[ k ] = n , r[k ++] = 1;
    for (i = 0 ; i < k ; ++ i)
        ans *= int (pow (p[ i ] * 1.0 , r[ i ] + 1) - 1) / (p[ i ] - 1);
    return ans;
}
int main(){
    uint n;
    while (cin >> n)
        cout << rou(n) << endl;   //如果不包括本身，可以自行减去
    return 0;
}
/*==================================================*\
|中国剩余定理
|比如 N=3x+2,N=5y+3,N=7x+2，求解 n，最小的是 23
|输入方程的个数，依次输入方程的系数，（3，2）（5，3）（7，2）
\*==================================================*/
#include <iostream>
using namespace std;
__int64 m[1000];                //除数
__int64 r[1000];                //余数
__int64 X,Y;
__int64 Extende_GCD(__int64 a, __int64 b){
    if (b==0){
        X=1;
        Y=0;
        return a;
    }
    __int64 d=Extende_GCD(b,a%b);
    __int64 t=X;
    X=Y;
    Y=t-a/b*Y;
    return d;
}
__int64 CHN_Rnd(long len){
    __int64 M=1;
    long i;
    for (i=0;i<len;++i){
        M*=m[i];
    }
    __int64 res=0;
    for (i=0;i<len;++i){
        __int64 Mi=M/m[i];
        Extende_GCD(Mi,m[i]);
        res= (res+Mi*X*r[i])%M;
    }
    if (res<0){
```

```
            res+=M;
        }
        return res;
}
int main(){
        long n;
        while (scanf("%ld",&n)!=EOF){
            long i;
            for (i=0;i<n;++i){
                scanf("%I64d %I64d",&m[i],&r[i]);
            }
            printf("%I64d\n",CHN_Rnd(n));
        }
        return 0;
}
```

/*===\\

|母函数

===/

/*===\\

| 数量给定

===/

```
#include <iostream>
using namespace std;
#define MAX 1001
int main()
{
        int nn[ 11 ] , tt[ 11 ];
    // nn 代表东西的种类（如货币的分值 ， 1 分，2 分……），
    // tt 代表对应种类的数量
int dp[ MAX ] , bb[ MAX ];     // dp 存值 ， bb 充当媒介
int n , m , i , k , kk , j;
int N;
while (cin >> n >> m){
                N = 0;
                for (i = 1 ; i <= m ; ++ i)
                {
                        cin >> nn[ i ] >> tt[ i ];
                        N += nn[ i ] * tt[ i ];
                                        // N 代表总共价值不能超过题目给定数量
                }
                for (i = 0 ; i <= n ; ++ i)
                        dp[ i ] = bb[ i ] = 0;           // 付初值
                dp[ 0 ] = 1;                            // dp[ 0 ] = 1;
                for (i = 1 ; i <= m ; ++ i)
                {
                        for (j = 0 ; j <= n ; ++ j)
                        {
                                if (!dp[ j ])
                                        continue;
                                for (k = 0 , kk = 0; k + j <= n && kk <= tt[ i ];
++ kk , k += nn[ i ])                           // kk 给定的值

                                        bb[k + j] += dp[ j ];
                        }
                        for (j = 0 ; j <= n ; ++ j)
                        {
                                dp[ j ] = bb[ j ];
                                bb[ j ] = 0;
                        }
                }
                cout << dp[ n ] << endl;
                        //循环一次就能算出 dp[ 0 ]-----dp[ n ]的所有值
        }
        return 0;
}
```

/*===\\

| 数量无穷

===/

```
include <iostream>
using namespace std;
#define MAX 120
int main(){
    int c[MAX + 1] , bb[MAX + 1];
    int i , j , k , n;
    for (i = 0 ; i <= MAX ; ++ i){
        c[ i ] = 1;
        bb[ i ] = 0;
    }
    for (i = 2 ; i <= MAX ; ++ i)
    {
        for (j = 0 ; j <= MAX ; ++ j)
            for (k = 0 ; k + j <= MAX ; k += i)    // k += i , i 与货币的变化有关
                bb[k + j] += c[ j ];
            for (j = 1 ; j <= MAX ; ++ j)
            {
                c[ j ] = bb[ j ];
                bb[ j ] = 0;
            }
    }
    while (cin >> n)
        cout << c[ n ] << endl;
    return 0;
}
```

/*===\\

|高斯消元（线性方程组求秩）

|用于求整数解得方程组.

|浮点数线性方程组的求法类似，但是要在判断是否为 0 时，加入 EPS，

|以消除精度问题

===/

```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;
const int maxn = 105;
int equ, var;
// 有 equ 个方程,var 个变元。增广阵行数为 equ, 分别为 0 到 equ - 1,
列数为 var + 1, 分别为 0 到 var.
int a[maxn][maxn];
int x[maxn];                            // 解集.
bool free_x[maxn];                      // 判断是否是不确定的变元.
int free_num;
void Debug(void){
        int i, j;
        for (i = 0; i < equ; i++){
            for (j = 0; j < var + 1; j++){
                cout << a[i][j] << " ";
            }
            cout << endl;
        }
        cout << endl;
}
inline int gcd(int a, int b){
        int t;
        while (b != 0){
            t = b;
            b = a % b;
            a = t;
        }
        return a;
}
inline int lcm(int a, int b){
        return a * b / gcd(a, b);
}
// 高斯消元法解方程组(Gauss-Jordan elimination).(-2 表示有浮点数
解，但无整数解，-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并
返回自由变元的个数)
int Gauss(void){
        int i, j, k;
        int max_r;                      // 当前这列绝对值最大的行.
        int col;                        // 当前处理的列.
        int ta, tb;
        int LCM;
        int temp;
        int free_x_num;
        int free_index;                 // 转换为阶梯阵.
        col = 0;                        // 当前处理的列.
        for (k = 0; k < equ && col < var; k++, col++){
            // 枚举当前处理的行.
            // 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除
法时减小误差)
            max_r = k;
```

```
            for (i = k + 1; i < equ; i++){
                if (abs(a[i][col]) > abs(a[max_r][col])) max_r = i;
            }
            if (max_r != k){ // 与第 k 行交换.
                for (j = k; j < var + 1; j++) swap(a[k][j], a[max_r][j]);
            }
            if (a[k][col] == 0) {
        // 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
                k--; continue;
            }
            for (i = k + 1; i < equ; i++){ // 枚举要删去的行.
                if (a[i][col] != 0) {
                    LCM = lcm(abs(a[i][col]), abs(a[k][col]));
                  ta = LCM / abs(a[i][col]), tb = LCM / abs(a[k][col]);
                    if (a[i][col] * a[k][col] < 0) tb = -tb; // 异号的情况是两个数相加.
                    for (j = col; j < var + 1; j++) {
                        a[i][j] = a[i][j] * ta - a[k][j] * tb;
                    }
        }
            }
        }
        Debug();
        // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
        for (i = k; i < equ; i++){
         // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
            if (a[i][col] != 0) return -1;
        }
        // 2. 无穷解的情况: 在 var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.
        // 且出现的行数即为自由变元的个数.
        if (k < var){
         // 首先，自由变元有 var - k 个，即不确定的变元至少有 var - k 个.
            for (i = k - 1; i >= 0; i--) {
                // 第 i 行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第 k 行到第 equ 行.
                // 同样，第 i 行一定不会是(0, 0, ..., a), a != 0 的情况，这样的无解的.
                free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它们仍然为不确定的变元.
                for (j = 0; j < var; j++){
                    if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
                }
                if (free_x_num > 1) continue; // 无法求解出确定的变元.
                // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
                temp = a[i][var];
                for (j = 0; j < var; j++){
                    if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
                }
                x[free_index] = temp / a[i][free_index]; // 求出该变元.
                free_x[free_index] = 0; // 该变元是确定的.
            }
            return var - k; // 自由变元有 var - k 个.
        }
        // 3. 唯一解的情况: 在 var * (var + 1)的增广阵中形成严格的上三角阵.
        // 计算出 Xn-1, Xn-2 ... X0.
        for (i = var - 1; i >= 0; i--){
            temp = a[i][var];
            for (j = i + 1; j < var; j++){
                if (a[i][j] != 0) temp -= a[i][j] * x[j];
            }
            if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
            x[i] = temp / a[i][i];
        }
    return 0;
}
int main(void){
        freopen("Input.txt", "r", stdin);
        int i, j;
         while (scanf("%d %d", &equ, &var) != EOF){
             memset(a, 0, sizeof(a));
        memset(x, 0, sizeof(x));
        memset(free_x, 1, sizeof(free_x)); // 一开始全是不确定的变元.
             for (i = 0; i < equ; i++){
                 for (j = 0; j < var + 1; j++)
                 {
                     scanf("%d", &a[i][j]);
                 }
             }
//           Debug();
             free_num = Gauss();
             if (free_num == -1) printf("无解!\n");
        else if (free_num == -2) printf("有浮点数解，无整数解!\n");
             else if (free_num > 0){
                 printf("无穷多解! 自由变元个数为%d\n", free_num);
                 for (i = 0; i < var; i++){
                     if (free_x[i]) printf("x%d 是不确定的\n", i + 1);
                     else printf("x%d: %d\n", i + 1, x[i]);
                 }
             }
             else{
                 for (i = 0; i < var; i++){
                     printf("x%d: %d\n", i + 1, x[i]);
                 }
             }
             printf("\n");
         }
         return 0;
}
```

```
/*==================================================*\
|皮克公式
\*==================================================*/
#include<iostream>
int gcd(int a,int b){
   if(a<0) a=-a;
   if(b<0) b=-b;
   if(b==0) return a;
   else   gcd(b,a%b);
}
int main(){
   int t,n,i,j;
   int x[101]={0},y[101]={0};
   int dx,dy,e,l;
   double sum,s;
   cin>>t;
   for(j=1;j<=t;j++){
        e=0;sum=0;
        cin>>n;
        for(i=1;i<=n;i++){
         cin>>dx>>dy;
           x[i]=x[i-1]+dx;
           y[i]=y[i-1]+dy;
           e+=gcd(dx,dy);
        }
        for(i=1;i<=n;i++){
            if(i==n) {
               x[i+1]=x[1];
               y[i+1]=y[1];
            }
         sum +=x[i]*y[i+1]-x[i+1]*y[i];
      }
        s=0.5*fabs(sum);
        l=int(s+1-e/2.0+0.5);
           //皮克公式: S = a + b/2 - 1 ;其面积 S 和内部格点数目 a、边上格点数目 b 的关系:
           printf("Scenario #%d:\n",j);
         printf("%d %d %.1f\n",l,e,s);
                cout<<endl;
   }
   return 0;}
```

```
/*==================================================*\
|扩展的 Euclid 算法
|返回 a.b 的最大公约数，   并使 ax+by=d;
\*==================================================*/
long exEuclid(long a, long b, long & x, long & y)
```

```
{
        long tmp,d;
        if(b==0)
        {
                x=1;
                y=0;
                return a;
        }
        d=exEuclid(b, a%b, x,y);
        tmp=x;
        x=y;
        y=tmp-a/b*y;
      return d;
}
/*==================================================*\
|解线性同余方程 ax=b(mod n)
|返回最小的 x
\*==================================================*/
long   modu(long a, long b, long n)
{
        long d,x=1,y=0;
          d=exEuclid(a,n,x,y);
          x=x*(b/d);
          x=(x%(n/d)+n/d)%(n/d);
          return x;
}
/*==================================================*\
|筛法求素数
\*==================================================*/
const maxn=100000;
bool   prime[maxn+1];
void   searchprime(long b[],long & k){
     int i ,j;
     memset(prime,0,sizeof(prime));
     prime[1]=1;
     for(i=2; i<sqrt(maxn); i++)
             if(!prime[i])
             {
                j=i*2;
                    while(j<=maxn)
                    {
                            prime[j]=1;
                            j+=i;
                    }
             }
   j=0;
   for(i=1; i<maxn; i++)
           if(prime[i]==0)
                         b[j++]=i;
     k=j;
}
/*==================================================*\
|判定素数  素数表
\*==================================================*/
bool isPrime(long x,long b[]){
        int i;
        i=1;
        while(b[i]*b[i]<=x){
           if(x%b[i]==0)
                     return 0;
             i++;
        }
        return true;
}
/*==================================================*\
|判定素数，概率方法
\*==================================================*/
bool passTest(long n){
        long l ,m,b,i,k;
        m=n-1;
        l=0;
        while(m%2==0)
        {
                l++;
              m/=2;
        }
        b=rand()%n+1;
          if(modexp(b,m,n)==1) return 1;
        k=m;
          for(i=0; i<l; i++)
          {
                  if(modexp(b,k,n)==n-1) return 1;
                  k*=2;
          }
          return 0;
}
/*==================================================*\
| 筛素数  [1..n]
\*==================================================*/
bool is[N]; int prm[M];
int getprm(int n){
        int i, j, k = 0;
        int s, e = (int)(sqrt(0.0 + n) + 1);
        memset(is, 1, sizeof(is));
        prm[k++] = 2; is[0] = is[1] = 0;
        for (i = 4; i < n; i += 2) is[i] = 0;
              for (i = 3; i < e; i += 2) if (is[i]) {
                      prm[k++] = i;
                      for (s = i * 2, j = i * i; j < n; j += s)
                               is[j] = 0;
                // 因为 j 是奇数，所以+奇数 i 后是偶数，不必处理！
              }
      for ( ; i < n; i += 2) if (is[i]) prm[k++] = i;
return k;                         // 返回素数的个数
}
/*==================================================*\
| 高效求小范围素数  [1..n]
\*==================================================*/
int prime[500],num,boo[2500];
for( i=2; i <= 2300; i++ ) boo[i] = 0;
for( i=2; i <= 50; i++ )
      for( k=i*2; k <= 2300; k += i )
                 boo[k] = 1;
int num = 0;
for( i=2; i <= 2300; i++ )
               if( boo[i] == 0 ) prime[num++] = i;
/*==================================================*\
| 随机素数测试(伪素数原理)
| CALL: bool res = miller(n);
| 快速测试 n 是否满足素数的'必要'条件，出错概率很小;
| 对于任意奇数  n>2  和正整数  s，算法出错概率  <= 2^(-s);
\*==================================================*/
int witness(int a, int n)
{
             int x, d=1, i = ceil(log(n - 1.0) / log(2.0)) - 1;
             for ( ; i >= 0; i--) {
                x = d; d = (d * d) % n;
                if (d==1 && x!=1 && x!=n-1) return 1;
                if (((n-1) & (1<<i)) > 0) d = (d * a) % n;
              }
      return (d == 1 ? 0 : 1);
}
int miller(int n, int s = 50)
{
           if (n == 2) return 1;
           if ((n % 2) == 0) return 0;
           int j, a;
for (j = 0; j < s; j++) {
      a = rand() * (n-2) / RAND_MAX + 1;
// rand()只能随机产生[0, RAND_MAX)内的整数
//  而且这个 RAND_MAX 只有 32768 直接%n 的话永远也产生不了
// [RAND-MAX, n)之间的数
           if (witness(a, n)) return 0;
      }
return 1;
}
/*==================================================*\
| 组合数学相关
\*==================================================*/
```

1. {1, 2, ... n}的 r 组合 a1, a2, ... ar 出现在所有 r 组合中的字典序位置编号, C(n, m)表示 n 中取 m 的组合数; index = C(n, r) - C(n - a1, r) - C(n - a2, r-1) - ... - C(n - ar, 1)

2. k * C(n, k) = n * C(n-1, k-1);
   C(n, 0) + C(n, 2) + ... = C(n, 1) + C(n, 3) + ...
   1 * C(n, 1) + 2 * C(n, 2) + ... + n * C(n, n) = n * 2^(n-1)

3. Catalan 数: C_n = C(2*n, n) / (n+1)
   C_n = (4*n-2)/(n+1) * C_n-1
   C_1 = 1

4. 第二类 Stirling 数: S(p, k) = k * S(p-1, k) + S(p-1, k-1).
   S(p, 0) = 0, (p>=1); S(p, p) = 1, (p>=0);
   且有 S(p, 1) = 1, (p>=1);
   S(p, 2) = 2^(p-1) - 1, (p>=2);
   S(p, p-1) = C(p, 2);

含义: 将 p 个元素划分到 k 个同样的盒子，每个盒子非空的方法数.

```
          1       k
S(p, k) = --- * sigma((-1)^t * C(k, t) * (k-t)^p)
          k!     t=0
```

5. Bell 数: B_p = S(p, 0) + S(p, 1) + ... + S(p, p)
   B_p = C(p-1,0)*B_0 + C(p-1,1)*B_1 + ... C(p-1,p-1)*B_(p-1)

6. 第一类 stirling 数:
   s(p, k)是将 p 个物体排成 k 个非空的循环排列的方法数.
   (或者: 把 p 个人排成 k 个非空圆圈的方法数)
   s(p, k) = (p-1) * s(p-1, k) + s(p-1, k-1);

```
/*==================================================*\
| 集合划分问题
| n 元集合分划为 k 类的方案数记为 S(n,k),称为第二类 Stirling 数。
| 如{A,B,C}可以划分{{A},{B},{C}}, {{A,B},{C}}, {{B,C},{A}},
| {{A,C},{B}},{{A,B,C}}。即一个集合可以划分为不同集合（1…n 个）
| 的种类数 HDU 一卡通大冒险
| CALL: compute(N); 每当输入一个 n，输出 B[n]
\*==================================================*/
const int N = 2001;
int data[N][N], B[N];
void NGetM(int m, int n)          // m 个数 n 个集合
{                     // data[i][j]:i 个数分成 j 个集合
        int min, i, j;
        data[0][0] = 1; //
        for( i = 1; i <= m; ++i ) data[i][0] = 0;
        for( i = 0; i <= m; ++i ) data[i][i+1] = 0;
        for( i = 1; i <= m; ++i ){
            if( i < n ) min = i;
            else min = n;
            for( j = 1; j <= min; ++j )
                data[i][j] = (j*data[i-1][j] + data[i-1][j-1]);
        }
}
void compute(int m){                  // b[i]:Bell 数
        NGetM(m, m);
        memset(B, 0, sizeof(B));
        int i, j;
        for( i=1; i <= m; ++i )
        for( j=0; j <= i; ++j ) B[i] += data[i][j];
}
/*==================================================*\
| 组合数 C(n, r)
\*==================================================*/
int com(int n, int r){                // return C(n, r)
        if( n-r > r ) r = n-r;        // C(n, r) = C(n, n-r)
            int i, j, s = 1;
        for( i=0, j=1; i < r; ++i ){
                s *= (n-i);
                for( ; j <= r && s%j == 0; ++j ) s /= j;
        }
    return s;
}
/*==================================================*\
| polya 定理
|一个项链有 n 个珠子，用 k 种颜色涂染会形成多少种不同的项链；对于
|异同的定义有三种：
|1.旋转得来的为同一种，翻转得来的也视为同一种
|2.旋转得来的为同一种，翻转得来的视为另一种
|3.旋转得来的为另一种，翻转得来的视为同一种
\*==================================================*/
#include <memory.h>
#include <math.h>
double ploya1(int c,int n) {            //旋转和翻转视为相同
        int i,j,k,x,y;
        double t=0.0;
        memset(b1,0,sizeof(b1));
        memset(b2,0,sizeof(b2));
        for (i=0;i<=n-1;i++){
            for (x=y=j=0;j<=n-1;j++){
                if (!b1[(i+j)%n])
                    for (x++,k=(i+j)%n;!b1[k];k=(i+k)%n)
                        b1[k]=true;
                if (!b2[n-1-(i+j)%n])
                for(y++,k=n-1-(i+j)%n;!b2[k];k=n-1-(i+k)%n)
                        b2[k]=true;
            }
            t=t+pow(c,x)+pow(c,y);
        }
        return t/(2*n);
}
double ploya2(int c,int n) {            //旋转视为相同，翻转为异
        int i,j,k,x;
        double t=0.0;
        memset(bj,0,sizeof(bj));
        for (i=0;i<=n-1;i++){
            for (x=y=j=0;j<=n-1;j++)
                if (!bj[(i+j)%n])
                    for (x++,k=(i+j)%n;!bj[k];k=(i+k)%n)
                        bj[k]=true;
            t=t+pow(c,x);
        }
        return t/n;
}
double ploya3(int c,int n) {        //翻转视为相同，旋转为异
        int x=n/2;
        if (n%2)        x++;
        return (pow(c,n)+pow(c,x))/2;
}
/*==================================================*\
| 线性方程组 a[][]x[]=b[]
\*==================================================*/
#define MAXN 100
#define fabs(x) ((x)>0?(x):-(x))
#define eps 1e-10
//列主元 gauss 消去求解 a[][]x[]=b[]
//返回是否有唯一解,若有解在 b[]中
int gauss_cpivot(int n,double a[][MAXN],double b[]){
      int i,j,k,row;
      double maxp,t;
      for (k=0;k<n;k++){
          for (maxp=0,i=k;i<n;i++)
             if (fabs(a[i][k])>fabs(maxp))
                 maxp=a[row=i][k];
             if (fabs(maxp)<eps)
                 return 0;
             if (row!=k){
                 for (j=k;j<n;j++)
                     t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;
                     t=b[k],b[k]=b[row],b[row]=t;
             }
            for (j=k+1;j<n;j++){
                a[k][j]/=maxp;
            for (i=k+1;i<n;i++)
                a[i][j]-=a[i][k]*a[k][j];
            }
           b[k]/=maxp;
           for (i=k+1;i<n;i++)
              b[i]-=b[k]*a[i][k];
      }
          for (i=n-1;i>=0;i--)
             for (j=i+1;j<n;j++)
                 b[i]-=a[i][j]*b[j];
return 1;
}
//全主元 gauss 消去解 a[][]x[]=b[]
//返回是否有唯一解,若有解在 b[]中
```

```
int gauss_tpivot(int n,double a[][MAXN],double b[]){
        int i,j,k,row,col,index[MAXN];
        double maxp,t;
        for (i=0;i<n;i++)
            index[i]=i;
        for (k=0;k<n;k++){
            for (maxp=0,i=k;i<n;i++)
                for (j=k;j<n;j++)
                    if (fabs(a[i][j])>fabs(maxp))
                        maxp=a[row=i][col=j];
                    if (fabs(maxp)<eps)
                        return 0;
                    if (col!=k){
                        for (i=0;i<n;i++)
                            t=a[i][col],a[i][col]=a[i][k],a[i][k]=t;
                        j=index[col],index[col]=index[k],index[k]=j;
                     }
                    if (row!=k){
                        for (j=k;j<n;j++)
                            t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;
                            t=b[k],b[k]=b[row],b[row]=t;
                    }
                for (j=k+1;j<n;j++){
                    a[k][j]/=maxp;
                for (i=k+1;i<n;i++)
                    a[i][j]-=a[i][k]*a[k][j];
            }
        b[k]/=maxp;
            for (i=k+1;i<n;i++)
                b[i]-=b[k]*a[i][k];
    }
        for (i=n-1;i>=0;i--)
            for (j=i+1;j<n;j++)
                b[i]-=a[i][j]*b[j];
        for (k=0;k<n;k++)
            a[0][index[k]]=b[k];
        for (k=0;k<n;k++)
            b[k]=a[0][k];
 return 1;
}
/*==================================================*\
| 追赶法解周期性方程
周期性方程定义：
| a1 b1 c1 ...                  |           = x1
|     a2 b2 c2 ...              |           = x2
|                 ...           |           = ...
|cn-1      ...    an-1 bn-1     |           = xn-1
|    bn cn                 an   |           = xn
输入：a[],b[],c[],x[]
输出：求解结果 X 在 x[]中
\*==================================================*/
void run(){
    c[0] /= b[0]; a[0] /= b[0]; x[0] /= b[0];
    for (int i = 1; i < N - 1; i ++) {
        double temp = b[i] - a[i] * c[i - 1];
        c[i] /= temp;
        x[i] = (x[i] - a[i] * x[i - 1]) / temp;
        a[i] = -a[i] * a[i - 1] / temp;
     }
    a[N - 2] = -a[N - 2] - c[N - 2];
    for (int i = N - 3; i >= 0; i --) {
        a[i] = -a[i] - c[i] * a[i + 1];
        x[i] -= c[i] * x[i + 1];
     }
    x[N - 1] -= (c[N - 1] * x[0] + a[N - 1] * x[N - 2]);
    x[N - 1] /= (c[N - 1] * a[0] + a[N - 1] * a[N - 2] + b[N- 1]);
    for (int i = N - 2; i >= 0; i --)
      x[i] += a[i] * x[N - 1];
}
/*==================================================*\
| 二分法
|HUTC   赶公交
\*==================================================*/
#include<stdio.h>
#include<math.h>
int main()
{
    long i,j,n,time;
    double x1,y1,x2,y2,right,left,v1,v2,sum,middle;
    scanf("%d",&n);
    while(n--){
        scanf("%lf%lf",&x1,&y1);
        scanf("%lf%lf",&x2,&y2);
        scanf("%lf%lf",&v1,&v2);
        scanf("%d",&time);
        left=x1;right=x2;
        while(right - left > 0.000001){
            middle=(left+right)/2.0;

    if(sqrt(y1*y1+(middle-x1-0.0001)*(middle-x1-0.0001))/v1 +
sqrt(y2*y2+(x2-middle+0.0001)*(x2-middle+0.0001))/v2 >

    sqrt(y1*y1+(middle-x1+0.0001)*(middle-x1+0.0001))/v1 +
sqrt(y2*y2+(x2-middle-0.0001)*(x2-middle-0.0001))/v2)
                left = middle;
            else right = middle;
        }
        sum=sqrt(y1*y1+(right-x1)*(right-x1))/v1 +
sqrt(y2*y2+(x2-right)*(x2-right))/v2;
        if(sum > time)printf("NO\n");
        else printf("YES\n");
    }
}
```

<table>
<tr><td rowspan="3">三角不等式</td><td colspan="2">$|a+b|\leqslant|a|+|b|$</td><td>$|a-b|\leqslant|a|+|b|$</td><td>$|a|\leqslant b \Leftrightarrow -b\leqslant a\leqslant b$</td></tr>
<tr><td colspan="2"></td><td></td><td></td></tr>
<tr><td colspan="2">$|a-b|\geqslant|a|-|b|$</td><td>$-|a|\leqslant a\leqslant|a|$</td><td></td></tr>
<tr><td>一元二次方程的解</td><td colspan="2">$-b+\sqrt{(b^2-4ac)}/2a$</td><td>$-b-\sqrt{(b^2-4ac)}/2a$</td><td></td></tr>
<tr><td colspan="2">根与系数的关系</td><td>X1+X2=-b/a</td><td>X1*X2=c/a</td><td>注：韦达定理</td></tr>
<tr><td rowspan="3">数列前 n 项和</td><td colspan="2">$1+2+3+4+5+6+7+8+9+\cdots+n=n(n+1)/2$</td><td colspan="2">$1+3+5+7+9+11+13+15+\cdots+(2n-1)=n^2$</td></tr>
<tr><td colspan="2">$2+4+6+8+10+12+14+\cdots+(2n)=n(n+1)$</td><td colspan="2">$1^2+2^2+3^2+4^2+5^2+6^2+7^2+8^2+\cdots+n^2=n(n+1)(2n+1)/6$</td></tr>
<tr><td colspan="2">$1^3+2^3+3^3+4^3+5^3+6^3+\cdots n^3=n^2(n+1)^2/4$</td><td colspan="2">$1*2+2*3+3*4+4*5+5*6+6*7+\cdots+n(n+1)=n(n+1)(n+2)/3$</td></tr>
<tr><td colspan="5">常用数学公式表:解析几何公式</td></tr>
<tr><td>圆的标准方程</td><td colspan="2">$(x-a)^2+(y-b)^2=r^2$</td><td colspan="2">注：（a,b）是圆心坐标</td></tr>
<tr><td>圆的一般方程</td><td colspan="2">$x^2+y^2+Dx+Ey+F=0$</td><td colspan="2">注：$D^2+E^2-4F>0$</td></tr>
<tr><td>抛物线标准方程</td><td>$y^2=2px$</td><td>$y^2=-2px$</td><td>$x^2=2py$</td><td>$x^2=-2py$</td></tr>
<tr><td colspan="5">常用数学公式表:几何图形公式</td></tr>
<tr><td>直棱柱侧面积</td><td colspan="2">S=c*h</td><td>斜棱柱侧面积</td><td>S=c’*h</td></tr>
<tr><td>正棱锥侧面积</td><td colspan="2">S=1/2c*h’</td><td>正棱台侧面积</td><td>S=1/2(c+c’)h’</td></tr>
<tr><td>圆台侧面积</td><td colspan="2">S=1/2(c+c’)l=pi(R+r)l</td><td>球的表面积</td><td>$S=4pi*r^2$</td></tr>
<tr><td>圆柱侧面积</td><td colspan="2">S=c*h=2pi*h</td><td>圆锥侧面积</td><td>S=1/2*c*l=pi*r*l</td></tr>
<tr><td>弧长公式</td><td colspan="2">l=a*r（a 是圆心角的弧度数 r>0）</td><td>扇形面积公式</td><td>s=1/2*l*r</td></tr>
<tr><td>锥体体积公式</td><td colspan="2">V=1/3*S*H</td><td>圆锥体体积公式</td><td>$V=1/3*pi*r^2h$</td></tr>
<tr><td>柱体体积公式</td><td colspan="2">V=s*h</td><td>圆柱体</td><td>$V=pi*r^2h$</td></tr>
<tr><td>斜棱柱体积</td><td colspan="2">V=S’L（S’是直截面面积，L 是侧棱长）</td><td colspan="2">注：pi=acos(-1.0);</td></tr>
</table>

完数

6，28、496，8128，33550336，8589869056(10 位)，137438691328(12 位)，
2305843008139952128(19 位)……

e

e=2.71828 18284 59045 23536 02874 71352 66249 77572 47093 69995 95749 66967 62772 40766 30353 54759 45713 82178 52516 64274 27466 39193 20030 59921 81741 35966 29043 57290 03342 95260 59563 07381 32328 62794 34907 63233 82988 07531 95251 01901 15738 34187 93070 21540 89149 93488 41675 09244 76146 06680 82264 80016 84774 11853 74234 54424 37107 53907 77449 92069 55170 27618 38606 26133 13845 83000 75204 49338 26560 29760 67371 13200 70932 87091 27443 74704 72306 96977 20931 01416 92836 81902 55151 08657 46377 21112 52389 78442 50569 53696 77078 54499 69967 94686 44549 05987 93163 68892 30098 79312 77361 78215 42499 92295 76351 48220 82698 95193 66803 31825 28869 39849 64651 05820 93923 98294 88793 32036 25094 43117 30123 81970 68416 14039 70198 37679 32068 32823 76464 80429 53118 02328 78250 98194 55815 30175 67173 61332 06981 12509 96181 88159 30416 90351 59888 85193 45807 27386 67385 89422 87922 84998 92086 80582 57492 79610 48419 84443 63463 24496 84875 60233 62482 70419 78623 20900 21609 90235 30436 99418 49146 31409 34317 38143 64054 62531 52096 18369 08887 07016 76839 64243 78140 59271 45635 49061 30310 72085 10383 75051 01157 47704 17189 86106 87396 96552 12671 54688 95703 50354 02123 40784 98193 34321 06817 01210 05627 88023 51930 33224 74501 58539 04730 41995 77770 93503 66041 69973 29725 08868 76966 40355 57071 62268 44716 25607 98826 51787 13419 51246 65201 03059 21236 67719 43252 78675 39855 89448 96970 96409 75459 18569 56380 23637 01621 12047 74272 28364 89613 42251 64450 78182 44235 29486 36372 14174 02388 93441 24796 35743 70263 75529 44483 37998 01612 54922 78509 25778 25620 92622 64832 62779 33386 56648 16277 25164 01910 59004 91644 99828 93150 56604 72580 27786 31864 15519 56532 44258 69829 46959 30801 91529 87211 72556 34754 63964 47910 14590 40905 86298 49679 12874 06870 50489 58586 71747 98546 67757 57320 56812 88459 20541 33405 39220 00113 78630 09455 60688 16674 00169 84205 58040 33637 95376 45203 04024 32256 61352 78369 51177 88386 38744 39662 53224 98506 54995 88623 42818 99707 73327 61717 83928 03494 65014 34558 89707 19425 86398 77275 47109 62953 74152 11151 36835 06275 26023 26484 72870 39207 64310 05958 41166 12054 52970 30236 47254 92966 69381 15137 32275 36450 98889 03136 02057 24817 65851 18063 03644 28123 14965 50704 75102 54465 01172 72115 55194 86685 08003 68532 28183 15219 60037 35625 27944 95158 28418 82947 87610 85263 98139

π

π≈3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745024811174502841027019385211055596446229489549303819644288109756659334461284756482337867831652712019091456485669234603486104543266482133936072602491412737245870066063155881748815209209628292540917153643678925903600113305305488204665213841469519415116094330572703657595919530921861173819326117931051185480744623799627495673518857527248912279381830119491298336733624406566430860213949463952247371907021798609437027705392171762931767523846748184676694051320005681271452635608277857713427577896091736371787214684409012249534301465495853710507922796892589235420199561121290219608640344181598136297747713099605187072113499999983729780499510597317328160963185950244594553469083026425223082533446850352619311881710100031378387528865875332083814206171776691473035982534904287554687311595628638823537875937519577818577805321712268066130019278766111959092164201989

黄金分割 (sqrt(5)-1)/2

0.6180339887 4989484820 4586834365 6381177203 0917980576
2862135448 6227052604 6281890244 9707207204 1893911374
8475408807 5386891752 1266338622 2353693179 3180060766
7263544333 8908659593 9582905638 3226613199 2829026788
0675208766 8925017116 9620703222 1043216269 5486262963
1361443814 9758701220 3408058879 5445474924 6185695364
8644492410 4432077134 4947049565 8467885098 7433944221
2544877066 4780915884 6074998871 2400765217 0575179788
3416625624 9407589069 7040002812 1042762177 1117778053
1531714101 1704666599 1466979873 1761356006 7087480710
1317952368 9427521948 4353056783 0022878569 9782977834
7845878228 9110976250 0302696156 1700250464 3382437764
8610283831 2683303724 2926752631 1653392473 1671112115
8818638513 3162038400 5222165791 2866752946 5490681131
7159934323 5973494985 0904094762 1322298101 7261070596
1164562990 9816290555 2085247903 5240602017 2799747175
3427775927 7862561943 2082750513 1218156285 5122248093
9471234145 1702237358 0577278616 0086883829 5230459264
7878017889 9219902707 7690389532 1968198615 1437803149
9741106926 0886742962 2675756052 3172777520 3536139362

素数---1230 个

1: 2	2: 3	3: 5	4: 7	5: 11
6: 13	7: 17	8: 19	9: 23	10: 29
11: 31	12: 37	13: 41	14: 43	15: 47
16: 53	17: 59	18: 61	19: 67	20: 71
21: 73	22: 79	23: 83	24: 89	25: 97
26: 101	27: 103	28: 107	29: 109	30: 113
31: 127	32: 131	33: 137	34: 139	35: 149
36: 151	37: 157	38: 163	39: 167	40: 173
41: 179	42: 181	43: 191	44: 193	45: 197
46: 199	47: 211	48: 223	49: 227	50: 229
51: 233	52: 239	53: 241	54: 251	55: 257
56: 263	57: 269	58: 271	59: 277	60: 281
61: 283	62: 293	63: 307	64: 311	65: 313
66: 317	67: 331	68: 337	69: 347	70: 349
71: 353	72: 359	73: 367	74: 373	75: 379
76: 383	77: 389	78: 397	79: 401	80: 409
81: 419	82: 421	83: 431	84: 433	85: 439
86: 443	87: 449	88: 457	89: 461	90: 463
91: 467	92: 479	93: 487	94: 491	95: 499
96: 503	97: 509	98: 521	99: 523	100: 541
101: 547	102: 557	103: 563	104: 569	105: 571
106: 577	107: 587	108: 593	109: 599	110: 601
111: 607	112: 613	113: 617	114: 619	115: 631
116: 641	117: 643	118: 647	119: 653	120: 659
121: 661	122: 673	123: 677	124: 683	125: 691
126: 701	127: 709	128: 719	129: 727	130: 733
131: 739	132: 743	133: 751	134: 757	135: 761
136: 769	137: 773	138: 787	139: 797	140: 809
141: 811	142: 821	143: 823	144: 827	145: 829
146: 839	147: 853	148: 857	149: 859	150: 863
151: 877	152: 881	153: 883	154: 887	155: 907
156: 911	157: 919	158: 929	159: 937	160: 941
161: 947	162: 953	163: 967	164: 971	165: 977
166: 983	167: 991	168: 997	169: 1009	170: 1013
171: 1019	172: 1021	173: 1031	174: 1033	175: 1039
176: 1049	177: 1051	178: 1061	179: 1063	180: 1069
181: 1087	182: 1091	183: 1093	184: 1097	185: 1103
186: 1109	187: 1117	188: 1123	189: 1129	190: 1151
191: 1153	192: 1163	193: 1171	194: 1181	195: 1187
196: 1193	197: 1201	198: 1213	199: 1217	200: 1223
201: 1229	202: 1231	203: 1237	204: 1249	205: 1259
206: 1277	207: 1279	208: 1283	209: 1289	210: 1291
211: 1297	212: 1301	213: 1303	214: 1307	215: 1319
216: 1321	217: 1327	218: 1361	219: 1367	220: 1373
221: 1381	222: 1399	223: 1409	224: 1423	225: 1427

226: 1429	227: 1433	228: 1439	229: 1447	230: 1451
231: 1453	232: 1459	233: 1471	234: 1481	235: 1483
236: 1487	237: 1489	238: 1493	239: 1499	240: 1511
241: 1523	242: 1531	243: 1543	244: 1549	245: 1553
246: 1559	247: 1567	248: 1571	249: 1579	250: 1583
251: 1597	252: 1601	253: 1607	254: 1609	255: 1613
256: 1619	257: 1621	258: 1627	259: 1637	260: 1657
261: 1663	262: 1667	263: 1669	264: 1693	265: 1697
266: 1699	267: 1709	268: 1721	269: 1723	270: 1733
271: 1741	272: 1747	273: 1753	274: 1759	275: 1777
276: 1783	277: 1787	278: 1789	279: 1801	280: 1811
281: 1823	282: 1831	283: 1847	284: 1861	285: 1867
286: 1871	287: 1873	288: 1877	289: 1879	290: 1889
291: 1901	292: 1907	293: 1913	294: 1931	295: 1933
296: 1949	297: 1951	298: 1973	299: 1979	300: 1987
301: 1993	302: 1997	303: 1999	304: 2003	305: 2011
306: 2017	307: 2027	308: 2029	309: 2039	310: 2053
311: 2063	312: 2069	313: 2081	314: 2083	315: 2087
316: 2089	317: 2099	318: 2111	319: 2113	320: 2129
321: 2131	322: 2137	323: 2141	324: 2143	325: 2153
326: 2161	327: 2179	328: 2203	329: 2207	330: 2213
331: 2221	332: 2237	333: 2239	334: 2243	335: 2251
336: 2267	337: 2269	338: 2273	339: 2281	340: 2287
341: 2293	342: 2297	343: 2309	344: 2311	345: 2333
346: 2339	347: 2341	348: 2347	349: 2351	350: 2357
351: 2371	352: 2377	353: 2381	354: 2383	355: 2389
356: 2393	357: 2399	358: 2411	359: 2417	360: 2423
361: 2437	362: 2441	363: 2447	364: 2459	365: 2467
366: 2473	367: 2477	368: 2503	369: 2521	370: 2531
371: 2539	372: 2543	373: 2549	374: 2551	375: 2557
376: 2579	377: 2591	378: 2593	379: 2609	380: 2617
381: 2621	382: 2633	383: 2647	384: 2657	385: 2659
386: 2663	387: 2671	388: 2677	389: 2683	390: 2687
391: 2689	392: 2693	393: 2699	394: 2707	395: 2711
396: 2713	397: 2719	398: 2729	399: 2731	400: 2741
401: 2749	402: 2753	403: 2767	404: 2777	405: 2789
406: 2791	407: 2797	408: 2801	409: 2803	410: 2819
411: 2833	412: 2837	413: 2843	414: 2851	415: 2857
416: 2861	417: 2879	418: 2887	419: 2897	420: 2903
421: 2909	422: 2917	423: 2927	424: 2939	425: 2953
426: 2957	427: 2963	428: 2969	429: 2971	430: 2999
431: 3001	432: 3011	433: 3019	434: 3023	435: 3037
436: 3041	437: 3049	438: 3061	439: 3067	440: 3079
441: 3083	442: 3089	443: 3109	444: 3119	445: 3121
446: 3137	447: 3163	448: 3167	449: 3169	450: 3181
451: 3187	452: 3191	453: 3203	454: 3209	455: 3217
456: 3221	457: 3229	458: 3251	459: 3253	460: 3257

461: 3259	462: 3271	463: 3299	464: 3301	465: 3307
466: 3313	467: 3319	468: 3323	469: 3329	470: 3331
471: 3343	472: 3347	473: 3359	474: 3361	475: 3371
476: 3373	477: 3389	478: 3391	479: 3407	480: 3413
481: 3433	482: 3449	483: 3457	484: 3461	485: 3463
486: 3467	487: 3469	488: 3491	489: 3499	490: 3511
491: 3517	492: 3527	493: 3529	494: 3533	495: 3539
496: 3541	497: 3547	498: 3557	499: 3559	500: 3571
501: 3581	502: 3583	503: 3593	504: 3607	505: 3613
506: 3617	507: 3623	508: 3631	509: 3637	510: 3643
511: 3659	512: 3671	513: 3673	514: 3677	515: 3691
516: 3697	517: 3701	518: 3709	519: 3719	520: 3727
521: 3733	522: 3739	523: 3761	524: 3767	525: 3769
526: 3779	527: 3793	528: 3797	529: 3803	530: 3821
531: 3823	532: 3833	533: 3847	534: 3851	535: 3853
536: 3863	537: 3877	538: 3881	539: 3889	540: 3907
541: 3911	542: 3917	543: 3919	544: 3923	545: 3929
546: 3931	547: 3943	548: 3947	549: 3967	550: 3989
551: 4001	552: 4003	553: 4007	554: 4013	555: 4019
556: 4021	557: 4027	558: 4049	559: 4051	560: 4057
561: 4073	562: 4079	563: 4091	564: 4093	565: 4099
566: 4111	567: 4127	568: 4129	569: 4133	570: 4139
571: 4153	572: 4157	573: 4159	574: 4177	575: 4201
576: 4211	577: 4217	578: 4219	579: 4229	580: 4231
581: 4241	582: 4243	583: 4253	584: 4259	585: 4261
586: 4271	587: 4273	588: 4283	589: 4289	590: 4297
591: 4327	592: 4337	593: 4339	594: 4349	595: 4357
596: 4363	597: 4373	598: 4391	599: 4397	600: 4409
601: 4421	602: 4423	603: 4441	604: 4447	605: 4451
606: 4457	607: 4463	608: 4481	609: 4483	610: 4493
611: 4507	612: 4513	613: 4517	614: 4519	615: 4523
616: 4547	617: 4549	618: 4561	619: 4567	620: 4583
621: 4591	622: 4597	623: 4603	624: 4621	625: 4637
626: 4639	627: 4643	628: 4649	629: 4651	630: 4657
631: 4663	632: 4673	633: 4679	634: 4691	635: 4703
636: 4721	637: 4723	638: 4729	639: 4733	640: 4751
641: 4759	642: 4783	643: 4787	644: 4789	645: 4793
646: 4799	647: 4801	648: 4813	649: 4817	650: 4831
651: 4861	652: 4871	653: 4877	654: 4889	655: 4903
656: 4909	657: 4919	658: 4931	659: 4933	660: 4937
661: 4943	662: 4951	663: 4957	664: 4967	665: 4969
666: 4973	667: 4987	668: 4993	669: 4999	670: 5003
671: 5009	672: 5011	673: 5021	674: 5023	675: 5039
676: 5051	677: 5059	678: 5077	679: 5081	680: 5087
681: 5099	682: 5101	683: 5107	684: 5113	685: 5119
686: 5147	687: 5153	688: 5167	689: 5171	690: 5179
691: 5189	692: 5197	693: 5209	694: 5227	695: 5231

696: 5233	697: 5237	698: 5261	699: 5273	700: 5279
701: 5281	702: 5297	703: 5303	704: 5309	705: 5323
706: 5333	707: 5347	708: 5351	709: 5381	710: 5387
711: 5393	712: 5399	713: 5407	714: 5413	715: 5417
716: 5419	717: 5431	718: 5437	719: 5441	720: 5443
721: 5449	722: 5471	723: 5477	724: 5479	725: 5483
726: 5501	727: 5503	728: 5507	729: 5519	730: 5521
731: 5527	732: 5531	733: 5557	734: 5563	735: 5569
736: 5573	737: 5581	738: 5591	739: 5623	740: 5639
741: 5641	742: 5647	743: 5651	744: 5653	745: 5657
746: 5659	747: 5669	748: 5683	749: 5689	750: 5693
751: 5701	752: 5711	753: 5717	754: 5737	755: 5741
756: 5743	757: 5749	758: 5779	759: 5783	760: 5791
761: 5801	762: 5807	763: 5813	764: 5821	765: 5827
766: 5839	767: 5843	768: 5849	769: 5851	770: 5857
771: 5861	772: 5867	773: 5869	774: 5879	775: 5881
776: 5897	777: 5903	778: 5923	779: 5927	780: 5939
781: 5953	782: 5981	783: 5987	784: 6007	785: 6011
786: 6029	787: 6037	788: 6043	789: 6047	790: 6053
791: 6067	792: 6073	793: 6079	794: 6089	795: 6091
796: 6101	797: 6113	798: 6121	799: 6131	800: 6133
801: 6143	802: 6151	803: 6163	804: 6173	805: 6197
806: 6199	807: 6203	808: 6211	809: 6217	810: 6221
811: 6229	812: 6247	813: 6257	814: 6263	815: 6269
816: 6271	817: 6277	818: 6287	819: 6299	820: 6301
821: 6311	822: 6317	823: 6323	824: 6329	825: 6337
826: 6343	827: 6353	828: 6359	829: 6361	830: 6367
831: 6373	832: 6379	833: 6389	834: 6397	835: 6421
836: 6427	837: 6449	838: 6451	839: 6469	840: 6473
841: 6481	842: 6491	843: 6521	844: 6529	845: 6547
846: 6551	847: 6553	848: 6563	849: 6569	850: 6571
851: 6577	852: 6581	853: 6599	854: 6607	855: 6619
856: 6637	857: 6653	858: 6659	859: 6661	860: 6673
861: 6679	862: 6689	863: 6691	864: 6701	865: 6703
866: 6709	867: 6719	868: 6733	869: 6737	870: 6761
871: 6763	872: 6779	873: 6781	874: 6791	875: 6793
876: 6803	877: 6823	878: 6827	879: 6829	880: 6833
881: 6841	882: 6857	883: 6863	884: 6869	885: 6871
886: 6883	887: 6899	888: 6907	889: 6911	890: 6917
891: 6947	892: 6949	893: 6959	894: 6961	895: 6967
896: 6971	897: 6977	898: 6983	899: 6991	900: 6997
901: 7001	902: 7013	903: 7019	904: 7027	905: 7039
906: 7043	907: 7057	908: 7069	909: 7079	910: 7103
911: 7109	912: 7121	913: 7127	914: 7129	915: 7151
916: 7159	917: 7177	918: 7187	919: 7193	920: 7207
921: 7211	922: 7213	923: 7219	924: 7229	925: 7237
926: 7243	927: 7247	928: 7253	929: 7283	930: 7297

931: 7307	932: 7309	933: 7321	934: 7331	935: 7333
936: 7349	937: 7351	938: 7369	939: 7393	940: 7411
941: 7417	942: 7433	943: 7451	944: 7457	945: 7459
946: 7477	947: 7481	948: 7487	949: 7489	950: 7499
951: 7507	952: 7517	953: 7523	954: 7529	955: 7537
956: 7541	957: 7547	958: 7549	959: 7559	960: 7561
961: 7573	962: 7577	963: 7583	964: 7589	965: 7591
966: 7603	967: 7607	968: 7621	969: 7639	970: 7643
971: 7649	972: 7669	973: 7673	974: 7681	975: 7687
976: 7691	977: 7699	978: 7703	979: 7717	980: 7723
981: 7727	982: 7741	983: 7753	984: 7757	985: 7759
986: 7789	987: 7793	988: 7817	989: 7823	990: 7829
991: 7841	992: 7853	993: 7867	994: 7873	995: 7877
996: 7879	997: 7883	998: 7901	999: 7907	1000: 7919
1001: 7927	1002: 7933	1003: 7937	1004: 7949	1005: 7951
1006: 7963	1007: 7993	1008: 8009	1009: 8011	1010: 8017
1011: 8039	1012: 8053	1013: 8059	1014: 8069	1015: 8081
1016: 8087	1017: 8089	1018: 8093	1019: 8101	1020: 8111
1021: 8117	1022: 8123	1023: 8147	1024: 8161	1025: 8167
1026: 8171	1027: 8179	1028: 8191	1029: 8209	1030: 8219
1031: 8221	1032: 8231	1033: 8233	1034: 8237	1035: 8243
1036: 8263	1037: 8269	1038: 8273	1039: 8287	1040: 8291
1041: 8293	1042: 8297	1043: 8311	1044: 8317	1045: 8329
1046: 8353	1047: 8363	1048: 8369	1049: 8377	1050: 8387
1051: 8389	1052: 8419	1053: 8423	1054: 8429	1055: 8431
1056: 8443	1057: 8447	1058: 8461	1059: 8467	1060: 8501
1061: 8513	1062: 8521	1063: 8527	1064: 8537	1065: 8539
1066: 8543	1067: 8563	1068: 8573	1069: 8581	1070: 8597
1071: 8599	1072: 8609	1073: 8623	1074: 8627	1075: 8629
1076: 8641	1077: 8647	1078: 8663	1079: 8669	1080: 8677
1081: 8681	1082: 8689	1083: 8693	1084: 8699	1085: 8707
1086: 8713	1087: 8719	1088: 8731	1089: 8737	1090: 8741
1091: 8747	1092: 8753	1093: 8761	1094: 8779	1095: 8783
1096: 8803	1097: 8807	1098: 8819	1099: 8821	1100: 8831
1101: 8837	1102: 8839	1103: 8849	1104: 8861	1105: 8863
1106: 8867	1107: 8887	1108: 8893	1109: 8923	1110: 8929
1111: 8933	1112: 8941	1113: 8951	1114: 8963	1115: 8969
1116: 8971	1117: 8999	1118: 9001	1119: 9007	1120: 9011
1121: 9013	1122: 9029	1123: 9041	1124: 9043	1125: 9049
1126: 9059	1127: 9067	1128: 9091	1129: 9103	1130: 9109
1131: 9127	1132: 9133	1133: 9137	1134: 9151	1135: 9157
1136: 9161	1137: 9173	1138: 9181	1139: 9187	1140: 9199
1141: 9203	1142: 9209	1143: 9221	1144: 9227	1145: 9239
1146: 9241	1147: 9257	1148: 9277	1149: 9281	1150: 9283
1151: 9293	1152: 9311	1153: 9319	1154: 9323	1155: 9337
1156: 9341	1157: 9343	1158: 9349	1159: 9371	1160: 9377
1161: 9391	1162: 9397	1163: 9403	1164: 9413	1165: 9419

1166: 9421	1167: 9431	1168: 9433	1169: 9437	1170: 9439
1171: 9461	1172: 9463	1173: 9467	1174: 9473	1175: 9479
1176: 9491	1177: 9497	1178: 9511	1179: 9521	1180: 9533
1181: 9539	1182: 9547	1183: 9551	1184: 9587	1185: 9601
1186: 9613	1187: 9619	1188: 9623	1189: 9629	1190: 9631
1191: 9643	1192: 9649	1193: 9661	1194: 9677	1195: 9679
1196: 9689	1197: 9697	1198: 9719	1199: 9721	1200: 9733
1201: 9739	1202: 9743	1203: 9749	1204: 9767	1205: 9769
1206: 9781	1207: 9787	1208: 9791	1209: 9803	1210: 9811
1211: 9817	1212: 9829	1213: 9833	1214: 9839	1215: 9851
1216: 9857	1217: 9859	1218: 9871	1219: 9883	1220: 9887
1221: 9901	1222: 9907	1223: 9923	1224: 9929	1225: 9931
1226: 9941	1227: 9949	1228: 9967	1229: 9973	1230: 10007

卡特兰数

001 :1
002 :2
003 :5
004 :14
005 :42
006 :132
007 :429
008 :1430
009 :4862
010 :16796
011 :58786
012 :208012
013 :742900
014 :2674440
015 :9694845
016 :35357670
017 :129644790
018 :477638700
019 :1767263190
020 :6564120420
021 :24466267020
022 :91482563640
023 :343059613650
024 :1289904147324
025 :4861946401452
026 :18367353072152
027 :69533550916004
028 :263747951750360
029 :1002242216651368
030 :3814986502092304
031 :14544636039226909
032 :55534064877048198
033 :212336130412243110
034 :812944042149730764
035 :3116285494907301262
036 :11959798385860453492
037 :45950804324621742364
038 :176733862787006701400
039 :680425371729975800390
040 :2622127042276492108820
041 :10113918591637898134020
042 :39044429911904443959240
043 :150853479205085351660700
044 :583300119592996693088040
045 :2257117854077248073253720
046 :8740328711533173390046320
047 :33868773757191046886429490
048 :131327898242169365477991900
049 :509552245179617138054608572
050 :1978261657756160653623774456
051 :7684785670514316385230816156
052 :29869166945772625950142417512
053 :116157871455782434250553845880
054 :451959718027953471447609509424
055 :1759414616608818870992479875972
056 :6852456927844873497549658464312
057 :26700952856774851904245220912664
058 :104088460289122304033498318812080
059 :405944995127576985730643443367112
060 :1583850964596120042686772779038896
061 :6182127958584855650487080847216336
062 :24139737743045626825711458546273312
063 :94295850558771979787935384946380125
064 :368479169875816659479009042713546950
065 :1440418573150919668872489894243865350
066 :5632681584560312734993915705849145100
067 :22033725021956517463358552614056949950
068 :86218923998960285726185640663701108500
069 :337485502510215975556783793455058624700
070 :1321422108420282270489942177190229544600
071 :5175569924646105559418940193995065716350
072 :20276890389709399862928998568254641025700
073 :79463489365077377841208237632349268884500
074 :311496878311103321137536291518809134027240
075 :1221395654430378811828760722007962130791020
076 :4790408930363303911328386208394864461024520
077 :18793142726809884575211361279087545193250040
078 :73745243611532458459690151854647329239335600
079 :289450081175264899454283846029490767264392230
080 :1136359577947336271931632877004667456667613940
081 :4462290049988320482463241297506133183499654740
082 :17526585015616776834735140517915655636396234280
083 :68854441132780194707888052034668647142985206100
084 :270557451039395118028642463289168566420671280440
085 :1063353702922273835973036658043476458723103404520
086 :4180080073556524734514695828170907458428751314320
087 :16435314834665426797069144960762886143367590394940
088 :64633260585762914370496637486146181462681535261000
089 :254224158304000796523953440778841647086547372026600
090 :1000134600800354781929399250536541864362461089950800
091 :3935312233584004685417853572763349509774031680023800
092 :15487357822491889407128326963778343232013931127835600
093 :60960876535340415751462563580829648891969728907438000
094 :239993345518077005168915776623476723006280827488229600
095 :944973797977428207852605870454939596837230758234904050
096 :3721443204405954385563870541379246659709506697378694300
097 :14657929356129575437016877846657032761712954950899755100
098 :57743358069601357782187700608042856334020731624756611000
099 :227508830794229349661819540395688853956041682601541047340
100 :896519947090131496687170070074100632420837521538745909320

BELL 数前 50 项

1 1
2 2
3 5
4 15
5 52
6 203
7 877
8 4140
9 21147
10 115975
11 678570
12 4213597
13 27644437
14 190899322
15 1382958545
16 10480142147
17 82864869804
18 682076806159
19 5832742205057
20 51724158235372
21 474869816156751
22 4506715738447323
23 44152005855084346
24 445958869294805289
25 4638590332229999353
26 49631246523618756274
27 545717047936059989389
28 6160539404599934652455
29 71339801938860275191172
30 846749014511809332450147
31 10293358946226376485095653
32 128064670049908713818925644
33 1629595892846007606764728147
34 21195039388640360462388656799
35 281600203019560266563340426570
36 3819714729894818339975525681317
37 52868366208550447901945575624941
38 746289892095625330523099540639146
39 10738823330774692832768857986425209
40 157450588391204931289324344702531067
41 2351152507740617628200694077243788988
42 35742549198872617291353508656626642567
43 552950118797165484321714693280737767385
44 8701963427387055089023600531855797148876
45 139258505266263669602347053993654079693415
46 2265418219334494002928484444705392276158355
47 37450059502461511196505342096431510120174682
48 628919796303118415420210454071849537746015761
49 10726137154573358400342215518590002633917247281
50 185724268771078270438257767181908917499221852770

目　录:

Chapter 3

Geometry Theory

3.0 注意

```
/*==================================================*\
\*==================================================*/
```

注意事项：

1. 注意舍入方式(0.5 的舍入方向);防止输出-0.
2. 几何题注意多测试不对称数据.
3. 整数几何注意 xmult 和 dmult 是否会出界;

   符点几何注意 eps 的使用.
4. 避免使用斜率;注意除数是否会为 0.
5. 公式一定要化简后再代入.
6. 判断同一个 2*PI 域内两角度差应该是

   abs(a1-a2)<beta||abs(a1-a2)>pi+pi-beta;

   相等应该是

   abs(a1-a2)<eps||abs(a1-a2)>pi+pi-eps;
7. 需要的话尽量使用 atan2,注意:atan2(0,0)=0,

   atan2(1,0)=pi/2,atan2(-1,0)=-pi/2,atan2(0,1)=0,atan2(0,-1)=pi.
8. cross product = |u|*|v|*sin(a)

   dot product = |u|*|v|*cos(a)
9. (P1-P0)x(P2-P0)结果的意义:

   正: <P0,P1>在<P0,P2>顺时针(0,pi)内

   负: <P0,P1>在<P0,P2>逆时针(0,pi)内

   0 : <P0,P1>,<P0,P2>共线,夹角为 0 或 pi
10. 误差限缺省使用 1e-8!
11. PI= 3.14159265358979323846264338327950

```
/*==================================================*\
\*==================================================*/
```

3.1 浮点几何函数库

```
/*==================================================*\
\*==================================================*/
#include <stdio.h>
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};
struct line{point a,b;};
/*==================================================*\
| 1、计算叉积
|计算 cross product (P1-P0)x(P2-P0)
\*==================================================*/
double xmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
	return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
/*==================================================*\
| 2、计算点积
|计算 dot product (P1-P0).(P2-P0)
\*==================================================*/
double dmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
double dmult(double x1,double y1,double x2,double y2,double x0,double y0){
	return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
}
/*==================================================*\
| 3、计算两点的距离
\*==================================================*/
double distance(point p1,point p2){
	return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double distance(double x1,double y1,double x2,double y2){
	return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
/*==================================================*\
| 4、判断三点是否共线（1 表示共线）
\*==================================================*/
int dots_inline(point p1,point p2,point p3){
	return zero(xmult(p1,p2,p3));
}
int dots_inline(double x1,double y1,double x2,double y2,double x3,double y3){
	return zero(xmult(x1,y1,x2,y2,x3,y3));
}
/*==================================================*\
| 5 判点是否在线段上,包括端点（1 表示在线段上）
\*==================================================*/
int dot_online_in(point p,line l){
	return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a.y-p.y)*(l.b.y-p.y)<eps;
}
int dot_online_in(point p,point l1,point l2){
	return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}
int dot_online_in(double x,double y,double x1,double y1,double x2,double y2){
	return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)*(x2-x)<eps&&(y1-y)*(y2-y)<eps;
}
/*==================================================*\
| 6、判点是否在线段上,不包括端点（1 表示在线段上）
\*==================================================*/
int dot_online_ex(point p,line l){
	return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));
}
int dot_online_ex(point p,point l1,point l2){
	return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))&&(!zero(p.x-l2.x)||!zero(p.y-l2.y));
}
int dot_online_ex(double x,double y,double x1,double y1,double x2,double y2){
	return dot_online_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-x2)||!zero(y-y2));
}
/*==================================================*\
| 7、判两点在线段同侧,点在线段上返回 0
\*==================================================*/
int same_side(point p1,point p2,line l){
	return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
}
int same_side(point p1,point p2,point l1,point l2){
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}
```

```
/*==================================================*\
| 8、判两点在线段异侧,点在线段上返回 0
\*==================================================*/
int opposite_side(point p1,point p2,line l){
	return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
}
int opposite_side(point p1,point p2,point l1,point l2){
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}
/*==================================================*\
| 9、点关于直线的对称点
| by lyt
|缺点：用了斜率
|也可以利用"点到直线上的最近点"来做，避免使用斜率。
\*==================================================*/
point symmetric_point(point p1, point l1, point l2) {
   point ret;
   if (l1.x > l2.x - eps && l1.x < l2.x + eps) {
      ret.x = (2 * l1.x - p1.x);
        ret.y = p1.y;
   } else {
      double k = (l1.y - l2.y ) / (l1.x - l2.x);
      ret.x = (2*k*k*l1.x + 2*k*p1.y - 2*k*l1.y - k*k*p1.x + p1.x) / (1 + k*k);
      ret.y = p1.y - (ret.x - p1.x ) / k;
   }
   return ret;
}
/*==================================================*\
| 10、判两直线平行
\*==================================================*/
int parallel(line u,line v){
	return zero((u.a.x-u.b.x)*(v.a.y-v.b.y)-(v.a.x-v.b.x)*(u.a.y-u.b.y));
}
int parallel(point u1,point u2,point v1,point v2){
	return zero((u1.x-u2.x)*(v1.y-v2.y)-(v1.x-v2.x)*(u1.y-u2.y));
}
/*==================================================*\
| 11、判两直线垂直
\*==================================================*/
int perpendicular(line u,line v){
	return zero((u.a.x-u.b.x)*(v.a.x-v.b.x)+(u.a.y-u.b.y)*(v.a.y-v.b.y));
}
int perpendicular(point u1,point u2,point v1,point v2){
	return zero((u1.x-u2.x)*(v1.x-v2.x)+(u1.y-u2.y)*(v1.y-v2.y));
}
/*==================================================*\
| 12、判两线段相交,包括端点和部分重合
\*==================================================*/
int intersect_in(line u,line v){
	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_online_in(v.b,u);
}
int intersect_in(point u1,point u2,point v1,point v2){
	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);
}
/*==================================================*\
| 13、判两线段相交,不包括端点和部分重合
\*==================================================*/
int intersect_ex(line u,line v){
	return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point u1,point u2,point v1,point v2){
	return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}
```

```
/*==================================================*\
| 14、计算两直线交点,注意事先判断直线是否平行!
|线段交点请另外判线段相交(同时还是要判断是否平行!)
\*==================================================*/
point intersection(line u,line v){
	point 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;
	return ret;
}
point intersection(point u1,point u2,point v1,point v2){
	point 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;
	return ret;
}
/*==================================================*\
| 15、点到直线上的最近点
\*==================================================*/
point ptoline(point p,line l){
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	return intersection(p,t,l.a,l.b);
}
point ptoline(point p,point l1,point l2){
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	return intersection(p,t,l1,l2);
}
/*==================================================*\
| 16、点到直线距离
\*==================================================*/
double disptoline(point p,line l){
	return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoline(point p,point l1,point l2){
	return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
double disptoline(double x,double y,double x1,double y1,double x2,double y2){
	return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);
}
/*==================================================*\
| 17、点到线段上的最近点
\*==================================================*/
point ptoseg(point p,line l){
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
		return distance(p,l.a)<distance(p,l.b)?l.a:l.b;
	return intersection(p,t,l.a,l.b);
}
point ptoseg(point p,point l1,point l2){
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
		return distance(p,l1)<distance(p,l2)?l1:l2;
	return intersection(p,t,l1,l2);
}
/*==================================================*\
| 18、点到线段距离
\*==================================================*/
double disptoseg(point p,line l){
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
		return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);
	return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoseg(point p,point l1,point l2){
	point t=p;
```

```
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
		return
distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);
	return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
/*==================================================*\
| 19、矢量 V 以 P 为顶点逆时针旋转 angle 并放大 scale 倍
\*==================================================*/
point rotate(point v,point p,double angle,double scale){
	point ret=p;
	v.x-=p.x,v.y-=p.y;
	p.x=scale*cos(angle);
	p.y=scale*sin(angle);
	ret.x+=v.x*p.x-v.y*p.y;
	ret.y+=v.x*p.y+v.y*p.x;
	return ret;
}
```

3.2 整数几何函数库

```
/*==================================================*\
|注意某些情况下整数运算会出界!
\*==================================================*/
#define sign(a) ((a)>0?1:(((a)<0?-1:0)))
struct point{int x,y;};
struct line{point a,b;};
/*==================================================*\
| 1、计算叉积
|计算 cross product (P1-P0)x(P2-P0)
\*==================================================*/
int xmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
int xmult(int x1,int y1,int x2,int y2,int x0,int y0){
	return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
/*==================================================*\
| 2、计算点积
|计算 dot product (P1-P0).(P2-P0)
\*==================================================*/
int dmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
int dmult(int x1,int y1,int x2,int y2,int x0,int y0){
	return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
}
/*==================================================*\
| 3、判三点共线
\*==================================================*/
int dots_inline(point p1,point p2,point p3){
	return !xmult(p1,p2,p3);
}
int dots_inline(int x1,int y1,int x2,int y2,int x3,int y3){
	return !xmult(x1,y1,x2,y2,x3,y3);
}
/*==================================================*\
| 4、判点是否在线段上,包括端点和部分重合
\*==================================================*/
int dot_online_in(point p,line l){
	return  !xmult(p,l.a,l.b)&&(l.a.x-p.x)*(l.b.x-p.x)<=0&&(l.a.y-p.
y)*(l.b.y-p.y)<=0;
}
int dot_online_in(point p,point l1,point l2){
	return !xmult(p,l1,l2)&&(l1.x-p.x)*(l2.x-p.x)<=0&&(l1.y-p.y)*(l
2.y-p.y)<=0;
}
int dot_online_in(int x,int y,int x1,int y1,int x2,int y2){
	return !xmult(x,y,x1,y1,x2,y2)&&(x1-x)*(x2-x)<=0&&(y1-y)*(y
2-y)<=0;
}
/*==================================================*\
| 5、判点是否在线段上,不包括端点
\*==================================================*/
int dot_online_ex(point p,line l){
	return
dot_online_in(p,l)&&(p.x!=l.a.x||p.y!=l.a.y)&&(p.x!=l.b.x||p.y!=l.b.y)
;}
int dot_online_ex(point p,point l1,point l2){
	return
dot_online_in(p,l1,l2)&&(p.x!=l1.x||p.y!=l1.y)&&(p.x!=l2.x||p.y!=l2.
y);
}
int dot_online_ex(int x,int y,int x1,int y1,int x2,int y2){
	return
dot_online_in(x,y,x1,y1,x2,y2)&&(x!=x1||y!=y1)&&(x!=x2||y!=y2);
}
/*==================================================*\
| 6、判两点在直线同侧,点在直线上返回 0
\*==================================================*/
int same_side(point p1,point p2,line l){
	return sign(xmult(l.a,p1,l.b))*xmult(l.a,p2,l.b)>0;
}
int same_side(point p1,point p2,point l1,point l2){
	return sign(xmult(l1,p1,l2))*xmult(l1,p2,l2)>0;
}
/*==================================================*\
|7、判两点在直线异侧,点在直线上返回 0
\*==================================================*/
int opposite_side(point p1,point p2,line l){
	return sign(xmult(l.a,p1,l.b))*xmult(l.a,p2,l.b)<0;
}
int opposite_side(point p1,point p2,point l1,point l2){
	return sign(xmult(l1,p1,l2))*xmult(l1,p2,l2)<0;
}
/*==================================================*\
| 8、判两直线平行
\*==================================================*/
int parallel(line u,line v){
	return (u.a.x-u.b.x)*(v.a.y-v.b.y)==(v.a.x-v.b.x)*(u.a.y-u.b.y);
}
int parallel(point u1,point u2,point v1,point v2){
	return (u1.x-u2.x)*(v1.y-v2.y)==(v1.x-v2.x)*(u1.y-u2.y);
}
/*==================================================*\
| 9、判两直线垂直
\*==================================================*/
int perpendicular(line u,line v){
	return (u.a.x-u.b.x)*(v.a.x-v.b.x)==-(u.a.y-u.b.y)*(v.a.y-v.b.y);
}
int perpendicular(point u1,point u2,point v1,point v2){
	return (u1.x-u2.x)*(v1.x-v2.x)==-(u1.y-u2.y)*(v1.y-v2.y);
}
/*==================================================*\
| 10、判两线段相交,包括端点和部分重合
\*==================================================*/
int intersect_in(line u,line v){
	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)||do
t_online_in(v.b,u);
}
int intersect_in(point u1,point u2,point v1,point v2){
	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(v
1,u1,u2)||dot_online_in(v2,u1,u2);
}
/*==================================================*\
| 11、判两线段相交,不包括端点和部分重合
\*==================================================*/
int intersect_ex(line u,line v){
	return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point u1,point u2,point v1,point v2){
	return
opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}
```

3.3 公式

```
/*============================================================*\
| A、关于三角形的公式:
\*============================================================*/
/*============================================================*\
|1. 半周长 P=(a+b+c)/2
|2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
|3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
|4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
|5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
|6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
|              =4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
|                =Ptan(A/2)tan(B/2)tan(C/2)
|7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))
\*============================================================*/
/*============================================================*\
| B、关于四边形的计算公式:
|D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角
\*============================================================*/
/*============================================================*\
|1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
|2. S=D1D2sin(A)/2
|(以下对圆的内接四边形)
|3. ac+bd=D1D2
|4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P 为半周长
\*============================================================*/
/*============================================================*\
| C、正 n 边形:
|R 为外接圆半径,r 为内切圆半径
\*============================================================*/
/*============================================================*\
|1. 中心角 A=2PI/n
|2. 内角 C=(n-2)PI/n
|3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
|4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))
\*============================================================*/
/*============================================================*\
| D、圆:
\*============================================================*/
/*============================================================*\
|1. 弧长 l=rA
|2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
|3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
|4. 扇形面积 S1=rl/2=r^2A/2
|5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2
\*============================================================*/
/*============================================================*\
| E、棱柱:
\*============================================================*/
/*============================================================*\
|1. 体积 V=Ah,A 为底面积,h 为高
|2. 侧面积 S=lp,l 为棱长,p 为直截面周长
|3. 全面积 T=S+2A
\*============================================================*/
/*============================================================*\
| F、棱锥:
\*============================================================*/
/*============================================================*\
|1. 体积 V=Ah/3,A 为底面积,h 为高
|(以下对正棱锥)
|2. 侧面积 S=lp/2,l 为斜高,p 为底面周长
|3. 全面积 T=S+A
\*============================================================*/
/*============================================================*\
| G、棱台:
\*============================================================*/
/*============================================================*\
|1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2 为上下底面积,h 为高
|(以下为正棱台)
|2. 侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
|3. 全面积 T=S+A1+A2
\*============================================================*/
/*============================================================*\
| H、圆柱:
\*============================================================*/
/*============================================================*\
|1. 侧面积 S=2PIrh
|2. 全面积 T=2PIr(h+r)
|3. 体积 V=PIr^2h
\*============================================================*/
/*============================================================*\
| I、圆锥:
\*============================================================*/
/*============================================================*\
|1. 母线 l=sqrt(h^2+r^2)
|2. 侧面积 S=PIrl
|3. 全面积 T=PIr(l+r)
|4. 体积 V=PIr^2h/3
\*============================================================*/
/*============================================================*\
| J、圆台:
\*============================================================*/
/*============================================================*\
|1. 母线 l=sqrt(h^2+(r1-r2)^2)
|2. 侧面积 S=PI(r1+r2)l
|3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
|4. 体积 V=PI(r1^2+r2^2+r1r2)h/3
\*============================================================*/
/*============================================================*\
| K、球:
\*============================================================*/
/*============================================================*\
|1. 全面积 T=4PIr^2
|2. 体积 V=4PIr^3/3
\*============================================================*/
/*============================================================*\
| L、球台:
\*============================================================*/
/*============================================================*\
|1. 侧面积 S=2PIrh
|2. 全面积 T=PI(2rh+r1^2+r2^2)
|3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6
\*============================================================*/
/*============================================================*\
| M、球扇形:
\*============================================================*/
/*============================================================*\
|1. 全面积 T=PIr(2h+r0),h 为球冠高,r0 为球冠底面半径
|2. 体积 V=2PIr^2h/3
\*============================================================*/
```

3.4 三角形

```
/*============================================================*\
|1、外心、内心、垂心、重心、费马点（到三角形三顶点距离之和最小的点）
\*============================================================*/
#include <math.h>
struct point{double x,y;};
struct line{point a,b;};
double distance(point p1,point p2){
	return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
point intersection(line u,line v){
	point 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;
	return ret;
}
```

```
/*==================================================*\
| //外心
\*==================================================*/
point circumcenter(point a,point b,point c){
        line u,v;
        u.a.x=(a.x+b.x)/2;
        u.a.y=(a.y+b.y)/2;
        u.b.x=u.a.x-a.y+b.y;
        u.b.y=u.a.y+a.x-b.x;
        v.a.x=(a.x+c.x)/2;
        v.a.y=(a.y+c.y)/2;
        v.b.x=v.a.x-a.y+c.y;
        v.b.y=v.a.y+a.x-c.x;
        return intersection(u,v);
}
/*==================================================*\
|//内心
\*==================================================*/
point incenter(point a,point b,point c){
        line u,v;
        double m,n;
        u.a=a;
        m=atan2(b.y-a.y,b.x-a.x);
        n=atan2(c.y-a.y,c.x-a.x);
        u.b.x=u.a.x+cos((m+n)/2);
        u.b.y=u.a.y+sin((m+n)/2);
        v.a=b;
        m=atan2(a.y-b.y,a.x-b.x);
        n=atan2(c.y-b.y,c.x-b.x);
        v.b.x=v.a.x+cos((m+n)/2);
        v.b.y=v.a.y+sin((m+n)/2);
        return intersection(u,v);
}
/*==================================================*\
|//垂心
\*==================================================*/
point perpencenter(point a,point b,point c){
        line u,v;
        u.a=c;
        u.b.x=u.a.x-a.y+b.y;
        u.b.y=u.a.y+a.x-b.x;
        v.a=b;
        v.b.x=v.a.x-a.y+c.y;
        v.b.y=v.a.y+a.x-c.x;
        return intersection(u,v);
}
/*==================================================*\
|//重心
|//到三角形三顶点距离的平方和最小的点
|//三角形内到三边距离之积最大的点
\*==================================================*/
point barycenter(point a,point b,point c){
        line u,v;
        u.a.x=(a.x+b.x)/2;
        u.a.y=(a.y+b.y)/2;
        u.b=c;
        v.a.x=(a.x+c.x)/2;
        v.a.y=(a.y+c.y)/2;
        v.b=b;
        return intersection(u,v);
}
/*==================================================*\
| //费马点
|//到三角形三顶点距离之和最小的点
\*==================================================*/
point fermentpoint(point a,point b,point c){
        point u,v;
        double
step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);
        int i,j,k;
        u.x=(a.x+b.x+c.x)/3;
        u.y=(a.y+b.y+c.y)/3;
        while (step>1e-10)
                for (k=0;k<10;step/=2,k++)
                        for (i=-1;i<=1;i++)
                                for (j=-1;j<=1;j++){
                                        v.x=u.x+step*i;
                                        v.y=u.y+step*j;
                                        if
(distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+distance(v,c))
                                                u=v;
                                }
        return u;
}
/*==================================================*\
|2、计算三角形的面积
\*==================================================*/
/*==================================================*\
|计算 叉积 cross product (P1-P0)x(P2-P0)
\*==================================================*/
double xmult(point p1,point p2,point p0){
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
        return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
/*==================================================*\
| //计算三角形面积,输入三顶点
\*==================================================*/
double area_triangle(point p1,point p2,point p3){
        return fabs(xmult(p1,p2,p3))/2;
}
double area_triangle(double x1,double y1,double x2,double y2,double x3,double y3){
        return fabs(xmult(x1,y1,x2,y2,x3,y3))/2;
}
/*==================================================*\
|//计算三角形面积,输入三边长
\*==================================================*/
double area_triangle(double a,double b,double c){
        double s=(a+b+c)/2;
        return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

3.5 多边形

```
/*==================================================*\
|1、计算多边形的面积
\*==================================================*/
/*==================================================*\
| //计算多边形面积,顶点按顺时针或逆时针给出
\*==================================================*/
double area_polygon(int n,point* p){
        double s1=0,s2=0;
        int i;
        for (i=0;i<n;i++)
                s1+=p[(i+1)%n].y*p[i].x,s2+=p[(i+1)%n].y*p[(i+2)%n].x;
        return fabs(s1-s2)/2;
}
/*==================================================*\
\*==================================================*/
#include <stdlib.h>
#include <math.h>
#include<stdio.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))
struct point{double x,y;};
struct line{point a,b;};

double xmult(point p1,point p2,point p0){
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

/*==================================================*\
|2、判断凸多边形
\*==================================================*/
```

```
/*==========================================================*\
|A、判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
| //注意：返回 1 表示是凸多边形， 0 为凹多边形
\*==========================================================*/
int is_convex(int n,point* p){
	int i,s[3]={1,1,1};
	for (i=0;i<n&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
	return s[1]|s[2];
}
/*==========================================================*\
| B、判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
| //注意：返回 1 表示是凸多边形， 0 为凹多边形
\*==========================================================*/
int is_convex_v2(int n,point* p){
	int i,s[3]={1,1,1};
	for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
	return s[0]&&s[1]|s[2];
}
/*==========================================================*\
| C、判 点 在凸多边形内或多边形边上,顶点按顺时针或逆时针给出
|//注意：返回 1 表示点在凸边形内或多边形上，返回 0 表示点在凸边|
形外，
\*==========================================================*/
int inside_convex(point q,int n,point* p){
	int i,s[3]={1,1,1};
	for (i=0;i<n&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
	return s[1]|s[2];
}
/*==========================================================*\
| D、判 点 在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返
|回 0
|//注意：返回 1 表示点在凸边形内，返回 0 表示点在凸边形 上 或 外，
\*==========================================================*/
int inside_convex_v2(point q,int n,point* p){
	int i,s[3]={1,1,1};
	for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
	return s[0]&&s[1]|s[2];
}
/*==========================================================*\
| E、判点在任意多边形内,顶点按顺时针或逆时针给出
|on_edge 表示点在多边形边上时的返回值,offset 为多边形坐标上限
\*==========================================================*/
int inside_polygon(point q,int n,point* p,int on_edge=1){
	point q2;
	int i=0,count;
	while (i<n)
		for
(count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
			if
(zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<ep
s&&(p[i].y-q.y)*(p[(i+1)%n].y-q.y)<eps)
				return on_edge;
			else if (zero(xmult(q,q2,p[i])))
				break;
			else                                                if
(xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+
1)%n])*xmult(p[i],q2,p[(i+1)%n])<-eps)
				count++;
	return count&1;
}
inline int opposite_side(point p1,point p2,point l1,point l2){
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}
inline int dot_online_in(point p,point l1,point l2){
	return
zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.
y)<eps;
}
/*==========================================================*\
|F、判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回
|1
|//注意：返回值是 1 表示线段与多边形相交（包括一个点相交），返回 0
|不相交，
\*==========================================================*/
int inside_polygon(point l1,point l2,int n,point* p){
	point t[MAXN],tt;
	int i,j,k=0;
	if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
		return 0;
	for (i=0;i<n;i++)
		if
(opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)
%n],l1,l2))
			return 0;
		else if (dot_online_in(l1,p[i],p[(i+1)%n]))
			t[k++]=l1;
		else if (dot_online_in(l2,p[i],p[(i+1)%n]))
			t[k++]=l2;
		else if (dot_online_in(p[i],l1,l2))
			t[k++]=p[i];
	for (i=0;i<k;i++)
		for (j=i+1;j<k;j++){
			tt.x=(t[i].x+t[j].x)/2;
			tt.y=(t[i].y+t[j].y)/2;
			if (!inside_polygon(tt,n,p))
				return 0;
		}
	return 1;
}
/*==========================================================*\
| G、求两条直线的交点
|// 注意调用该函数是要先判断是否平行
\*==========================================================*/
point intersection(line u,line v){
	point 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;
	return ret;
}
/*==========================================================*\
| H、   求三角形的重心
\*==========================================================*/
point barycenter(point a,point b,point c){
	line u,v;
	u.a.x=(a.x+b.x)/2;
	u.a.y=(a.y+b.y)/2;
	u.b=c;
	v.a.x=(a.x+c.x)/2;
	v.a.y=(a.y+c.y)/2;
	v.b=b;
	return intersection(u,v);
}
/*==========================================================*\
| I、多边形重心
\*==========================================================*/
point barycenter(int n,point* p){
	point ret,t;
	double t1=0,t2;
	int i;
	ret.x=ret.y=0;
	for (i=1;i<n-1;i++)
		if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
			t=barycenter(p[0],p[i],p[i+1]);
			ret.x+=t.x*t2;
			ret.y+=t.y*t2;
			t1+=t2;
		}
	if (fabs(t1)>eps)
		ret.x/=t1,ret.y/=t1;
	return ret;
}
```

3.6 凸包

```
/*==========================================================*\
| A、不能去掉点集中重合的点
```

```
\*==================================================*/
/*==================================================*\
|// CONVEX HULL I
|// modified by rr  不能去掉点集中重合的点
\*==================================================*/
#include <stdlib.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};
/*==================================================*\
|1、计算叉积；
|//计算 cross product (P1-P0)x(P2-P0)
\*==================================================*/
double xmult(point p1,point p2,point p0){
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
/*==================================================*\
|//graham 算法顺时针构造包含所有共线点的凸包,O(nlogn)
\*==================================================*/
point p1,p2;
int graham_cp(const void* a,const void* b){
        double ret=xmult(*((point*)a),*((point*)b),p1);
        return
zero(ret)?(xmult(*((point*)a),*((point*)b),p2)>0?1:-1):(ret>0?1:-1);
}
void _graham(int n,point* p,int& s,point* ch){
        int i,k=0;
        for (p1=p2=p[0],i=1;i<n;p2.x+=p[i].x,p2.y+=p[i].y,i++)
                if (p1.y-p[i].y>eps||(zero(p1.y-p[i].y)&&p1.x>p[i].x))
                        p1=p[k=i];
        p2.x/=n,p2.y/=n;
        p[k]=p[0],p[0]=p1;
        qsort(p+1,n-1,sizeof(point),graham_cp);
        for
(ch[0]=p[0],ch[1]=p[1],ch[2]=p[2],s=i=3;i<n;ch[s++]=p[i++])
                for (;s>2&&xmult(ch[s-2],p[i],ch[s-1])<-eps;s--);
}
/*==================================================*\
|2.构造凸包接口函数,传入原始点集大小 n,点集 p(p 原有顺序被打乱!)
|//返回凸包大小,凸包的点在 convex 中
|//参数 maxsize 为 1 包含共线点,为 0 不包含共线点,缺省为 1
|//参数 clockwise 为 1 顺时针构造,为 0 逆时针构造,缺省为 1
|//在输入仅有若干共线点时算法不稳定,可能有此类情况请另行处理!
|//不能去掉点集中重合的点
\*==================================================*/
int graham(int n,point* p,point* convex,int maxsize=1,int dir=1){
        point* temp=new point[n];
        int s,i;
        _graham(n,p,s,temp);
        for
(convex[0]=temp[0],n=1,i=(dir?1:(s-1));dir?(i<s):i;i+=(dir?1:-1))
                if
(maxsize||!zero(xmult(temp[i-1],temp[i],temp[(i+1)%s])))
                        convex[n++]=temp[i];
        delete []temp;
        return n;
}
/*==================================================*\
```

| B、去掉点集中重合的点

```
\*==================================================*/
/*==================================================*\
|// CONVEX HULL II
// modified by mgmg  去掉点集中重合的点
\*==================================================*/
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};
/*==================================================*\
|1、计算叉积 cross product (P1-P0)x(P2-P0)
\*==================================================*/
double xmult(point p1,point p2,point p0){
      return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
/*==================================================*\
|2、graham 算法顺时针构造包含所有共线点的凸包,O(nlogn)
\*==================================================*/
point p1,p2;
int graham_cp(const void* a,const void* b){
        double ret=xmult(*((point*)a),*((point*)b),p1);
        return
zero(ret)?(xmult(*((point*)a),*((point*)b),p2)>0?1:-1):(ret>0?1:-1);
}
void _graham(int n,point* p,int& s,point* ch){
        int i,k=0;
        for (p1=p2=p[0],i=1;i<n;p2.x+=p[i].x,p2.y+=p[i].y,i++)
                if (p1.y-p[i].y>eps||(zero(p1.y-p[i].y)&&p1.x>p[i].x))
                        p1=p[k=i];
        p2.x/=n,p2.y/=n;
        p[k]=p[0],p[0]=p1;
        qsort(p+1,n-1,sizeof(point),graham_cp);
        for
(ch[0]=p[0],ch[1]=p[1],ch[2]=p[2],s=i=3;i<n;ch[s++]=p[i++])
                for (;s>2&&xmult(ch[s-2],p[i],ch[s-1])<-eps;s--);
}
int wipesame_cp(const void *a, const void *b)
{
        if ((*(point *)a).y < (*(point *)b).y - eps) return -1;
        else if ((*(point *)a).y > (*(point *)b).y + eps) return 1;
        else if ((*(point *)a).x < (*(point *)b).x - eps) return -1;
        else if ((*(point *)a).x > (*(point *)b).x + eps) return 1;
        else return 0;
}
int _wipesame(point * p, int n)
{
        int i, k;
        qsort(p, n, sizeof(point), wipesame_cp);
        for (k=i=1;i<n;i++)
                if (wipesame_cp(p+i,p+i-1)!=0) p[k++]=p[i];
        return k;
}
/*==================================================*\
|3.构造凸包接口函数,传入原始点集大小 n,点集 p(p 原有顺序被打乱!)
|//返回凸包大小,凸包的点在 convex 中
|//参数 maxsize 为 1 包含共线点,为 0 不包含共线点,缺省为 1
|//参数 clockwise 为 1 顺时针构造,为 0 逆时针构造,缺省为 1
|//在输入仅有若干共线点时算法不稳定,可能有此类情况请另行处理!
\*==================================================*/
int graham(int n,point* p,point* convex,int maxsize=1,int dir=1){
        point* temp=new point[n];
        int s,i;
        n = _wipesame(p,n);
        _graham(n,p,s,temp);
        for
(convex[0]=temp[0],n=1,i=(dir?1:(s-1));dir?(i<s):i;i+=(dir?1:-1))
                if
(maxsize||!zero(xmult(temp[i-1],temp[i],temp[(i+1)%s])))
                        convex[n++]=temp[i];
        delete []temp;
        return n;
}
```

3.7 关于圆的函数

```
/*==================================================*\
\*==================================================*/
#include <stdio.h>
#include <math.h>
#define eps 1e-8
struct point{double x,y;};
double xmult(point p1,point p2,point p0){
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double distance(point p1,point p2){
        return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double disptoline(point p,point l1,point l2){
        return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
point intersection(point u1,point u2,point v1,point v2){
        point 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;
	return ret;
}
/*=======================================================*\
|1、判 直线 和圆相交,包括相切
|//相交或相切返回 1 ，相离返回 0
\*=======================================================*/
int intersect_line_circle(point c,double r,point l1,point l2){
	return disptoline(c,l1,l2)<r+eps;
}
/*=======================================================*\
|2、判 线段 和圆相交,包括端点和相切
|//相交或相切返回 1 ，相离返回 0 ；
\*=======================================================*/
int intersect_seg_circle(point c,double r,point l1,point l2){
	double t1=distance(c,l1)-r,t2=distance(c,l2)-r;
	point t=c;
	if (t1<eps||t2<eps)
		return t1>-eps||t2>-eps;
	t.x+=l1.y-l2.y;
	t.y+=l2.x-l1.x;
	return
xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
}
/*=======================================================*\
|3、判圆和圆相交,包括相切
|//相交或相切返回 1 ，相离返回 0 ；
\*=======================================================*/
int intersect_circle_circle(point c1,double r1,point c2,double r2){
	return
distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;
}
/*=======================================================*\
|4、计算圆上到点 p 最近点,如 p 与圆心重合,返回 p 本身
\*=======================================================*/
point dot_to_circle(point c,double r,point p){
	point u,v;
	if (distance(p,c)<eps)
		return p;
	u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
	u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-
1:1);
	v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
	v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1
:1);
	return distance(u,p)<distance(v,p)?u:v;
}
/*=======================================================*\
|5、计算 直线 与圆的交点,保证直线与圆有交点
|//计算 线段 与圆的交点可用这个函数后判点是否在线段上
\*=======================================================*/
void intersection_line_circle(point c,double r,point l1,point
l2,point& p1,point& p2){
	point p=c;
	double t;
	p.x+=l1.y-l2.y;
	p.y+=l2.x-l1.x;
	p=intersection(p,c,l1,l2);
	t=sqrt(r*r-distance(p,c)*distance(p,c))/distance(l1,l2);
	p1.x=p.x+(l2.x-l1.x)*t;
	p1.y=p.y+(l2.y-l1.y)*t;
	p2.x=p.x-(l2.x-l1.x)*t;
	p2.y=p.y-(l2.y-l1.y)*t;
}
/*=======================================================*\
|6、计算 圆 与圆的交点,保证圆与圆有交点,圆心不重合
\*=======================================================*/
void intersection_circle_circle(point c1,double r1,point
c2,double r2,point& p1,point& p2){
	point u,v;
	double t;
	t=(1+(r1*r1-r2*r2)/distance(c1,c2)/distance(c1,c2))/2;
	u.x=c1.x+(c2.x-c1.x)*t;
	u.y=c1.y+(c2.y-c1.y)*t;
	v.x=u.x+c1.y-c2.y;
	v.y=u.y-c1.x+c2.x;
	intersection_line_circle(c1,r1,u,v,p1,p2);
}
```

3.8 关于球的函数

```
/*=======================================================*\
\*=======================================================*/
#include <math.h>
#include <stdio.h>
const double pi=acos(-1);
/*=======================================================*\
|1、计算地球的 圆心角 lat 表示纬度,-90<=w<=90, lng 表示经度
//返回两点所在大圆劣弧对应圆心角,0<=angle<=pi
\*=======================================================*/
double angle(double lng1,double lat1,double lng2,double lat2){
	double dlng=fabs(lng1-lng2)*pi/180;
	while (dlng>=pi+pi)
		dlng-=pi+pi;
	if (dlng>pi)
		dlng=pi+pi-dlng;
	lat1*=pi/180,lat2*=pi/180;
	return
acos(cos(lat1)*cos(lat2)*cos(dlng)+sin(lat1)*sin(lat2));
}
/*=======================================================*\
|2、计算地球两点的球面 距离 ,r 为球半径，已知两点的经度 lng、纬度
lat
\*=======================================================*/
double line_dist(double r,double lng1,double lat1,double
lng2,double lat2){
	double dlng=fabs(lng1-lng2)*pi/180;
	while (dlng>=pi+pi)
		dlng-=pi+pi;
	if (dlng>pi)
		dlng=pi+pi-dlng;
	lat1*=pi/180,lat2*=pi/180;
	return
r*sqrt(2-2*(cos(lat1)*cos(lat2)*cos(dlng)+sin(lat1)*sin(lat2)));
}
/*=======================================================*\
|3、计算球面距离,r 为球半径
\*=======================================================*/
inline double sphere_dist(double r,double lng1,double
lat1,double lng2,double lat2){
	return r*angle(lng1,lat1,lng2,lat2);
}
```

3.9 三维几何函数库

```
/*=======================================================*\
\*=======================================================*/
#include <stdio.h>
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point3{double x,y,z;};
struct line3{point3 a,b;};
struct plane3{point3 a,b,c;};
/*=======================================================*\
|1、计算 cross product U x V ( 差积 )
\*=======================================================*/
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;
}
/*=======================================================*\
|2、计算 dot product U . V （ 点积 ）
\*=======================================================*/
double dmult(point3 u,point3 v){
	return u.x*v.x+u.y*v.y+u.z*v.z;
}
```

```
/*==================================================*\
|3、矢量差 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;
}
/*==================================================*\
|4、取平面法向量
\*==================================================*/
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));
}
/*==================================================*\
|5、两点距离,单参数取向量大小
\*==================================================*/
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));
}
/*==================================================*\
|6、向量大小
\*==================================================*/
double vlen(point3 p){
	return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}
/*==================================================*\
|7、判三点共线
\*==================================================*/
int dots_inline(point3 p1,point3 p2,point3 p3){
	return vlen(xmult(subt(p1,p2),subt(p2,p3)))<eps;
}
/*==================================================*\
|8、判四点共面
\*==================================================*/
int dots_onplane(point3 a,point3 b,point3 c,point3 d){
	return zero(dmult(pvec(a,b,c),subt(d,a)));
}
/*==================================================*\
|9、判点是否在线段上,包括端点和共线
\*==================================================*/
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)<e
ps&&(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;
}
/*==================================================*\
|10、判点是否在线段上,不包括端点
\*==================================================*/
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-l
1.z))&&(!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
}
/*==================================================*\
|11、判点是否在空间三角形上,包括边界,三点共线无意义
\*==================================================*/
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),s
ubt(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),s
ubt(p,s2)))-
	vlen(xmult(subt(p,s2),subt(p,s3)))-vlen(xmult(subt(p,s3),sub
t(p,s1))));
}
/*==================================================*\
|12、判点是否在空间三角形上,不包括边界,三点共线无意义
\*==================================================*/
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;
}
/*==================================================*\
| 13、判两点在线段同侧,点在线段上返回 0,不共面无意义
\*==================================================*/
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)))>eps;
}
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;
}
/*==================================================*\
|14、判两点在线段异侧,点在线段上返回 0,不共面无意义
\*==================================================*/
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;
}
/*==================================================*\
|15、判两点在平面同侧,点在平面上返回 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;
}
/*==================================================*\
|16、判两点在平面异侧,点在平面上返回 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;
```

```
}
/*==================================================*\
|17、判两直线平行
\*==================================================*/
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;}
/*==================================================*\
|18、判两平面平行
\*==================================================*/
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;}
/*==================================================*\
|19、判直线与平面平行
\*==================================================*/
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)));}
/*==================================================*\
|20、判两直线垂直
\*==================================================*/
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)));}
/*==================================================*\
|21、判两平面垂直
\*==================================================*/
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)));}
/*==================================================*\
|22、判直线与平面平行
\*==================================================*/
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;}
/*==================================================*\
|23、判两线段相交,包括端点和部分重合
\*==================================================*/
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_online_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);
}
/*==================================================*\
|24、判两线段相交,不包括端点和部分重合
\*==================================================*/
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_side(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_side(v1,v2,u1,u2);
}
/*==================================================*\
|25、判线段与空间三角形相交,包括交于边界和(部分)包含
\*==================================================*/
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);
}
/*==================================================*\
| 26、判线段与空间三角形相交,不包括交于边界和(部分)包含
\*==================================================*/
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);
}
/*==================================================*\
|27、计算两直线交点,注意事先判断直线是否共面和平行!
|//线段交点请另外判线段相交(同时还是要判断是否平行!)
\*==================================================*/
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;
}
/*==================================================*\
|28、计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!
|//线段和空间三角形交点请另外判断
\*==================================================*/
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;
```

```
}
```

/*==*\
|29、计算两平面交线,注意事先判断是否平行,并保证三点不共线!
==/

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

/*==*\
|30、点到直线距离
==/

```
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);
}
```

/*==*\
|31、点到平面距离
==/

```
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));
}
```

/*==*\
|32、直线到直线距离
==/

```
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);
}
```

/*==*\
|33、两直线夹角 cos 值
==/

```
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));
}
```

/*==*\
|34、两平面夹角 cos 值
==/

```
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));
}
```

/*==*\
|35、直线平面夹角 sin 值
==/

```
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));
}
```

/*==*\
|36、已知六条边求四面体体积
==/

```
#include <iostream>
#include <math.h>
using namespace std;
int main(){
   double a[ 6 ] , cosarfa , cosbeta , cosgama , ans;
   while (cin >> a[ 0 ] >> a[ 1 ] >> a[ 2 ] >> a[ 3 ] >> a[ 4 ] >> a[ 5 ]){
      cosarfa = (a[ 0 ] * a[ 0 ] + a[ 2 ] * a[ 2 ] - a[ 4 ] * a[ 4 ]) / 2.0 / a[ 0 ] / a[ 2 ];
       cosbeta = (a[ 0 ] * a[ 0 ] + a[ 1 ] * a[ 1 ] - a[ 3 ] * a[ 3 ]) / 2.0 / a[ 0 ] / a[ 1 ];
      cosgama = (a[ 1 ] * a[ 1 ] + a[ 2 ] * a[ 2 ] - a[ 5 ] * a[ 5 ]) / 2.0 / a[ 1 ] / a[ 2 ];
      ans = a[ 0 ] * a[ 1 ] * a[ 2 ] / 6.0 * sqrt(1.0 - cosarfa * cosarfa - cosbeta * cosbeta - cosgama * cosgama + 2 * cosarfa * cosbeta * cosgama);
   printf ("%.4f\n" , ans);
   }
   return 0;
}
```

/*==*\
|半平面求交的面积
==/

```
#include <cmath>
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <iostream>
using namespace std;
typedef double TYPE;
#define MaxPoint 1550
#define Epsilon    1e-10
/*验证*///精度的范围 ,根据不同的情况调整精度值
#define Abs(x)      (((x)>0)?(x):(-(x)))
 /*验证*///空间中的点，可以用来作为二维点来用
struct POINT {/*验证*/
	TYPE x; TYPE y; TYPE z;
	POINT() : x(0), y(0), z(0) {};
	POINT(TYPE _x_, TYPE _y_, TYPE _z_ = 0)	: x(_x_), y(_y_), z(_z_) {};	//要用 G++ 提交 ， 可以不用这个
	POINT operator =(const POINT &A){	x = A.x; y = A.y; z = A.z;	}
};			// 多边形 ,逆时针或顺时针给出 x,y
struct POLY {/*验证*/		//n 个点
	int n;		//x,y 为点的指针， 首尾必须重合
	TYPE * x; TYPE * y;
	POLY() : n(0), x(NULL), y(NULL) {};
	POLY(int _n_, const TYPE * _x_, const TYPE * _y_) {
		n = _n_;
		x = new TYPE[n + 1];
		memcpy(x, _x_, n*sizeof(TYPE));
		x[n] = _x_[0];
		y = new TYPE[n + 1];
		memcpy(y, _y_, n*sizeof(TYPE));
		y[n] = _y_[0];
	}
};
//判断 x 是正数还是负数
inline int Sign(TYPE x){/*验证*/
	return x<-Epsilon?-1:x>Epsilon;
}
```

```
void Interect(POINT x,POINT y,TYPE a,TYPE b,TYPE c,int &s,POINT q[]){
	TYPE u=fabs(a*x.x+b*x.y+c);
	TYPE v=fabs(a*y.x+b*y.y+c);
	q[++s].x=(x.x*v+y.x*u)/(u+v);
	q[s].y=(x.y*v+y.y*u)/(u+v);
}
//利用半平面切割
void Cut(TYPE a,TYPE b,TYPE c,int &KarnalPoint,POINT p[]){
	int s=0;
	int i;
	POINT q[MaxPoint];
	for(i=1; i<=KarnalPoint; i++){//遍历所有顶点是否能观察到该边
		if(Sign(a* p[i].x+ b*p[i].y+ c)>= 0){
//因为线段是顺时针给出的，如果是逆时针就是<=0
			q[++s]= p[i];		//若是则存储
		}
		else{
			if(Sign(a* p[i-1].x+ b* p[i-1].y+ c)> 0)
							//逆时针就是<0
				Interect(p[i-1], p[i], a, b, c, s, q);
			if(Sign(a* p[i+1].x+ b* p[i+1].y+ c)> 0)
							//逆时针就是<0
				Interect(p[i+1], p[i], a, b, c, s, q);
		}
	}	//最后的 p 数组存放半平面的点集合
	for(i=1;i<=s;i++)
		p[i]=q[i];
	p[s+1]=p[1],p[0]=p[s];
	KarnalPoint=s;
}
POLY PolygonKernal(int n, POINT point[]){
	int KarnalPoint= n;
	POINT p[MaxPoint];//p 的大小和 tr 的大小一样
	for(int i= 0; i< n; i++){
		p[i+1]= point[i]; //初始化边界
	}
	point[n]= point[0];
	p[n+1]= p[1];
	p[0]	= p[n];
	TYPE a,b,c;
	for(int i=0;i<n;i++){
		a=point[i+1].y- point[i].y	; //计算出相邻两点所在直线 ax+by+c=0
		b=point[i].x	- point[i+1].x;
		c=point[i+1].x* point[i].y- point[i].x* point[i+1].y;
		Cut(a, b, c, KarnalPoint, p);
	}
	TYPE X[MaxPoint],Y[MaxPoint];
	for(int i= 0; i< KarnalPoint; i++){
		X[i]= p[i].x;
		Y[i]= p[i].y;
	}
	POLY poly(KarnalPoint, X, Y);
	return poly;
}
//求多边形面积	拍好序的点 (返回的有可能是负数，Abs 一下)
TYPE Area(const POLY & poly) { /*验证*/
	if ( poly.n < 3)
		return TYPE(0);
	double	s = poly.y[0] * (poly.x[poly.n - 1] - poly.x[1]);
	for (int i = 1; i < poly.n; i++) {
		s += poly.y[i] * (poly.x[i - 1] - poly.x[(i + 1) % poly.n]);
	}
	return s/2;
}
int main(){
	int n,t;
	POINT point[MaxPoint];
	cin>>t;
	while(t--)	{
		cin>>n;
		for(int i=0;i<n;i++){
			scanf("%lf%lf",&point[i].x,&point[i].y);
		}
		POLY poly= PolygonKernal(n, point);
		TYPE s= Area(poly);		//之前用自定义的 Abs 宏定义，出现错误，改成下面的就 AC 了
		if(s< 0) printf("%.2f\n",-s);
		else	printf("%.2f\n", s);
	}
	return 0;
}
```

```
/*=====================================================*\
|旋转卡壳算法 求凸包上最远距离
\*=====================================================*/
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
#include<algorithm>
using namespace std;
#define N 50100
#define PI (3.141592653589793)
#define EPS (1e-6)
#define INF (1e250)
#define feqs(x,y) (fabs((x)-(y))<EPS)
struct node{
	int x,y;
};
struct pt{
	int x,y;
};
struct polygon{
	pt p[N];
	int ct;
};
node P[N];
int H[N],np;
int D[N*2],top,bot;
inline int crossP(struct node v1,struct node v2){
      return v1.x*v2.y-v2.x*v1.y;
};
struct node MAKE_VECTOR(int p1,int p2){
	struct node ans;
	ans.x=P[p2].x-P[p1].x;
	ans.y=P[p2].y-P[p1].y;
	return ans;
};
int isleft(int p1,int p2,int p3){
   return crossP(MAKE_VECTOR(p1,p2),MAKE_VECTOR(p2,p3));
};
int compar(const void* a,const void* b){
   int dx, dy;
   dx=((struct node*)a)->x - ((struct node*)b)->x;
   dy=((struct node*)a)->y - ((struct node*)b)->y;
   return dx ? dx : dy;
};
int area3(int i,int j,int k){
   return ( P[j].x-P[i].x ) * ( P[k].y-P[j].y ) -
      ( P[j].y-P[i].y ) * ( P[k].x-P[j].x );
};
void convex_hull2(int n){
   int i, j, k;
   qsort(P,n,sizeof(P[0]),compar);
   np = 0;
   // Lower Concave
   for ( i = 0; i < n; ++i) {
      H[ np++ ] = i;
      while ( np > 2 && area3( H[np-3],H[np-2],H[np-1]) < 0) {
         H[np-2]=H[np-1];
         --np;
      }
      while ( i < n -1 && P[i+1].x == P[i].x ) ++i;
   }
   // Upper Concave
   for ( i = n - 2; i > 0; --i) {
      H[np++]=i;
      while ( np > 2 && area3( H[ np-3 ], H[ np-2 ], H[np-1] ) < 0) {
         H[np-2] = H[np-1];
         --np;
      }
```

```
        while ( i > 1 && P[i-1].x == P[i].x ) --i;
    }
}
double area(polygon p){
        double ans=0;
        for (int i=1;i<=p.ct;i++)

        ans+=(double)(p.p[i-1].x*p.p[i%p.ct].y-p.p[i%p.ct].x*p.p[i-1].
y);
        return .5*ans;
};
void MirrorClockWise(polygon &p){
        for (int i=0;i<=(p.ct-1)/2;i++) swap(p.p[i],p.p[p.ct-i-1]);
};
double angle(pt p1,pt p2,double s){
        double ans;
        pt p;
        p.x= p2.x - p1.x;
        p.y= p2.y - p1.y;
        if (p.x==0){
                if (p.y>0) ans= .5 * PI;
                else ans = 1.5 * PI;
        }
        else{
                ans = atan((double)p.y / (double)p.x);
                if (p.x < 0) ans += PI;
        }
        while(ans < 0) ans+= 2.0 * PI;
        if (ans>=PI) s += PI;
        if (ans>s) ans-=s;
        else ans=PI-(s-ans);
        while(ans>=PI) ans-=PI;
        if (feqs(ans,PI)) ans=0;
        return ans;
};
int dist(pt p1, pt p2){
        int dx=(p1.x-p2.x);
        int dy=(p1.y-p2.y);
        return dx*dx+dy*dy;
};
int dot(node p1,node p2){
        return p1.x*p2.x+p1.y*p2.y;
};
bool cmp(node n1,node n2){
        if (n1.x==n2.x)
                return n1.y<n2.y;
        else return n1.x<n2.x;
}
int main(){
        int n;
        int k1,k2;
        int ymin=(1<<29),ymax=-(1<<29);
        scanf("%d",&n);
        //memset(D,0,sizeof(D));
        //memset(H,0,sizeof(H));
        //memset(P,0,sizeof(P));
        bool iline=1;
        for (int i=0;i<n;i++) {
                scanf("%d %d",&P[i].x,&P[i].y);
                if (i>1 && iline){
                if (crossP(MAKE_VECTOR(0,1),MAKE_VECTOR(0,i)))
                                iline=0;
                }
        }
        if (iline){
                sort(P,P+n,cmp);
                int dx=(P[0].x-P[n-1].x);
                int dy=(P[0].y-P[n-1].y);
                printf("%d\n",dx*dx+dy*dy);
                return 0;
        }
        if (n==1){
                puts("0");
                return 0;
        }
        if (n==2){
                int dx=(P[0].x-P[1].x);
                int dy=(P[0].y-P[1].y);
                printf("%d\n",dx*dx+dy*dy);
                return 0;
        }
        convex_hull2(n);
        polygon p;
        p.ct=np;
        ymin=(1<<29); ymax=-(1<<29);
        for (int i=0;i<np;i++) {
                p.p[i].x=P[H[i]].x;
                p.p[i].y=P[H[i]].y;
                if (p.p[i].y>ymax){
                        ymax=p.p[i].y;
                        k2=i;
                }
                if (p.p[i].y<ymin){
                        ymin=p.p[i].y;
                        k1=i;
                }
        }
        if (area(p)<0) MirrorClockWise(p);
        int ans=-1;
        double s=0,rotate=0;
        double a1,a2;
        double ct1=0,ct2=0;
        while(ct1<2*np || ct2<2*np){
                        while(s>=PI) s-=PI;
                        if (fabs(PI-s)<EPS) s=0;
                        a1=angle(p.p[k1],p.p[(k1+1)%np],s);
                        a2=angle(p.p[k2],p.p[(k2+1)%np],s);
                        ans=max(dist(p.p[k1],p.p[k2]),ans);
                        if (feqs(a1,a2)){
                                k1=(k1+1)%np;
                                k2=(k2+1)%np;
                                ct1++;
                                ct2++;
                        }
                        else if (a1<a2){
                                k1=(k1+1)%np;
                                ct1++;
                        }
                        else {
                                k2=(k2+1)%np;
                                ct2++;
                        }
                        s+=min(a1,a2);
                }
        printf("%d\n",ans);
        return 0;
}
/*==================================================*\
|旋转卡壳算法 求两凸包的最近距离
\*==================================================*/
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<algorithm>
using namespace std;
#define PI (3.141592653589793)
#define EPS (1e-6)
#define INF (1e250)
#define feqs(x,y) (fabs((x)-(y))<EPS)
#define N 10100
struct pt{
        double x,y;
};
struct polygon{
        pt p[N];
        int ct;
};
struct line{
        double a,b,c;
};
double inline dist(pt p1,pt p2){
        return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
```

```
};
line SegToLine(pt p1,pt p2){
	line ans;
	ans.a=p2.y-p1.y;
	ans.b=p1.x-p2.x;
	ans.c=p2.x*p1.y-p1.x*p2.y;
	return ans;
};
double area(polygon p){
	double ans=0;
	for (int i=1;i<=p.ct;i++)
		ans+=(p.p[i-1].x*p.p[i%p.ct].y-p.p[i%p.ct].x*p.p[i-1].y);
	return .5*ans;
};
void MirrorClockWise(polygon &p){
	for (int i=0;i<=(p.ct-1)/2;i++) swap(p.p[i],p.p[p.ct-i-1]);
};
double dist_ptToSeg(pt p,pt p1,pt p2){
	double a=dist(p,p1);
	double b=dist(p,p2);
	double c=dist(p1,p2);
	if (feqs(a+b,c)) return 0;
	if (feqs(a+c,b) || feqs(b+c,a)) return min(a,b);
	double t1=-a * a + b * b + c * c;
	double t2= a * a - b * b + c * c;
	if (t1 <=0 || t2 <=0 ) return min(a,b);
	line   l1= SegToLine(p1,p2);
	return fabs(l1.a*p.x+l1.b*p.y+l1.c)/sqrt(l1.a*l1.a+l1.b*l1.b);
};
double dist_SegToSeg(pt p1,pt p2,pt p3,pt p4){
	return
min(min(dist_ptToSeg(p1,p3,p4),dist_ptToSeg(p2,p3,p4)),
		min(dist_ptToSeg(p3,p1,p2),dist_ptToSeg(p4,p1,p2)));
};
double angle(pt p1,pt p2,double s){
	double ans;
	pt p;
	p.x= p2.x - p1.x;
	p.y= p2.y - p1.y;
	if (feqs(p.x,0.0)){
		if (p.y>0) ans= .5 * PI;
		else ans = 1.5 * PI;
	}
	else{
		ans = atan(p.y / p.x);
		if (p.x < 0) ans += PI;
	}
	while(ans < 0) ans+= 2.0 * PI;
	if (ans>=PI) s += PI;
	if (ans>s) ans-=s;
	else ans=PI-(s-ans);
	while(ans>=PI) ans-=PI;
	if (feqs(ans,PI)) ans=0;
	return ans;
};
int main(){
	int n,m;
	polygon p1,p2;
	double ymax,ymin,ans,d;
	int k2,k1;
	while(scanf("%d%d",&n,&m),(n||m)){
		memset(p1.p,0,sizeof(p1.p));
		memset(p2.p,0,sizeof(p2.p));
		p1.ct=n;
		p2.ct=m;
		for(int i=0;i<n;i++)
scanf("%lf %lf",&p1.p[i].x,&p1.p[i].y);
		for (int i=0;i<m;i++)
scanf("%lf %lf",&p2.p[i].x,&p2.p[i].y);
		if (area(p1)<0) MirrorClockWise(p1);
		if (area(p2)<0) MirrorClockWise(p2);
		ymin=INF; ymax=-INF;
		for (int i=0;i<n;i++)
			if (p1.p[i].y<ymin){
				ymin=p1.p[i].y;
				k1=i;
			}
		for (int i=0;i<m;i++)
			if (p2.p[i].y>ymax){
				ymax=p2.p[i].y;
				k2=i;
			}
		double s=0,rotate=0,ans=INF;
		double a1,a2;
		int c1=0,c2=0;
		while(c1<2*n || c2<2*m){
			while(s>=PI) s-=PI;
			if (fabs(PI-s)<EPS) s=0;
			a1=angle(p1.p[k1],p1.p[(k1+1)%n],s);
			a2=angle(p2.p[k2],p2.p[(k2+1)%m],s);
			if (feqs(a1,a2)){
	d=dist_SegToSeg(p1.p[k1],p1.p[(k1+1)%n],p2.p[k2],p2.p[(k
2+1)%m]);
				ans=min(ans,d);
				k1=(k1+1)%n;
				k2=(k2+1)%m;
				c1++;
				c2++;
			}
			else if (a1<a2){
		d=dist_ptToSeg(p2.p[k2],p1.p[k1],p1.p[(k1+1)%n]);
				ans=min(ans,d);
				k1=(k1+1)%n;
				c1++;
			}
			else {
		d=dist_ptToSeg(p1.p[k1],p2.p[k2],p2.p[(k2+1)%m]);
				ans=min(d,ans);
				k2=(k2+1)%m;
				c2++;
			}
			s+=min(a1,a2);
			rotate+=min(a1,a2);
		}
		printf("%.5lf\n",ans);
	}
	return 0;
}
```

3.10 网格

```
/*==================================================*\
\*==================================================*/
#define abs(x) ((x)>0?(x):-(x))
struct point{int x,y;};

int gcd(int a,int b){
	return b?gcd(b,a%b):a;
}
/*==================================================*\
|1、多边形上的网格点个数
\*==================================================*/
int grid_onedge(int n,point* p){
	int i,ret=0;
	for (i=0;i<n;i++)
	ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-p[(i+1)%n].y));
	return ret;}
/*==================================================*\
|2、多边形内的网格点个数
\*==================================================*/
int grid_inside(int n,point* p){
	int i,ret=0;
	for (i=0;i<n;i++)
		ret+=p[(i+1)%n].y*(p[i].x-p[(i+2)%n].x);
	return (abs(ret)-grid_onedge(n,p))/2+1;
}
```

目　　录：

Chapter 4

Advanced Data structures and Algorithms

4.1 树

```
/*==================================================*\
|树形 DP(边带值)
\*==================================================*/
#include <iostream>
#define MIN(a, b) a<b?a:b
#include <vector>
#define inf 1000000001
#include <queue>
#define MAXN 500009
using namespace std;
#define MAX(a, b) a>b?a:b
struct ver {
    int v;
    int val;
    int next;
};
ver e[MAXN];
int p[MAXN], aid;
int dp[MAXN];
int n , L, R;
int dist[MAXN];
/*
在结点 u 判断选择是否满足条件时,
需要多记录一个 dist[u]表示从根 0 到 u 的距离
dp[v]则表示叶子到 v 的最优值.
知道某个节点，就知道了根 0 到它的距离以及轮到谁决策
这是很有用的性质，最终答案就是 dist[u]+dp[v]+w
if(dist[u] + dp[v] + w <= R && dist[u] + dp[v] + w >= L)
才更新答案
*/
void add(int a, int b, int val) {
    e[aid].next = p[ a ];
    e[aid].v = b;
    e[aid].val = val;
    p[ a ] = aid ++;
}
void dfs (int x, bool flag) {
    if(dist[ x ] > R) {
        dp[ x ] = 0;
        return ;
    }
    dp[ x ] = flag ? 0 : inf;
    if(p[ x ] == -1) dp[ x ] = 0;
    for(int j = p[ x ]; j != -1; j = e[ j ].next) {
        dist[e[ j ].v] = dist[ x ] + e[ j ].val;
        dfs (e[ j ].v, !flag);
        int len = dist[ x ] + dp[e[ j ].v] + e[ j ].val;
        if(len >= L && len <= R) {
            if(flag) {
                if(dp[ x ] < dp[e[ j ].v] + e[ j ].val) {
                    dp[ x ] = dp[e[ j ].v] + e[ j ].val;
                }
                //dp[ x ] = MAX(dp[ x ], dp[e[ j ].v] + e[ j ].val);
            }
            else {
                if(dp[ x ] > dp[e[ j ].v] + e[ j ].val) {
                    dp[ x ] = dp[e[ j ].v] + e[ j ].val;
                }
                //dp[ x ] = MIN(dp[ x ], dp[e[ j ].v] + e[ j ].val);
            }
        }
    }
}
int main() {
    int i, a, b , val;
    while(~scanf("%d %d %d", &n, &L, &R)) {
        memset(p, -1, sizeof(p));
        aid = 0;
        for(i = 1; i < n ; ++ i) {
            scanf("%d %d %d", &a, &b, &val);
            add(a, b, val);
        }
        dfs (0, 1);
        if(L <= dp[ 0 ] && dp[ 0 ] <= R) {
            printf("%d\n", dp[ 0 ]);
        }
        else {
            puts("Oh, my god!");
        }
    }
    return 0;
}
/*==================================================*\
|树形 DP(点带值)
\*==================================================*/
#include <iostream>
#include <vector>
#include <queue>
#define MAXN 111
using namespace std;
#define MAX(a, b) a>b?a:b
int val[MAXN][ 2 ];
int dp[MAXN][MAXN];
int n , c;
bool vist[MAXN];
struct ver {
    int v;
    int next;
};
ver e[MAXN*2];
int p[MAXN], aid;
void add(int a, int b) {
    e[aid].next = p[ a ];
    e[aid].v = b;
    p[ a ] = aid ++;
    swap(a, b);
    e[aid].next = p[ a ];
    e[aid].v = b;
    p[ a ] = aid ++;
}
//dp[ i ][ k ]  表示以 i 为根节点用了 k 个人去消灭虫的最大值
//dp[ 1 ][ c ]  为结果
//当 trooper = 0 时，即使不花费任何代价也拿不了，和背包搞混了，杯
具
void dfs (int x) {
    int i, j, k;
    int num = (val[ x ][ 0 ] + 19) / 20;
    for(i = num; i <= c; ++ i) {
        dp[ x ][ i ] = val[ x ][ 1 ];
    }
    vist[ x ] = true;
    for(i = p[ x ]; i != -1; i = e[ i ].next) {
        int y = e[ i ].v;
        if(vist[ y ]) continue;
        dfs (y);
        for(j = c; j >= num; -- j) {
            for(k = 1; j + k <= c; ++ k) {
                if(dp[ y ][ k ]) {
                    dp[ x ][j + k] = MAX(dp[ x ][j + k], dp[ x ][ j ] +
dp[ y ][ k ]);
                }
            }
        }
    }
}
int main() {
    int i, m;
```

```
while(cin >> n >> c) {
      if(n == -1 && c == -1) break;
      memset(dp, 0, sizeof(dp));
      memset(vist, false, sizeof(vist));
      memset(p, -1, sizeof(p));
      aid = 0;
      for(i = 1; i <= n ; ++ i) {
            cin >> val[ i ][ 0 ] >> val[ i ][ 1 ];
      }
      m = n;
      while(-- m) {
            int a, b;
            cin >> a >> b;
            add(a, b);
      }
      if(c == 0) {
            cout << 0 << endl;
            continue;
      }
      dfs (1);
      cout << dp[ 1 ][ c ] << endl;
}
return 0;
}
/*===================================================*\
|归并树（log(n)^2）
|求区间【l，r】中询问第 k 小的数字，其实从小到大排序后就是问第几
|个，n 个数字，m 个询问
\*===================================================*/
#include <iostream>
using namespace std;
#define MAX_SIZE 100010
struct Tree//线段树结构体
{
      int l, r, bit;
}tree[3 * MAX_SIZE];

int hash[22][MAX_SIZE];
void MakeTree(int k, int bit, int l, int r) {
//建立线段树，同时记录归并排序过程
        tree[k].bit = bit;
        tree[k].l = l, tree[k].r = r;
        if (l == r){
              hash[bit][l] = hash[0][l];
              return ;
        }
        int low = (l + r) >> 1;
        MakeTree(k << 1, bit + 1, l, low);
        MakeTree((k << 1) + 1, bit + 1, low + 1, r);
        int inf = l, i = l, j = low + 1;
        while (i <= low && j <= r){
               if (hash[bit + 1][i] < hash[bit + 1][j])
                   hash[bit][inf++] = hash[bit + 1][i++];
               else
                   hash[bit][inf++] = hash[bit + 1][j++];
        }
        while (i <= low) hash[bit][inf++] = hash[bit + 1][i++];
        while (j <= r) hash[bit][inf++] = hash[bit + 1][j++];
}
int find(int k, int x) {
//查找 x 在区间[tree[k].l, tree[k].r]，比 x 小的个数
      int l = tree[k].l, r = tree[k].r, bit = tree[k].bit;
      int mid;
      while (l <= r){
              mid = (l + r) >> 1;
              if (hash[bit][mid] < x)
              {
                    l = mid + 1;
              }
              else
                   r = mid - 1;
      }
      return l - tree[k].l;
}
int rank(int k, int x, int l, int r) {
//查找在区间[l, r]中比 x 小的个数
        if (l == tree[k].l && r == tree[k].r){
              return find(k, x);
        }
        int low = (tree[k].l + tree[k].r) >> 1;
        int c1 = 0, c2 = 0;
        if (r <= low)
        c1 = rank(2 * k, x, l, r);
        else if (l > low){
               c2 = rank(2 * k + 1, x, l, r);
        }
        else{
              c1 = rank(2 * k, x, l, low);
              c2 = rank(2 * k + 1, x, low + 1, r);
        }
        return c1 + c2;
}
bool check(int x, int l, int r, int k){
        int rs = rank(1, x, l, r);
        return rs < k;
}
int main(){
      int i, s, p, k, n, t, m, l, r, mid;
      while (scanf("%d%d", &n, &m) != EOF){
            for (i = 1; i <= n; i++){
                  scanf("%d", &hash[0][i]);
            }
            MakeTree(1, 0, 1, n);
            for (i = 0; i < m; i++) {
                  scanf("%d%d%d", &s, &p, &k);
                  if (s > p){
                        t = s;
                        s = p;
                        p = t;
                  }
                  int re = 1;
                  l = 1, r = n;
                  while (l <= r){
                        mid = (l + r) >> 1;
                        if (check(hash[0][mid], s, p, k)){
                              re = mid;
                              l = mid + 1;
                        }
                        else r = mid - 1;
                  }
                  printf("%d\n", hash[0][re]);
            }
      }
      return 0;
}
/*===================================================*\
|划分树（log(n)）
|划分树，求解给定区间[l, r]的第 k 小。
|和归并是相反，相当于把归并树倒过来。
|当访问到节点 t 时，sum[bit][i](left <= i <= right)表示从第 left 个数到
|第 i 个数中有 sum[bit][i]个数进入了左子树。
|假如我们在区间[left, right]中求解[l, r]中的第 k 小的时候(left <= l <= r
|<= right)，看在区间[l,r]中有多少个数进入了左子树，如果有 s 个，如
|果 s>=k,则递归到左节点，但是要注意的是，当访问的左节点的时候，
|求解的区间[l, r]要发生变化 ,即找到区间[l, r]中第一次出现在左子树的
|位置 ll，和最后一次出现在左子树的位置 rr,然后访问左子树的区间
[ll, rr]，一直到 left=right，返回元数列 sa[left]为所求
\*===================================================*/
#include <iostream>
#include <algorithm>
using namespace std;
#define MAX_SIZE 100010
struct Tree
{
      int l, r, bit;
}tree[3 * MAX_SIZE];

int sum[24][MAX_SIZE]; //记录划分过程的数组，需要空间为 n*logn
int a[2][MAX_SIZE];    // 滚动数组， 在划分过程中，保存上一层的序列,最开始时将
                       //将数组存放到 a[0][i]中
int sa[MAX_SIZE];      // 排序数组
```

```
void MakeTree(int k, int bit, int l, int r){
        tree[k].bit = bit;
        tree[k].l = l, tree[k].r = r;
        if (l == r)
             return ;
        int inf = (bit & 1);
        int low = (l + r) >> 1;
        int temp = 0;
        int t = 0;
        int c = 0;
        while (low - temp >= l && sa[low - temp] == sa[low])temp++;
        int i = l, j = low + 1;
        for (;l <= r; l++){
              if (a[inf][l] < sa[low]){
                    c++;
                    sum[bit][l] = c;
                    a[1 - inf][i++] = a[inf][l];
              }
              else if (a[inf][l] == sa[low] && t < temp){
                    c++;
                    sum[bit][l] = c;
                    a[1 - inf][i++] = a[inf][l];
                    t++;
              }
              else {
                     sum[bit][l] = c;
                     a[1 - inf][j++] = a[inf][l];
              }
        }
        l = tree[k].l;
        MakeTree(2 * k, bit + 1, l, low);
        MakeTree(2 * k + 1, bit + 1, low + 1, r);
}
int Find_rank(int k, int l, int r, int t){
      int bit = tree[k].bit;
      int left = tree[k].l, right = tree[k].r;
      int low = (left + right) >> 1;
      if (right == left){
         return sa[left];
      }
      int s;
      if (l == left)
         s = sum[bit][r];
      else
         s = sum[bit][r] - sum[bit][l - 1];
      if (s >= t){
         if (l == left)
         return Find_rank(2 * k, left, left + sum[bit][r] - 1, t);
         else
         return Find_rank(2 * k, left + sum[bit][l - 1], left + sum[bit][r] - 1, t);
      }
      else{
          if (l == left)
          return Find_rank(2 * k + 1, low + 1, low + 1 + r - l - s, t - s);
          else
          return Find_rank(2 * k + 1, low + 1 + l - left - sum[bit][l - 1], low + r - left + 1 - sum[bit][r], t - s);
      }
}
int main(){
      int i, j, k, l, m, n, t, r;
      while (scanf("%d%d", &n, &m) != EOF){
            memset(sum, 0, sizeof(sum));
            for (i = 1; i <= n; i++) {
                  scanf("%d", &a[0][i]);
                  sa[i] = a[0][i];
            }
            sort(sa + 1, sa + n + 1);
            MakeTree(1, 0, 1, n);
            for (i = 0; i < m; i++){
                  scanf("%d%d%d", &l, &r, &k);
                  printf("%d\n", Find_rank(1, l, r, k));
            }
      }
      return 0;
}
/*==================================================*\
| 左偏树 合并复杂度 O(log N)
| INIT: init()读入数据并进行初始化;
| CALL: merge() 合并两棵左偏树; ins() 插入一个新节点;
|         top() 取得最小结点; pop() 取得并删除最小结点;
|         del() 删除某结点; add() 增/减一个结点的键值;
|         iroot() 获取结点 i 的根;
\*==================================================*/
#define typec int                      // type of key val
const int na = -1;
struct node { typec key; int l, r, f, dist; } tr[N];
int iroot(int i){                      // find i's root
        if (i == na) return i;
        while (tr[i].f != na) i = tr[i].f;
        return i;
}
int merge(int rx, int ry){                     // two root: rx, ry
         if (rx == na) return ry;
         if (ry == na) return rx;
         if (tr[rx].key > tr[ry].key) swap(rx, ry);
         int r = merge(tr[rx].r, ry);
         tr[rx].r = r; tr[r].f = rx;
         if (tr[r].dist > tr[tr[rx].l].dist)
             swap(tr[rx].l, tr[rx].r);
         if (tr[rx].r == na) tr[rx].dist = 0;
         else tr[rx].dist = tr[tr[rx].r].dist + 1;
 return rx;                             // return new root
}
int ins(int i, typec key, int root){          // add a new node(i, key)
             tr[i].key = key;
             tr[i].l = tr[i].r = tr[i].f = na;
             tr[i].dist = 0;
return root = merge(root, i);                 // return new root
}
int del(int i) {                              // delete node i
           if (i == na) return i;
           int x, y, l, r;
           l = tr[i].l; r = tr[i].r; y = tr[i].f;
           tr[i].l = tr[i].r = tr[i].f = na;
           tr[x = merge(l, r)].f = y;
           if (y != na && tr[y].l == i) tr[y].l = x;
           if (y != na && tr[y].r == i) tr[y].r = x;
           for ( ; y != na; x = y, y = tr[y].f) {
                if (tr[tr[y].l].dist < tr[tr[y].r].dist)
                      swap(tr[y].l, tr[y].r);
                if (tr[tr[y].r].dist + 1 == tr[y].dist) break;
           tr[y].dist = tr[tr[y].r].dist + 1;
           }
        if (x != na) return iroot(x);              // return new root
   else return iroot(y);
}
node top(int root){
return tr[root];
}
node pop(int &root){
           node out = tr[root];
           int l = tr[root].l, r = tr[root].r;
           tr[root].l = tr[root].r = tr[root].f = na;
           tr[l].f = tr[r].f = na;
           root = merge(l, r);
return out;
}
int add(int i, typec val) // tr[i].key += val
{
           if (i == na) return i;
           if (tr[i].l == na && tr[i].r == na && tr[i].f == na) {
                  tr[i].key += val;
                  return i;
           }
         typec key = tr[i].key + val;
         int rt = del(i);
   return ins(i, key, rt);
}
void init(int n){
```

```
    for (int i = 1; i <= n; i++) {
        scanf("%d", &tr[i].key);            //%d: type of key
        tr[i].l = tr[i].r = tr[i].f = na;
        tr[i].dist = 0;
    }
}
```

4.2 Trie 字典树（前缀树）

```
/*==================================================*\
|字典树（动态）内存小
\*==================================================*/
#include <iostream>
using namespace std;
const int MAXM = 30,KIND = 26;
int m;
struct node{
    char* s;
    int prefix;
    bool isword;
    node* next[KIND];
    node() {
        s = NULL;
        prefix = 0;
        isword = false;
        memset(next,0,sizeof(next));
    }
}*root;                              //根
void insert(node *root,char *s) {    //插入
    node *p = root;
    for (int i = 0;s[i];i++){
        int x = s[i] - 'a';
        p->s = s+i;
        if (p->next[x] == NULL)
            p->next[x] = new node;
        p = p->next[x];
        p->prefix++;
    }
    p->isword = true;
}
bool del(node *root,char *s) {       //删除
    node *p = root;
    for (int i = 0;s[i];i++){
        int x = s[i] - 'a';
        if (p->next[x] == NULL)
            return false;
        p = p->next[x];
    }
    if (p->isword)
        p->isword = false;
    else
        return false;
    return true;
}
bool search(node *root,char* s) {    //查找
    node* p = root;
    for (int i = 0;s[i];i++){
        int x = s[i]-'a';
        if (p->next[x] == NULL)
            return false;
        p = p->next[x];
    }
    return p->isword;
}
int count(node *root,char *s) {      //统计后缀
    node *p = root;
    for (int i = 0;s[i];i++){
        int x = s[i] - 'a';
        if (p->next[x] == NULL)
            return 0;
        p = p->next[x];
    }
    return p->prefix;
}
int main(){
    m = 0;
    root = new node;
    char s[MAXM];
    while (gets(s)){
        if (strcmp(s,"") == 0)
            break;
        insert(root,s);
    }
    while (gets(s))
        printf("%d\n",count(root,s));
}
/*==================================================*\
|字典树（静态）速度快
\*==================================================*/
#include <iostream>
using namespace std;
const int MAXN=10010,MAXM=30,kind=26;
int m;
struct node{
    char *s;
    int count;
    bool isword;
    node *next[kind];
    void init(){
        s=NULL;
        count=0;
        isword=false;
        memset(next,0,sizeof(next));
    }
}a[MAXN*MAXM],*root;
void insert(node *root,char *s){
    int i,index;
    node *p=root;
    for(i=0;s[i];i++){
        index=s[i]-'a';
        p->s=s+i;
        if(p->next[index]==NULL)
        {
            a[m].init();
            p->next[index]=&a[m++];
        }
        p=p->next[index];
        p->count++;
    }
    p->isword=true;
}
bool del(node *root,char *s){
    node *p=root;
    int i,index;
    for(i=0;s[i];i++){
        index=s[i]-'a';
        if(p->next[index]==NULL)
            return false;
        p=p->next[index];
    }
    if(p->isword)
        p->isword=false;
    else
        return false;
    return true;
}
bool search(node *root,char *s){
    node *p=root;
    int i,index;
    for(i=0;s[i];i++){
        index=s[i]-'a';
        if(p->next[index]==NULL)
            return false;
        p=p->next[index];
    }
    return p->isword;
}
int count(node *root,char *s){
    node *p=root;
    int i,index;
    for(i=0;s[i];i++){
        index=s[i]-'a';
```

```
        if(p->next[index]==NULL)
            return 0;
        p=p->next[index];
    }
    return p->count;
}
int main(){
    int m=0;
    a[m].init();
    root=&a[m++];
    char s[MAXM];
    while(gets(s)){
        if(strcmp(s,"")==0)
            break;
        insert(root,s);
    }
    while(gets(s))
        printf("%d\n",search(root,s));
    return 0;
}
/*==================================================*\
| Trie 树(k 叉)
| INIT: init();
| 注: tree[i][tk]>0 时表示单词存在，当然也可赋予它更多含义;
\*==================================================*/
const int tk = 26, tb = 'a';            // tk 叉; 起始字母为 tb;
int top, tree[N][tk + 1];               // N: 最大结点个数
void init(){
        top = 1;
    memset(tree[0], 0, sizeof(tree[0]));
}
int search(char *s){                    // 失败返回 0
    for (int rt = 0; rt = tree[rt][*s - tb]; )
      if (*(++s) == 0) return tree[rt][tk];
   return 0;
}
void insert(char *s, int rank = 1){
        int rt, nxt;
        for (rt = 0; *s; rt = nxt, ++s) {
           nxt = tree[rt][*s - tb];
        if (0 == nxt) {
           tree[rt][*s - tb] = nxt = top;
      memset(tree[top], 0, sizeof(tree[top]));
            top++;
        }
     }
 tree[rt][tk] = rank;//1 表示存在 0 表示不存在，也可以赋予其其他含义
}
void delete(char *s){                   // 只做标记，假定 s 一定存在
        int rt = 0;
        for ( ; *s; ++s) rt = tree[rt][*s - tb];
          tree[rt][tk]=0;
}
int prefix(char *s){                    // 最长前缀
     int rt = 0, lv;
     for (lv = 0; *s; ++s, ++lv) {
         rt = tree[rt][*s - tb];
   if (rt == 0) break;
    }
  return lv;
}
/*==================================================*\
| Trie 树(左儿子右兄弟)
| INIT: init();
\*==================================================*/
int top;
struct trie { char c; int l, r, rk; } tree[N];
void init(){
  top = 1;
  memset(tree, 0, sizeof(tree[0]));
}
int search(char *s) {                   // 失败返回 0
       int rt;
       for (rt = 0; *s; ++s) {
           for (rt = tree[rt].l; rt; rt = tree[rt].r)
             if (tree[rt].c == *s) break;
            if (rt == 0) return 0;
    }
    return tree[rt].rk;
}
void insert(char *s, int rk = 1){       //rk: 权或者标记
     int i, rt;
     for (rt = 0; *s; ++s, rt=i) {
        for (i = tree[rt].l; i; i = tree[i].r)
           if (tree[i].c == *s) break;
              if (i == 0) {
                 tree[top].r = tree[rt].l;
                 tree[top].l = 0;
                 tree[top].c = *s;
                 tree[top].rk = 0;
                 tree[rt].l = top;
                 i = top++;
              }
    }
  tree[rt].rk=rk;
}
void delete(char *s){                   // 假定 s 已经存在，只做标记
        int rt;
        for (rt = 0; *s; ++s) {
           for (rt = tree[rt].l; rt; rt = tree[rt].r)
              if (tree[rt].c == *s) break;
        }
     tree[rt].rk = 0;
}
int prefix(char *s){                    // 最长前缀
    int rt = 0, lv;
    for (lv = 0; *s; ++s, ++lv) {
        for (rt = tree[rt].l; rt; rt = tree[rt].r)
           if (tree[rt].c == *s) break;
           if (rt == 0) break;
        }
  return lv;
}
```

4.3 树状数组

```
/*==================================================*\
|树状数组（一维）
|功能：可以快速的访问某个区间的和，改变一个节点的值，
|优点： 其效率非常高，而且代码长度较短，很方便使用，占用空间较
|少。
|缺点：功能较少，每次操作只能改变一个节点的值，遇到改变一段
|       节点的值的时候，只能用线段树解决。
|注意：不能改变节点为 0 的值，下表是从 1-n 记录的，当遇到有下标为
|0 的时候，要特殊处理。
\*==================================================*/
#include<iostream>
using namespace std;
#define INT int
#define MAX_SIZE 100000
INT c[MAX_SIZE],N;
INT lowbit(INT x)
{
    return x^(x&(x-1));
}
void add(INT i,INT m)
{
    while(i<=N)
    {
        c[i]+=m;
        i+=lowbit(i);
    }
}
INT Find(INT i)
{
    INT s=0;
    while(i)
    {
        s+=c[i];
        i-=lowbit(i);
    }
    return s;
```

```
}
int main()
{
	return 0;
}
/*======================================================*\
|树状数组（二维）
|二维数状数组，用于查找某个子矩阵所有元素的和，可以改变矩阵中某
|个元素的值，时间复杂度都是 lon(n)*long(n)，矩阵大小是 M 行 N 列，
|下标范围 (1,1)到(M,N),
|注意：下标不是从(0,0)开始的，也不能改变(0,0)的值，否则超时，进入
|死循环。
\*======================================================*/
#include<iostream>
using namespace std;
#define INT int
#define MAX_SIZE 1010
INT c[MAX_SIZE][MAX_SIZE],M,N;
INT lowbit(INT x){
	return x^(x&(x-1));
}
void add(INT i,INT j,INT x)
{
	INT k;
	while(i<=M){
		k=j;
		while(k<=N)
		{
			c[i][k]+=x;
			k+=lowbit(k);
		}
		i+=lowbit(i);
	}
}
INT Find(INT i,INT j){
	INT k,s=0;
	while(i){
		k=j;
		while(k)
		{
			s+=c[i][k];
			k-=lowbit(k);
		}
		i-=lowbit(i);
	}
	return s;
}
int main()
{}
```

4.4 线段树

```
/*======================================================*\
|线段树----区间求和
|线段数的建立，插入和访问
|给定一个序列 a[],有 m 次操作 ，操作包括访问 a[i]到 a[j]（i<=j）的和。
|修改 a[i]到 a[j]的所有数都加上 cove
|算法复杂度 n*(long n)
\*======================================================*/
poj  3468
#include<iostream>
using namespace std;
#define MAX 100010
#define INT int
INT a[MAX+4];
struct
{
	INT l,r;		//区间
	INT s,cove;	//s 表示 l-r 的和，cove 表示在这段数中每个数
都加了 cove
}line[3*MAX];
void Make_Tree(INT k,INT l,INT r)	//建立 1-n 的线段树
{
	line[k].cove=0;
	if(l==r)	//当 l=r 时，表示只有一个点，直接把这个数给 line[k].s
	{
		line[k].l=line[k].r=r;
		line[k].s=a[r];
		return ;
	}
	int m=(l+r)/2;
	line[k].l=l;
	line[k].r=r;
	Make_Tree(2*k,l,m);
	Make_Tree(2*k+1,m+1,r);
	line[k].s=line[2*k].s+line[2*k+1].s;
}

//插入线段 l-r，表示在区间 L 到 r 上的每个数都增加 cove，k 第一次调
用时为 k=1
void Insert(INT k,INT l,INT r,INT cove)
 {
	if(line[k].l==l&&line[k].r==r)//当插入线段与当前线段相同，直接
赋值 cove 结束
	{
		line[k].cove+=cove;
		return ;
	}
	if(r<=line[2*k].r)	//如果插入的线段在当前线段的左边，则插
入到左儿子
	{
		Insert(2*k,l,r,cove);
	}
	else if(l>=line[2*k+1].l)//在右边时，插入到右儿子
	{
		Insert(2*k+1,l,r,cove);
	}
	else //在左右之间，分别插入，但和上面的插入有所不同
	{
		Insert(2*k,l,line[2*k].r,cove);
		Insert(2*k+1,line[2*k+1].l,r,cove);
	}
	line[k].s=line[2*k].s+line[2*k+1].s+line[2*k].cove*(line[2*k].
r-line[2*k].l+1)+line[2*k+1].cove*(line[2*k+1].r-line[2*k+1].l+1);
	  //  通过插入，更新 sum 的值，
}
INT Find(INT k,INT l,INT r)	//访问线段 l-r
{
	INT s;
	if(line[k].l==l&&line[k].r==r)
	   //当前线段与访问线段相同，直接返回
	{
		return line[k].s+line[k].cove*(r-l+1);
	      //返回 sum 和加上去的 cove
	}
	if(r<=line[2*k].r)
	 //要访问的线段在当前线段的左边，则访问左儿子
	{
		s=Find(2*k,l,r);
	}
	else if(l>=line[2*k+1].l)//在当前线段的右边访问右儿子
	{
		s=Find(2*k+1,l,r);
	}
/ /在当前线段的左右儿子之间，分别访问求和
	else s=Find(2*k,l,line[2*k].r)+Find(2*k+1,line[2*k+1].l,r);
	return s+line[k].cove*(r-l+1);
//将访问所求的和再加上增加的  cove  值为该区间的总和即要求的值(因
为插入过程中，当插入到当前线段的时候，下面的线段没有有插入)
}
int main()
{
		return 0;
}
/*======================================================*\
|线段树----涂色问题
|线段树：  涂色问题，给一个线段涂色，每次涂色的区间为 l-r，
|通过不断的涂色，先涂的颜色将被后涂的覆盖掉，求最后的涂色情况，
|输入出每种颜色各有多少段，如果没有就不用输出
\*======================================================*/
#include<iostream>
using namespace std;
```

```
#define MAX 8010
#define INT int
INT sum[MAX];
INT T[MAX];
struct tree
{
	INT l,r;	//线段区间
	INT cove;	//	颜色
}line[4*MAX];
void Make_Tree(INT k,INT l,INT r)	//建立线段树
{
	line[k].cove=-1;	//最开始都标记未被涂色
	line[k].l=l;
	line[k].r=r;
	if(l==r){
		return ;
	}
	int m=(l+r)/2;
	Make_Tree(2*k,l,m);
	Make_Tree(2*k+1,m+1,r);
}
void Insert(INT k,INT l,INT r,INT cove)	//线段插入
{
	int m;
	if(line[k].l==l&&line[k].r==r)//当插入的线段刚好为当前线段时，直接插入，结束
	{
		line[k].cove=cove;
		return ;
	}
	if(line[k].cove==cove)//当插入的颜色与当前颜色相同时结束
		return ;
	if(line[k].cove!=-1)	//如果当前线段为被涂色，把他的颜色给他的左右儿子(因为在前次插入到当前线段的时候就已经结束，当前线段以下的线段没有涂色)
	{
		line[2*k].cove=line[2*k+1].cove=line[k].cove;
	}
	line[k].cove=-1;	//标记当前线段为被涂色
	if(r<=line[2*k].r)// 给他的左儿子涂色
	{
		Insert(2*k,l,r,cove);
	}
	else if(l>=line[2*k+1].l)//给他的右儿子涂色
	{
		Insert(2*k+1,l,r,cove);
	}
	else			//左右儿子涂色
	{
		Insert(2*k,l,line[2*k].r,cove);
		Insert(2*k+1,line[2*k+1].l,r,cove);
	}
}
void Tab(INT l,INT r,INT cove)	//记录找到的被涂色线段
{
	int i;
	for(i=l;i<=r;i++)
		sum[i]=cove;
	return ;
}
void Find(INT k)//查找线段的涂色情况，并调用 Tab 函数记录
{
	if(line[k].cove!=-1)//如此线段被涂色，记录
	{
		Tab(line[k].l,line[k].r,line[k].cove);
		return ;
	}
	if(line[k].l==line[k].r)//当为一个单位线段时，直接记录
	{
		sum[line[k].l]=line[k].cove;
		return ;
	}
	Find(2*k);	//记录左儿子
	Find(2*k+1);//记录右儿子
}
int main(){
		memset(sum,-1,sizeof(sum));
memset(T,0,sizeof(T));Make_Tree(1,1,8001);}
/*=====================================================*\
|线段树----离散化求矩形并的周长（线段树+离散化+扫描线）
|HDU	1828
\*=====================================================*/
#include <iostream>
#include <algorithm>
using namespace std;

const int M = 5003;
const int N = 10002;
typedef struct
{
	int s,e,pp;//三点确定一条直线位置
	// 1 stand for the start line
	// 0 stand for the end line
	int status;
}Line;
bool cmp(Line a, Line b)
{
	if(a.pp == b.pp)
	{
		return a.status > b.status;
	}
	else
		return a.pp < b.pp;
}
Line Lx[2 * M], Ly[2 * M];
int n, ans;
int *level;
void init(){
	int x1, y1, x2, y2;
	int kk = 0;
	for(int i = 0; i < n; i++){
		scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
		/**///////////////初始化始边//////////////////////////////////////
	 Lx[kk].s = x1; Lx[kk].e = x2; Lx[kk].pp = y1; Lx[kk].status = 1;
	 Ly[kk].s = y1; Ly[kk].e = y2; Ly[kk].pp = x1; Ly[kk].status = 1;
		kk++;
		/**///////////////初始化终边///////////////////////////////////
	 Lx[kk].s = x1; Lx[kk].e = x2; Lx[kk].pp = y2; Lx[kk].status = 0;
	 Ly[kk].s = y1; Ly[kk].e = y2; Ly[kk].pp = x2; Ly[kk].status = 0;
		kk++;
	}
	n = kk;
	ans = 0;
	sort(Lx, Lx + n, cmp);
	sort(Ly, Ly + n, cmp);
}
void solve(Line * str){
	int i, j;
	/**/////////////对层次初始化/////////////
	for(i = -10000; i <= 10000; i++)
		level[i] = 0;
	for(i = 0; i < n; i++){
		if(str[i].status == 1){			//对始边做处理
			for(j = str[i].s ; j < str[i].e; j++){
				level[j]++;
				/**//////////////始边由-> 1，可以确定是边缘边/////////////
				if(level[j] == 1)
					ans++;
			}
		}
		else{
			for(j = str[i].s ; j < str[i].e; j++) {	//对终边做处理
				level[j]--;
				/**//////////////终边由-> 0，可以确定是边缘边/////////////
				if(level[j] == 0)
					ans++;
			}
		}
	}
}
int main(){
```

```
    level = new int[2 * N];
    level += N;
    while(cin>>n){
        init();
        solve(Lx);
        solve(Ly);
        cout<<ans<<endl;
    }
    return 0;
}
/*==================================================*\
|线段树----离散化求矩形并的面积（线段树+离散化+扫描线）
|POJ   1151
|本题与 poj 1177 picture 极相似,现在回想起来甚至比 1177 还要简
|单一些.与 1177 不同的是，本题中的坐标是浮点类型的，故不能将坐
|标直接离散.我们必须为它们建立一个对应关系,用一个整数去对应一个
|浮点数这样的对应关系在本题的数组 y[] 中
\*==================================================*/
#include<iostream>
#include<algorithm>
#include<cmath>
#include<iomanip>
using namespace std;
struct node
{
   int st, ed, c;              //c : 区间被覆盖的层数, m: 区间的测度
   double m;
}ST[802];
struct line
{
    double x,y1,y2;
//纵方向直线, x:直线横坐标, y1 y2:直线上的下面与上面的两个纵坐标
bool s;         //s = 1 : 直线为矩形的左边, s = 0:直线为矩形的右边
}Line[205];
double y[205],ty[205];
 //y[]  整数与浮点数的对应数组; ty[]:用来求 y[]的辅助数组
void build(int root, int st, int ed)
{
    ST[root].st = st;
    ST[root].ed = ed;
    ST[root].c = 0;
    ST[root].m = 0;
    if(ed - st > 1){
        int mid = (st+ed)/2;
        build(root*2, st, mid);
        build(root*2+1, mid, ed);
     }
}
inline void updata(int root){
if(ST[root].c > 0)
//将线段树上区间的端点分别映射到 y[]数组所对应的浮点数上,由此计算出测度
    ST[root].m = y[ST[root].ed-1] - y[ST[root].st-1];
else if(ST[root].ed - ST[root].st == 1)
    ST[root].m = 0;
else ST[root].m = ST[root*2].m + ST[root*2+1].m;
}
void insert(int root, int st, int ed){
     if(st <= ST[root].st && ST[root].ed <= ed){
          ST[root].c++;
          updata(root);
          return ;
     }
if(ST[root].ed - ST[root].st == 1)return ;//不出错的话这句话就是冗余的
      int mid = (ST[root].ed + ST[root].st)/2;
      if(st < mid) insert(root*2, st, ed);
      if(ed > mid) insert(root*2+1, st, ed);
      updata(root);
}
void Delete(int root, int st, int ed){
      if(st <= ST[root].st && ST[root].ed <= ed){
        ST[root].c--; updata(root);
        return ;
      }
if(ST[root].ed - ST[root].st == 1)return ; //不出错的话这句话就是冗余的
      int mid = (ST[root].st + ST[root].ed)/2;
      if(st < mid) Delete(root*2, st, ed);
      if(ed > mid) Delete(root*2+1, st, ed);
      updata(root);
  }
int Correspond(int n, double t){
//二分查找出浮点数 t 在数组 y[]中的位置(此即所谓的映射关系)
      int low,high,mid;
      low = 0; high = n-1;
      while(low < high){
         mid = (low+high)/2;
         if(t > y[mid])
            low = mid + 1;
         else high = mid;
       }
    return high+1;
}
bool cmp(line l1, line l2){
     return l1.x < l2.x;
}
int main()
{
        int n,i,num,l,r,c=0;
        double area,x1,x2,y1,y2;
        while(cin>>n, n){
            for(i = 0; i < n; i++){
                cin>>x1>>y1>>x2>>y2;
                Line[2*i].x = x1; Line[2*i].y1 = y1;
                Line[2*i].y2 = y2; Line[2*i].s = 1;
                Line[2*i+1].x = x2; Line[2*i+1].y1 = y1;
                Line[2*i+1].y2 = y2; Line[2*i+1].s = 0;
                ty[2*i] = y1; ty[2*i+1] = y2;
             }
        n <<= 1;
        sort(Line, Line+n, cmp);
        sort(ty, ty+n);
         y[0] = ty[0];
//处理数组 ty[]使之不含重覆元素,得到新的数组存放到数组 y[]中
       for(i=num=1; i < n; i++)
           if(ty[i] != ty[i-1])
               y[num++] = ty[i];
build(1, 1, num); //树的叶子节点与数组 y[]中的元素个数相同,以便建立一一对应的关系
       area = 0;
       for(i = 0; i < n-1; i++){
           //由对应关系计算出线段两端在树中的位置
            l = Correspond(num, Line[i].y1);
            r = Correspond(num, Line[i].y2);
       if(Line[i].s)                       //插入矩形的左边
            insert(1, l, r);
       else                                //删除矩形的右边
            Delete(1, l, r);
        area += ST[1].m * (Line[i+1].x -Line[i].x);
    }
    cout<<"Test case #"<<++c<<endl<<"Total explored area: ";
    cout<<fixed<<setprecision(2)<<area<<endl<<endl;
  }
return 0;
}
/*==================================================*\
| 区间最大频率
| You are given a sequence of n integers a1 , a2 , ... , an
in non-decreasing order. In addition to that, you are given
several queries consisting of indices i and j (1 ≤ i ≤ j ≤n). For
each query, determine the most frequent value among
the integers ai , ... , aj. POJ 3368 Frequent values
求区间中数出现的最大频率

方法一:线段树.
先离散化。因为序列是升序，所以先将所有值相同的点缩成一点。这样 n 规模就缩小了。建立一个数据结构
记录缩点的属性：在原序列中的值 id,和该值有多少个 num
比如序列
10
-1 -1 1 1 1 1 3 10 10 10
```

缩点后为:下标 1 2 3 4
id -1 1 3 10
num 2 4 1 3
然后建树，树的属性有区间最大值(也就是频率)和区间总和。
接受询问的时候。接受的是原来序列的区间[be,ed]
我们先搜索一下两个区间分别在离散化区间后的下标。
比如接受[2,3]时候相应下标区间就是[1,2];[3,10]的相应下标区间是[2,4];处理频率的时候，我们发现两个极端，也就是左右两个端点的频率不好处理。因为它们是不完全的频率也就是说有部分不在区间内。但是如果对于完全区间，也就是说左右端点下标值完全在所求区间内。
比如上例的[2,3]不好处理。但是如果是[1,6]，或是[1,10]就很好处理了，只要像 RMQ 一样询问区间最大值就可以了。

方法二:RMQ. 我们可以转化一下问题。将左右端点分开来考虑。
现在对于离散后的询问区间我们可以分成 3 个部分.左端点，中间完全区间，右端点。对于中间完全区间线段树或 RMQ 都能轻松搞定。只要特判一左右的比较一下就得最后解了。
===/

```
const int N = 100010;
struct NODE{
int b, e;                    // 区间[b, e]
int l, r;                    // 左右子节点下标
int number;                  // 区间内的最大频率值
int last;
// 以 data[e] 结尾且与 data[e] 相同的个数:data[e-last+1]...data[e]
}node[N*2+1];
int len, data[N];
int main(void){
        int n;
        while( scanf("%d", &n), n ){
            int i, q, a, b;
            scanf("%d", &q);
            for( i=0; i < n; i++ ) scanf("%d", &data[i]);
                len = 0;                // 下标
               build(0, n-1);
           while( q-- ){
              scanf("%d%d", &a, &b);
              printf("%d\n", query(0, a-1, b-1));
              // 输出区间的最大频率值,而非 data[]
           }
        }
 return 0;
}
int build(int a, int b){             // 建立线段树
        int temp = len, mid = (a+b)/2;
        node[temp].b = a, node[temp].e = b;
        len++;
        if( a == b ){
          node[temp].number = 1;
          node[temp].last = 1; //
       return temp;
}
         node[temp].l = build(a, mid);
         node[temp].r = build(mid+1, b);
         int left_c=node[temp].l, right_c=node[temp].r, p,
         lcount=0, rcount=0, rec, max=0;
         rec = data[mid]; p = mid;
         while( p >= a && data[p] == rec ) { p--, lcount++; }
         node[left_c].last = lcount; //
         rec = data[mid+1]; p = mid+1;
         while( p <= b && data[p] == rec ) { p++, rcount++; }
         node[right_c].last = rcount;//
         if( data[mid] == data[mid+1] ) max = lcount+rcount;
if( node[left_c].number > max ) max =node[left_c].number;
if( node[right_c].number > max ) max =node[right_c].number;
node[temp].number = max;
  return temp;
}
int query(int index, int a, int b){
           int begin=node[index].b, end=node[index].e,
           mid=(begin+end)/2;
if( a == begin && b == end ) return node[index].number;
if( a > mid ) return query(node[index].r, a, b);
if( b < mid+1 ) return query(node[index].l, a, b);
        int temp1, temp2, max;
if( node[index].l > 0 ) temp1 = query(node[index].l,a, mid);
if( node[index].r > 0 ) temp2 = query(node[index].r,mid+1, b);
max = temp1 > temp2 ? temp1 : temp2;
if( data[mid] != data[mid+1] ) return max;
temp1 = node[ node[index].l ].last > (mid-a+1) ?
(mid-a+1) : node[ node[index].l ].last;
temp2 = node[ node[index].r ].last > (b-mid) ? (b-mid) :
node[ node[index].r ].last;
if( max < temp1+temp2 ) max = temp1+temp2;
  return max;
}
```

4.5 并查集

/*===*\
|第一种并查集的实现:
|这种并查集在进行集合号更改的时候需要遍历整个数组所以要看输入
|对数和集合中元素数目是不是超过时间限制(切记
|分析)
===/

```
#define MAX 1001
int un[MAX],t,n,m;
void Init()
{
        int i;
        for(i=1;i<=n;i++)
                un[i]=i;   //初始化每个节点都是一棵独立的树
}
int Find(int x)   //查找每个元素所属的集合好选择是不是要并如果集合号相同的话,就证明他们是一个集合的(有回路)
{
        return un[x];
}
void Merge(int a,int b)//时间复杂度 O(N)   //所以如果输入的元素对数过多就不能用这个并查集了
{
        int mmax,mmin,i,temp;
        mmax=Find(a);
        mmin=Find(b);
        if(mmax==mmin)   //如果原本是一个集合,那么就不需要并了,这里也可以证明是否形成回路
                return;
        if(mmin>mmax)
        {
                temp=mmin;
                mmin=mmax;
                mmax=temp;
        }
        for(i=1;i<=n;i++)   //这里选择每次将集合号大的并到集合号小的上面
        {
                if(un[i]==mmax)
                        un[i]=mmin;
        }
}
int main()
{
        int a,b,i;
        scanf("%d",&t);
        while(t--)
        {
                scanf("%d%d",&n,&m);
                Init();   //初始化
                while(m--)
                {
                        scanf("%d%d",&a,&b);   输入 m 对顶点
                        Merge(a,b);
                }
                int sum=0;
                for(i=1;i<=n;i++)   //用于统计连通分量的个数
                {
                        if(un[i]==i)
                                sum++;
                }
                printf("%d\n",sum);
        }
        return 0;
```

```
}
/*==================================================*\
|第二种并查集的实现:
|树形结构（带路径压缩）压缩路径有利于查找某个集合中的元素个数
|实际上就是 kruskal 求最小生成树的思想,这里是一种只需要统计哪个
|集合的元素个数最多
\*==================================================*/
#define MAX 10000002   //因为这个题目内存限制是 102400 K ,所以开了这么大的数组
struct node
{
	int parent;
	int w;
}data[MAX];
int MM;   //统计最大集合元素个数
void Init()
{
	int i;
	for(i=1;i<MAX;i++)
	{
		data[i].parent=i;   //这里利用的是 kruskal 的思想,这里父亲结点全初始化为 i
		data[i].w=1;   //初始化高度,权值为 1
	}
}
//路径压缩的思想：每次查找的时候，如果路径较长，则修改信息，以便下次查找的时候速度更快
步骤:第一步，找到根结点
第二步，修改查找路径上的所有节点，将它们都指向根结点
int Find(int x)   //这里是用来查找根的,就是当前这个 x 的根节点
{
	int y=x;
	while(y!=data[y].parent)
		y=data[y].parent;
	while(x != y)   //此处进行路径压缩
      {
		int tmp = data[x].parent;
		data[x].parent = y;
		x = tmp;
      }
      return y;
}
void Merge(int a,int b)
{
	if(data[a].w>data[b].w)
	{
		data[a].w+=data[b].w;
		data[b].parent=a;
		if(data[a].w>MM)
				//这里用来统计哪个集合的元素个数最多
			MM=data[a].w;
	}
	else
	{
		data[b].w+=data[a].w;
		data[a].parent=b;
		if(data[b].w>MM)
			MM=data[b].w;
	}
}
int main()
{
	int n;
	while(scanf("%d",&n)!=EOF)
	{
		MM=1;
		int a,b,x,y;
		Init();
		while(n--)
		{
			scanf("%d%d",&a,&b);
			x=Find(a);
			y=Find(b);
			if(x!=y)
				Merge(x,y);
		}
			printf("%d\n",MM);
	}
	return 0;
}
/*==================================================*\
| 带权值的并查集
| INIT: makeset(n);
| CALL: findset(x); unin(x, y);
\*==================================================*/
struct lset{
    int p[N], rank[N], sz;
    void link(int x, int y) {
        if (x == y) return;
        if (rank[x] > rank[y]) p[y] = x;
        else p[x] = y;
        if (rank[x] == rank[y]) rank[y]++;
    }
    void makeset(int n) {
      sz = n;
      for (int i=0;i<sz;i++) {
          p[i] = i; rank[i] = 0;
      }
    }
    int findset(int x) {
            if (x != p[x]) p[x] = findset(p[x]);
        return p[x];
    }
    void unin(int x, int y) {
          link(findset(x), findset(y));
    }
    void compress() {
        for (int i = 0; i < sz; i++) findset(i);
    }
};
```

4.6 RMQ

```
/*==================================================*\
| 一维 RMQ
|POJ 3264 Balanced Lineup
\*==================================================*/
#include<cstdio>
#include<algorithm>
#include<iostream>
using namespace std;
#define M 50001
int val[M];
int Max[20][M];
int Min[20][M];
int idx[M];
void initRMQ(int n) {
	idx[0] = -1;
	for(int i = 1; i <= n ; i ++) {
		idx[i] = (i&(i-1)) ? idx[i-1] : idx[i-1] + 1;
		Min[0][i] = Max[0][i] = val[i];
	}
	for(int i = 1; i <= idx[n] ; i ++) {
		int limit = n + 1 - (1<<i);
		for(int j = 1; j <= limit ; j ++) {
			Min[i][j] = min(Min[i-1][j] , Min[i-1][j+(1<<i>>1)]);
			Max[i][j]= max(Max[i-1][j] , Max[i-1][j+(1<<i>>1)]);
		}
	}
}
int getval(int a,int b) {
	int t = idx[b-a+1];
	b -= (1<<t) - 1;
	return max(Max[t][a] , Max[t][b]) - min(Min[t][a] , Min[t][b]);
//返回最大值减最小值
}
int main() {
	int n , m;
	scanf("%d%d",&n,&m);
	for(int i = 1; i <= n ; i ++) {		//下标要从 1 开始
		scanf("%d",&val[i]);
	}
```

```
	initRMQ(n);
	while(m --) {
		int a , b ;
		scanf("%d%d",&a,&b);
		printf("%d\n",getval(a,b));
	}
	return 0;
}
```

/*==*\
| 二维 RMQ
|hdu 2888 Check Corners
|复杂度 n*m*log(n)*log(m)
==/

```
#define M 301
int val[M][M];
int Max[9][9][M][M];
int idx[M];
void initRMQ(int n , int m) {
	for(int i = 1; i <= n ; i ++) {
		for(int j = 1 ; j <= m ; j ++) {
			Max[0][0][i][j] = val[i][j];
		}
	}
	for(int i = 0; i <= idx[n] ; i ++) {
		int limit1 = n + 1 - (1<<i);
		for(int j = 0 ; j <= idx[m] ; j ++) {
			if(!i && !j)  continue;
			int limit2 = m + 1 - (1<<j);
			for(int ii = 1; ii <= limit1 ; ii ++) {
				for(int jj = 1; jj <= limit2 ; jj ++) {
	if(i)  Max[i][j][ii][jj] = max( Max[i-1][j][ii+(1<<i>>1)][jj] , Max[i-1][j][ii][jj] );
				else Max[i][j][ii][jj] =max(Max[i][j-1][ii][jj] , Max[i][j-1][ii][jj+(1<<j>>1)] );
				}
			}
		}
	}
}
int query(int a,int b,int c,int d) {
	int n = idx[c-a+1],   m = idx[d-b+1];
	c -= (1<<n) - 1;   d -= (1<<m) - 1;
	return max(max(Max[n][m][a][b],Max[n][m][a][d]),max(Max[n][m][c][b], Max[n][m][c][d]));
}
int main() {
	idx[0] = -1;
	for(int i = 1; i <= 300 ; i ++) {
		idx[i] = (i&(i-1)) ? idx[i-1] : idx[i-1] + 1;//放在外边计算
	}
	int n , m , Q;
	while(~scanf("%d%d",&n,&m)) {
		for(int i = 1; i <= n ; i ++) {
			for(int j = 1; j <= m ; j ++) {
				SS(val[i][j]);
			}
		}
		initRMQ(n,m);
		SS(Q);
		while(Q --) {
			int a,b,c,d;
			scanf("%d%d%d%d",&a,&b,&c,&d);
			if(a > c) swap(a,c);
			if(b > d) swap(b,d);
			int key = query(a,b,c,d);
			printf("%d ",key);
			if(key == val[a][b] || key == val[a][d] || key == val[c][b] || key == val[c][d]) {
				puts("yes");
			} else {
				puts("no");
			}
		}
	}
}
```

/*==*\
| RMQ 问题 ST 算法
| RMQ 问题是求给定区间中的最值问题。当然，最简单的算法是 O(n)
|的，但是对于查询次数很多（设置多大 100 万次），O(n)的算法效率不|
够。可以用线段树将算法优化到 O(logn)（在线段树中保存线段的最值）
|不过，Sparse_Table 算法才是最好的：它可以在 O(nlogn)的预处理以
|后实现 O(1)的查询效率
==/

```
#include<iostream>
#include<cmath>
using namespace std;
#define MAXN 1000000
#define mmin(a, b)    ((a)<=(b)?(a):(b))
#define mmax(a, b)    ((a)>=(b)?(a):(b))
int num[MAXN];
int f1[MAXN][100];
int f2[MAXN][100];
//测试输出所有的 f(i, j)
void dump(int n){
	int i, j;
	for(i = 0; i < n; i++){
		for(j = 0; i + (1<<j) - 1 < n; j++){
			printf("f[%d, %d] = %d\t", i, j, f1[i][j]);
		}
		printf("\n");
	}
	for(i = 0; i < n; i++)
		printf("%d ", num[i]);
	printf("\n");
	for(i = 0; i < n; i++){
		for(j = 0; i + (1<<j) - 1 < n; j++){
			printf("f[%d, %d] = %d\t", i, j, f2[i][j]);
		}
		printf("\n");
	}
	for(i = 0; i < n; i++)
		printf("%d ", num[i]);
	printf("\n");
}

//sparse table 算法
void st(int n){
	int i, j, k, m;
	k = (int) (log((double)n) / log(2.0));
	for(i = 0; i < n; i++) {
		f1[i][0] = num[i]; //递推的初值
		f2[i][0] = num[i];
	}
	for(j = 1; j <= k; j++)
	{ //自底向上递推
		for(i = 0; i + (1 << j) - 1 < n; i++){
			m = i + (1 << (j - 1)); //求出中间的那个值
			f1[i][j] = mmax(f1[i][j-1], f1[m][j-1]);
			f2[i][j] = mmin(f2[i][j-1], f2[m][j-1]);
		}
	}
}
//查询 i 和 j 之间的最值,注意 i 是从 0 开始的
void rmq(int i, int j) {
	int k = (int)(log(double(j-i+1)) / log(2.0)), t1, t2; //用对 2 去对数的方法求出 k
	t1 = mmax(f1[i][k], f1[j - (1<<k) + 1][k]);
	t2 = mmin(f2[i][k], f2[j - (1<<k) + 1][k]);
	printf("%d\n",t1 - t2);
}
int main(){
	int i,N,Q,A,B;
	scanf("%d %d", &N, &Q);
	for (i = 0; i < N; ++i){
		scanf("%d", num+i);
	}
	st(N); //初始化
	//dump(N); //测试输出所有 f(i, j)
	while(Q--){
		scanf("%d %d",&A,&B);
		rmq(A-1, B-1);
```

```
    }
    return 0;
}
/*==================================================*\
|RMQ 求区间最值
|这里给出最大值，若要求最小，把所有 max 改成 min 即可
\*==================================================*/

#include <iostream>
using namespace std;
struct
{
    int left , right;
    int max;
}tree[600000];
int init[200001];
int max (int a , int b)
{
    return a > b ? a : b;
}
void create (int l, int r, int node)
{
    tree[ node ].left = l;
    tree[ node ].right = r;
    if (l == r)
    {
        tree[ node ].max = init[ l ];
        return;
    }
    int mid = (l + r) / 2;
    create (l , mid , node * 2);
    create (mid + 1 , r , node * 2 + 1);
    tree[ node ].max = max(tree[node * 2].max , tree[node * 2 +
1].max);
}
void update (int num, int val, int node)
{
    if (tree[ node ].left == tree[ node ].right)
    {
        tree[ node ].max = val;
        return;
    }
    if (tree[node * 2].right >= num)
        update (num , val , node * 2);
    else
        update (num , val , node * 2 + 1);
    tree[ node ].max = max(tree[node * 2].max , tree[node * 2 +
1].max);
}
int query (int l , int r , int node)
{
    if (tree[ node ].left == l && tree[ node ].right == r)
        return tree[ node ].max;
    else
        if(tree[node * 2].right >= r)
            return query (l , r , node * 2);
        else
            if(tree[node * 2 + 1].left <= l)
                return query (l,    r , node * 2 + 1);
            else
                return max(query (l , tree[node * 2].right , node *
2) , query(tree[node * 2 + 1].left , r , node * 2 + 1));
}
int main()
{
    char c;
    int i , len , q , a , b;
    while (scanf("%d %d", &len, &q) != EOF)
    {
        for (i = 1;i <= len;++ i)
            scanf ("%d", &init[ i ]);
        create (1 , len , 1);
        for (i = 0; i < q; ++i)
        {
            scanf("%*c%c %d %d", &c, &a, &b);
            if (c == 'Q')
                printf ("%d\n", query(a , b , 1));
            else
                update (a , b , 1);
        }
    }
    return 0;
}
/*==================================================*\
| RMQ  离线算法  O(N*logN)+O(1)
| INIT: val[]置为待查询数组; initrmq(n);
\*==================================================*/
int st[20][N], ln[N], val[N];
void initrmq(int n){
    int i, j, k, sk;
    ln[0] = ln[1] = 0;
    for (i = 0; i < n; i++) st[0][i] = val[i];
    for (i = 1, k = 2; k < n; i++, k <<= 1) {
        for (j = 0, sk = (k >> 1); j < n; ++j, ++sk) {
            st[i][j] = st[i-1][j];
            if (sk < n && st[i][j] > st[i-1][sk])
            st[i][j] = st[i-1][sk];
        }
        for (j=(k>>1)+1; j <= k; ++j) ln[j] = ln[k>>1] + 1;
    }
        for (j=(k>>1)+1; j <= k; ++j) ln[j] = ln[k>>1] + 1;
}
int query(int x, int y) // min of { val[x] ... val[y] }{
     int bl = ln[y - x + 1];
    return min(st[bl][x], st[bl][y-(1<<bl)+1]);
}
/*==================================================*\
| RMQ(Range Minimum/Maximum Query)-st  算法(O(nlogn + Q))
| Readln()  初始化数组 a[0...n-1];
| InitRMQ()利用 st 算法( O(nlogn) )进行预处理;
| Query()根据输入的下标查询最值(O(Q))
| Hint:  下标范围:0...n-1,如果为 1...n 须稍做修改;  此处实现的的是求
|  最大值，如果求最小值需要把 max->min
| Call: Readln(n); InitRMQ(n); Query(Q);
\*==================================================*/
const int N = 200001;
int a[N], d[20];
int st[N][20];
int main(void){
int n, Q;
  while( scanf("%d%d", &n, &Q) != EOF )
      Readln(n); InitRMQ(n); Query(Q);
 return 0;
}
void Readln(const int &n){
     int i;
     for( i=0; i < n; ++i ) scanf("%d", &a[i]);
}
inline int max(const int &arg1, const int &arg2){
    return arg1 > arg2 ? arg1 : arg2;
}
void InitRMQ(const int &n){
      int i, j;
      for( d[0]=1, i=1; i < 21; ++i ) d[i] = 2*d[i-1];
      for( i=0; i < n; ++i ) st[i][0] = a[i];
         int k = int( log(double(n))/log(2) ) + 1;
      for( j=1; j < k; ++j )
         for( i=0; i < n; ++i ){
           if( i+d[j-1]-1 < n ){
              st[i][j] = max(st[i][j-1],
              st[i+d[j-1]][j-1]);
           }
           else break;              // st[i][j] = st[i][j-1];
         }
}
void Query(const int &Q){
       int i;
       for( i=0; i < Q; ++i ){
       int x, y, k;                // x, y 均为下标:0...n-1
       scanf("%d%d", &x, &y);
       k = int( log(double(y-x+1))/log(2.0) );
       printf("%d\n", max(st[x][k], st[y-d[k]+1][k]));
```

```
        }
}
/*==================================================*\
| RMQ 离线算法 O(N*logN)+O(1)求解 LCA
| INIT: val[]置为待查询数组; initrmq(n);
\*==================================================*/
const int N = 10001; // 1<<20;
int pnt[N], next[N], head[N];        // 邻接表
int e;                               // 边数
bool visited[N];              // 初始为 0，从根遍历
int id;
int dep[2*N+1], E[2*N+1], R[N];
    // dep:dfs 遍历节点深度, E:dfs 序列, R:第一次被遍历的下标
void DFS(int u, int d);
int d[20], st[2*N+1][20];
void Answer(void){
        int i, Q;
        scanf("%d", &Q);
        for( i=0; i < Q; ++i ){
            int x, y;
            scanf("%d%d", &x, &y);      // 查询 x,y 的 LCA
            x = R[x]; y = R[y];
        if( x > y )
            int tmp = x; x = y; y = tmp;
      printf("%d\n", E[ Query(x, y) ]);
        }
}
void DFS(int u, int d){
        visited[u] = 1;
        R[u] = id; E[id] = u; dep[id++] = d;
        for( int i=head[u]; i != -1; i=next[i] )
        if( visited[ pnt[i] ] == 0 ){
            DFS(pnt[i], d+1);
            E[id] = u; dep[id++] = d;
        }
}
void InitRMQ(const int &id){
        int i, j;
        for( d[0]=1, i=1; i < 20; ++i ) d[i] = 2*d[i-1];
        for( i=0; i < id; ++i ) st[i][0] = i;
        int k = int( log(double(n))/log(2.0) ) + 1;
        for( j=1; j < k; ++j )
      for( i=0; i < id; ++i ){
            if( i+d[j-1]-1 < id ){
st[i][j] = dep[ st[i][j-1] ] >dep[ st[i+d[j-1]][j-1] ] ? st[i+d[j-1]][j-1] :
st[i][j-1];
            }
            else break;        // st[i][j] = st[i][j-1];
        }
}
int Query(int x, int y){
        int k;                  // x, y 均为下标:0...n-1
        k = int( log(double(y-x+1))/log(2.0) );
        return dep[ st[x][k] ] > dep[ st[y-d[k]+1][k] ] ?
    st[y-d[k]+1][k] : st[x][k];
}
/*==================================================*\
| LCA 离线算法 O(E)+O(1)
| INIT: id[]置为-1; g[]置为邻接矩阵;
| CALL: for (i=0; i<n; ++i) if (-1==st[i]) dfs(i, n);
| LCA 转化为 RMQ 的方法: 对树进行 DFS 遍历，每当进入或回溯到
| 某个结点 i 时，将 i 的深度存入数组 e[]最后一位. 同时记录结点 i 在
| 数组中第一次出现的位置，记做 r[i]. 结点 e[i]的深度记做 d[i].
| LCA(T,u,v), 等价于求 E[RMQ(d,r[u],r[v])], (r[u]<r[v]).
\*==================================================*/
int id[N], lcs[N][N], g[N][N];
int get(int i){
        if (id[i] == i) return i;
return id[i] = get(id[i]);
}
void unin(int i, int j){
        id[get(i)] = get(j);
}
void dfs(int rt, int n) {           // 使用邻接表可优化为 O(E)+O(1)
    int i;
    id[rt] = rt;
        for (i = 0; i < n; ++i) if (g[rt][i] && -1 == id[i])
            dfs(i, n); unin(i, rt);
        for (i = 0; i < n; ++i) if (-1 != id[i])
            lcs[rt][i] = lcs[i][rt] = get(i);
}
/*==================================================*\
| Tarjan 离线算法求 LCA
\*==================================================*/
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
#define MAXV 100001
#define INF 1000000000
typedef struct {
    int v;
     int val;
     int next;
}Edge;
Edge e[MAXV*2], q[MAXV*2];
int eid, qid;
int pe[MAXV], pq[MAXV], ans[MAXV*2];
bool vist[MAXV];
int fa[MAXV];
void init() {
    memset(pe, -1, sizeof(pe));
    memset(pq, -1, sizeof(pq));
    memset(vist, false, sizeof(vist));
    memset(fa, -1, sizeof(fa));
    eid = qid = 0;
}
void add(int u,int v) {
    e[ eid ].next = pe[ u ];
     e[ eid ].v = v;
     pe[ u ] = eid ++;
}
void addQ(int u,int v,int val) {
    q[ qid ].next = pq[ u ];
     q[ qid ].v = v;
     q[ qid ].val = val;
     pq[ u ] = qid ++;
}
int find(int x) {
    if(fa[ x ] == -1) return x;
     return fa[ x ] = find(fa[ x ]);
}
void tarjan(int u,int curval) {
    int i, v, val;
    vist[ u ] = true;
    for(i = pq[ u ]; i != -1; i = q[ i ].next) {
        v = q[ i ].v;
          val = q[ i ].val;
          if(vist[ v ]) ans[ val ] = find(v);
    }
    for(i = pe[ u ]; i != -1; i = e[ i ].next) {
          v = e[ i ].v;
          val = e[ i ].val;
        if(!vist[ v ]) {
            tarjan(v, curval + val);
            fa[ v ] = u;
        }
    }
}
int main() {
     int n, i, k;
     int u, v;
     while(~scanf("%d", &n)) {
        init();
        for(i = 1; i <= n; ++ i) {
            scanf("%d", &v);
            add(i, v);
            add(v, i);
        }
          scanf("%d", &k);
        for(i = 1; i <= k; ++ i) {
            scanf("%d %d", &u, &v);
```

```
            //i 为编号
            addQ(u, v, i);
            addQ(v, u, i);
        }
        tarjan(1, 0);
        for(i = 1; i <= k; ++ i) {
            printf("%d\n", ans[ i ]);
        }
    }
    return 0;
}
```

4.7 AC 自动机

```
/*==================================================*\
|AC 自动机
|给 n 个字符串，和一个目标串，问最多能匹配上几个
\*==================================================*/

#include <stdio.h>
#include <string.h>
#define MAXKLEN 55
#define MAXDLEN 1000005
#define MAXQSIZE 500005
typedef struct TNode{
    TNode(){
        count=0;
        fail=NULL;
        memset(next,NULL,sizeof(next));
    }
    int count;
    TNode *fail;
    TNode *next[26];
}*Trie;
TNode *que[MAXQSIZE];
void insert(Trie p,char *key){
    int i;
    while(*key){
        i = *key - 'a';
        if(!p->next[i]) p->next[i] = new TNode;
        p = p->next[i];
        key++;
    }
    p->count++;
}
void build_ac_automation(Trie root){
    root->fail = NULL;
    int front=-1,rear=-1;
    que[++rear] = root;
    TNode *out;
    while(front < rear){
        out = que[++front];
        for(int i=0;i<26;i++){
            if(out->next[i]){
                que[++rear] = out->next[i];
                if(out == root) out->next[i]->fail = root;
                else{
                    TNode *tmp = out->fail;
                    while(tmp){
                        if(tmp->next[i]){
                            out->next[i]->fail = tmp->next[i];
                            break;
                        }
                        tmp = tmp->fail;
                    }
                    if(!tmp) out->next[i]->fail = root;
                }
            }
        }
    }
}
int search(Trie root,char *pat){
    int i,cnt=0;
    TNode *p = root;
    while(*pat){
        i = *pat - 'a';
        while(!p->next[i] && p != root) p=p->fail;
        p = p->next[i];
        if(!p) p = root;
        TNode *tmp = p;
        while(tmp->count!=-1 && tmp != root){
            cnt+=tmp->count;
//          printf("tmp->count=%d pat=%s\n",tmp->count,pat);
            tmp->count = -1;
            tmp = tmp->fail;
        }
        pat++;
//      printf("pat=%s cnt=%d\n",pat,cnt);
    }
    return cnt;
}
inline void destroy(Trie &root){
    if(!root) return;
    for(int i=0;i<26;i++) destroy(root->next[i]);
    delete root;
}
char description[MAXDLEN];
int main(){
    #ifndef ONLINE_JUDGE
    freopen("tdata.txt","r",stdin);
    #endif
    int T,n;
    char keys[MAXKLEN];
    scanf("%d",&T);
    while(T--){
        Trie tree = new TNode;
        scanf("%d",&n);
        while(n--){
            scanf("%s",keys);
            insert(tree,keys);
        }
        build_ac_automation(tree);
//      printf("build count\n");
        scanf("%s",description);
        printf("%d\n",search(tree,description));
//      destroy(tree);
    }
    return 0;
}
```

4.8 后缀数组

```
/*==================================================*\
|倍增算法
|时间复杂度 O(nlogn)   25 行
\*==================================================*/
#define maxn 1000001
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int cmp(int *r,int a,int b,int l)
{return r[a]==r[b]&&r[a+l]==r[b+l];}
void da(int *r,int *sa,int n,int m)
{
     int i,j,p,*x=wa,*y=wb,*t;
     for(i=0;i<m;i++) ws[i]=0;
     for(i=0;i<n;i++) ws[x[i]=r[i]]++;
     for(i=1;i<m;i++) ws[i]+=ws[i-1];
     for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
     for(j=1,p=1;p<n;j*=2,m=p)
     {
       for(p=0,i=n-j;i<n;i++) y[p++]=i;
       for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
       for(i=0;i<n;i++) wv[i]=x[y[i]];
       for(i=0;i<m;i++) ws[i]=0;
       for(i=0;i<n;i++) ws[wv[i]]++;
       for(i=1;i<m;i++) ws[i]+=ws[i-1];
       for(i=n-1;i>=0;i--) sa[--ws[wv[i]]]=y[i];
       for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
       x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
     }
     return;
}
int rank[maxn],height[maxn];
```

```
void calheight(int *r,int *sa,int n)
{
        int i,j,k=0;
        for(i=1;i<=n;i++) rank[sa[i]]=i;
        for(i=0;i<n;height[rank[i++]]=k)
        for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);
        return;
}
int RMQ[maxn];
int mm[maxn];
int best[20][maxn];
void initRMQ(int n)
{
        int i,j,a,b;
        for(mm[0]=-1,i=1;i<=n;i++)
        mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
        for(i=1;i<=n;i++) best[0][i]=i;
        for(i=1;i<=mm[n];i++)
        for(j=1;j<=n+1-(1<<i);j++)
        {
           a=best[i-1][j];
           b=best[i-1][j+(1<<(i-1))];
           if(RMQ[a]<RMQ[b]) best[i][j]=a;
           else best[i][j]=b;
        }
        return;
}
int askRMQ(int a,int b)
{
      int t;
      t=mm[b-a+1];b-=(1<<t)-1;
      a=best[t][a];b=best[t][b];
      return RMQ[a]<RMQ[b]?a:b;
}
int lcp(int a,int b)
{
      int t;
      a=rank[a];b=rank[b];
      if(a>b) {t=a;a=b;b=t;}
      return(height[askRMQ(a+1,b)]);
}
```

```
/*==================================================*\
| DC3 算法
|时间复杂度 O(n)        40 行
\*==================================================*/
#define maxn 1000003
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int c0(int *r,int a,int b)
{return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];}
int c12(int k,int *r,int a,int b)
{if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
  else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];}
void sort(int *r,int *a,int *b,int n,int m)
{
        int i;
        for(i=0;i<n;i++) wv[i]=r[a[i]];
        for(i=0;i<m;i++) ws[i]=0;
        for(i=0;i<n;i++) ws[wv[i]]++;
        for(i=1;i<m;i++) ws[i]+=ws[i-1];
        for(i=n-1;i>=0;i--) b[--ws[wv[i]]]=a[i];
        return;
}
void dc3(int *r,int *sa,int n,int m)
{
        int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
        r[n]=r[n+1]=0;
        for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;
        sort(r+2,wa,wb,tbc,m);
        sort(r+1,wb,wa,tbc,m);
        sort(r,wa,wb,tbc,m);
        for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
        rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
        if(p<tbc) dc3(rn,san,tbc,p);
        else for(i=0;i<tbc;i++) san[rn[i]]=i;
        for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
        if(n%3==1) wb[ta++]=n-1;
        sort(r,wb,wa,ta,m);
        for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
        for(i=0,j=0,p=0;i<ta && j<tbc;p++)
        sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
        for(;i<ta;p++) sa[p]=wa[i++];
        for(;j<tbc;p++) sa[p]=wb[j++];
        return;
}
int rank[maxn],height[maxn];
void calheight(int *r,int *sa,int n)
{
        int i,j,k=0;
        for(i=1;i<=n;i++) rank[sa[i]]=i;
        for(i=0;i<n;height[rank[i++]]=k)
        for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);
        return;
}
int RMQ[maxn];
int mm[maxn];
int best[20][maxn];
void initRMQ(int n)
{
        int i,j,a,b;
        for(mm[0]=-1,i=1;i<=n;i++)
        mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
        for(i=1;i<=n;i++) best[0][i]=i;
        for(i=1;i<=mm[n];i++)
        for(j=1;j<=n+1-(1<<i);j++)
        {
           a=best[i-1][j];
           b=best[i-1][j+(1<<(i-1))];
           if(RMQ[a]<RMQ[b]) best[i][j]=a;
           else best[i][j]=b;
        }
        return;
}
int askRMQ(int a,int b)
{
      int t;
      t=mm[b-a+1];b-=(1<<t)-1;
      a=best[t][a];b=best[t][b];
      return RMQ[a]<RMQ[b]?a:b;
}
int lcp(int a,int b)
{
      int t;
      a=rank[a];b=rank[b];
      if(a>b) {t=a;a=b;b=t;}
      return(height[askRMQ(a+1,b)]);
}
```

4.9 查找与排序

```
/*==================================================*\
| 快速排序
\*==================================================*/
void ksort(int l, int h, int a[]){
      if (h < l + 2) return;
      int e = h, p = l;
      while (l < h) {
         while (++l < e && a[l] <= a[p]);
         while (--h > p && a[h] >= a[p]);
        if (l < h) swap(a[l], a[h]);
      }
   swap(a[h], a[p]);
  ksort(p, h, a); ksort(l, e, a);
}
```

```
/*==================================================*\
| 二分查找
|在[l, r)范围内查找值 v，返回下标
|假设 a 数组已经按从小到大排序
|失败返回-1
\*==================================================*/
```

```
int bs(int a[], int l, int h, int v){
        int m;
        while ( l < h ){
            m = ( l + h ) >> 1;
            if (a[m] == v) return m;
            if (a[m] < v) l=m+1;
            else h=m;
        }
return -1;
}
/*==================================================*\
| 二分查找（大于等于 v 的第一个值）
|传入参数必须 l <= h
|返回值 l 总是合理的
\*==================================================*/
int bs(int a[], int l, int h, int v{
          int m;
          while ( l < h ){
                m = ( l + h ) >> 1;
                if (a[m] < v) l=m+1;
                else h=m;
          }
 return l;
}
```

4.10 堆

```
/*==================================================*\
| 堆栈
\*==================================================*/
const int MAXSIZE = 10000;
int a[MAXSIZE]，heapsize;
inline void swap(int i, int j){
      int temp = a[i]; a[i] = a[j]; a[j] = temp;
}
inline int Parent(int i){ return i >> 1; }
inline int Left(int i){ return 1 << i; }
inline int Right(int i){ return (1 << i) + 1; }
// 保持堆的性质
void MaxHeapify(int i){
      int l = Left(i), r = Right(i), largest;
      if( l <= heapsize && a[l] > a[i] ) largest = l;
      else largest = i;
      if( r <= heapsize && a[r] > a[largest] ) largest = r;
            if( largest != i )
               swap(i, largest); MaxHeapify(largest);
}
void BuildMaxHeap(int *arr, int n){
         heapsize = n;
         for( int i=heapsize/2; i > 0; --i ) MaxHeapify(i);
}
void HeapSort(int *arr, int n){
      BuildMaxHeap(arr, n);
      for( int i=n; i > 1; --i ){
          swap(1, i); heapsize--;
       MaxHeapify(1);
       }
}
```

目　　录：

Chapter 5

Simulate Problem

5.1 日期

```
/*==================================================*\
|日期有关的函数
\*==================================================*/
#include <iostream>
#include <cstdio>
#include <cstring>
const int MAXN = 37198;
#include <string>
using namespace std;
int val[ 2 ][ 13 ] = {
	{31, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
	{31, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};
int yearday[ 2 ] = {365, 366};
const int excnt = 730793;
const int exy = 2001;
const int exm = 11;
const int exd = 4;
const int lowcnt = 693596;
string dayval[ 7 ] = {"Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday"};
int leapYear(int y) {
	return y%4==0 && y%100 || y%400==0;
}
void toDate(int &y, int &m, int &d, int cnt) {
	int v[ 1 ] = {146097};
	int g[ 1 ] = {400};
	int ty, tm, td;
	ty = tm = td = 1;
	cnt --;
	int tmp = cnt / v[ 0 ], le;
	cnt = cnt % v[ 0 ];
	ty += tmp * g[ 0 ];
	while(true) {
		le = leapYear(ty);
		if(cnt < yearday[ le ]) break;
		ty ++;
		cnt -= yearday[ le ];
	}
	while(cnt >= val[ le ][ tm ]) {
		cnt -= val[ le ][ tm ];
		tm ++;
	}
	td += cnt;
	y = ty, m = tm, d = td;
}
int toDays(int &y, int &m, int &d) {
	int yf = y - 1, le;
	int cnt = yf * 365 + d;
	int v[ 3 ] = {97, 24, 1};
	int g[ 3 ] = {400, 100, 4};
	int ty = y, tm = m, td = d;
	le = leapYear(ty);
	for(int i = 1; i < tm; ++ i) {
		cnt += val[ le ][ i ];
	}
	for(int k = 0; k < 3; ++ k) {
		int tmpy = yf / g[ k ];
		yf = yf % g[ k ];
		cnt += tmpy * v[ k ];
	}
	return cnt;
}
/*==================================================*\
|2 日期之隔天数
\*==================================================*/
int getSeparateDays(int &y, int &m, int &d, int &yy, int &mm, int
&dd) {
	int ans = toDays(y, m, d) - toDays(yy, mm, dd);
	return ans > 0 ? ans : -ans;
}
bool hasLatterDate(int &y, int &m, int &d, int n) {
	if(excnt <= n + toDays(y, m, d)) return false;
	return true;
}
/*==================================================*\
|第 n 天后的日期
\*==================================================*/
void latterDateValueOf(int &y, int &m, int &d, int n) {
	int tc = toDays(y, m, d) + n;
	toDate(y, m, d, tc);
}
/*==================================================*\
|后一天的日期
\*==================================================*/
void latterDate(int &y, int &m, int &d) {
	d ++;
	int le = leapYear(y);
	if(d > val[ le ][ m ]) {
		d -= val[ le ][ m ];
		m ++;
	}
	if(m > 12) {
		m -= 12;
		y ++;
	}
}
/*==================================================*\
|第 n 天前的日期
\*==================================================*/
void formerDateValueOf(int &y, int &m, int &d, int n) {
	int tc = toDays(y, m, d) - n;
	toDate(y, m, d, tc);
}
/*==================================================*\
|是否存在第前 n 的日期，日期是从 1 年 1 月 1 日开始
\*==================================================*/
bool hasFormerDate(int &y, int &m, int &d, int n) {
	if(toDays(y, m, d) <= n) return false;
	return true;
}
/*==================================================*\
|前一天的日期
\*==================================================*/
void formerDate(int &y, int &m, int &d, int cnt) {
	d --;
	int le = leapYear(y);
	if(d <= 0) {
		m --;
		d = val[ le ][ m ];
	}
	if(m < 1) {
		m = 12; y --;
	}
}
bool isCorrectDate(int &y, int &m, int &d) {
	int le = leapYear(y);
	if(m > 12 || m < 1) return false;
	if(d > val[ le ][ m ]) return false;
	return true;
}
int main() {
	int y, m, d, n, cnt, g;
	while(cin >> n) {
		if(n < 0) break;
		y = 2000, m = 1, d = 1; g = 6;
```

```
            cnt = toDays(y, m, d);
            latterDateValueOf(y, m, d, n);
            g = (g + n) % 7;
            printf("%04d-%02d-%02d ", y, m, d);
            cout << dayval[ g ] << endl;
        }
        return 0;
}
```

5.2 K-th largest

```
/*==================================================*\
| K-th largest
\*==================================================*/
#include <iostream>
#include <algorithm>
#define MAXN 10003
#include <cstdio>
using namespace std;
int sa[MAXN], sb[MAXN];
int n, m;
int main() {
      int i;
      int cas; scanf("%d", &cas);
      while(cas --) {
            scanf("%d %d", &n, &m);
            for (i = 0; i < n; ++ i) scanf("%d", sa + i);
            for (i = 0; i < n; ++ i) scanf("%d", sb + i);
            int curr = 0, result = 0;
            sort(sa, sa + n);
            sort(sb, sb + n);
            int l=sa[ 0 ]*sb[ 0 ], r=sa[n - 1]*sb[n - 1];
               //原来是求第 m 小的，转一下变成 m 大.
            m = n * n - m + 1;
            while(l <= r) {
                  int mid = (l + r) >> 1;
                  int lt = 0;
                  for (i = 0; i < n; ++ i) {
                        int a = 0, b = n - 1;
                        while (a <= b) {
                              int c = (a + b) >> 1;
                              int t = sa[ i ] * sb[ c ];
                              if (t > mid) b = c - 1;
                              else a = c + 1;
                        }
                        lt += a;
                        if (lt > m) break;
                  }
                  if (lt >= m) r =   mid - 1;
                  else l = mid + 1;
            }
            printf("%d\n", l);
      }
      return 0;
}
/*==================================================*\
| nth_element
|作用：求取满足条件的第 n 个元素，例如 10 个学生，求第五名的学生
|就可以用这个，就的
|结果就在数组下标为 4 的位置，注意，在调用的时候，输入参数(num,
|num+4, num+n);
\*==================================================*/
#include <vector>
#include <algorithm>
#include <functional>          // For greater<int>( )
#include <iostream>
using namespace std;
// Return whether first element is greater than the second
bool UDgreater (int elem1, int elem2) {
        return elem1 > elem2;
}
int main() {
        vector <int> v1;int n,data,t;
        vector <int>::iterator Iter1;
      cin >> t;
        while (t -- && cin >> n){
v1.clear ();
                while (n --){
cin >> data;       v1.push_back(data);
                }
            //To sort in riseing order
                nth_element(v1.begin( ), v1.begin( ) + 2, v1.end( ));
            cout << v1[ 1 ] << endl;
                // To sort in descending order, specify binary
predicate
                nth_element( v1.begin(), v1.begin( ) + 2, v1.end( ),
greater<int>( ));
                //convert the order of sequence
                random_shuffle(v1.begin( ), v1.end( ) );
        // A user-defined (UD) binary predicate can also be used
                nth_element( v1.begin(), v1.begin() + 2, v1.end(),
UDgreater );
        }
        return 0;
}
/*==================================================*\
| Permutation 函数
\*==================================================*/
#include<iostream>
#include<algorithm>
#include<cstdio>
using namespace std;
int main(){
     char a[1000];
     int n,i,count;
     while(cin>>n){
            scanf("%s",a);
            sort (a , a + n);
     count=0;
     do {
          count++;
          cout << a;
            cout<<endl;
     }while(next_permutation(a, a + n));
     cout<<count<<endl;
     }
     return 0;
}
```

5.3 工作调度

```
/*==================================================*\
| 2 台机器工作调度
|2 台机器, n 件任务，必须先在 S1 上做，再在 S2 上做．任务之间先做后
|做任意．求最早的完工时间．这是一个经典问题: 2 台机器的情况下有多
|项式算法(Johnson 算法), 3 台或以上的机器是 NP-hard 的. Johnson
|算法:
|(1) 把作业按工序加工时间分成两个子集，
|第一个集合中在 S1 上做的时间比在 S2 上少，
|其它的作业放到第二个集合.
|先完成第一个集合里面的作业，再完成第二个集合里的作业.
|(2) 对于第一个集合，其中的作业顺序是按在 S1 上的时间的不减排列;
|对于第二个集合，其中的作业顺序是按在  S2  上的时间的不增排列.
|Johnson 算法的时间取决于对作业集合的排序 ，因此 ，在最怀情况
|下算法的时间复杂度为 O(nlogn), 所需的空间复杂度为 O(n).
\*==================================================*/
Struct Triplet{              // 三元组结构
     Int   Operator<(Triplet b)const {return t <b.t;}
     Int jobNo,t,ab;       //jobNo 为作业 ，体委处理时间 ,ab 为设备号
};
Void FlowShop(int n,int *a,int*b,int*c) {
   Triplet d[mSize]={{0,0,0}};
     For(int i=0;i<n;i++)           // 算法步骤 (1), 生成三元组表 d
        If (a[i]<b[i]){
     d[i].jobNo=i;d[i].ab=0;d[i].t=a[i];
}
Else {
   d[i].jobNo=i;d[i].ab=1;d[i].t=b[i];
}
Sort(d,n);              // 算法步骤 (2), 任意排序算法
   Int    left=0,right=n-1;
For(i=0;i<n;i++)    // 算法步骤 (3), 生成最优解
```

```
    If (d[i].ab==0)c[left++]=d[i].jobNo;
    Else c[right--]=d[i].jobNo;
}
```

5.4 游戏

/*===*\
| 棋盘分割
| 将一个 8 * 8 的棋盘进行如下分割：将原棋盘割下一块矩形棋盘并使剩
|下部分也是矩形，再将剩下的部分继续如此分割，这样割了(n-1)次后，
|连同最后剩下的矩形棋盘共有 n 块矩形棋盘。(每次切割都只能沿着棋
|盘格子的边进行) 原棋盘上每一格有一个分值，一块矩形棋盘的总分为
|其所含各格分值之和。现在需要把棋盘按上述规则分割成 n 块矩形棋盘，
|并使各矩形棋盘总分的均方差最小。 均方差…，其中平均值…，xi 为|
第 i 块矩形棋盘的|
| 总分。请编程对给出的棋盘及 n，求出 O'的最小值。
| POJ 1191 棋盘分割
===/

```
#define min(a, b) ( (a) < (b) ? (a) : (b) )
const int oo = 10000000;
int map[8][8];
double C[16][8][8][8][8];//c[k][si][ei][sj][ej]:
    //对矩阵 map[si...sj][ei...ej]分割成 k 个矩形(切割 k-1 刀)的结果
double ans;          // 平均值
int n;               // 分成 n 块矩形棋盘
void input(void);
void reset(void);
double caluate(int i1, int j1, int i2, int j2);
void dp(int m, int si, int sj, int ei, int ej);
int main(void){
    int m, i, j, k, l;
    while( scanf("%d", &n) != EOF ){
        input(); reset();
        for( m=1; m <= n; m++ )
            for( i=0; i < 8; i++ )
                for( j=0; j < 8; j++ )
                    for( k=0; k < 8; k++ )
                        for( l=0; l < 8; l++ ){
                            if( (k-i+1)*(l-j+1) < m )
                                C[m][i][j][k][l] = oo;
                            else{
                                if( m == 1 ){
                                    C[m][i][j][k][l] =
                                    pow( (caluate(i,j,k,l)-ans), 2 );
                                }
                                else
                                    dp(m, i, j, k, l);
                            }
                        }
        printf("%.3lf\n", sqrt(C[n][0][0][7][7]/n));
    }
    return 0;
}
void input(void){
    int i, j;
    double sum = 0;
    for( i=0; i < 8; i++ )
        for( j=0; j < 8; j++ ){
            scanf("%d", &map[i][j]);
            sum += map[i][j];
        }
    ans = sum/double(n);       // 平均值
}
void reset(void){
    int i, j, k, l, m;
    for( m=0; m <= n; m++ )
        for( i=0; i < 8; i++ )
            for( j=0; j < 8; j++ )
                for( k=0; k < 8; k++ )
                    for( l=0; l < 8; l++ )
                        C[m][i][j][k][l] = 0;
}
double caluate(int i1, int j1, int i2, int j2){
double sum=0;
    int i, j;
    for( i=i1; i <= i2; i++ )
        for( j=j1; j <= j2; j++ ) sum += map[i][j];
    return sum;
}
void dp(int m, int si, int sj, int ei, int ej){
    int i, j;
    double mins = oo;
    for( j=sj; j < ej; j++ ) {            // 竖刀
        mins = min(mins,
        C[1][si][sj][ei][j]+C[m-1][si][j+1][ei][ej]);
        mins = min(mins,
        C[m-1][si][sj][ei][j]+C[1][si][j+1][ei][ej]);
    }
    for( i=si; i < ei; i++ ) {            // 横刀
        mins = min(mins,
        C[1][si][sj][i][ej]+C[m-1][i+1][sj][ei][ej]);
        mins = min(mins,
        C[m-1][si][sj][i][ej]+C[1][i+1][sj][ei][ej]);
    }
    C[m][si][sj][ei][ej] = mins;
}
```

/*===*\
| 汉诺塔
| 1,2,...,n 表示 n 个盘子．数字大盘子就大．n 个盘子放在第 1 根柱子|
上．大盘不能放在小盘上．在第 1 根柱子上的盘子是 a[1],a[2],...,a[n].
|a[1]=n,a[2]=n-1,...,a[n]=1.即 a[1]是最下面的盘子．把 n 个盘子
|移动到第 3 根柱子．每次只能移动 1 个盘子，且大盘不能放在小盘上．问
|第 m 次移动的是哪一个盘子，从哪根柱子移到哪根柱子.例如：n=3,m=2.
|回答是 ：2 1 2，即移动的是 2 号盘，从第 1 根柱子移动到第 2 根柱|
子 。
| HDU 2511 汉诺塔 X
===/

一号柱有 n 个盘子,叫做源柱.移往 3 号柱,叫做目的柱.2 号柱叫做中间柱.
全部移往 3 号柱要 f(n) =（2^n）- 1 次.
最大盘 n 号盘在整个移动过程中只移动一次,n-1 号移动 2 次,i 号盘移动 2^(n-i)次.
1 号盘移动次数最多,每 2 次移动一次.
第 2k+1 次移动的是 1 号盘,且是第 k+1 次移动 1 号盘.
第 4k+2 次移动的是 2 号盘,且是第 k+1 次移动 2 号盘.
第(2^s)k+2^(s-1)移动的是 s 号盘,这时 s 号盘已被移动了 k+1 次.
每 2^s 次就有一次是移动 s 号盘.
第一次移动 s 号盘是在第 2^(s-1)次.
第二次移动 s 号盘是在第 2^s+2^(s-1)次.
......
第 k+1 次移动 s 号盘是在第 k*2^s+2^(s-1)次.
1--2--3--1 叫做顺时针方向,1--3--2--1 叫做逆时针方向.
最大盘 n 号盘只移动一次:1--3,它是逆时针移动.
n-1 移动 2 次:1--2--3,是顺时针移动.
如果 n 和 k 奇偶性相同,则 k 号盘按逆时针移动,否则顺时针.

```
int main(void){
    int i, k;
    scanf("%d", &k);
    for( i=0; i < k; i++ ){
        int n, l;
        int64 m, j;
        __int64 s, t;
        scanf("%d%I64d", &n, &m);
            s = 1; t = 2;
        for( l=1; l <= n; l++ ){
            if( m%t == s ) break;
            s = t; t *= 2;
        }
        printf("%d ", l);
        j = m/t;
    if( n%2 == l%2 ){                // 逆时针
    if( (j+1)%3 == 0 ) printf("2 1\n");
    if( (j+1)%3 == 1 ) printf("1 3\n");
    if( (j+1)%3 == 2 ) printf("3 2\n");
    }
    else{                            // 逆时针
    if( (j+1)%3 == 0 ) printf("3 1\n");
    if( (j+1)%3 == 1 ) printf("1 2\n");
    if( (j+1)%3 == 2 ) printf("2 3\n");
    }
  }
return 0;}
```

ASCII 码表

下面的 ASCII 码表包含数值在 0-127 之间的字符的十进制、八进制以及十六进制表示.

十进制	八进制	十六进制	字符	描述
0	0	00	NUL	
1	1	01	SOH	start of header
2	2	02	STX	start of text
3	3	03	ETX	end of text
4	4	04	EOT	end of transmission
5	5	05	ENQ	enquiry
6	6	06	ACK	acknowledge
7	7	07	BEL	bell
8	10	08	BS	backspace
9	11	09	HT	horizontal tab
10	12	0A	LF	line feed
11	13	0B	VT	vertical tab
12	14	0C	FF	form feed
13	15	0D	CR	carriage return
14	16	0E	SO	shift out
15	17	0F	SI	shift in
16	20	10	DLE	data link escape
17	21	11	DC1	no assignment, but usually XON
18	22	12	DC2	
19	23	13	DC3	no assignment, but usually XOFF
20	24	14	DC4	
21	25	15	NAK	negative acknowledge
22	26	16	SYN	synchronous idle
23	27	17	ETB	end of transmission block
24	30	18	CAN	cancel
25	31	19	EM	end of medium
26	32	1A	SUB	substitute
27	33	1B	ESC	escape
28	34	1C	FS	file seperator
29	35	1D	GS	group seperator
30	36	1E	RS	record seperator

31	37	1F	US	unit seperator
32	40	20	SPC	space
33	41	21	!	
34	42	22	″	
35	43	23	#	
36	44	24	$	
37	45	25	%	
38	46	26	&	
39	47	27	’	
40	50	28	(	
41	51	29	)	
42	52	2A	*	
43	53	2B	+	
44	54	2C	,	
45	55	2D	-	
46	56	2E	.	
47	57	2F	/	
48	60	30	0	
49	61	31	1	
50	62	32	2	
51	63	33	3	
52	64	34	4	
53	65	35	5	
54	66	36	6	
55	67	37	7	
56	70	38	8	
57	71	39	9	
58	72	3A	:	
59	73	3B	;	
60	74	3C	<	
61	75	3D	=	
62	76	3E	>	
63	77	3F	?	
64	100	40	@	
65	101	41	A	
66	102	42	B	
67	103	43	C	

68	104	44	D	
69	105	45	E	
70	106	46	F	
71	107	47	G	
72	110	48	H	
73	111	49	I	
74	112	4A	J	
75	113	4B	K	
76	114	4C	L	
77	115	4D	M	
78	116	4E	N	
79	117	4F	O	
80	120	50	P	
81	121	51	Q	
82	122	52	R	
83	123	53	S	
84	124	54	T	
85	125	55	U	
86	126	56	V	
87	127	57	W	
88	130	58	X	
89	131	59	Y	
90	132	5A	Z	
91	133	5B	[	
92	134	5C	\	
93	135	5D	]	
94	136	5E	^	
95	137	5F	_	
96	140	60	`	
97	141	61	a	
98	142	62	b	
99	143	63	c	
100	144	64	d	
101	145	65	e	
102	146	66	f	
103	147	67	g	
104	150	68	h	

105	151	69	i	
106	152	6A	j	
107	153	6B	k	
108	154	6C	l	
109	155	6D	m	
110	156	6E	n	
111	157	6F	o	
112	160	70	p	
113	161	71	q	
114	162	72	r	
115	163	73	s	
116	164	74	t	
117	165	75	u	
118	166	76	v	
119	167	77	w	
120	170	78	x	
121	171	79	y	
122	172	7A	z	
123	173	7B	{	
124	174	7C	|	
125	175	7D	}	
126	176	7E	~	
127	177	7F	DEL	delete

ACM 中 java 的使用

这里指的 java 速成，只限于 java 语法，包括输入输出，运算处理，字符串和高精度的处理，进制之间的转换等，能解决 OJ 上的一些高精度题目。

```
/*==================================================*\
|1. 输入：
|格式为：
|Scanner cin = new Scanner (new
|                         BufferedInputStream(System.in));
\*==================================================*/
```

例子：

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main
{
    public static void main(String[] args)
    {
        Scanner cin = new Scanner (new
                BufferedInputStream(System.in));
        int a; double b; BigInteger c; String st;
        a = cin.nextInt(); b = cin.nextDouble(); c = cin.nextBigInteger(); d = cin.nextLine(); // 每种类型都有相应的输入函数.
    }
}
```

```
/*==================================================*\
|2. 输出
|函数：System.out.print(); System.out.println();System.out.printf()
|System.out.print();          // cout << …;
|System.out.println();        // cout << … << endl;
|System.out.printf();         // 与 C 中的 printf 用法类似.
\*==================================================*/
```

例子：

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main
{
    public static void main(String[] args)
    {
        Scanner cin = new Scanner (new BufferedInputStream(System.in));
        int a; double b;
        a = 12345; b = 1.234567;
        System.out.println(a + " " + b);
        System.out.printf("%d %10.5f\n", a, b); // 输入 b 为字宽为 10，右对齐，保留小数点后 5 位，四舍五入.
    }
}
```

规格化的输出：

函数：

// 这里 0 指一位数字，#指除 0 以外的数字(如果是 0，则不显示),四舍五入.

```
    DecimalFormat fd = new DecimalFormat("#.00#");
    DecimalFormat gd = new DecimalFormat("0.000");
    System.out.println("x =" + fd.format(x));
    System.out.println("x =" + gd.format(x));
```

```
/*==================================================*\
|3. 字符串处理
|java 中字符串 String 是不可以修改的，要修改只能转换为字符数组.
\*==================================================*/
```

例子：

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main
{
    public static void main(String[] args)
    {
        int i;
        Scanner cin = new Scanner (new BufferedInputStream(System.in));
        String st = "abcdefg";
        System.out.println(st.charAt(0)); // st.charAt(i)就相当于st[i].
        char [] ch;
        ch = st.toCharArray(); // 字符串转换为字符数组.
        for (i = 0; i < ch.length; i++) ch[i] += 1;
        System.out.println(ch); // 输入为“bcdefgh”.
if (st.startsWith("a")) // 如果字符串以'0'开头.
        {
            st = st.substring(1); // 则从第 1 位开始 copy(开头为第 0 位).
        }
    }
}
```

```
/*==================================================*\
|4. 高精度
|BigInteger 和 BigDecimal 可以说是 acmer 选择 java 的首要原因。
|函数：add, subtract, divide, mod, compareTo 等，其中加减乘除模|
都要求是 BigInteger(BigDecimal)和 BigInteger(BigDecimal)之间的|
运算，所以需要把 int(double)类型转换为 BigInteger(BigDecimal)，
|用函数 BigInteger.valueOf().
\*==================================================*/
```

例子：

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main
{
    public static void main(String[] args)
    {
        Scanner cin = new Scanner (new BufferedInputStream(System.in));
        int a = 123, b = 456, c = 7890;
        BigInteger x, y, z, ans;
        x = BigInteger.valueOf(a); y = BigInteger.valueOf(b); z = BigInteger.valueOf(c);
        ans = x.add(y); System.out.println(ans);
        ans = z.divide(y); System.out.println(ans);
        ans = x.mod(z); System.out.println(ans);
        if (ans.compareTo(x) == 0) System.out.println("1");
       System.out.println(BigInteger.valueOf(a).pow(b));
/////a^b

    }
}
```

```
/*==================================================*\
|5. 进制转换
java 很强大的一个功能。
函数：
String st = Integer.toString(num, base); // 把 num 当做 10 进制的数转成 base 进制的 st(base <= 35).
int num = Integer.parseInt(st, base); // 把 st 当做 base 进制，转成 10 进制的 int(parseInt 有两个参数,第一个为要转的字符串,第二个为说明是什么进制).
BigInter m = new BigInteger(st, base); // st 是字符串，base 是 st 的进制.
//Added by abilitytao
1. 如果要将一个大数以 2 进制形式读入 可以使用 cin.nextBigInteger(2);
当然也可以使用其他进制方式读入；
2. 如果要将一个大数转换成其他进制形式的字符串 使用
```

ACM 中 java 的使用

cin.toString(2);//将它转换成 2 进制表示的字符串

==/

例子：POJ 2305

```
import java.io.*;
import java.util.*;
import java.math.*;
public class Main
{
    public static void main(String[] args)
    {
        int b;
        BigInteger p,m,ans;
        String str ;
        Scanner cin = new Scanner (new BufferedInputStream(System.in));
        while(cin.hasNext())
        {
            b=cin.nextInt();
            if(b==0)
                break;
            p=cin.nextBigInteger(b);
            m=cin.nextBigInteger(b);
            ans=p.mod(m);
            str=ans.toString(b);
            System.out.println(str);
        }
    }
}
//End by abilitytao
```

/*==*\

|6. 排序

函数：Arrays.sort();至于怎么排序结构体，像 C++里写个 cmp 的方法，在 java 还不太清楚，希望有人指点下～～

==/

例子：

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main
{
    public static void main(String[] args)
    {
        Scanner cin = new Scanner (new BufferedInputStream(System.in));
        int n = cin.nextInt();
        int a[] = new int [n];
        for (int i = 0; i < n; i++) a[i] = cin.nextInt();
        Arrays.sort(a);
        for (int i = 0; i < n; i++) System.out.print(a[i] + " ");
    }
}
```

/*==*\

对 node 类构成的数组按照 dist 从大到小排序，若相等，按照 x 的从小到大排序

==/

```
import java.io.*;
import java.util.*;
class node implements Comparable
{
    public int x;
    public int dist;
  public node(int _x, int _dist)
    {
      this.x = _x;
      this.dist = _dist;
    }
public int compareTo(Object obj)
{
    if (obj instanceof node)
  {
      node b = (node) obj;
      if(this.dist!=b.dist)
          return   (this.dist<b.dist)?1:0;
      else
          return   (this.x>b.x)?1:0;
    }
  /*if (this.dist < b.dist)
        return -1;
    else if (this.dist > b.dist)
    return 1;
    }*/
    return 0;
    }
}

 public class Main
 {
     static int n;
     static node a[];
  public static void main(String args[]) throws Exception
  {
     Scanner cin = new Scanner(System.in);
      n = cin.nextInt();
     a = new node[n];
for (int i = 0; i < n; i++)
 {
    int t   = cin.nextInt();
    int tt = cin.nextInt();
   a[i] = new node(t, tt);
 }
Arrays.sort(a);
    for (int i = 0; i < n; i++)
      {
        System.out.print(a[i].x);
        System.out.print(" "+a[i].dist);
        System.out.println();
      }
   }
}
```

/*==*\

|7. POJ 高精度题目汇总：

==/

POJ 1131 1205 1220 1405 1503 1604 1894 2084 2305 2325 2389 2413 3101 3199

http://blog.sina.com.cn/s/blog_6635898a0100ovel.html#post

http://blog.sina.com.cn/s/blog_6635898a0100ovun.html

转自：czyuan 大牛

http://hi.baidu.com/czyuan_acm/blog/item/d0bf7a439d90d21b72f05d69.html

/*==*\

|一个整数各个位的数字相乘，可以得到一个新的数，继续各个位相乘，

|最后得到一个数，比如：

|679 -> 378 -> 168 -> 48 -> 32 -> 6

|问给定一个数，求一个最小的数，使得其各位乘积是这个数

==/

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main {
public static void main(String[] args) {
    Scanner cin = new Scanner(System.in);
    int i, cnt[] = new int[12];
    BigInteger out, one, ten, num;
    out = BigInteger.valueOf(-1);
    one = BigInteger.valueOf(1);
    ten = BigInteger.valueOf(10);
    while(cin.hasNext()){
        num = cin.nextBigInteger();
        if(num.compareTo(out) == 0) break;
        if(num.compareTo(ten) < 0){ //  小于 10 的情况。
```

```
            System.out.println("1"+num); continue;
        }
for(i = 9; i >= 2; i --){           // 按贪心的思想分解。
            cnt[i] = 0;
    BigInteger BigI = BigInteger.valueOf(i);
    while(num.mod(BigI).compareTo(BigInteger.ZERO) == 0){
            num = num.divide(BigI);
            cnt[i] ++;
        }
}
if(num.compareTo(one) != 0){   // 含有大于等于 10 质数因子的情况。
    System.out.println("There is no such number.");
}
else{       // 按从小到大输出。
for(i = 2; i <= 9; i ++)
while((cnt[i] --) != 0)
            System.out.print(i);
            System.out.println();
        }
    }
  System.exit(0);
  }
}
/*=====================================================*\
|将一行中以空格中分开的四个数据分别读入四个整型变量
\*=====================================================*/
import java.util.Scanner;
public class Test{
  public static void main(String args[]){
    Scanner scan=new Scanner(System.in);
      int p=0;
      int e=0;
      int i=0;
      int d=0;
       while (p!=-1) {//或者 while (scan.hasNext())
          p = scan.nextInt();
          e = scan.nextInt();
          i = scan.nextInt();
          d = scan.nextInt();
          System.out.printf("p=%d,e=%d,i=%d,d=%d\n",p,e,i,d);
       }
   }
}
C:\java>java   Test
1 2 3 4
p=1,e=2,i=3,d=4
5 6 7 8
p=5,e=6,i=7,d=8
-1 2 3 4
p=-1,e=2,i=3,d=4
/*=====================================================*\
|下面程序从文件中读并输出到文件,
|例子计算 n!
\*=====================================================*/
import java.io.*;
import java.util.*;
import java.math.*;
public class FileInAndOut {

public static void main(String[] args) throws Exception{

    //input.in 为输入文件,在当前文件夹下
   System.setIn(new BufferedInputStream(new
FileInputStream("input.txt")));

   //output.in 为输出文件,在当前文件夹下
   System.setOut(new PrintStream(new File("output.txt")));
   //以上代码可作为文件输入输出模板,下面的是程序要实现的功能
   Scanner sc=new Scanner(System.in);
    while(sc.hasNextLong()){
     BigInteger ans=new BigInteger("1");
     long n=sc.nextLong();
        for(int i=1;i<=n;i++)
          ans=ans.multiply(BigInteger.valueOf(i));
     System.out.println(n+"!="+ans);

     }
   }
}
input.txt:
1 2 3 4 5 6 7 8
9 10 11
12
output.txt:
1!=1
2!=2
3!=6
4!=24

/*=====================================================*\
|求 num^n
\*=====================================================*/
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main {
        public static void main(String[] args) {

          Scanner cin = new Scanner(System.in);
          BigDecimal num;
          int ep, sta, end, i;
          String st;
          while(cin.hasNext()){      // 相当于 c++的!=EOF。
          num = cin.nextBigDecimal();
          ep = cin.nextInt();
          num = num.pow(ep);       // 计算 num^ep。
          st = new String(num.toPlainString()); // toString()会有科学记数法。
          sta = 0;
          while(st.charAt(sta) == '0') sta ++; // 去掉前缀的 0。
          end = st.length() - 1;
          while(st.charAt(end) == '0') end --; // 去掉后缀的 0。
        if(st.charAt(end) == '.') end --; // 若小数点后没 0,去掉小数点。
           for(i = sta; i <= end; i ++)
               System.out.print(st.charAt(i));
           System.out.println();

       }

    System.exit(0);

  }
}
/*=====================================================*\
|计算 R^N,R 为浮点数
| PKU 1001
|BigInteger/BigDecimal 常用方法(所有方法均为 1 个参数)
|add()加 subtract()减 multiply()乘 divide()除
|abs()绝对值 max()最大值 min()最小值
|compareTo()比较大小 toString()转为字符串
|仅 BigInteger:
|mod()取余 gcd()求最大公约数
|and()求与 or()求或 not()求反 xor()求异或
\*=====================================================*/
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main
{
        public static void main(String[] args)
        {
              Scanner cin=new Scanner(System.in);
```

```
            while(cin.hasNext())
            {
                BigDecimal x=cin.nextBigDecimal();
                BigDecimal y=BigDecimal.valueOf(1.0);
                int n=cin.nextInt();
                for(int i=1;i<=n;i++)
                    y=y.multiply(x);
                String out=new String(y.toPlainString());
                boolean flag=false;
                int q=out.length()-1;
                //去掉前导零
                while(out.charAt(q)=='0') q--;
                if (out.charAt(q)=='.') q--;
                int p=0;
                //去掉小数点后多余的零
                while(out.charAt(p)=='0') p++;
                for(int i=p;i<=q;i++)
                    System.out.print(out.charAt(i));
                System.out.println();
            }
        }
}
```

/*==*\
|大数的进制转换。将 a 进制的正整数 num 转换为 b 进制的正整数
==/

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main {
public static void main(String[] args) {
    Scanner cin = new Scanner(System.in);
     char c;
    int testCase, i, w;
    BigInteger ba1, ba2, t, sum;
    testCase = cin.nextInt();
    while((testCase --) != 0){
    ba1 = cin.nextBigInteger();        // 读入大数。
    ba2 = cin.nextBigInteger();
String st = cin.next(); // 读入字符串，无空格，而cin.nextline()有。
 t = new BigInteger("1");    // 等价于 t = BigInteger.valueOf(1)。
 sum = new BigInteger("0");
 for(i = st.length() - 1; i >= 0; i --){   // 先将num转换为10进制。
     c = st.charAt(i);
     if(c >= '0' && c <= '9') w = c - '0';
     else if(c >= 'A' && c <= 'Z') w = c - 'A' + 10;
     else w = c - 'a' + 36;
   sum = sum.add(BigInteger.valueOf(w).multiply(t));
    t = t.multiply(ba1);
 }
 BigInteger zero = BigInteger.valueOf(0);
  int top = 0, stack[] = new int[2000];
while(sum.compareTo(zero) != 0){
   // 转化为ba2进制的数，存在stack[]中。
stack[++ top] = sum.mod(ba2).intValue();
sum = sum.divide(ba2);
}
System.out.print(ba1+" "+st+"\n"+ba2+" ");
if(top == 0) System.out.print(0);    // 要注意为0的情况。
while(top != 0){    // 按所给字符输出。
w = stack[top --];
if(w < 10) c = (char)('0' + w);
else if(w < 36) c = (char)(w - 10 + 'A');
else c = (char)(w - 36 + 'a');
System.out.print(c);
 }
  System.out.print("\n\n");
 }
 System.exit(0);
 }
}
```

/*==*\
|求一个八进制小数的十进制
==/

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main {
public static void main(String[] args) {
     Scanner cin = new Scanner(System.in);
     BigDecimal ans, t, tmp;
      while(cin.hasNext()){
         String st = cin.nextLine(); // 字符串读入。
         t = BigDecimal.valueOf(1);
         ans = BigDecimal.valueOf(0);
int i, sta = st.indexOf('.'); // sta = 2是一样的，这样保险点，预防前缀0。
      for(i = sta + 1; i < st.length(); i ++){
           tmp = BigDecimal.valueOf(st.charAt(i) - '0');
           t = t.divide(new BigDecimal("8"));
           tmp = tmp.multiply(t);
             //System.out.println(t);
      ans = ans.add(tmp);
    }
   System.out.println(st+" [8] = "+ans+" [10]");
 }
System.exit(0);
}
}
```

/*==*\
|大数计算 Catalan 数
==/

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        BigInteger t1, t2, dp[] = new BigInteger[1005];
        dp[1] = BigInteger.valueOf(1);
        for(int i = 2; i <= 1000; i ++){
            t1 = BigInteger.valueOf(4 * i - 2);     //   要注意
(4n-2)/(n+1)可能为小数。
            t2 = BigInteger.valueOf(i + 1);
            dp[i] = dp[i-1].multiply(t1).divide(t2);
        }
        while(cin.hasNext()){
            int id = cin.nextInt();
            if(id == -1) break;
            System.out.println(dp[id]);
        }
        System.exit(0);
    }
}
```

/*==*\
|求出在[a, b]范围内斐波那契数的个数，a <= b <= 10 的 100 次幂
==/

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;


public class Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        int i, cnt;
        BigInteger zero, a, b;
        BigInteger Fib[] = new BigInteger[505];
        zero = BigInteger.valueOf(0);
        Fib[1] = BigInteger.valueOf(1);
        Fib[2] = BigInteger.valueOf(2);
```

```
        for(i = 3; i < 500; i ++)
            Fib[i] = Fib[i-1].add(Fib[i-2]);
        while(cin.hasNext()){
            a = cin.nextBigInteger();
            b = cin.nextBigInteger();
            if(a.compareTo(zero) == 0 && b.compareTo(zero) ==
0) break;
            i = 1; cnt = 0;
            while(Fib[i].compareTo(a) < 0) i ++;
            while(Fib[i ++].compareTo(b) <= 0) cnt ++;
            System.out.println(cnt);
        }
        System.exit(0);
    }
}
/*====================================================*\
|求 n！高精度   注意 0 ！= 1
\*====================================================*/
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main
{
    public static void main(String[] args)
    {
        Scanner cin = new Scanner (new
BufferedInputStream(System.in));
        int  b ;
        while(cin.hasNext())
        {
            BigInteger x= new BigInteger("1");
            b=cin.nextInt();
            //if(b<0)  x=new BigInteger("0");
            //else
            for(;b>0;b--)  x = x.multiply(new BigInteger(new
Integer(b).toString()));
            System.out.println(x);

        }
    }
}
```

ACM 中 Java 的应用

Chapter I.

Java 的优缺点各种书上都有，这里只说说用 Java 做 ACM-ICPC 的特点:
(1) 最明显的好处是，学会 Java，可以参加 Java Challenge :)
(2) 对于熟悉 C/C++的程序员来说，Java 并不难学，找本书，一两周业余时间就可以搞定了。当然，这里只是指一般编程，想熟悉所有的 Java 库还是需要些时间的。

事实上,Java 只相当于 C++的一个改进版,所有的语法都几乎是 C++的，很少有变动。

(3) 在一般比赛中，Java 程序会有额外的时间和空间，而实际上经过实验，在执行计算密集任务的时候 Java 并不比 C/C++慢多少，只是 IO 操作较慢而已。
(4) Java 简单而功能强大，有些东西用 Java 实现起来更为方便，比如高精度。
(5) 用 Java 不易犯细微的错误，比如 C/C++中的指针，“if (n = m) ... ”等
(6) 目前来看 Eclipse 已成基本配置，写 Java 程序反而比 C/C++更方便调试。在具体竞赛时也算多一种选择。
(7) 学会 Java 对以后工作有好处。现在国外很多地方会 Java 的人比会 C/C++的人多。
(8) 会 Java 可以使你看起来更像偶蹄类动物（牛） hoho~

Chapter II.

下面说一下 ACM-ICPC 队员初用 Java 编程所遇到的一些问题:
1. 基本输入输出:
(1)
JDK 1.5.0 新增的 Scanner 类为输入提供了良好的基础，简直就是为 ACM-ICPC 而设的。
一般用法为:

```
import java.io.*
import java.util.*
public class Main
{
	public static void main(String args[])
	{
		Scanner cin = new Scanner(new BufferedInputStream(System.in));
		...
	}
}
```

当然也可以直接 Scanner cin = new Scanner(System.in);
只是加 Buffer 可能会快一些
(2)
读一个整数: int n = cin.nextInt(); 相当于 scanf("%d", &n); 或 cin >> n;
读一个字符串: String s = cin.next(); 相当于 scanf("%s", s); 或 cin >> s;
读一个浮点数: double t = cin.nextDouble(); 相当于 scanf("%lf", &t); 或 cin >> t;
读一整行: String s = cin.nextLine(); 相当于 gets(s); 或 cin.getline(...);
判断是否有下一个输入可以用 cin.hasNext() 或 cin.hasNextInt() 或 cin.hasNextDouble() 等，具体见 TOJ 1001 例程。
(3)
输出一般可以直接用 System.out.print() 和 System.out.println()，前者不输出换行，而后者输出。
比如：System.out.println(n); // n 为 int 型
同一行输出多个整数可以用

System.out.println(new Integer(n).toString() + " " + new

Integer(m).toString());
也可重新定义:
static PrintWriter cout = new PrintWriter(new BufferedOutputStream(System.out));
cout.println(n);

(4)
对于输出浮点数保留几位小数的问题，可以使用 DecimalFormat 类，
import java.text.*;
DecimalFormat f = new DecimalFormat("#.00#");
DecimalFormat g = new DecimalFormat("0.000");
double a = 123.45678, b = 0.12;
System.out.println(f.format(a));
System.out.println(f.format(b));
System.out.println(g.format(b));
这里 0 指一位数字，#指除 0 以外的数字。

2. 大数字
BigInteger 和 BigDecimal 是在 java.math 包中已有的类，前者表示整数，后者表示浮点数
用法:
不能直接用符号如+、-来使用大数字，例如:
(import java.math.*) // 需要引入 java.math 包
BigInteger a = BigInteger.valueOf(100);
BigInteger b = BigInteger.valueOf(50);
BigInteger c = a.add(b) // c = a + b;
主要有以下方法可以使用:
BigInteger add(BigInteger other)
BigInteger subtract(BigInteger other)
BigInteger multiply(BigInteger other)
BigInteger divide(BigInteger other)
BigInteger mod(BigInteger other)
int compareTo(BigInteger other)
static BigInteger valueOf(long x) ///初始化
输出大数字时直接使用 System.out.println(a) 即可。

3. 字符串
String 类用来存储字符串，可以用 charAt 方法来取出其中某一字节，计数从 0 开始:
String a = "Hello"; // a.charAt(1) = 'e'
用 substring 方法可得到子串，如上例
System.out.println(a.substring(0, 4)) // output "Hell"
注意第 2 个参数位置上的字符不包括进来。这样做使得 s.substring(a, b) 总是有 b-a 个字符。
字符串连接可以直接用 + 号，如
String a = "Hello";
String b = "world";
System.out.println(a + ", " + b + "!"); // output "Hello, world!"
如想直接将字符串中的某字节改变，可以使用另外的 StringBuffer 类。

4. 调用递归（或其他动态方法）
在主类中 main 方法必须是 public static void 的，在 main 中调用非 static 类时会有警告信息，
可以先建立对象，然后通过对象调用方法:

ACM 中 java 的使用

```
public class Main
{
	...
	void dfs(int a)
	{
		if (...) return;
		...
		dfs(a+1);
	}

	public static void main(String args[])
	{
		...
		Main e = new Main();
		e.dfs(0);
		...
	}
}
```

5. 其他注意的事项

(1) Java 是面向对象的语言，思考方法需要变换一下，里面的函数统称为方法，不要搞错。

(2) Java 里的数组有些变动，多维数组的内部其实都是指针，所以 Java 不支持 fill 多维数组。

数组定义后必须初始化，如 int[] a = new int[100];

(3) 布尔类型为 boolean，只有 true 和 false 二值，在 if (...) / while (...) 等语句的条件中必须为 boolean 类型。

在 C/C++中的 if (n % 2) ... 在 Java 中无法编译通过。

(4) 下面在 java.util 包里 Arrays 类的几个方法可替代 C/C++里的 memset、qsort/sort 和 bsearch:

Arrays.fill()

Arrays.sort()

Arrays.binarySearch()

转自：http://hi.baidu.com/oak_wesley/blog/item/35839200fd9dc10e1d9583de.html

Java 进制转换~集锦

由于 Unicode 兼容 ASCII（0～255），因此，上面得到的 Unicode 就是 ASCII。

java 中进行二进制，八进制，十六进制，十进制间进行相互转换

Integer.toHexString(int i)

十进制转成十六进制

Integer.toOctalString(int i)

十进制转成八进制

Integer.toBinaryString(int i)

十进制转成二进制

Integer.valueOf("FFFF",16).toString()

十六进制转成十进制

Integer.valueOf("876",8).toString()

八进制转成十进制

Integer.valueOf("0101",2).toString()

二进制转十进制

至于转换成二进制或其他进制，Java API 提供了方便函数，你可以查 Java 的 API 手册。

ACM 中 java 的使用

以字符 a 的 ASCII 为例：
int i = 'a';
String iBin = Integer.toBinaryString(i);//二进制
String iHex = Integer.toHexString(i);//十六进制
String iOct = Integer.toOctalString(i);//八进制
String iWoKao = Integer.toString(i,3);//三进制或任何你想要的 35 进制以下的进制
DEC

有什么方法可以直接将 2,8,16 进制直接转换为 10 进制的吗?
java.lang.Integer 类
parseInt(String s, int radix)
使用第二个参数指定的基数，将字符串参数解析为有符号的整数。
examples from jdk:
parseInt("0", 10) returns 0
parseInt("473", 10) returns 473
parseInt("-0", 10) returns 0
parseInt("-FF", 16) returns -255
parseInt("1100110", 2) returns 102
parseInt("2147483647", 10) returns 2147483647
parseInt("-2147483648", 10) returns -2147483648
parseInt("2147483648", 10) throws a NumberFormatException
parseInt("99", 8) throws a NumberFormatException
parseInt("Kona", 10) throws a NumberFormatException
parseInt("Kona", 27) returns 411787

进制转换如何写（二，八，十六）不用算法
Integer.toBinaryString
Integer.toOctalString
Integer.toHexString

例一：

```
public class Test{
public static void main(String args[]){

int i=100;
String binStr=Integer.toBinaryString(i);
String otcStr=Integer.toOctalString(i);
String hexStr=Integer.toHexString(i);
System.out.println(binStr);
```

例二：

```
public class TestStringFormat {
public static void main(String[] args) {
if (args.length == 0) {
System.out.println("usage: java TestStringFormat <a number>");
System.exit(0);
}

Integer factor = Integer.valueOf(args[0]);

String s;
```

```
s = String.format("%d", factor);
System.out.println(s);
s = String.format("%x", factor);
System.out.println(s);
s = String.format("%o", factor);
System.out.println(s);
}
}
```

各种数字类型转换成字符串型：

```
String s = String.valueOf( value); // 其中 value 为任意一种数字类型。
```

字符串型转换成各种数字类型：

```
String s = "169";
byte b = Byte.parseByte( s );
short t = Short.parseShort( s );
int i = Integer.parseInt( s );
long l = Long.parseLong( s );
Float f = Float.parseFloat( s );
Double d = Double.parseDouble( s );
```

数字类型与数字类对象之间的转换：

```
byte b = 169;
Byte bo = new Byte( b );
b = bo.byteValue();

short t = 169;
Short to = new Short( t );
t = to.shortValue();

int i = 169;
b = bo.byteValue();

short t = 169;
Short to = new Short( t );
t = to.shortValue();

int i = 169;
Integer io = new Integer( i );
i = io.intValue();

long l = 169;
Long lo = new Long( l );
l = lo.longValue();

float f = 169f;
Float fo = new Float( f );
f = fo.floatValue();
```

ACM 中 java 的使用

```
double d = 169f;
Double dObj = new Double( d );
d = dObj.doubleValue();
```

算法

数据结构的算法

数论与代数算法

- 最大公约数
- 最小公倍数
- 分解质因数
- 素数判定
- 进制转换
- 高精度计算

几何的算法

- 凸包
  - Gift wrapping
  - Graham scan

图论的算法

- 哈夫曼编码、哈夫曼树（最优二叉树）
- 树的遍历
- 最短路径算法
- 欧拉路
- 哈密尔顿回路
- 差分约束系统
- 最小生成树算法

- 最小树形图
- 强连通分量
- 网络流算法
- 匹配算法

动态规划

- 背包问题
  - 01 背包
  - 完全背包
  - 多重背包
  - 混合背包
  - 二维费用背包
  - 分组背包问题
  - 有依赖的背包
- 最长不下降子序列
- 最长公共字序列
- 树型动态规划
- 动态规划的优化

其他

- 数值分析
- 加密算法
- 排序算法

- 检索算法
- 随机化算法
- 并行算法
- 模拟退火算法
- 蚁群优化算法
- 遗传算法
- 人工神经网络
- 禁忌搜索算法
- 粒子群优化算法

数学

图论

- 有向图
    - 有向网 (带权有向图)
    - 有向无环图 (DAG)
        - AOE 网 (带权有向无环图)
        - AOV 网
- 无向图
    - 无向网 (带权无向图)
- 连通图
    - 强连通图
- 完全图

- ☐ 稀疏图
- ☐ 稠密图

整数论

- ☐ 整除问题
- ☐ 素数
- ☐ 进位制
- ☐ 同余
- ☐ 欧拉函数
- ☐ 扩展欧几里得的算法

组合数学

- ☐ 置换群
- ☐ 递推关系
- ☐ 母函数
- ☐ 离散变换
- ☐ 康托展开

线性代数

- ☐ 矩阵
- ☐ 向量
- ☐ 线性方程组
- ☐ 线性规划

几何

- 线段的基本问题
- 多边形和多面体相关问题
- 凸包及其应用

博弈论

- Normal Play
  - 必胜态与必败态
  - SG 函数
    - Nim-Game
    - Graph-Game
  - 游戏的和
- Misere Play

数理逻辑

- 命题逻辑
- 谓词逻辑

重要的算法

- 求有向图的强连通分支 (Strongerst Connected Component)
  - Kosaraju 算法
  - Gabow 算法
  - Tarjan 算法
- 求最小生成树 (Minimal Spanning Trees)
  - Kruskal 算法
  - Prim 算法
- 最小树形图
  - 朱永津刘振宏算法
- 最短路径问题
  - SSSP(Single-source Shortest Paths) (单源最短路径)
    - Dijkstra 算法
    - Bellman-Ford 算法(改进：SPFA 算法)
  - APSP(All-pairs Shortest Paths) (多源最短路径)
    - Floyd-Warshall 算法
    - Johnson 算法
- 网络流问题
  - 最大网络流

- 增广路算法
  - Ford-Fulkerson 算法
  - Edmonds-Karp 算法 O(V*E^2)
  - 最短路径增殖 EK-2（MPLA）(也就是很流行的 SAP 算法

    Shortest Augmenting Paths) O(V^2*E)
  - Dinic O(V^2*E)
- 预流推进算法 HLPP

□ 最小费用流

□ 图匹配问题

□ 匈牙利算法

□ Hopcroft Karp 算法

□ Kuhn-Munkres 算法

□ Edmonds' blossom-contraction 算法

ACM 比赛经验

1. 比赛中评测会有些慢，偶尔还会碰到隔 10 分钟以上才返回结果的情况，这段时间不能等结果，必须开工其他题，如果 WA，两道题同时做。交完每道题都要先打印。

2. 比赛时发的饭不是让你当时就吃的，那是给你赛后吃的。基本上比赛中前几名的队都没人吃，除非领先很多。

3. 很多选手，尤其是第一次参加比赛的，到一个新环境，全当旅游了，参观的参观，找同学的找同学，玩玩乐乐就把正事抛到脑后了，结果比赛自然没什么好成绩，这样的例子太多了。所以到参赛地后要时刻不忘自己是来比赛的，好好休息、备战。

4. 参赛前一天要睡 10 个小时以上，非常有助于保持比赛中的精力，很多时候比赛到 3 个多小时队员就没劲了就是这个原因。前一天晚饭与当天早饭要吃好，理由同上，要知道下顿饭得下午 3 点赛后才能吃。

5. 到新环境，时刻注意远离疾病，感冒肠炎病不大，却是成绩的天敌。

6. 英语不好，看不懂的，要勤查词典，懒一次就少一道题，远离奖牌。

7. 可以紧张，杜绝慌张，慌张是出题的敌人，任何时候，如果发现自己或者队友出现慌张的情况，提醒深呼吸。

8. 照着纸敲代码和 sample 数据时不要敲错，特别注意文字信息。

9. 第一道简单题交给队中最稳的人做，万一遇到麻烦也不要慌，如果有很多队都出了就更不必着急了，它必定是简单题，必定是可以很快做出来的，晚几分钟也比罚掉 20 分好。另外注意不要 PE。

10. 最后一小时是出题高峰，谁松懈，谁落后。最后一小时出一道是正常，出两道更好。

以上各条均有出处，每条都包含着以往教训，每条都可能浪费掉你一年的努力，不可小视。

以下各条有些来自于其他学校，有些是总结：

11. 无论是否有人通过，所有题必须全读过，最好每道题都有两人以上读过，尽量杜绝讲题现象。要完全弄清题意，正确的判断出题目的难易，不要想当然。

12. 虽然讨论有助于出题，但是以往每赛区第一名基本都是各自为战，但是互相了解，觉得一道题适合其他人做就转手。

13. 保持头脑灵活，在正常方法不行时想想歪门邪道，比如换种不常见的特殊的数据结构，加预处理，限时搜索等。效率是第一位的，如果觉得 DP 麻烦就用记忆化搜索，总之考虑清楚后就要在最短时间出题。

14. 竞赛中 更需要比平时稳定，程序出来后要检查重点地方，尽量 1Y。对于 WA 的题，不要改一处就交，很可能还有错的地方，要稳，要懂得在压力下也要仔细。对 WA 的题 测试时要完整，必须每个点都测到，但不一定特别复杂。要考虑到测试的各种边界情况，比如矩阵可能为 1*1 或 1*n 或 m*1。

15. 除非做出的人很多，否则最后考虑复杂几何题，精度造成的问题太多了。对 double 型操作要小心判断大小、绝对值等情况。一般情况下不要用 float 型。

16. 块复制要小心，检查相应的部分是否已经正确修改。

17. 纸上写程序要尽量完整，每道题上机时间（包括输入、测试和调试）不

要超过一小时。程序出错如果一时无法排除就应该打印出来阅读而把机器让出来。

18. 提交时注意题号，不要交错题。由于 PC^2 的界面，这种情况时有发生。

19. 尽可能想到题目可以用到的数学的东西。

20. 初始化必不可少。

21. 数组行列下标不要弄反，位运算或字符串哪头是 0 和 n 不要搞反。

22. 提交时记得把所有的调试信息都关掉。

23. 实在迫不得已才可换人做题。

24. 有想法后，写程序之前想好时空效率。比赛中一般不会出现时限 30 秒以上的题（国外赛区除外），10 秒及以上的一般不会超过 3 道。

25. 竞赛机会每年只有一次，训练了很长时间，如果比赛中出现疏失，那么今后一年都会后悔。对于不准备明年参赛的同学，更是要珍惜最后一次参赛机会。

附以前所写《组队赛说明》

1 要有做题比较多的队员，对于各种题型都有所涉及，做题稳，一般对前两道简单题能够保证快速，并且 99%以上一次 AC。

2 要有人专门应付数学与几何题，但复杂的几何题要放在最后做，对一些常用的函数要有模版准备。如精度控制，叉积，凸包等。

3 要有人能够对付麻烦的题，并保证一定的通过率，大多数的比赛都至少有一道这样的题，如 POJ 1913，TOJ 1092。

4 要有人对 DP 非常之熟，单次、双次、相对等情况都不在话下。对经典 DP 手到擒来。

5 要有人对稀奇古怪的算法都做过程序，涉猎广，对于数论、图论中的一些特殊结论都知道。如 TOJ 1584，ZOJ 1015，UVA 10733。

6 要有人对复杂的通用算法做过程序，如网络流中的最小费用最大流等等一系列的流，求割点/割边，启发式搜索/搏弈等。

7 模版要自己写，并且另两个人都认真读过，用以往题目进行多次的测试。模版要全，但要控制篇幅，因为很多赛区已开始限制页数。

8 要有人对 Linux/vi/gcc 系统熟悉，对 PC^2 熟悉，一定注意正式比赛时不要出现提交错题的情况。另外也要试用 Dev-C++等 Windows 下的免费软件。总之熟悉比赛环境。

9 每次练习赛都要当作正式比赛来做，要确保所有的题都看过，赛后要把没做出来的题尽量补上。

10 可能的话多看看以往比赛的总结、照片和录象，缩短与正式竞赛的距离，避免正式竞赛时紧张得做不出题等情况。

最好的情况就是对于各种题目三个队员都能做，但是又各有侧重。

要保证出来一道题能够有人会做、敢做，至少也要知道做法。