ACM/ICPC

Fudan University

LeGenD.N

Reference Document

Coach	教练
Weiwei Sun	孙未未

Team Member	队员
Zhan Yu	喻展
Kailun Chen	陈凯伦
Liaoliang Ye	叶寥亮

Contents

Part I
Mathematics

1 Chapter 1 Analytic Geometry 解析几何

1.1 空间的平面和直线

1.1.1 空间平面方程

点式法

$$A(x-x_0)+B(y-y_0)+C(z-z_0)=0$$

其中(A,B,C)是平面法向量。由平面三个点可以利用叉积求的法向量

1.1.2 空间直线方程

一般式

直线 L 作为两个平面的交线：

$$\begin{cases}A_1x+B_1y+C_1z+D_1=0\\A_2x+B_2y+C_2z+D_2=0\end{cases}$$

方向数：

$$p=\begin{vmatrix}B_1 & C_1\\B_2 & C_2\end{vmatrix}\quad q=\begin{vmatrix}C_1 & A_1\\C_2 & A_2\end{vmatrix}\quad r=\begin{vmatrix}A_1 & B_1\\A_2 & B_2\end{vmatrix}$$

对称式直线 L 通过点 M(x0,y0,z0)且具有方向数 p,q,r

$$\frac{x-x_0}{p}=\frac{y-y_0}{q}=\frac{z-z_0}{r}$$

两点式

$$\frac{x-x_1}{x_2-x_1}=\frac{y-y_1}{y_2-y_1}=\frac{z-z_1}{z_2-z_1}$$

1.1.3 空间点到直线距离

$$L:\frac{x-x_0}{p}=\frac{y-y_0}{q}=\frac{z-z_0}{r}$$

$$d=\frac{\begin{vmatrix}\vec{i} & \vec{j} & \vec{k}\\p & q & r\\x-x_0 & y-y_0 & z-z_0\end{vmatrix}}{p^2+q^2+r^2}$$

式中 d 为点 M(x0,y0,z0)到直线 L 的距离

1.1.4 点到平面距离

一般式：$Ax+By+Cz+D=0$

$$d=\frac{|Ax_0+By_0+Cz_0+D|}{\sqrt{A^2+B^2+C^2}}$$

1.1.5 空间中两直线距离

两不平行直线的最短距离

$$L_1:\frac{x-x_1}{p_1}=\frac{y-y_1}{q_1}=\frac{z-z_1}{r_1}$$

$$L_2:\frac{x-x_2}{p_2}=\frac{y-y_2}{q_2}=\frac{z-z_2}{r_2}$$

$$d=\frac{\begin{vmatrix}x_2-x_1 & y_2-y_1 & z_2-z_1\\p_1 & q_1 & r_1\\p_2 & q_2 & r_2\end{vmatrix}}{\sqrt{\begin{vmatrix}p_1 & q_1\\p_2 & q_2\end{vmatrix}^2+\begin{vmatrix}q_1 & r_1\\q_2 & r_2\end{vmatrix}^2+\begin{vmatrix}r_1 & p_1\\r_2 & p_2\end{vmatrix}^2}}$$

d是指L_1，L_2的公关垂线与此两焦点之间的距离。由此可推出两直线共面条件为$d=0$，所在平面方程：

$$\begin{vmatrix}x_2-x_1 & y_2-y_1 & z_2-z_1\\p_1 & q_1 & r_1\\p_2 & q_2 & r_2\end{vmatrix}=0$$

1.1.6 空间中直线与直线的夹角

$$L_1:\frac{x-x_1}{p_1}=\frac{y-y_1}{q_1}=\frac{z-z_1}{r_1}$$

$$L_2:\frac{x-x_2}{p_2}=\frac{y-y_2}{q_2}=\frac{z-z_2}{r_2}$$

$$\cos\theta=\frac{p_1p_2+q_1q_2+r_1r_2}{\sqrt{p_1^2+q_1^2+r_1^2}\sqrt{p_2^2+q_2^2+r_2^2}}$$

1.1.7 直线与平面的夹角

$$L:\ \frac{x-x_0}{p}=\frac{y-y_0}{q}=\frac{z-z_0}{r}$$

$$P:\ Ax+By+Cz+D=0$$

$$\sin\theta=\frac{|pA+qB+rC|}{\sqrt{p^2+q^2+r^2}\sqrt{A^2+B^2+C^2}}$$

1.1.8 平面与平面的夹角

$$P_1:A_1x+B_1y+C_1z+D_1=0$$

$$P_2:A_2x+B_2y+C_2z+D_2=0$$

$$\cos\theta=\frac{A_1A_2+B_1B_2+C_1C_2}{\sqrt{A_1^2+B_1^2+C_1^2}\sqrt{A_2^2+B_2^2+C_2^2}}$$

1.2 三维坐标绕轴旋转

三维向量绕向量(u_x,u_y,u_z)逆时针转t角度的矩阵

$$R=\begin{bmatrix}u_x^2+cost(1-u_x^2) & u_xu_y(1-cost)+u_zsint & u_zu_x(1-cost)+u_ysint\\u_xu_y(1-cost)+u_zsint & u_y^2+cost(1-u_y^2) & u_yu_z(1-cost)+u_xsint\\u_zu_x(1-cost)+u_ysint & u_yu_z(1-cost)+u_xsint & u_z^2+cost(1-u_z^2)\end{bmatrix}$$

1.3 其他

1.3.1 圆及其性质

过圆$(x-x_0)^2+(y-y_0)^2=r^2$上点$P(x_1,y_1)$的

切线$(x-x_0)(x_1-x_0)+(y-y_0)(y_1-y_0)=r^2$

法线$(y-y_1)(x_1-x_0)=(x-x_1)(y_1-y_0)$

1.3.2 外心

$$2R=\frac{abc}{2S}=\frac{a}{\sin A}=\frac{b}{\sin B}=\frac{c}{\sin C}$$

$$x=x_a-\frac{\begin{vmatrix}y_b-y_a & c^2\\y_c-y_a & b^2\end{vmatrix}}{2\begin{vmatrix}x_b-x_a & y_b-y_a\\x_c-x_a & y_c-y_a\end{vmatrix}}=\frac{\begin{vmatrix}x_a^2+y_a^2 & y_a & 1\\x_b^2+y_b^2 & y_b & 1\\x_c^2+y_c^2 & y_c & 1\end{vmatrix}}{2\begin{vmatrix}x_a & y_a & 1\\x_b & y_b & 1\\x_c & y_c & 1\end{vmatrix}}$$

$$y=y_a-\frac{\begin{vmatrix}x_b-x_a & c^2\\x_c-x_a & b^2\end{vmatrix}}{2\begin{vmatrix}x_b-x_a & y_b-y_a\\x_c-x_a & y_c-y_a\end{vmatrix}}=\frac{\begin{vmatrix}x_a & x_a^2+y_a^2 & 1\\x_b & x_b^2+y_b^2 & 1\\x_c & x_c^2+y_c^2 & 1\end{vmatrix}}{2\begin{vmatrix}x_a & y_a & 1\\x_b & y_b & 1\\x_c & y_c & 1\end{vmatrix}}$$

1.3.3 垂心

$$x=\frac{\begin{vmatrix}y_a & x_bx_c+y_a^2 & 1\\ y_b & x_ax_c+y_b^2 & 1\\ y_c & x_bx_a+y_c^2 & 1\end{vmatrix}}{2\begin{vmatrix}x_a & y_a & 1\\ x_b & y_b & 1\\ x_c & y_c & 1\end{vmatrix}},y=\frac{\begin{vmatrix}x_a^2+y_by_c & x_a & 1\\ x_b^2+y_ay_c & x_b & 1\\ x_c^2+y_by_a & x_c & 1\end{vmatrix}}{2\begin{vmatrix}x_a & y_a & 1\\ x_b & y_b & 1\\ x_c & y_c & 1\end{vmatrix}}$$

设 O 是△ABC 的外心，则有 H=A+B+C-2O

1.3.4 四面体由边长求体积

$$288V^2=\begin{vmatrix}0 & 1 & 1 & 1 & 1\\ 1 & 0 & d_{12}^2 & d_{13}^2 & d_{14}^2\\ 1 & d_{21}^2 & 0 & d_{23}^2 & d_{24}^2\\ 1 & d_{31}^2 & d_{32}^2 & 0 & d_{34}^2\\ 1 & d_{41}^2 & d_{42}^2 & d_{43}^2 & 0\end{vmatrix}$$

2 Chapter 2 Algebra 代数

2.1 求和

2.1.1 常用求和公式

$$\sum_{i=1}^{n} i^2=\frac{1}{6}n(n+1)(2n+1)$$

$$\sum_{i=1}^{n} i^3=\frac{1}{4}n^2(n+1)^2$$

$$\sum_{i=1}^{n} i^4=\frac{1}{30}n(n+1)(2n+1)(3n^2+3n-1)$$

$$\sum_{i=1}^{n} i^5=\frac{1}{12}n^2(n+1)^2(2n^2+2n-1)$$

$$\sum_{i=1}^{n}\frac{1}{i(i+1)(i+2)}=\frac{1}{4}-\frac{1}{2(n+1)(n+2)}$$

$$\sum_{i=1}^{n}\frac{1}{i(i+1)(i+2)(i+3)}=\frac{1}{18}-\frac{1}{3(n+1)(n+2)(n+3)}$$

3 Chapter 3 Combinatorial Mathematics 组合数学

3.1 错位排列

$1,2,\dots,n$的错位排序是一个排列$i_1,i_2,\dots,i_n$ 使得$i\neq k$记D_n为$1,2,\dots,n$的错位排序个数，则

$$D_n=n!\,(1-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+\cdots+(-1)^n\frac{1}{n!})$$

$$D_n=nD_{n-1}+(-1)^n$$

$$\lim_{\mathrm{n}\to\infty}\frac{D_n}{n!}=e^{-1}$$

n的排列恰有k个位置正确的个数：$\mathrm{C}_n^{\mathrm{k}}D_{n-k}$

3.2 Polya 定理

设 G 是p个对象的一个置换群，用k种颜色涂染这p个对象，若一种染色方案在群 G 的作用下变为另一种染色方案，则这两种方案当作是同一种方案，这样不同的染色方案数为：

$$l=\frac{1}{|G|}\sum_{f\in G}k^{c(f)}$$

3.3 Catalan 数及其拓展

m个 0，n个 $1(m\geq n)$的数列，其任意前k项满足 0 的个数多于 1 的个数。这样的数列个数为：$\mathrm{C}_{m+n}^{m}-\mathrm{C}_{m+n}^{m+1}$

特别的当$m=n$ 时即为$Catalan$ 数

$$C_n=\frac{C_{2n}^{n}}{n+1}$$

3.4 无向图生成树个数

设 G 是无向图， A 是其度数对角阵， B 是其邻接矩阵。则 G 的生成树个数是 A - B 的任一 n - 1 阶子式的行列式。

4 Chapter 4 Other 其他

4.1 皮克公式

设 A 是格点多边形面积， B 是多边形边上的点个数，I 是多边形内的点的个数，则：

$$\mathrm{A}=\mathrm{I}+\frac{1}{2}B-1$$

4.2 欧拉公式

1.对凸多面体 $\mathrm{V}-\mathrm{E}+\mathrm{F}=2$

2.对平面图 $|\mathrm{F}|=|\mathrm{E}|-|\mathrm{V}|+$连通块个数$+1$

4.3 anti-NIM 定理

走完最后一步者输，先手必胜当且仅当：

1. 所有堆的石子数都为 1 且游戏的 SG 值为 0;
2. 有些堆的石子数大于 1 且游戏的 SG 值不为 0.

4.4 素数表

100003,100019,200003,200017,300007,300017,400009,500009,600011,700001,800011,900001,900019,1000003,2000003,3000017,4100011,5000011,8000009,9000011,10000019,20000003,50000017,50100007,100000007,100200011,200100007,250000019,300000007

Part II
Standard Code Library

1 Chapter 1 Graph Theory 图论

1.1 求割点、桥、双连通分量

- 若 u 是搜索树根且具有超过 1 个的搜索子树，则 u 是割点。
- 若 u 不是搜索树根且存在搜索树枝边(u,v)满足 low(v)>=dfn(u)，则 u 是割点。
- 桥：如果搜索树枝边(u,v) 满足 dfn(u) < low(v)，则(u,v) 是桥，同一个双联通分量中的 low 值相同。

```
int dfn[MXN], low[MXN];
int time;
bool cut[MXN];
void dfs(int u, int fa, int root){
	dfn[u] = low[u] = ++time;
	int tot = 0;
	for (int i = 1; i <= n; ++i) if (g[u][i]) {
		if (!dfn[i]) {
			dfs(i, u, root);
			if ((u==root&&++tot>1)||(u!=root&& low[i] >= dfn[u])) cut[u] = true;
			low[u] = min(low[u], low[i]);
		} else if (i != fa) low[u] = min(low[u], dfn[i]);
	}
}
```

1.2 欧拉回路

- Edge :边，邻接表，id 表示节点 t 对应边的位置
- a :Ans

```
struct node{
	int t,id;
};
vector < node > Edge[MaxN];
void Ins(int x,int y){
	node o;
	o.t=y,o.id=Edge[y].size();
	Edge[x].push_back(o);
	o.t=x,o.id=Edge[x].size()-1;
	Edge[y].push_back(o);
}
void Del(int x,int id){
	Edge[x][id]=Edge[x][Edge[x].size()-1];
	Edge[x].pop_back();
	node o=Edge[x][id];
	Edge[o.t][o.id].id=id;
}
void DFS(int x){
	if (Edge[x].size()==0){
		a[tot++]=x;
		return;
	}
	while (Edge[x].size()){
		node o=Edge[x][Edge[x].size()-1];
```

```
                Del(x,Edge[x].size()-1);
                Del(o.t,o.id);
                DFS(o.t);
        }
        a[tot++]=x;
}
```

1.3 Tarjan 强连通分量

- Vec　　：为邻接表，存边;
- stop, cnt, scnt 初始化为 0;
- pre 初始化为-1;

```
vector <int> vec[V];
int id[V],pre[V],low[V],s[V],stop,cnt,scnt;
void tarjan(int v, int n){ // vertex: 0 ~ n-1
   int t,minc=low[v]=pre[v]=cnt++;
   vector <int> ::iterator pv;
   s[stop++] = v;
   for (pv = vec[v].begin(); pv != vec[v].end(); ++pv) {
       if (-1 == pre[*pv]) tarjan(*pv, n);
       if (low[*pv] < minc) minc=low[*pv];
   }
   if(minc < low[v]){
       low[v]=minc;
       return;
   }
   do{
       id[t=s[--stop]]=scnt;
       low[t]=n;
   }while(t!=v);
   ++scnt; // 强连通分量的个数
}
```

1.4 带花树（任意图匹配）

- g[i][j]存放关系图　：i,j 是否有边,match[i]存放 i 所匹配的点
- solve 返回最大匹配值,匹配出的边可以查看 match 数组

```
#define MAXE 250*250*2
#define MAXN 250
#define SET(a,b) memset(a,b,sizeof(a))deque<int> Q;
bool g[MAXN][MAXN],inque[MAXN],inblossom[MAXN];int match[MAXN],pre[MAXN],base[MAXN];
//找公共祖先
int findancestor(int u,int v){
    bool inpath[MAXN]={false};
    while(1)
     {
             u=base[u];
             inpath[u]=true;
             if (match[u]==-1) break;
             u=pre[match[u]];
     }
```

```
    while(1)
    {
              v=base[v];
              if (inpath[v]) return  v;
              v=pre[match[v]];
    }
}
//压缩花
void reset(int u,int anc){
     while(u!=anc)
     {
              int v=match[u];
              inblossom[base[u]]=1;
              inblossom[base[v]]=1;
              v=pre[v];
              if(base[v]!=anc)pre[v]=match[u];
              u=v;
     }
}
void contract(int u,int v,int n){
     int anc=findancestor(u,v);
     SET(inblossom,0);
     reset(u,anc);reset(v,anc);
     if(base[u]!=anc)pre[u]=v;
     if(base[v]!=anc)pre[v]=u;
     for(int i=1;i<=n;i++)
     if(inblossom[base[i]])
     {
                  base[i]=anc;
                  if(!inque[i])
                  {
                                  Q.push_back(i);
                                  inque[i]=1;
                  }
     }
}
bool dfs(int S,int n){
     for(int i=0;i<=n;i++)pre[i]=-1,inque[i]=0,base[i]=i;
     Q.clear();Q.push_back(S);inque[S]=1;
     while(!Q.empty())
     {
              int u=Q.front();Q.pop_front();
              for(int v=1;v<=n;v++)
              {
                  if(g[u][v]&&base[v]!=base[u]&&match[u]!=v)
                  {
                      if(v==S||(match[v]!=-1&&pre[match[v]]!=-1))contract(u,v,n);
```

```
                        else if(pre[v]==-1)
                        {
                            pre[v]=u;
                            if(match[v]!=-1)Q.push_back(match[v]),inque[match[v]]=1;
                             else
                             {
                                 u=v;
                                 while(u!=-1)
                                 {
                                     v=pre[u];
                                     int w=match[v];
                                     match[u]=v;
                                     match[v]=u;
                                     u=w;
                                 }
                              return true;
                             }
                        }
                    }
                }
        }
        return  false;
}
int solve(int n){
    SET(match,-1);
    int ans=0;
    for(int i=1;i<=n;i++)
        if(match[i]==-1&&dfs(i,n))
            ans++;
    return ans;
}
```

1.5 2-SAT

- 添加限制边的时候请注意 **对称** 性质
- dfn[id[i]]为最后 i 点的颜色

```
int tot, cnt,n;
int list[MXN];
int id[MXN], dfn[MXN];
int contain[MXN];
struct arc {
   int v;
   arc *next;
   arc() {}
   arc(int v, arc *next) : v(v), next(next) {}
} *adj[MXN], *rev[MXN], *scc[MXN], Mem[MXM << 1];
int Memcnt;
inline void addedge(int u, int v){
   adj[u] = &(Mem[Memcnt++] = arc(v, adj[u]));
```

```
    rev[v] = &(Mem[Memcnt++] = arc(u, rev[v]));
}
void dfs1(int u){
    id[u] = -1;
    for (arc *p = adj[u]; p; p = p->next)
    if (id[p->v] != -1) dfs1(p->v);
    list[tot--] = u;
}
void dfs2(int u, int c){
    id[u] = c;
    for (arc *p = rev[u]; p; p = p->next)
    if (id[p->v] == -1) dfs2(p->v, c);
}
void dfs3(int u){
    dfn[u] = -1;
    for (arc *p = scc[u]; p; p = p->next)
    if (dfn[p->v] != -1) dfs3(p->v);
    list[--tot] = u;
}
void dfs4(int u){
    dfn[u] = 2;
    for (arc *p = scc[u]; p; p = p->next)
    if (dfn[p->v] == -1) dfs4(p->v);
}
bool test(){
    cnt = 0;
    tot = n * 2;
    for (int i = 1; i <= n * 2; ++i)
    if (id[i] != -1)
    dfs1(i);
    for (int i = 1; i <= n * 2; ++i)
    if (id[list[i]] == -1)
    dfs2(list[i], cnt++);
    for (int i = 1; i <= n; ++i)
    if (id[i] == id[n + i])
    return  false;
    memset(scc, 0, sizeof(scc));
    memset(contain, -1, sizeof(contain));
    for (int i = 1; i <= n * 2; ++i)
    for (arc *p = adj[i]; p; p = p->next)
    if (id[i] != id[p->v])
    scc[id[p->v]] = &(Mem[Memcnt++] = arc(id[i], scc[id[p->v]]));
    for (int i = 1; i <= n * 2; ++i)
    if (contain[id[i]] == -1)
    contain[id[i]] = i;
    tot = cnt;
    for (int i = 0; i < cnt; ++i)
```

```
    if (dfn[i] != -1) dfs3(i);
    for (int i = 0; i < cnt; ++i)
    if (dfn[list[i]] == -1) {
        int a = contain[list[i]], b = a <= n ? a + n : a - n;
        dfn[list[i]] = 1;
        if (dfn[id[b]] == -1) dfs4(id[b]);
    }
    return true;
}
```

1.6 2-sat(喻展版)

```
void dfs(long x){
        long i,r;
        f[x]=t[x]=w++;
        st[++l]=x;
        b[x]=1;
        for (i=hd[x];i>0;i=e[i].next){
                r=e[i].r;
                if (t[r]<0){
                        dfs(r);
                        if (flag) return;
                        if (f[r]<f[x]) f[x]=f[r];
                }
                else if (b[r] && t[r]<f[x]) f[x]=t[r];
        }
        if (f[x]==t[x]){
                while (st[l]!=x){
                        if (st[l]==x+n || st[l]+n==x)      //i 和 i+n 是对应点{
                                flag=1;
                                return;
                        }
                        s[st[l]]=x;
                        b[st[l--]]=0;
                }
                b[st[l--]]=0;
        }
}
bool two_sat(){
        flag=w=l=0;
        memset(t,-1,sizeof(t));
        memset(b,0,sizeof(b));
        for (i=0;i<n+n;i++) s[i]=i;
        for (i=0;i<n+n;i++) if (!flag && t[i]<0) dfs(i);
        if (flag) return 0;
        return 1;
}
```

1.7 最小树形图

- $O(EV)$

- 有向图的最小生成树
- 注意重边的情况。
- 不固定根的最小树形图：新加一个点，和每个点连权相同的边，这个权大于原图所有边的权值之和，这样这个图固定根的最小树形图和原图不固定根的最小树形图就是对应的了。

```
int g[MXN][MXN];
int pre[MXN];
bool vis[MXN], del[MXN];

inline void update(int &x, int y){
   if (x > y) x = y;
}
int minTreeGraph(int root){
   memset(del, 0, sizeof del);
   int ret = 0;
   while (true) {
       int check = 1, k;
       for (int i = 0; i < n; ++i) if (!del[i] && i != root) {
           pre[i] = i;
           g[i][i] = INF;
           for (int j = 0; j < n; ++j) if (!del[j])
           if (g[j][i] < g[pre[i]][i]) pre[i] = j;
       }
       for (int i = 0; i < n; ++i) if (!del[i] && i != root) {
           memset(vis, 0, sizeof vis);
           for (k = i; k != root && !vis[k]; k = pre[k])
           vis[k] = true;
           check = 0;
           int tmp = k;
           ret += g[pre[k]][k];
           for (k = pre[k]; k != tmp; k = pre[k]) {
               ret += g[pre[k]][k];
               del[node] = true;
           }
           for (int j = 0; j < n; ++j) if (!del[j])
           if (g[j][tmp] != INF) g[j][tmp] -= g[pre[tmp]][tmp];
           for (k = pre[tmp]; k != tmp; k = pre[k])
           for (int j = 0; j < n; ++j) if (!del[j]) {
               update(g[tmp][j], g[k][j]);
               if (g[j][k] != INF) update(g[j][tmp], g[j][k] - g[pre[k]][k]);
           }
           break;
       }
       if (check) {
           for (int i = 0; i < n; ++i)
           if (!del[i] && i != root) ret += g[pre[i]][i];
           break;
       }
```

```
    }
    return ret;
}
```

- $O(E\log V)$
- ans 是有向图最小生成树的代价，总是以 1 号节点为 root，节点标号 1~n
- lst[]是取的边的编号

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <string>
#include <cmath>
#include <vector>
using namespace std;
const int N = 100000 + 10;
const int M = 100000 + 10;
struct Cost;
vector <Cost*> csts;
struct Cost {
        int c;
        Cost*a, *b;
        int id;
        int nUsed;
        bool operator<(const Cost&o) const {
                return c < o.c;
        }
        Cost(int c, int id) {
                this->c = c;
                this->id = id;
                a = b = 0;
                nUsed = 0;
                csts.push_back(this);
        }
        Cost(Cost* a, Cost* b) {
                this->a = a;
                this->b = b;
                id = -1;
                c = a->c - b->c;
                nUsed = 0;
                csts.push_back(this);
        }
        void  push() {
                if (id == -1) {
                        a->nUsed += nUsed;
                        b->nUsed -= nUsed;
                }
        }
```

```
        void  useIt() {
                ++nUsed;
        }
};
struct edge {
        int u, v, id;
        Cost* cost;
        edge() {
        }
        edge(int u, int v, int c, int id) :
        u(u), v(v) {
                cost = new Cost(c, id);
        }
} e[M];
int pre[N], hash1[N], vis[N];
Cost* In[N];
bool better(Cost* a, Cost* b) {
        if (a == 0 || b == 0)
        return b == 0;
        return a->c < b->c;
}
int Directed_MST(int root, int n, int m) {
        int ret = 0;
        while (true) {
                for (int i = 0; i < n; i++)
                In[i] = 0;
                for (int i = 0; i < m; i++) {
                        int u = e[i].u;
                        int v = e[i].v;
                        if (better(e[i].cost, In[v]) && u != v) {
                                pre[v] = u;
                                In[v] = e[i].cost;
                        }
                }
                for (int i = 0; i < n; i++) {
                        if (i == root)
                        continue;
                        if (In[i] == 0)
                        return -1;
                }
                int cntnode = 0;
                memset(hash1, -1, sizeof(hash1));
                memset(vis, -1, sizeof(vis));
                for (int i = 0; i < n; i++)
                if (i != root) {
                        ret += In[i]->c;
                        In[i]->useIt();
```

```
                        int v = i;
                        while (vis[v] != i && hash1[v] == -1 && v != root) {
                                vis[v] = i;
                                v = pre[v];
                        }
                        if (v != root && hash1[v] == -1) {
                                for (int u = pre[v]; u != v; u = pre[u])
                                hash1[u] = cntnode;
                                hash1[v] = cntnode++;
                        }
                }
                if (cntnode == 0)
                break;
                for (int i = 0; i < n; i++)
                if (hash1[i] == -1)
                hash1[i] = cntnode++;
                for (int i = 0; i < m; i++){
                        int v = e[i].v;
                        e[i].u = hash1[e[i].u];
                        e[i].v = hash1[e[i].v];
                        if (e[i].u != e[i].v) {
                                e[i].cost = new Cost(e[i].cost, In[v]);
                        }
                }
                n = cntnode;
                root = hash1[root];
        }
        return ret;
}
int n, m;
int main() {
        scanf("%d %d", &n, &m);
        int mm = 0;
        for (int i = 0; i < m; i++) {
                int  a, b, c;
                scanf("%d%d%d", &a, &b, &c);
                a--, b--;
                e[mm++] = edge(a, b, c, i + 1);
        }
        int ans = Directed_MST(0, n, mm);
        if (ans == -1) puts("-1");else {
                cout << ans << endl;
                for (int i = csts.size() - 1; i >= 0; --i) {
                        csts[i]->push();
                }
                vector <int> lst;
                for (int i = 0; i < csts.size(); ++i) {
```

```
                        Cost*c = csts[i];
                        if (c->id != -1 && c->c > 0 && c->nUsed > 0) {
                                lst.push_back(c->id);
                        }
                }
                sort(lst.begin(), lst.end());
                for (int i = 0; i < lst.size(); ++i) {
                        cout << lst[i] << " ";
                }
                cout << endl;
        }
        return 0;
}
```

1.8 最大流 SAP

```
struct arc {
   int v, c;
   arc *next, *o;
   arc() {}
   arc(int v, int c, arc *next) : v(v), c(c), next(next) {}
};
inline void addedge(arc **a, int u, int v, int c){
   a[u] = &(mem[memCnt++] = arc(v, c, a[u]));
   a[v] = &(mem[memCnt++] = arc(u, 0, a[v]));
   a[u]->o = a[v];
   a[v]->o = a[u];
}
int aug(int u, int m){
   if (u == t) return m;
   int mind = N;
   for (arc *&p = cur[u]; p; p = p->next)
   if (p->c && d[p->v] + 1 == d[u]) {
       int drt = aug(p->v, min(m, p->c));
       if (d[s] == N) return 0;
       if (drt > 0) {
       p->c -= drt; p->o->c += drt;
       return  drt;
       }
   }
   for (arc *p = cur[u] = a[u]; p; p = p->next)
   if (p->c) mind = min(mind, d[p->v] + 1);
   if (!--cnt[d[u]]) d[s] = N; else  ++cnt[d[u] = mind];
   return  0;
}
int main(){
   memmove(cur, a, sizeof  a);
   cnt[0] = N;
   while (d[s] < N) ans += aug(s, INF);
```

```
}
```

1.9 最大流 DINIC

```
struct Edge
{
       long l,r,c,bk,next;
}e[maxm]; //l=边头, r=边尾, c=流量, bk=对应边;
long n,m,ans,hd[maxn];
void  AddEdge(long l,long r,long c)
{
     e[++m].l=l;
     e[m].r=r;
     e[m].c=c;
     e[m].bk=m+1;
     e[m].next=hd[l];
     hd[l]=m;
     e[++m].r=l;
     e[m].l=r;
     e[m].c=0;          //无向图此处改为 e[m].c=c;
     e[m].bk=m-1;
     e[m].next=hd[r];
     hd[r]=m;
}
void  Dinic(long s,long t)
{
     long q[maxn],z[maxn],f[maxn],l,r,x; //q:队列/栈  z:当前弧  hd:链表头  f:深度标记
     long i;
     while (1)
     {
           memset(f,-1,sizeof(f));
           f[s]=1;
           q[l=0]=s;
           r=1;
           while (l<r)
           {
                 x=q[l++];
                 for (i=hd[x];i>0;i=e[i].next)
                     if (e[i].c>0 && f[e[i].r]<0)
                     {
                        f[e[i].r]=f[x]+1;
                        q[r++]=e[i].r;
                     }
           }
           if (f[t]<0) break;
           memcpy(z,hd,sizeof(z));
           e[0].r=s;
           q[r=0]=0;
           while (r>=0)
```

```
            {
                  x=e[q[r]].r;
                  if (x!=t)
                     if (z[x]>0)
                     {
                        for (;z[x]>0;z[x]=e[z[x]].next) if (e[z[x]].c>0 && f[x]+1==f[e[z[x]].r]) break;
                        if (z[x]==0) goto  re;
                        q[++r]=z[x];
                        z[x]=e[z[x]].next;
                     }
                     else
                     {
                        re:
                        r--;
                        f[x]=-1;
                     }
                  else
                  {
                     x=0x7fffffff;
                     for (i=1;i<=r;i++) if (x>e[q[i]].c) x=e[q[i]].c;
                     for (i=1;i<=r;i++)
                     {
                         e[q[i]].c-=x;
                         e[e[q[i]].bk].c+=x;
                     }
                     ans+=x;
                     for (r=0;e[q[r+1]].c>0;r++);
                  }
            }
      }
}
```

1.10 ZKW 最小费用流

```
long augment(long x,long w)
{
     if (x==t) return flow+=w,cost+=d[s]*w,w;
     b[x]=1;
     for (long i=hd[x];i;i=e[i].next)
         if (e[i].c && !b[e[i].r] && d[e[i].r]+e[i].b==d[x])
            if (dt=augment(e[i].r,w<e[i].c ? w:e[i].c)) return e[i].c-=dt,e[e[i].bk].c+=dt,dt;
     return 0;
}
bool modlabel()
{
     dt=0x7fffffff;
     for (x=0;x<n;x++) if (b[x])
         for (i=hd[x];i;i=e[i].next)
```

```
            if (e[i].c && !b[e[i].r] && dt>e[i].b-d[x]+d[e[i].r]) dt=e[i].b-d[x]+d[e[i].r];
    if (dt==0x7fffffff) return 0;
    for (i=0;i<n;i++) if (b[i]) d[i]+=dt;
    return 1;
}
    memset(d,0,sizeof(d));//spfa();
    do do memset(b,0,sizeof(b)); while (augment(s,0x7fffffff)); while (modlabel());
```

1.11 无向图最小割 Stoer-Wangner

- 1.min = MAXINT，固定一个顶点 P
- 2.从点 P 用类似 Prim 的算法扩展出“最大生成树”，记录最后扩展的顶点和最后扩展的边
- 3.计算最后扩展到的顶点的切割值（即与此顶点相连的所有边权和），若比 min 小更新 min
- 4.合并最后扩展的那条边的两个端点为一个顶点（当然他们的边也要合并）
- 5.转到 2，合并 n - 1 次后结束
- 6.min 即为所求，输出 min

Prim 算法本身复杂度是$O(n^2)$，合并$n-1$ 次，算法复杂度即为$O(n^3)$ ，如果在 Prim 中加堆优化，复杂度会降为 $O(n^2 \log n)$

1.12 无向图任意点对最小割

```
struct node{
        int v,c,po,pre;
};
struct Ege{
        int U,V,C;
};
node edrec[MaxM];
int N,M,ed[MaxM],Belong[MaxN],Level[MaxN],F[MaxN][MaxN],flin[MaxN],flou[MaxN],ednum,MaxFlow;
Ege Edge[MaxM];
void AddEdge( int u , int v , int c ){
        ednum++;
        edrec[ednum].v = v;
        edrec[ednum].c = c;
        edrec[ednum].po = ednum+1;
        edrec[ednum].pre = ed[u];
        ed[u] = ednum;
        ednum++;
        edrec[ednum].v = u;
        edrec[ednum].c = c;
        edrec[ednum].po = ednum-1;
        edrec[ednum].pre = ed[v];
        ed[v] = ednum;
}
void Init(){
        scanf("%d%d" , &N , &M);
        memset(ed , 0 , sizeof(ed));
        ednum = 0;
        for(int i=1;i<=M;i++){
                scanf("%d%d%d" , &Edge[i].U , &Edge[i].V , &Edge[i].C);
                AddEdge( Edge[i].U , Edge[i].V , Edge[i].C );
```

```
        }
}
void Work(){
        for(int i=1;i<=N;i++) Belong[i] = 1;
        for(int i=1;i<=N;i++)
        for(int j=1;j<=N;j++) F[i][j] = 2000000000;
        for(int t=1;t<=N-1;t++){
                int Source = -1 , Sink = -1;
                for(int i=1;i<=N-1;i++)
                for(int j=i+1;j<=N;j++) if( Belong[i]==Belong[j] ){
                        Source = i; Sink = j;
                }
                int Flow = MaxFlow( Source , Sink , N+(t-1)*2 );//求 Source 到 Sink, N+(t-1)*2 个节点的最大流
                int Id = Belong[Source];
                for(int i=1;i<=N;i++) if( Belong[i]==Id && Level[i]==0 ) Belong[i] = t+1;
                memset(flin , 0 , sizeof(flin));
                memset(flou , 0 , sizeof(flou));
                int Num = 0;
                for(int i=1;i<=M;i++) if( (Level[Edge[i].U]!=0)+(Level[Edge[i].V]!=0) == 1 ){
                        if( Level[Edge[i].U] != 0 ) flou[Edge[i].U] += Edge[i].C;
                        else flou[Edge[i].V] += Edge[i].C;
                        if( Level[Edge[i].V] == 0 ) flin[Edge[i].V] += Edge[i].C;
                        else flin[Edge[i].U] += Edge[i].C;
                }
                else Edge[++Num] = Edge[i];
                M = Num;
                int a = N+2*t-1 , b = N+2*t;
                Edge[++M].U = a;
                Edge[ M].V = b;
                Edge[ M].C = Flow;
                for(int i=1;i<=N+2*t;i++){
                        if( flou[i] > 0 ){
                                Edge[++M].U = i; Edge[M].V = a; Edge[M].C = flou[i];
                        }
                        if( flin[i] > 0 ){
                                Edge[++M].U = b; Edge[M].V = i; Edge[M].C = flin[i];
                        }
                }
                memset(ed , 0 , sizeof(ed));
                ednum = 0;
                for(int i=1;i<=M;i++) AddEdge( Edge[i].U , Edge[i].V , Edge[i].C );
                for(int i=1;i<=N;i++) if( Level[i] > 0 )
                for(int j=1;j<=N;j++) if( Level[j] == 0 ) F[i][j] = min( F[i][j] , Flow );
        }
}
void Output(){
```

```
        for(int i=1;i<=N;i++)
        for(int j=1;j<=N;j++) F[i][j] = min( F[i][j] , F[j][i] );
        int Q;
        scanf("%d" , &Q);
        for(int t=1;t<=Q;t++){
                int Range;
                scanf("%d" , &Range);
                int Res =0;
                for(int i=1;i<=N-1;i++)
                for(int j=i+1;j<=N;j++) Res += F[i][j]<=Range;
                printf("%d\n" , Res);
        }
}
```

1.13 匈牙利算法(邻接表)

- Map ：为邻接表，存边;
- Match ：存储右边点匹配的对象;
- 左边点 ：1~N

```
int DFS(int x){
   for (int i=0;i<Map[x].size();i++)
   if (!vis[Map[x][i]]){
       vis[Map[x][i]]=1;
       if (Match[Map[x][i]]==-1 || DFS(Match[Map[x][i]])){
           Match[Map[x][i]]=x;
           return 1;
       }
   }
   return 0;
}
int Max Match (){
   int ret=0;
   memset(Match,-1,sizeof(Match));
   for (int i=1;i<=N;i++){
       memset(vis,0,sizeof(vis));
       if (DFS(i)) ret++;
   }
   return ret;
}
```

1.14 有上下限的最小（最大）流

- 上限为 c，下限为 l,
- 添加一个源 s 和汇 t，对于每个下限容量 l 不为 0 的边(u, v),
- 将其下限去掉, 上限改为 c-l, 增加两条边(u, t), (s, v),
- 容量均为 l. 原网络存在循环流等价于新网络最大流是满流.

1.15 稳定婚姻问题

- 有 n 个男性和 n 个女性，每个男性对所有 n 个女性有他自己的喜欢程度排序，每个女性对所有的 n 个男性也有她自己的喜欢程度排序。一个婚姻是这 n 男 n 女的一个匹配，这个婚姻被称为稳定的，当且仅当不村子两对夫妇(A,B）和(C,D)，满足相对于 B，A 更喜欢 D；同时相对于 C，D 更喜欢 A。

1. 设置所有人味未婚状态。转 2.

2. 如果村子一个未婚男性 A，转 3；否则算法结束。
3. 取出 A“最喜欢”的女性，设为 B。转 4.
4. 如果 B 未婚，将(A,B)加入匹配（结婚），并转 2；否则转 5.
5. 设 B 目前的丈夫是 A’，如果相对于 A’，B 更喜欢 A，转 6；否则转 7.
6. 将（A’,B）拆散，同时(A,B)加入匹配，并将 B 从 A’的列表中删除。转 2.
7. 将 B 从 A 的列表中删除，转 2.

1.16 最大权完美匹配 KM

- 复杂度 $O(N^3)$

```
int w[MXN][MXN];
int lx[MXN], ly[MXN];
int slack[MXN];
int matchy[MXN];
bool visx[MXN], visy[MXN];

bool find(int u){
   visx[u] = true;
   for (int i = 0; i < n; ++i)
   if (!visy[i]) {
       if (lx[u] + ly[i] == w[u][i]) {
           visy[i] = true;
           if (matchy[i] == -1 || find(matchy[i])) {
               matchy[i] = u;
               return  true;
           }
       } else  slack[i] = min(slack[i], lx[u] + ly[i] - w[u][i]);
   }
   return  false;
}
int km(int n){
   memset(matchy, -1, sizeof matchy);
   memset(lx, 0, sizeof lx);
   memset(ly, 0, sizeof ly);
   for (int i = 0; i < n; ++i)
   for (int j = 0; j < n; ++j)
   lx[i] = max(lx[i], w[i][j]);
   for (int k = 0; k < n; ++k)
   while (true) {
       memset(slack, 0x3f, sizeof slack);
       memset(visx, 0, sizeof visx);
       memset(visy, 0, sizeof visy);
       if (find(k)) break;
       int derta = 0x3f3f3f3f;
       for (int i = 0; i < n; ++i)
       if (!visy[i]) derta = min(derta, slack[i]);
       for (int i = 0; i < n; ++i)
       if (visx[i]) lx[i] -= derta;
       for (int i = 0; i < n; ++i)
```

```
        if (visy[i]) ly[i] += derta;
    }
    int ret = 0;
    for (int i = 0; i < n; ++i)
    ret += w[matchy[i]][i];
    return ret;
}
```

2 Chapter 2 Number Theory 数论

2.1 拓展 GCD

- ex_gcd(a, b) = ax + by

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

2.2 平方剩余

- $x^2 \equiv a \mod n, n$是质数
- $\gcd(a,b) \equiv 1 \mod n$否则无解返回$-1$
- $power(a,x,n) = a^x \mod n$

```
int modSqrt(int a, int n){
    int b, k, i, x;
    if (n == 2) return a % n;
    if (power(a, (n - 1) / 2, n) == 1) {
        if (n % 4 == 3) x = power(a, (n + 1) / 4, n);
        else {
            for (b = 1; power(b, (n - 1) / 2, n) == 1; ++b);
            i = (n - 1) / 2;
            k = 0;
            do {
                i /= 2; k /= 2;
                if ((power(a, i, n) * (long long) power(b, k, n) + 1) % n == 0)
                k += (n - 1) / 2;
            } while (i % 2 == 0);
            x = power(a, (i + 1) / 2, n) * (long long) power(b, k / 2, n) % n;
        }
        if (x * 2 > n) x = n - x;
        return x;
    }
    return -1;
}
```

2.3 质数判断 Pollard-Rho 和 Miller-Rabin 算法

- Miller 过程判断一个数是否为质数，rho 过程返回某个因子

```
long long ans;
long long gcd(long long a, long long b){
    if (b == 0) return a;
    return gcd(b, a % b);
}
long long mulMod(long long a, long long b, long long p){
    long long y = (long long) (1.0 * a * b / p + 0.5);
    long long ret = a * b - y * p;
    if (ret < 0) ret += p;
    return ret;
}
long long powMod(long long x, long long k, long long p){
    long long ret = 1;
    while (k > 0) {
        if (k & 1) ret = mulMod(ret, x, p);
        x = mulMod(x, x, p);
        k >>= 1;
    }
    return ret;
}
bool witness(int a, long long n){
    long long u = n - 1;
    int t = 0;
    for (; u % 2 == 0; u /= 2) ++t;
    long long x = powMod(a, u, n);
    for (int i = 0; i < t; ++i) {
        long long nx = mulMod(x, x, n);
        if (nx == 1 && x != 1 && x != n - 1) return true;
        x = nx;
    }
    return x != 1;
}
bool millerRabin(long long n){ // n >= 2; {2, 3, 5, 7} => 10^9; {2, 3} => 10^6
    static const int prime[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    for (int i = 0; i < 10; ++i)
    if (n == prime[i]) return true;
    for (int i = 0; i < 10; ++i)
    if (witness(prime[i], n)) return false;
    return true;
}
long long pollardRho(long long n){
    if (n % 2 == 0) return 2;
    while (true) {
        int i = 1, k = 2;
        long long x = (long long) rand() * rand() % n, y = x;
        while (true) {
            ++i;
```

```
            x = mulMod(x, x, n);
            if (x == 0) x = n - 1; else --x;
            long long d = gcd(abs(x - y), n);
            if (d != 1 && d != n) return d;
            if (d == n) break;
            if (i == k) { y = x; k <<= 1; }
        }
    }
}

void getMinFactor(long long x)//求 x 最小的质因数
{
    if (millerRabin(x)) {
        ans = min(ans, x);
        return;
    }
    long long d = pollardRho(x);
    getMinFactor(d); getMinFactor(x / d);
}
```

2.4 欧拉函数

- 递推求欧拉函数

```
for (i = 1; i <= MaxN; i++) phi[i] = i;
for (i = 2; i <= MaxN; i += 2) phi[i] /= 2;
for (i = 3; i <= MaxN; i += 2)
if(phi[i] == i) {
    for (j = i; j <= maxn; j += i)
    phi[j] = phi[j] / i * (i - 1);
}
```

- 公式法

```
long long euler(unsigned x){
    long long i, res=x;
    for (i = 2; i*i<=x; i++)
    if(x%i==0) {
        res = res / i * (i - 1);
        while (x%i == 0) x /= i;
    }
    if (x > 1) res = res / x * (x - 1);
    return res;
}
```

2.5 原根 $x^k \equiv a\ mod\ p$

- $x^k = a\ (mod\ p)$ 设 g 为原根，$x = g^i,\ a = g^j$, 则$g^{ik} = g^j\ (mod\ p)$ $=> ik = j\ (mod\ \ p-1)$
- $g^i = a -> g^{\sqrt{p}*t+w} = a -> g^w = a*(g^{(\sqrt{p}*t)(p-2)})$
- 求 i 时,枚举 w ,用数组记录$p[g^w\%p] = w$,再枚举 t ,判断$p[a*(g^{(\sqrt{p}*t)(p-2)})\%p]$是否存在
- 若存在则$i = \sqrt{p}*t + p[a*(g^{(\sqrt{p}*t)(p-2)})\%p]$

```
int Cal_YuanGen(int m){
    int phi=euler(m),tmp;
    tmp=phi;
```

```
    int p[100],n=0;
    for (int i=2;i*i<=tmp;i++)
    if (tmp%i==0){
        p[n++]=i;
        while (tmp%i==0) tmp/=i;
    }
    if (tmp!=1) p[n++]=tmp;
    for (int g=2;g<m;g++){
        int flag=1;
        for (int i=0;i<n && flag;i++)
        if (PowerMul(g,phi/p[i],m)==1) flag=0;
        if (flag) return g;
    }
    return -1;
}
```

2.6 Euler 筛法求质数

```
int prime[MaxN],TotPrime,p[MaxN];
void GetPrime(){
    TotPrime=0;
    memset(p,0,sizeof(p));
    for (int i=2;i<MaxN;i++)
    if (!p[i]){
        p[i]=i;
        prime[TotPrime++]=i;
        for (int j=0;j<TotPrime;j++){
            int q=prime[j];
            if (i*q>=MaxN) break;
            p[i*q]=q;
            if (p[i]==q) break;
        }
    }
}
```

2.7 大数开根号

```
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(1){
        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");
}
```

2.8 高精度计算

```
class bignum{
        public:
        int op, len, a[MXN];
        bignum(int = 0);
        ~bignum();
        int max(int a, int b);
        void check();
        void operator=(bignum m);
        void operator=(int m);
        void operator=(char *s);
        bool operator<(bignum m);
        bool operator<=(bignum m);
        bool operator>(bignum m);
        bool operator>=(bignum m);
        bool operator==(bignum m);
        bool operator!=(bignum m);
        bignum operator-();
        bignum operator+(bignum m);
        void operator+=(bignum m);
        bignum operator-(bignum m);
        void operator-=(bignum m);
        bignum operator*(bignum m);
        bignum operator*(int m);
        void operator*=(bignum m);
        void operator*=(int m);
        bignum operator/(bignum m);
        bignum operator/(int m);
        void operator/=(bignum m);
        void operator/=(int m);
        bignum operator%(bignum m);
        bignum operator%(int m);
        void operator%=(bignum m);
```

```
        void operator%=(int m);
};
bignum abs(bignum m);
bool read(bignum &m);
void write(bignum m);
void swrite(char *s, bignum m);
void writeln(bignum m);
bignum sqr(bignum m);
bignum sqrt(bignum m);
bignum gcd(bignum a, bignum b);
bignum lcm(bignum a, bignum b);
int bignum::max(int a, int b){
        return (a > b) ? a : b;
}
bignum::bignum(int v){
        (*this) = v;
        this->check();
}
bignum::~bignum(){
}
void bignum::check(){
for (;len > 0 && a[len] == 0; --len) {}
        if (len == 0)
        op=1;
}
void bignum::operator=(bignum m){
        op = m.op;
        len = m.len;
        memcpy(a, m.a, MXN << 2);
        this->check();
}
void bignum::operator=(int m){
        op = (m > 0) ? 1 : -1;
        m = abs(m);
        memset(a, 0, MXN << 2);
        for (len = 0; m > 0; m = m / 10000)
        a[++len] = m % 10000;
        this->check();
}
void bignum::operator=(char *s){
        int L;
        (*this) = 0;
        if (s[0] == ’-’ || s[0] == ’+’) {
                if (s[0] == ’-’)
                op = -1;
                L = strlen(s);
                for (int i = 0; i < L; ++i)
```

```
                s[i] = s[i + 1];
        }
        L = strlen(s);
        len = (L + 3) / 4;
        for (int i = 0; i < L; ++i)
        a[(L - i + 3) / 4] = a[(L - i + 3)/4] * 10 + (s[i] - 48);
        this->check();
}
bool bignum::operator<(bignum m)
{
        if (op != m.op)
        return op < m.op;
        if (len != m.len)
        return op * len < m.len * op;
        for (int i = len; i >= 1; --i)
        if (a[i] != m.a[i])
        return a[i] * op < m.a[i] * op;
        return false;
}
bool bignum::operator<=(bignum m){
        return !(m < (*this));
}
bool bignum::operator>(bignum m){
        return m < (*this);
}
bool bignum::operator>=(bignum m){
        return !((*this) < m);
}
bool bignum::operator==(bignum m){
        return (!((*this) < m)) && (!(m < (*this)));
}
bool bignum::operator!=(bignum m){
        return ((*this) < m) || (m < (*this));
}
bignum bignum::operator-(){
        bignum c = (*this);
        c.op = -c.op;
        c.check();
        return c;
}
bignum abs(bignum m){
        bignum c = m;
        c.op = abs(c.op);
        c.check();
        return c;
}
bignum bignum::operator+(bignum m){
```

```
        if (m.len == 0)
        return (*this);
        if (len == 0)
        return m;
        if (op == m.op) {
                bignum c;
                c.op = op;
                c.len = max(len, m.len) + 1;
                for (int i = 1, temp = 0; i <= c.len; ++i)
                c.a[i] = (temp = (temp / 10000 + a[i] + m.a[i])) % 10000;
                c.check();
                return c;
        }
        return (*this) - (-m);
}
bignum bignum::operator-(bignum m){
        if (m.len == 0)
        return (*this);
        if (len == 0)
        return (-m);
        if (op == m.op) {
                bignum c;
                if (abs(*this) >= abs(m)) {
                        c.op = op;
                        c.len = len;
                        for (int i = 1, temp = 0; i <= len; ++i)
                        c.a[i] = ((temp = (-int(temp < 0) + a[i] - m.a[i])) + 10000) % 10000;
                        c.check();
                        return c;
                }
                return -(m - (*this));
        }
        return (*this) + (-m);
}
bool read(bignum &m){
        char s[maxlen * 4 + 10];
        if (scanf("%s", &s) ==- 1)
        return false;
        for (int i = 0; s[i]; ++i)
        if (!(s[i] >= '0' && s[i] <= '9' || (s[i] == '+' || s[i] == '-') && i == 0))
        return false;
        m = s;
        return true;
}
void swrite(char *s, bignum m){
        int L = 0;
        if (m.op == -1)
```

```
        s[L++] = '-';
        sprintf(s + L, "%d", m.a[m.len]);
for (;s[L] != 0; ++L) {}
        for (int i = m.len - 1; i >= 1; --i) {
                sprintf(s + L, "%04d", m.a[i]);
                L += 4;
        }
        s[L] = 0;
}
void write(bignum m){
        if (m.op == -1)
        printf("-");
        printf("%d", m.a[m.len]);
        for (int i = m.len - 1; i >= 1; --i)
        printf("%04d", m.a[i]);
}
void writeln(bignum m){
        write(m);
        printf("\n");
}
bignum bignum::operator*(bignum m){
        bignum c;
        c.op = op * m.op;

        c.len = len + m.len;
        for (int i = 1; i <= m.len; ++i) {
                int number = m.a[i], j, temp = 0;
                for (j = 1; j <= len; ++j)
                c.a[i + j - 1] += number * a[j];
                if (i % 10 == 0 || i == m.len)
                for (j = 1; j <= c.len; ++j)
                c.a[j] = (temp = (temp / 10000) + c.a[j]) % 10000;
        }
        c.check();
        return c;
}
bignum bignum::operator*(int m){
        if (m < 0)
        return -((*this) * (-m));
        if (m > 100000)
        return (*this) * bignum(m);
        bignum c;
        c.len = len + 2;
        c.op = op;
        int t = 0;
        for (int i = 1; i <= c.len; ++i)
        c.a[i] = (t = (t / 10000 + a[i] * m)) % 10000;
```

```
        c.check();
        return c;
}
bignum bignum::operator/(bignum m){
        if (m.len == 0) {
                printf("Division by zero.\n");
                exit(0);
        }
        if (abs(*this) < abs(m))
        return 0;
        bignum c, left;
        c.op = op / m.op;
        m.op = 1;
        c.len = len - m.len + 1;
        left.len = m.len - 1;
        memcpy(left.a + 1, a + len - left.len + 1, left.len << 2);
        for (int i = c.len; i >= 1; --i) {
                left = left * 10000 + a[i];
                int head = 0, tail = 10000, mid;
                while (head + 1 < tail) {
                        mid = (head + tail) >> 1;
                        if (m * mid <= left)
                        head = mid;
                        else
                        tail = mid;
                }
                c.a[i] = head;
                left -= m * head;
        }
        c.check();
        return c;
}
bignum bignum::operator/(int m){
        if (m < 0)
        return -((*this) / (-m));

        if (m > 100000)
        return (*this) / bignum(m);
        bignum c;
        c.op = op;
        c.len = len;
        int t = 0;
        for (int i = c.len; i >= 1; --i)
        c.a[i] = (t = (t % m * 10000 + a[i])) / m;
        c.check();
        return c;
}
```

```
bignum bignum::operator %(bignum m){
        return (*this) - ((*this) / m) * m;
}
bignum bignum::operator%(int m){
        if (m < 0)
        return -((*this) % (-m));
        if (m > 100000)
        return (*this) % bignum(m);
        int t = 0;
        for (int i = len; i >= 1; --i)
        t = (t * 10000 + a[i]) % m;
        return t;
}
bignum sqr(bignum m){
        return m * m;
}
bignum sqrt(bignum m){
        if (m.op < 0 || m.len == 0)
        return 0;
        bignum c, last, now, templast;
        c.len = (m.len + 1) >> 1;
        c.a[c.len] = int(sqrt((double)m.a[c.len*2] * 10000 + m.a[c.len * 2 - 1]) + 1e-6);
        templast.len = c.len * 2;
        templast.a[c.len * 2 - 1] = (c.a[c.len] * c.a[c.len]) % 10000;
        templast.a[c.len * 2] = (c.a[c.len] * c.a[c.len]) / 10000;
        templast.check();
        for (int i = c.len - 1; i >= 1; --i) {
                last = templast;
                int head = 0, tail = 10000, mid, j, temp;
                while (head + 1 < tail) {
                        mid = (head + tail) >> 1;
                        now = last;
                        now.a[2 * i - 1] += mid * mid;
                        for (j = i + 1; j <= c.len; ++j)
                        now.a[i + j - 1] += mid * c.a[j] * 2;
                        ++now.len;
                        for (j = 2 * i - 1, temp = 0; j <= now.len; ++j)
                        now.a[j] = (temp = (temp / 10000 + now.a[j])) % 10000;
                        now.check();
                        if (now <= m) {
                                templast = now;
                                head = mid;
                        } else
                        tail = mid;
                }
                c.a[i] = head;

```

```
        }
        c.check();
        return c;
}
bignum gcd(bignum a, bignum b){
        return (b == 0) ? a : gcd(b, a % b);
}
bignum lcm(bignum a, bignum b){
        return a * b / gcd(a, b);
}
void bignum::operator+=(bignum m){
        (*this) = (*this) + m;
}
void bignum::operator-=(bignum m){
        (*this) = (*this) - m;
}
void bignum::operator*=(bignum m){
        (*this) = (*this) * m;
}
void bignum::operator/=(bignum m){
        (*this) = (*this) / m;
}
void bignum::operator%=(bignum m){
        (*this) = (*this) % m;
}
void bignum::operator*=(int m){
        (*this) = (*this) * m;
}
void bignum::operator/=(int m){
        (*this) = (*this) / m;
}
void bignum::operator%=(int m){
        (*this) = (*this) % m;
}
```

3 Chapter 3 Data structure 数据结构

3.1 主席树

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
#define MX 100010
typedef pair<int,int> PI;
#define F first
#define S second
#define MP(x,y) make_pair(x,y)
struct node{
```

```
        PI mx;
        node* left;
        node* right;
        node(){ left=NULL; right=NULL; mx=MP(-1,-1); }
};
PI maxi(PI a, PI b){
        PI r;
        if(a.F>=b.F) r.F = a.F, r.S = max(a.S,b.F);
        else r.F = b.F, r.S = max(b.S,a.F);
        return r;
}
node* roots[MX];
int initRatings[MX], n, m, A, B, C, D, T;
node* build(int l,int h){
        node* tmp = new node();
        if(l==h){ tmp->mx = MP(initRatings[l],-1); return tmp; }
        tmp->left = build(l,(l+h)/2);
        tmp->right = build((l+h)/2+1,h);
        tmp->mx = maxi(tmp->left->mx, tmp->right->mx);
        return tmp;
}
PI query(node* cur,int l,int h,int a,int b){
        if(b<l || a>h) return MP(-1,-1);
        if(a<=l && h<=b) return cur->mx;
        return maxi(query(cur->left,l,(l+h)/2,a,b),query(cur->right,(l+h)/2+1,h,a,b));
}
node* update(node* cur,int l,int h,int ind,int val){
        node* tmp = new node();
        if(l==h){
                tmp->mx = MP(val,-1);
                return tmp;
        }
        int mid = (l+h)/2;
        if(ind>mid){
                tmp->left = cur->left;
                tmp->right = update(cur->right,mid+1,h,ind,val);
        }
        else{
                tmp->right = cur->right;
                tmp->left = update(cur->left,l,mid,ind,val);
        }
        tmp->mx = maxi(tmp->left->mx, tmp->right->mx);
        return tmp;
}
int main(){
        scanf("%d %d %d %d %d %d",&n,&m,&A,&B,&C,&D);
        for(int i=0;i<n;i++) scanf("%d",&initRatings[i]);
```

```
        roots[0] = build(0,n-1);
        scanf("%d",&T);
        PI ans = MP(0,0);
        for(int i=1;i<=T;i++){
                int l, h, t = (int)((A*1LL*ans.F+D)%i);
                scanf("%d %d",&l,&h);
                ans = query(roots[t],0,n-1,l,h);
                printf("%d %d\n",ans.F,ans.S);
                int ind = (int)((B*1LL*ans.F+D)%n), val = (int)((C*1LL*ans.S+D)%m);
                roots[i] = update(roots[i-1],0,n-1,ind,val);
        }
        return 0;
}
```

3.2 Splay

```
struct Tsplay{
        int l,r,p,s,v;
        bool rev;
};
Tsplay  tree[MaxN];
int  a[MaxN],N,L,R,tot,root,F;
bool Root(int x){
        return !tree[x].p || tree[tree[x].p].l!=x && tree[tree[x].p].r!=x;
}
void Down(int x){
        if (x==0) return;
        if (tree[x].rev){
                int t=tree[x].l;
                tree[x].l=tree[x].r;
                tree[x].r=t;
                tree[tree[x].l].rev^=1;
                tree[tree[x].r].rev^=1;
                tree[x].rev=0;
        }
}
void Zig(int x){
        int y=tree[x].p,z=tree[y].p;
        tree[y].l=tree[x].r;
        tree[tree[x].r].p=y;
        tree[x].p=z;
        if (y==tree[z].l) tree[z].l=x; else
        if (y==tree[z].r) tree[z].r=x;
        tree[x].r=y;
        tree[y].p=x;
        tree[y].s=tree[tree[y].l].s+tree[tree[y].r].s+1;
        tree[x].s=tree[tree[x].l].s+tree[tree[x].r].s+1;
}
void Zag(int x){
```

```
        int y=tree[x].p,z=tree[y].p;
        tree[y].r=tree[x].l;
        tree[tree[x].l].p=y;
        tree[x].p=z;
        if (y==tree[z].l) tree[z].l=x; else
        if (y==tree[z].r) tree[z].r=x;
        tree[x].l=y;
        tree[y].p=x;
        tree[y].s=tree[tree[y].l].s+tree[tree[y].r].s+1;
        tree[x].s=tree[tree[x].l].s+tree[tree[x].r].s+1;
}
void Splay(int x){
        a[a[0]=1]=x;
        for (int i=x;!Root(i);a[++a[0]]=i=tree[i].p);
        for (int i=a[0];i>=1;i--) Down(a[i]);
        while(!Root(x)){
                int y=tree[x].p,z=tree[y].p;
                if (Root(y)) if (x==tree[y].l) Zig(x); else Zag(x); else
                if (y==tree[z].l)
                if (x==tree[y].l) Zig(y),Zig(x); else Zag(x),Zig(x); else
                if (x==tree[y].r) Zag(y),Zag(x); else Zig(x),Zag(x);
        }
        if (tree[x].p==0) root=x;
}
void met(int x){
        tree[x].l=tree[x].r=tree[x].p=tree[x].v=tree[x].s=tree[x].rev=0;
}
int Find(int root,int S){//查找第S小的数
        if (root==0) return 0;
        while (1){
                Down(root);
                if (S<=tree[tree[root].l].s) root=tree[root].l;else
                if (S==tree[tree[root].l].s+1) return root;else{
                        S=S-tree[tree[root].l].s-1;
                        root=tree[root].r;
                }
        }
}
void Del(int x){
        if (N==1){
                root=0;
                return;
        }
        if (x==1){
                int Y=Find(root,x+1);
                Splay(Y);
                tree[Y].l=0;
```

```
				tree[Y].s--;
				return;
		}
		int y=Find(root,x-1);
		int z;
		if (x!=N) z=Find(root,x+1);
		Splay(y);
		tree[y].s--;
		tree[y].r=0;
		if (x==N) return;
		Splay(z);
		tree[z].l=0;
		tree[z].s--;
		tree[y].r=z;
}
void Rev(int l,int r){
		if (l==1){
				if (r==N){
						tree[root].rev^=1;
						return;
				}
				int y=Find(root,r+1);
				Splay(y);
				tree[tree[y].l].rev^=1;
		}else{
				int y=Find(root,l-1);
				Splay(y);
				if (r==N){
						tree[tree[root].r].rev^=1;
						return;
				}
				int z=Find(root,r+1);
				tree[y].r=0;
				Splay(z);
				tree[tree[z].l].rev^=1;
				tree[y].r=z;
		}
}
void Ins(int x,int v){
		met(++tot);
		tree[tot].v=v;
		tree[tot].s=1;
		if (root==0) root=tot;else{
				int now=root;
				while (1){
						if (x==0){
								while (1){
```

```
                    Down(now);
                    tree[now].s++;
                    if (tree[now].l==0){
                        tree[now].l=tot;
                        tree[tot].p=now;
                        return;
                    }
                    now=tree[now].l;
                }
            }else{
                Down(now);
                tree[now].s++;
                if (tree[tree[now].l].s>=x){
                    now=tree[now].l;
                    continue;
                }else
                if (tree[now].r==0){
                    tree[now].r=tot;
                    tree[tot].p=now;
                    return;
                }else{
                    x-=tree[tree[now].l].s+1;
                    now=tree[now].r;
                }
            }
        }
    }
}
```

3.3 动态树

- 实边相连组成的以深度为 key 的 Splay，左边的深度比右边的小

```
struct Tsplay{l=左子树, r=右子树, p=parent 父亲, rev=是否交换了左右子树, 用作节点变根操作
    int l,r,p;
    bool rev;
};
Tsplay  tree[MaxN];
int  a[MaxN];
bool Root(int x){//点 x 是否为某棵 Splay 的根
    return !tree[x].p||tree[tree[x].p].l!=x&&tree[tree[x].p].r!=x;
}
void Down(int x){//向下更新节点的内容
    if (tree[x].rev){
        int t=tree[x].l;
        tree[x].l=tree[x].r;
        tree[x].r=t;
        tree[tree[x].l].rev^=1;
        tree[tree[x].r].rev^=1;
        tree[x].rev=0;
```

```
        }
}
void Zig(int x){//右旋
        int y=tree[x].p,z=tree[y].p;
        tree[y].l=tree[x].r;
        tree[tree[x].r].p=y;
        tree[x].p=z;
        if (y==tree[z].l) tree[z].l=x; else
        if (y==tree[z].r) tree[z].r=x;
        tree[x].r=y;
        tree[y].p=x;
        Update(y);
}
void Zag(int x){//左旋
        int y=tree[x].p,z=tree[y].p;
        tree[y].r=tree[x].l;
        tree[tree[x].l].p=y;
        tree[x].p=z;
        if (y==tree[z].l) tree[z].l=x; else
        if (y==tree[z].r) tree[z].r=x;
        tree[x].l=y;
        tree[y].p=x;
        Update(y);
}
void Splay(int x){
        a[a[0]=1]=x;
        for (int i=x;!Root(i);a[++a[0]]=i=tree[i].p);
        for (int i=a[0];i>=1;i--) Down(a[i]);
        while(!Root(x)){
                int y=tree[x].p,z=tree[y].p;
                if (Root(y)) if (x==tree[y].l) Zig(x); else Zag(x); else
                if (y==tree[z].l)
                if (x==tree[y].l) Zig(y),Zig(x); else Zag(x),Zig(x); else
                if (x==tree[y].r) Zag(y),Zag(x); else Zig(x),Zag(x);
        }
        Update(x);
}
void Access(int x){//虚实边转换，从 x 到原树根的都变为实根，x 的儿子与 x 都连虚根
        for (int y=0;x;x=tree[x].p){
                Splay(x);
                tree[x].r=y;
                Update(y=x);
        }
}
int GetRoot(int x){//返回 x 的根
        Access(x);
        Splay(x);
```

```
        for (;tree[x].l;x=tree[x].l);
        Splay(x);
        return x;
}
int GetFather(int x){//返回 x 的父亲
        Access(x);
        Splay(x);
        if (!tree[x].l) return 0;
        for (Down(x),x=tree[x].l;Down(x),tree[x].r;x=tree[x].r);
        Splay(x);
        return x;
}
void MakeRoot(int x){//将节点 x 变为原树的根
        Access(x);
        Splay(x);
        tree[x].rev=1;
}
void Cut(int x){//将 x 与 x 的父亲断开，及分离成两棵树，一颗的根是 x
        Access(x);
        Splay(x);
        tree[tree[x].l].p=0;
        tree[x].l=0;
}
void Join(int x,int y){//将 x 与 y 相连，合并树
        if (GetRoot(x)==GetRoot(y))  return; else{
                MakeRoot(y);
                tree[y].p=x;
                Access(y);
        }
}
```

3.4 SBT

```
struct SBT{
        int key,l,r,s;
};
SBT tree[MaxN];
void Rrotate(int &t){
        int k=tree[t].l;
        tree[t].l=tree[k].r;
        tree[k].r=t;
        tree[k].s=tree[t].s;
        tree[t].s=tree[tree[t].l].s+tree[tree[t].r].s+1;
        t=k;
}
void Lrotate(int &t){
        int k=tree[t].r;
        tree[t].r=tree[k].l;
        tree[k].l=t;
```

```
		tree[k].s=tree[t].s;
		tree[t].s=tree[tree[t].l].s+tree[tree[t].r].s+1;
		t=k;
}
void MainTain(int &t, int flag){
		if (!flag){
				if (tree[tree[tree[t].l].l].s>tree[tree[t].r].s) Rrotate(t);else
				if (tree[tree[tree[t].l].r].s>tree[tree[t].r].s){
						Lrotate(tree[t].l);
						Rrotate(t);
				}else return;
		}else{
				if (tree[tree[tree[t].r].r].s>tree[tree[t].l].s) Lrotate(t);else
				if (tree[tree[tree[t].r].l].s>tree[tree[t].l].s){
						Rrotate(tree[t].r);
						Lrotate(t);
				}else return;
		}
		Maintain(tree[t].l,0);
		Maintain(tree[t].r,1);
		Maintain(t,1);
		Maintain(t,0);
}
void Insert(int &t,int v){
		if (t==0){
				tree[t=++tot].key=v;
				tree[t].s=1;
				tree[t].l=tree[t].r=0;
		}else{
				tree[t].s++;
				if (v<tree[t].key) Insert(tree[t].l,v);else
				Insert(tree[t].r,v);
				Maintain(t,v>=tree[t].key);
		}
}
int Delete(int &t,int v){
		int ret=0;
		s[t]--;
			if (v==tree[t].key || (v<tree[t].key && tree[t].l==0) || (v>tree[t].key && tree[t].r==0)){
				ret=tree[t].key;
				if (tree[t].l==0 || tree[t].r==0) t=tree[t].l+tree[t].r;else
				tree[t].key=Delete(tree[t].l,tree[t].key+1);
		}else
		if (v<tree[t].key) ret=Delete(tree[t].l,v);else
		ret=Delete(tree[t].r,v);
		return ret;
```

```
}
```

3.5 KD-tree

- K 维度中找离询问点最近的 M 个点

```
#include <cstdio>
#include <queue>
#include <cmath>
#include <cstring>
#include <algorithm>
using namespace std;
#define lchd idx << 1
#define rchd idx << 1 | 1
const int MAXN = 200000;
const int inf = 1000000000;
double sqr(double x){ return x * x; }
int k, n; //k 是维数, n 是点数
struct point{
        int x[5];
        friend bool operator == (point p1, point p2){
                for(int i = 0; i < k; i++)
                if(p1.x[i] != p2.x[i])
                return 0;
                return 1;
        }
}po[50010];
struct kd_Tree{
        point p;
        int succeed; //后裔的个数，判断是否为叶子
}tree[MAXN];
struct node{
        point p;
        double dis;
        friend bool operator < (node n1, node n2){
                return n1.dis < n2.dis;
        }
};
priority_queue<node> nq; //保存前 m 个最近点
point le[MAXN], ri[MAXN]; //two array of merging
double cald(point p1, point p2){
        double d = 0;
        for(int i = 0; i < k; i++)
        d += sqr(p1.x[i] - p2.x[i]);
        return d;
}
void merge(point p[], int l, int m, int r, int dim){
        for(int i = l; i <= m; i++)
        le[i - l] = p[i];
        for(int i = m + 1; i <= r; i++)
```

```
        ri[i - m - 1] = p[i];
        le[m - l + 1].x[dim] = inf;
        ri[r - m].x[dim] = inf;
        int ltop = 0, rtop = 0;
        for(int i = l; i <= r; ){
                if(le[ltop].x[dim] < ri[rtop].x[dim])
                p[i++] = le[ltop++];
                else p[i++] = ri[rtop++];
        }
}
void mergesort(point p[], int l, int r, int dim){
        int m = (l + r) >> 1;
        if(l < r){
                mergesort(p, l, m, dim);
                mergesort(p, m + 1, r, dim);
                merge(p, l, m, r, dim);
        }
}
//build the tree
void build(point po[], int l, int r, int idx, int dep){
        if(l > r) return;
        tree[idx].succeed = r - l;
        tree[lchd].succeed = tree[rchd].succeed = -1;
        int dim = dep % k;
        //printf("idx:%d size:%d\n", idx, size);
        //for(int i = 0; i < size; i++) print(p[i]);
        mergesort(po, l, r, dim); //sort according to one dimension
        int mid = (l + r) >> 1;
        tree[idx].p = po[mid];
        build(po, l, mid - 1, lchd, dep + 1);
        build(po, mid + 1, r, rchd, dep + 1);
}
// Query the m nearest point
// It is similar as the most nearest, go through to the leaf, and then if the current count of point is less than m
// search every sub-tree
// Maintain a heap which size is m, when the distance of current node is less than the top of heap
// push the current one into heap, pop up the top one.
// the rest is just similar to the most nearest.
void query(point p, int idx, int dep, int m){
        if(tree[idx].succeed == -1) return;
        node nd; nd.p = tree[idx].p;
        nd.dis = cald(nd.p, p);
        if(tree[idx].succeed == 0){
                if(nq.size() < m) nq.push(nd);
                else{
                        if(nd.dis < nq.top().dis){
```

```
                        nq.pop();
                        nq.push(nd);
                    }
                }
                return;
        }
        int dim = dep % k;
        if(p.x[dim] < tree[idx].p.x[dim]){
                query(p, lchd, dep + 1, m);
                if(nq.size() < m){
                        nq.push(nd);
                        query(p, rchd, dep + 1, m);
                }
                else{
                        if(nd.dis < nq.top().dis){
                                nq.pop();
                                nq.push(nd);
                        }
                        double mx = nq.top().dis;
                        /* if you want to query the distance from one plane is
                        less than the distance of the heap top
                        you should go to another side to query.
                        */
                        if(sqr(p.x[dim] - tree[idx].p.x[dim]) < mx)
                        query(p, rchd, dep + 1, m);
                }
        }
        else{
                query(p, rchd, dep + 1, m);
                if(nq.size() < m){
                        nq.push(nd);
                        query(p, lchd, dep + 1, m);
                }
                else{
                        if(nd.dis < nq.top().dis){
                                nq.pop();
                                nq.push(nd);
                        }
                        double mx = nq.top().dis;
                        if(sqr(p.x[dim] - tree[idx].p.x[dim]) < mx)
                        query(p, lchd, dep + 1, m);
                }
        }
}
//output the node
void print(point p){
        for(int j = 0; j < k; j++){
```

```
                printf("%d", p.x[j]);
                j == k - 1 ? puts("") : printf(" ");
        }
}
int main(){
        while(scanf("%d%d", &n, &k) != EOF){
                for(int i = 0; i < n; i++)
                for(int j = 0; j < k; j++)
                scanf("%d", &po[i].x[j]);
                build(po, 0, n - 1, 1, 0);
                int t;
                scanf("%d", &t);
                node nd[10];
                for(int i = 0; i < t; i++){
                        point ask;
                        for(int j = 0; j < k; j++)
                        scanf("%d", &ask.x[j]);
                        int m;
                        scanf("%d", &m);
                        query(ask, 1, 0, m);
                        printf("the closest %d points are:\n", m);
                        for(int j = 0; !nq.empty(); j++)
                        nd[j].p = nq.top().p, nq.pop();
                        for(int j = m - 1; j >= 0; j--)
                        print(nd[j].p);
                }
        }
        return 0;
}
```

3.6 Dancing-Links 精确覆盖

```
int N;
int U[MXD], D[MXD], L[MXD], R[MXD], CH[MXD], RH[MXD];
int size[MXC];

int ans[MXR], ansN;

int addNode(int u, int d, int l, int r){
        U[N] = u; D[N] = d; L[N] = l; R[N] = r;
        U[d] = D[u] = L[r] = R[l] = N;
        return N++;
}

inline void remove(int c){
        L[R[c]] = L[c]; R[L[c]] = R[c];
        for (int i = D[c]; i != c; i = D[i])
        for (int j = R[i]; j != i; j = R[j]) {
                --size[CH[j]];
```

```
                U[D[j]] = U[j]; D[U[j]] = D[j];
        }
}

inline void resume(int c){
        for (int i = U[c]; i != c; i = U[i])
        for (int j = L[i]; j != i; j = L[j]) {
                U[D[j]] = D[U[j]] = j;
                ++size[CH[j]];
        }
        R[L[c]] = L[R[c]] = c;
}

bool dfs(int dep){
        if (R[0] == 0) {
                ansN = dep - 1;
                return true;
        }
        int col = -1;
        for (int i = R[0]; i; i = R[i])
        if (col == -1 || size[i] < size[col]) col = i;


        remove(col);
        for (int i = D[col]; i != col; i = D[i]) {
                ans[dep] = RH[i];
                for (int j = R[i]; j != i; j = R[j])
                remove(CH[j]);
                if (dfs(dep + 1)) return true;
                for (int j = L[i]; j != i; j = L[j])
                resume(CH[j]);
        }
        resume(col);
        return false;
}

void init(int P, int Q){
        N = 0;
        memset(size, 0, sizeof size);

        addNode(0, 0, 0, 0);
        for (int i = 1; i <= Q; ++i)
        addNode(i, i, L[0], 0);
        for (int i = 1; i <= P; ++i) {
                int row = -1, k;
                for (int j = 1; j <= Q; ++j)
                if (mat[i][j]) {
```

```
                        CH[N] = j; ++size[j];
                        if (row == -1) {
                                row = addNode(U[j], j, N, N);
                                RH[row] = i;
                        } else {
                                k = addNode(U[j], j, L[row], row);
                                RH[k] = i;
                        }
                }
        }
}
```

3.7 Dancing-Links 重复覆盖

```
int R[MXD], L[MXD], D[MXD], U[MXD], CH[MXD], RH[MXD];
int size[MXN];
bool Hash[MXN];
inline int addNode(int u, int d, int l, int r){
        U[N] = u; D[N] = d; L[N] = l; R[N] = r;
        U[d] = D[u] = L[r] = R[l] = N;
        return N++;
}
inline void cov(int c){
        for (int i = D[c]; i != c; i = D[i]) {
                R[L[i]] = R[i];
                L[R[i]] = L[i];
        }
}
inline void res(int c){
        for (int i = U[c]; i != c; i = U[i]) {
                R[L[i]] = i;
                L[R[i]] = i;
        }
}
int h(){
        memset(Hash, 0, sizeof Hash);
        int ret = 0;
        for (int i = R[0]; i != 0; i = R[i])
        if (!Hash[i]) {
                ++ret;
                Hash[i] = true;
                for (int j = D[i]; j != i; j = D[j])
                for (int k = R[j]; k != j; k = R[k])
                Hash[CH[k]] = true;
        }
        return ret;
}
bool dfs(int u){
        if (u + h() >= ansN) return  false;
```

```
        if (R[0] == 0) {
                if (u < ansN) {
                        ansN = u;
                        memmove(ans, tmp, u << 2);
                }
                return true;
        }
        int c = R[0];
        for (int i = D[c]; i != c; i = D[i]) {
                cov(i);
                for (int j = R[i]; j != i; j = R[j]) cov(j);
                tmp[u] = RH[i];
                dfs(u + 1);
                put = false;
                for (int j = L[i]; j != i; j = L[j]) res(j);
                res(i);
        }
        return false;
}
```

4 Chapter 4 String 字符串

4.1 KMP 字符串匹配

```
int p[MaxN];
char s[MaxN],ss[MaxN];
void KMP(){
        int j=0;
        p[1]=0;
        int Lens=strlen(s);
        for (int i=2;i<=Lens;i++){
                while (j && s[i-1]!=s[j]) j=p[j];
                if (s[i-1]==s[j]) j++;
                p[i]=j;
        }
        j=0;
        int Lenss=strlen(ss);
        for (int i=1;i<=Len;i++){
                while (j && ss[i-1]!=s[j]) j=p[j];
                if (ss[i-1]==s[j]) j++;
                if (j==Lens){
                        cout << i-j+1;
                        j=p[j];
                }
        }
}
```

4.2 拓展 KMP

- 复杂度 $O(N)$
- p[i]　　：s[i]与 t 的最长公共前缀长度

```
void ExKMP(char s[],char t[]){
	int j,k,Len,L,nxt[100000],p[100000];
	int LenS,LenT;
	LenS=strlen(s);
	LenT=strlen(t);
	j=0;
	while (t[j+1]==t[j] && j+1<LenT) j++;
	nxt[1]=j,k=1;
	for (int i=2;i<LenT;i++){
		Len=k+nxt[k],L=nxt[i-k];
		if (Len>L+i) nxt[i]=L;else{
			j=Len-i>0 ? Len-i : 0;
			while (t[i+j]==t[j] && i+j<LenT) j++;
			nxt[i]=j,k=i;
		}
	}
	j=0;
	while ( s[j]==t[j] && j<LenT && j<LenS) j++;
	p[0]=j,k=0;
	for (int i=1;i<LenS;i++){
		Len=k+p[k],L=nxt[i-k];
		if (Len>L+i) p[i]=L;else{
			j=Len-i > 0 ? Len-i : 0;
			while (s[i+j]==t[j] && i+j<LenS && j<LenT) j++;
			p[i]=j,k=i;
		}
	}
}
```

4.3 后缀数组 DC3 算法

- 复杂度 $O(N)$
- num[0~len-1]为有效值 就是输入的字符串字符的大小数组，从 1 开始，0 是终止符
- sa[0~len-1]为有效值 sa[i]=a 则代表排在第 i 位的是第 a 个后缀
- rank[0~len-1]是有效值 rank[i]=b 则代表第 i 个后缀排在第 b 位
- height[0~len-1]是有效值 height[i]=c 则代表排在第 i 位的后缀和排在第 i-1 的后缀的最长前缀长度是 c

```
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x)((x)<tb?(x)*3+1:((x)-tb)*3+2)
const int MaxN=100000;
using namespace std;
int wa[MaxN],wb[MaxN],wv[MaxN],wss[MaxN];
int sa[MaxN*3], rank[MaxN*3], height[MaxN],num[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){
```

```
        for(int i=0;i<n;i++) wv[i]=r[a[i]];
        for(int i=0;i<m;i++) wss[i]=0;
        for(int i=0;i<n;i++) wss[wv[i]]++;
        for(int i=1;i<m;i++) wss[i]+=wss[i-1];
        for(int i=n-1;i>=0;i--) b[--wss[wv[i]]]=a[i];
}
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++];
}
void calHeight(int *r, 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 ++);
}
int main(){
        char str[MaxN];
        int m=30, ans,len;
        while(scanf("%s",str)!=EOF){
                len=strlen(str);
                for(int i=0;i<=len;i++) num[i]=str[i]-'0'+1;
                num[len]=0;
                dc3(num, sa, len+1, m);
                calHeight(num, len);
                for (int i=0;i<len;i++)
                sa[i]=sa[i+1],height[i]=height[i+1];
        }
        return 0;
}
```

4.4 后缀数组

- 复杂度 $O(N\ log\ N)$

```
char s[MXN];
int sa[MXN], rk[MXN], h[MXN];
int st[20][MXN], mm[MXN];

#define SUFDIFF(a, b) LS[a] != LS[b] || LS[a + m] != LS[b + m]
void Suffix(char *s, int L, int C){//C 大于 s 串中最大的字符
	int *RK = rk, *LS = h, *sum = mm;
	for (int i = 1; i <= L; ++i) {
		RK[i] = s[i];
		sa[i] = i;
	}
	for (int m = 0; m <= L; !m ? m = 1 : m <<= 1) {

		int cnt = m;
		for (int i = 1; i <= m; ++i) LS[i] = L - m + i;
		for (int i = 1; i <= L; ++i)
		if (sa[i] > m) LS[++cnt] = sa[i] - m;

		memset(sum, 0, (C + 1) << 2);
		for (int i = 1; i <= L; ++i) ++sum[RK[LS[i]]];
		for (int i = 1; i <= C; ++i) sum[i] += sum[i - 1];
		for (int i = L; i >= 1; --i) sa[sum[RK[LS[i]]]--] = LS[i];

		swap(RK, LS);
		int tot = 0; RK[sa[1]] = ++tot;
		for (int i = 2; i <= L; ++i)
		if (SUFDIFF(sa[i - 1], sa[i])) RK[sa[i]] = ++tot;
		else RK[sa[i]] = tot;
		C = tot;
	}
	memmove(rk, RK, (L + 1) << 2);
}

void predo(char *s, int L){
	for (int i = 1; i <= L; ++i) {
		h[i] = max(0, h[i - 1] - 1);
		int tmp = sa[rk[i] - 1];
		if (tmp == 0) continue;
		while (s[i + h[i]] == s[tmp + h[i]]) ++h[i];
	}

	mm[0] = -1;
	for (int i = 1; i <= L; ++i)
	mm[i] = mm[i - 1] + ((i & (i - 1)) == 0);
	for (int i = 1; i <= L; ++i) st[0][i] = h[sa[i]];

```

```
        for (int i = 1; i <= mm[L]; ++i)
        for (int j = 1; j <= L - (1 << i) + 1; ++j)
        st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
}

inline int LCP(int a, int b){
        if (a == b) return strlen(s + a) - 1;
        a = rk[a]; b = rk[b];
        if (a > b) swap(a, b);
        int len = mm[b - ++a + 1];
        return min(st[len][a], st[len][b - (1 << len) + 1]);
}
```

4.5 AC 自动机

```
struct AC {
        static const int UNDEF = 0;
        static const int MXN = 2048;
        static const int CHARSET = 2;
        int tot;
        int tag[MXN];
        int fail[MXN];
        int trie[MXN][CHARSET];


        void init(){
                tag[0] = UNDEF;
                memset(trie[0], -1, sizeof trie[0]);
                tot = 1;
        }

        void add(int m, const int* s, int t){
                int p = 0;
                for (int i = 0; i < m; ++i) {
                        if (trie[p][*s] == -1) {
                                tag[tot] = UNDEF;
                                memset(trie[tot], -1, sizeof trie[tot]);
                                trie[p][*s] = tot++;
                        }
                        p = trie[p][*s];
                        ++s;
                }
                tag[p] = t;
        }

        void build(){
                queue<int> q;
                fail[0] = 0;
                for (int i = 0; i < CHARSET; ++i) {
```

```
            if (trie[0][i] != -1) {
                fail[trie[0][i]] = 0;
                q.push(trie[0][i]);
            } else {
                trie[0][i] = 0;
            }
        }
        while (!q.empty()) {
            int p = q.front();
            tag[p] |= tag[fail[p]]; // operate
            q.pop();
            for (int i = 0; i < CHARSET; ++i) {
                if (trie[p][i] != -1) {
                    fail[trie[p][i]] = trie[fail[p]][i];
                    q.push(trie[p][i]);
                } else {
                    trie[p][i] = trie[fail[p]][i];
                }
            }
        }
    }
};
```

4.6 后缀自动机

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
const int MaxN=250000+1000;//要构造后缀自动机的母串的长度
const int MaxChr=30;//母串所包含的字符种类个数
const char LAST='#';//若要输出所有后缀取一不在母串中的字符当做结束符或利用 pre 对 node 标号
using namespace std;
int tot,Len,id[300];//对于母串中的字符 c, ch[id[c]=t]=c;t 表示字符 c 在母串所含字符中的位置
struct node{
    int deep;
    node *ch[MaxChr],*pre;
    node(){
        memset(ch,0,sizeof ch);
    }
};
node pool[MaxN*2],*tail,head;//head=root,tail 指向最后一个加入的字符
char st[MaxN],s[MaxN],ch[MaxChr];
void Add(int c,int len){//加字符
    node *p=tail,*np=&pool[++tot];
    np->deep=len;
    for (;p && !p->ch[c];p=p->pre) p->ch[c]=np;
```

```
		tail=np;
		if (!p) np->pre=&head;else
		if (p->ch[c]->deep==p->deep+1) np->pre=p->ch[c];else{
			node *q=p->ch[c],*r=&pool[++tot];
			*r=*q;
			r->deep=p->deep+1;
			q->pre=np->pre=r;
			for (;p && p->ch[c]==q;p=p->pre) p->ch[c]=r;
		}
}
void build(){//构造后缀自动机，本程序中假设母串字符数为 26 个分别编号 0~25，LAST 编号 26
		for (char C='a';C<='z';C++)
		ch[id[C]=C-'a']=C;
		ch[id[LAST]=26]=LAST;
		for (int i=0;i<Len+Len;i++){
			node *Ne=&pool[i];
			for (int j=0;j<MaxChr;j++)
			Ne->ch[j]=NULL;
		}
		tot=0;
		head.pre=NULL;
		head.deep=0;
		memset(head.ch,0,sizeof head.ch);
		tail=&head;
		for (int i=0;i<Len;i++)
		Add(id[st[i]],i+1);
}
int main(){
		gets(st);
		Len=strlen(st);
		build();
		memset(st,0,sizeof(st));
		gets(st);
		int Ans=0,tmp=0,N=strlen(st);
		node *last;
		int i;
		for (i=0,last=&head;i<N;i++,Ans=max(Ans,tmp)){
			if (last->ch[id[st[i]]]) tmp++,last=last->ch[id[st[i]]];else{
				for (;last && !last->ch[id[st[i]]];last=last->pre);
				if (!last) last=&head,tmp=0;else
				tmp=last->deep+1,last=last->ch[id[st[i]]];
			}
		}
		printf("%d\n", Ans);
		return 0;
}
```

4.7 后缀树

```
#define NUM 27
#define STARTCHAR 'a'
#define SPECIALCHAR '{'
#define ERROR -1
#define TYPE1 1
#define TYPE2 2
#define LEAF 1
#define NOTLEAF 2
struct SuffixTrie {
        int Start, End;
        SuffixTrie * Next[NUM];
        SuffixTrie * Link;
        SuffixTrie * Father;
        int Flag;
        int Length;
};
char str[100010], buf[100010];
SuffixTrie head;
SuffixTrie*P, *G, *U, *V, *q;
int W[3], len, len2;
void CreateNode(SuffixTrie * & Node) {
        int i;
        Node = (SuffixTrie * ) malloc(sizeof(SuffixTrie));
        Node -> Start = Node -> End = Node -> Length = ERROR;
        for (i = 0; i < NUM; i++) Node -> Next[i] = NULL;
        Node -> Link = Node -> Father = NULL;
        Node -> Flag = LEAF;
}
void Init(SuffixTrie &h, char s[]) {
        int i;
        h.Start = h.End = ERROR;
        for (i = 0; i < NUM; i++) h.Next[i] = NULL;
        h.Link = & h;
        h.Father = NULL;
        h.Flag = LEAF;
        h.Length = 0;
        len = strlen(s);
        s[len] = SPECIALCHAR;
        s[len + 1] = '\0';
        len++;
}
int FindV(char s[]) {
        int old;
        SuffixTrie * t, * newt;
        t = U -> Next[s[W[0]] - STARTCHAR];
        old = 0;
        while (W[2] > (t -> End) - (t -> Start) + 1 + old) {
```

```
                old += (t -> End - t -> Start + 1);
                t = t -> Next[s[W[0] + old] - STARTCHAR];
        }
        if (W[2] == (t -> End) - (t -> Start) + 1 + old) {
                V = t;
                P -> Link = V;
                return TYPE1;
                } else {
                CreateNode(newt);
                newt -> Start = t -> Start;
                newt -> End = t -> Start + W[2] - old - 1;
                newt -> Father = t -> Father;
                newt ->
                Length = newt -> Father -> Length + newt -> End - newt ->
                Start + 1;
                t -> Father -> Next[s[t -> Start] - STARTCHAR] = newt;
                t -> Start = newt -> End + 1;
                newt -> Next[s[t -> Start] - STARTCHAR] = t;
                t -> Father = newt;
                V = newt;
                P -> Link = V;
                return TYPE2;
        }
}
int Insert(SuffixTrie * Node, int start, char s[]) {
        int i, posbegin, posend;
        SuffixTrie * t;
        if (Node -> Next[s[start] - STARTCHAR] == NULL) {
                CreateNode(Node -> Next[s[start] - STARTCHAR]);
                Node -> Next[s[start] - STARTCHAR] -> Start = start;
                Node -> Next[s[start] - STARTCHAR] -> End = len - 1;
                Node -> Next[s[start] - STARTCHAR] -> Father = Node;
                Node -> Next[s[start] - STARTCHAR] ->
                Length = Node -> Length + len - start;
                Node -> Flag = NOTLEAF;
                P = Node;
                return TYPE1;
                } else {
                posbegin = Node -> Next[s[start] - STARTCHAR] -> Start;
                posend = Node -> Next[s[start] - STARTCHAR] -> End;
                for (i = posbegin; i <= posend; i++) {
                        if (s[i] != s[start + i - posbegin]) break;
                }
                if (i == posend + 1)
                return Insert(Node->Next[s[start]-STARTCHAR], start+i-posbegin,s);
                else {
                        CreateNode(t);
```

```
                        t -> Start = posbegin;
                        t -> End = i - 1;
                        t -> Flag = NOTLEAF;
                        Node -> Next[s[start] - STARTCHAR] -> Start = i;
                        t -> Next[s[i] - STARTCHAR] = Node -> Next[s[start] - STARTCHAR];
                        t -> Next[s[i] - STARTCHAR] -> Father = t;
                        Node -> Next[s[start] - STARTCHAR] = t;
                        t -> Father = Node;
                        t -> Length = Node -> Length + t -> End - t -> Start + 1;
                        Insert(t, start + i - posbegin, s);
                        G = Node;
                        P = t;
                        return TYPE2;
                }
        }
}
int Select(int start, char s[], int type) {
        int result1, result2, result;
        if (type == TYPE1) {
                U = P -> Link;
                result = Insert(U, start + U -> Length, s);
                } else {
                U = G -> Link;
                if (G -> Link == G) {
                        W[0] = P -> Start + 1;
                        W[1] = P -> End;
                        W[2] = P -> End - P -> Start;
                        } else {
                        W[0] = P -> Start;
                        W[1] = P -> End;
                        W[2] = P -> End - P -> Start + 1;
                }
                if (W[2] == 0) {
                        V = G;
                        P -> Link = V;
                        result = Insert(V, start, s);
                        } else {
                        result1 = FindV(s);
                        result2 = Insert(V, start + V -> Length, s);
                        if (result1 == TYPE2) {
                                G = P -> Father;
                                result = result1;
                        } else result = result2;
                }
        }
        return result;
}
```

```
void BuildSuffixTrie(SuffixTrie & h, char s[]) {
        int i;
        int type;
        len = strlen(s);
        CreateNode(h.Next[s[0] - STARTCHAR]);
        h.Next[s[0] - STARTCHAR] -> Start = 0;
        h.Next[s[0] - STARTCHAR] -> End = len - 1;
        h.Next[s[0] - STARTCHAR] -> Father = & h;
        h.Next[s[0] - STARTCHAR] -> Length = h.Length + h.Next[s[0] - STARTCHAR] -> End - h.Next[s[0] - STARTCHAR] -> Start + 1;
        h.Flag = NOTLEAF;
        type = TYPE1;
        P = & h;
        for (i = 1; i < len; i++) type = Select(i, s, type);
}
void DeleteSuffixTrie(SuffixTrie * & Node) {
        int i;
        for (i = 0; i < NUM; i++) {
                if (Node -> Next[i] != NULL) {
                        DeleteSuffixTrie(Node -> Next[i]);
                        Node -> Next[i] = NULL;
                }
        }
        free(Node);
}
int FindString(int start, char s[]) {
        int result;
        int i;
        int temp;
        SuffixTrie * x;
        x = P -> Next[s[start] - STARTCHAR];
        result = P -> Length;
        if (x == NULL) {
                P = P -> Link;
                return result;
        }
        temp = 0;
        for (i = start; i < len2; i++) {
                if (x -> Start + i - start - temp > x -> End) {
                        temp = i - start;
                        P = x;
                        x = x -> Next[s[start + temp] - STARTCHAR];
                        if (x == NULL) break;
                }
                if (s[i] != str[x -> Start + i - start - temp]) break;
                result++;
        }
```

```
        P = P -> Link;
        return result;
}
int Search(SuffixTrie &h, char s[]) {
        int result;
        int i;
        int temp;
        len2 = strlen(s);
        result = 0;
        P = & head;
        for (i = 0; i < len2; i++) {
                temp = FindString(i + P -> Length, s);
                if (result < temp) result = temp;
                if (result >= len2 - i) break;
        }
        return result;
}
int Search2(SuffixTrie & h, char s[]) {
        int result;
        int i;
        int temp;
        len2 = strlen(s);
        result = 0;
        P = & head;
        result=FindString(P -> Length, s);
        return result;
}
int main() {
        int result;
        while (scanf("%s", str) != EOF) {
                Init(head, str);
                BuildSuffixTrie(head, str);
                scanf("%s", buf);
                result = Search(head, buf); //该 re 为最大公共子串长度
                printf("%d\n", result);
        }
}
```

5 Chapter 5 Computational Geometry 计算几何

5.1 判断线段相交

```
const double Eps=1e-10;
struct point {
        double x, y;
};
double xmul(point sp, point ep, point op){
        return (sp.x - op.x) * (ep.y - op.y) - (ep.x - op.x) * (sp.y - op.y);
}
```

```
bool inter(point a, point b, point c, point d){
        if ( min(a.x, b.x) > max(c.x, d.x) ||
        min(a.y, b.y) > max(c.y, d.y) ||
        min(c.x, d.x) > max(a.x, b.x) ||
        min(c.y, d.y) > max(a.y, b.y) ) return 0;
        double h, i, j, k;
        h = xmul(b,c,a);
        i = xmul(b,d,a);
        j = xmul(d,a,c);
        k = xmul(d,b,c);
        return h * i <= Eps && j * k <= Eps;
}
```

5.2 坐标旋转

- 顺时针旋转 v，若逆时针则 3、4 行中+，-互换

```
1 node rotate(node o,node a,double v){
2         node ret;
3         ret.x=(a.x-o.x)*cos(v)+(a.y-o.y)*sin(v);
4         ret.y=(a.y-o.y)*cos(v)-(a.x-o.x)*sin(v);
5         ret.x+=o.x;
6         ret.y+=o.y;
7         return ret;
8 }
```

5.3 二维凸包

- 复杂度 $O(N\ log\ N)$

```
struct point {
        double x, y;
};
bool mult(point sp, point ep, point op){
        return (sp.x - op.x) * (ep.y - op.y)>= (ep.x - op.x) * (sp.y - op.y);
}
bool operator < (const point &l, const point &r){
        return l.y < r.y || (l.y == r.y && l.x < r.x);
}
int graham(point pnt[], int n, point res[]){
        int i, len, k = 0, top = 1;
        sort(pnt, pnt + n);
        if (n == 0) return 0; res[0] = pnt[0];
        if (n == 1) return 1; res[1] = pnt[1];
        if (n == 2) return 2; res[2] = pnt[2];
        for (i = 2; i < n; i++) {
                while (top && mult(pnt[i], res[top], res[top-1]))
                top--;
                res[++top] = pnt[i];
        }
        len = top; res[++top] = pnt[n - 2];
        for (i = n - 3; i >= 0; i--) {
                while (top!=len && mult(pnt[i], res[top],res[top-1])) top--;
```

```
                res[++top] = pnt[i];
        }
        return top; // 返回凸包中点的个数
}
```

5.4 三维凸包

```
const double EPS = 1e-8;
inline int sgn(double x){
        if (fabs(x) < EPS) return 0;
        return x < 0 ? -1 : 1;
}
struct point {
        double x, y, z;
        point() {}
        point(double x, double y, double z) : x(x), y(y), z(z) {}
        point operator-(const point &a) const{
                return point(x - a.x, y - a.y, z - a.z);
        }
        point operator+(const point &a) const{
                return point(x + a.x, y + a.y, z + a.z);
        }
        point operator/(double k) const{
                return point(x / k, y / k, z / k);
        }
        point operator*(double k) const{
                return point(x * k, y * k, z * k);
        }
        bool operator!=(const point &a){
                return sgn(x - a.x) || sgn(y - a.y) || sgn(z - a.z);
        }
};
inline double dot(const point &a, const point &b){
        return a.x * b.x + a.y * b.y + a.z * b.z;
}
inline double abs(const point &a){
        return sqrt(dot(a, a));
}
inline point cross(const point &a, const point &b){
        return point(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
}
struct Convex {
        struct Tface {
                int a, b, c;
                bool ok;
                Tface() {}
                Tface(int a, int b, int c) : a(a), b(b), c(c), ok(true) {}
        } add, fc[MXN << 2];
        point pt[MXN];
```

```
	point center;
	int n, cnt, tot;
	int to[MXN][MXN];
	int q[MXN << 2], r;
	void init(){//这是初始化，勿忘
		cnt = tot = 0;
		r = 0;
		memset(to, -1, sizeof to);
	}
	void read(){
		for (int i = 0; i < n; ++i)
		scanf("%lf%lf%lf", &pt[i].x, &pt[i].y, &pt[i].z);
	}
	bool onLine(const point &a, const point &b, const point &c){
		return sgn(abs(cross(b - a, c - a))) == 0;
	}
	double ptof(const point &p, const Tface &f){
		point m = pt[f.b] - pt[f.a], n = pt[f.c] - pt[f.a], t = p - pt[f.a];
		return dot(cross(m, n), t);
	}
	int addFace(const Tface &f){
		int x = r > 0 ? q[--r] : cnt++;
		fc[x] = f;
		return x;
	}
	Tface delFace(int i){
		q[r++] = i;
		fc[i].ok = false;
		to[fc[i].a][fc[i].b] = to[fc[i].b][fc[i].c] = to[fc[i].c][fc[i].a] = -1;
		return fc[i];
	}
	void deal(int p, int a, int b){
		int f = to[a][b];
		if (f != -1 && fc[f].ok) {
			if (sgn(ptof(pt[p], fc[f])) >= 0) dfs(p, f);
			else {
				add = Tface(b, a, p);
				to[p][b] = to[a][p] = to[b][a] = addFace(add);
			}
		}
	}
	void dfs(int p, int cur){
		Tface f = delFace(cur);
		deal(p, f.b, f.a);
		deal(p, f.c, f.b);
		deal(p, f.a, f.c);
	}
```

```
	void calc(){//求三维凸包
		tot = 1;
		for (int i = 1; i < n; ++i) {
			if (tot == 1) {
				if (pt[0] != pt[i])
				swap(pt[tot++], pt[i]);
				} else if (tot == 2) {
				if (!onLine(pt[0], pt[1], pt[i])) {
					swap(pt[tot++], pt[i]);
					add = Tface(0, 1, 2);
				}
				} else if (tot == 3) {
				if (sgn(ptof(pt[i], add)) != 0)
				swap(pt[tot++], pt[i]);
			}
		}
		if (tot < 4) {//三维凸包上的点不到 4 个构不成三维凸包时
			/*----------DEAL WITH IT!---------*/
		}
		cnt = 0;
		for (int i = 0; i < 4; ++i) {
			add = Tface((i + 1) % 4, (i + 2) % 4, (i + 3) % 4);
			if (sgn(ptof(pt[i], add)) == 1)
			swap(add.b, add.c);
			to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;
			fc[cnt++] = add;
		}
		for (int i = 4; i < n; ++i)
		for (int j = 0; j < cnt; ++j)
		if (fc[j].ok && sgn(ptof(pt[i], fc[j])) == 1) {
			dfs(i, j);
			break;
		}
		int tmp = 0;
		for (int i = 0; i < cnt; ++i)
		if (fc[i].ok) fc[tmp++] = fc[i];
		cnt = tmp;
	}
	double getSur(){//表面积和
		double ret = 0;
		for (int i = 0; i < cnt; ++i)
		ret += abs(cross(pt[fc[i].b] - pt[fc[i].a], pt[fc[i].c] - pt[fc[i].a]));
		return 0.5 * ret;
	}
	double getVolFace(int f){
		point fp[3] = {pt[fc[f].c], pt[fc[f].b], pt[fc[f].a]};
		double ret = 0;
```

```
		for (int i = 0; i < 3; ++i) {
			point p1 = fp[i] - fp[0], p2 = fp[(i + 1) % 3] - fp[0];
			ret += dot(cross(p1, p2), fp[0]);
		}
		return ret / 6;
	}
	double getVol(){//总体积，加绝对值
		double ret = 0;
		for (int i = 0; i < cnt; ++i)
		ret += getVolFace(i);
		return ret;
	}
	bool same(int a, int b){
		return sgn(ptof(pt[fc[a].a], fc[b])) == 0 &&
		sgn(ptof(pt[fc[a].b], fc[b])) == 0 &&
		sgn(ptof(pt[fc[a].c], fc[b])) == 0;
	}
	int getFaceNum(){//不同面的个数
		int ret = 0;
		for (int i = 0; i < cnt; ++i) {
			bool flag = true;
			for (int j = 0; j < i; ++j)
			if (same(i, j)) {
				flag = false;
				break;
			}
			if (flag) ++ret;
		}
		return ret;
	}
	double getDistFace(int f){//点 center 到面 F 的距离
		point fp[3] = {pt[fc[f].c], pt[fc[f].b], pt[fc[f].a]};
		double A = (fp[1].y - fp[0].y) * (fp[2].z - fp[0].z)
		- (fp[1].z - fp[0].z) * (fp[2].y - fp[0].y);
		double B = (fp[1].z - fp[0].z) * (fp[2].x - fp[0].x)
		- (fp[1].x - fp[0].x) * (fp[2].z - fp[0].z);
		double C = (fp[1].x - fp[0].x) * (fp[2].y - fp[0].y)
		- (fp[1].y - fp[0].y) * (fp[2].x - fp[0].x);
		double D = -A * fp[0].x - B * fp[0].y - C * fp[0].z;
		return fabs(A * center.x + B * center.y + C * center.z + D)
		/ sqrt(A * A + B * B + C * C);
	}
	void calcCenter(){
		center = point(0, 0, 0);
		for (int i = 0; i < cnt; ++i) {
			point fp[3] = {pt[fc[i].c], pt[fc[i].b], pt[fc[i].a]};
			center = center
```

```
                        + ((fp[0] + fp[1] + fp[2]) / 4.0 * getVolFace(i));
                }
                center = center / getVol();
        }
        double getDist(){//点 center 到三维凸包的距离（到所有面的距离的最小值）
                calcCenter();//当点给定时不用进行 calcCenter
                double ret = 1e7;
                for (int i = 0; i < cnt; ++i)
                ret = min(ret, getDistFace(i));
                return ret;
        }
}
```

```
const double eps = 1e-8;
const double pi = acos(-1.0);
inline int Sign(double x){
        if (x<-eps) return -1;
        return x>eps;
}
inline double sqr(double x){
        return x*x;
}
struct Point{
        double x,y,z;
        Point(){x=y=z=0;}
        Point(double x,double y,double z):x(x),y(y),z(z){}
        inline double norm(){
                return x*x+y*y+z*z;
        }
        inline double length(){
                return sqrt(norm());
        }
        inline void read(){
                scanf("%lf%lf%lf",&x,&y,&z);
        }
};
inline Point operator +(const Point &a,const Point &b){return Point(a.x+b.x,a.y+b.y,a.
z+b.z);}
inline Point operator -(const Point &a,const Point &b){return Point(a.x-b.x,a.y-b.y,a.
z-b.z);}
inline bool operator <(const Point &a,const Point &b){return Sign(a.x-b.x)<0 || Sign(a
        .x-b.x)==0 && Sign(a.y-b.y)<0 || Sign(a.x-b.x)==0 && Sign(a.y-b.y)==0 && Sign(a.z-b
.z)<0;}
inline bool operator ==(const Point &a,const Point &b){return Sign(a.x-b.x)==0 && Sign
(a.y-b.y)==0 && Sign(a.z-b.z)==0;}
inline Point operator *(const Point &a,const double &b){return Point(a.x*b,a.y*b,a.z*b
);}
```

```
inline Point operator /(const Point &a,const double &b){return Point(a.x/b,a.y/b,a.z/b
);}
inline Point det(const Point &a,const Point &b){
        return Point(a.y*b.z-a.z*b.y, -( a.x*b.z-a.z*b.x ),a.x*b.y-a.y*b.x);
}
inline double dot(const Point &a,const Point &b){
        return a.x*b.x+a.y*b.y+a.z*b.z;
}
//====================================================
int mark[1005][1005];
Point info[1005];
int n,cnt;
double mix(const Point &a,const Point &b,const Point &c){
        return dot(a,det(b,c));
}
double area(int a,int b,int c){
        return (info[b]-info[a],info[c]-info[a]).length();
}
double volume(int a,int b,int c,int d){
        return mix(info[b]-info[a],info[c]-info[a],info[d]-info[a]);
}
struct Face{
        int a,b,c;
        Face(){}
        Face(int a,int b,int c):a(a),b(b),c(c){}
        int& operator [](int k){
                if (k==0) return a;
                if (k==1) return b;
                return c;
        }
};
vector <Face> face;
inline void insert(int a,int b,int c){
        face.push_back(Face(a,b,c));
}
inline void add(int v){
        vector <Face> tmp;
        int a,b,c;
        ++cnt;
        for (int i=0;i<face.size();++i){
                a=face[i][0];
                b=face[i][1];
                c=face[i][2];
                if (Sign(volume(v,a,b,c))<0)
                mark[a][b]=mark[b][a]=mark[b][c]=mark[c][b]=mark[c][a]=mark[a][c]=cnt;
                else
                tmp.push_back(face[i]);
```

```
        }
        face=tmp;
        for (int i=0;i<tmp.size();++i){
                a=face[i][0];
                b=face[i][1];
                c=face[i][2];
                if (mark[a][b]==cnt) insert(b,a,v);
                if (mark[b][c]==cnt) insert(c,b,v);
                if (mark[c][a]==cnt) insert(a,c,v);
        }
}
inline int Find(){
        for (int i=2;i<n;++i){
                Point ndir=det(info[0]-info[i],info[1]-info[i]);
                if (ndir==Point()) continue;
                swap(info[i],info[2]);
                for (int j=i+1;j<n;++j){
                        if (Sign(volume(0,1,2,j))!=0){
                                swap(info[j],info[3]);
                                insert(0,1,2);
                                insert(0,2,1);
                                return 1;
                        }
                }
        }
        return 0;
}
int main(){
        for (;scanf("%d",&n)==1;){
                for (int i=0;i<n;++i) info[i].read();
                sort(info,info+n);
                n=unique(info,info+n)-info;
                face.clear();
                random_shuffle(info,info+n);
                if (Find()){
                        memset(mark,0,sizeof(mark));
                        cnt=0;
                        for (int i=3;i<n;++i) add(i);
                        Point ans(0,0,0),o=info[0];
                        double total=0;//  total/6 就是体积
                        for (int i=0;i<face.size();++i){
                                double volume=fabs(mix(info[face[i][0]]-o,info[face[i][1]]-o,info[face[i][2]]-o));
                                total+=volume;
                                ans=ans+(o+info[face[i][0]]+info[face[i][1]]+info[face[i][2]])/4.0*
```

```
                                volume;
                        }
                        ans=ans/total;
                        double len=(ans-info[0]).length();
                        for (int i=0;i<face.size();++i){
                                Point ndir=det(info[face[i][1]]-info[face[i][0]],info[face[i][2]]-info[face[i][0]]);
                                len=min(len,fabs(dot(ans-info[face[i][0]],ndir))/ndir.length());
                        }
                        printf("%.3f\n",len);
                }
        }
        return 0;
}
```

5.5 半平面交

- 复杂度 $O(N\ log\, N)$

```
#include<cstdio>
#include<vector>
#include<cmath>
#include<algorithm>
using namespace std;
const double eps=1e-10,big=10000.0;
const int maxn = 20010;
struct point{ double x,y; };
struct polygon{ //存放最后半平面交中相邻边的交点，就是一个多边形的所有点
        int n;
        point p[maxn];
};
struct line{ //半平面，这里是线段
        point a,b;
};
double at2[maxn];
int ord[maxn],dq[maxn+1],lnum,n;
polygon pg;
line ls[maxn]; //半平面集合
inline int sig(double k) { //判是不是等于 0，返回-1，0，1，分别是小于，等于，大于
        return (k < -eps)? -1: k > eps;
}
//叉积>0 代表在左边，<0 代表在右边，=0 代表共线
//e 是否在 o->s 的左边 onleft(sig(multi))>=0
inline double multi(point o, point s, point e){//构造向量，然后返回叉积
        return (s.x-o.x)*(e.y-o.y)-(e.x-o.x)*(s.y-o.y);
}
//直线求交点
point isIntersected(point s1, point e1, point s2, point e2){
```

```
        double dot1,dot2;
        point pp;
        dot1 = multi(s2,e1,s1); dot2 = multi(e1,e2,s1);
        pp.x = (s2.x * dot2 + e2.x * dot1) / (dot2 + dot1);
        pp.y = (s2.y * dot2 + e2.y * dot1) / (dot2 + dot1);
        return pp;
}
//象限排序
inline bool cmp(int u,int v) {
        if(sig(at2[u]-at2[v])==0)
        return sig(multi(ls[v].a,ls[v].b,ls[u].b))>=0;
        return at2[u]<at2[v];
}
//判断半平面的交点在当前半平面外
bool judgein(int x,int y,int z){
        point pnt = isIntersected(ls[x].a, ls[x].b, ls[y].a, ls[y].b); //求交点
        return sig(multi(ls[z].a,ls[z].b,pnt)) < 0;
        //判断交点位置，如果在右面，返回 true，如果要排除三点共线，改成<=
}
//半平面交
void HalfPlaneIntersection() { //预处理
        int n = lnum , tmpn , i;
        /* 对于 atan2(y,x)
        结果为正表示从 X 轴逆时针旋转的角度，结果为负表示从 X 轴顺时针旋转的角度。
        atan2(a, b) 与 atan(a/b)稍有不同，atan2(a,b)的取值范围介于 -pi 到 pi 之间(不包括 -pi),
        而 atan(a/b)的取值范围介于-pi/2 到 pi/2 之间 (不包括±pi)*/
        for(i = 0 ; i < n ; i ++){ //atan2(y,x)求出每条线段对应坐标系的角度
                at2[i] = atan2( ls[i].b.y - ls[i].a.y, ls[i].b.x - ls[i].a.x);
                ord[i] = i;
        }
        sort(ord , ord + n , cmp);
        for (i = 1 , tmpn = 1 ; i < n ; i++) //处理重线的情况
        if( sig(at2[ord[i-1]] - at2[ord[i]]) != 0 ) ord[tmpn++] = ord[i];
        n = tmpn;
        //圈地
        int bot = 1,top = bot + 1; //双端栈，bot 为栈底，top 为栈顶
        dq[bot] = ord[0]; dq[top] = ord[1]; //先压两根线进栈
        for(i = 2 ; i < n ; i ++){
                //bot < top 表示要保证栈里至少有 2 条线段，如果剩下 1 条，就不继续退栈
                //judgein，判断如果栈中两条线的交点如果在当前插入先的右边，就退栈
                while( bot < top && judgein(dq[top-1] , dq[top] , ord[i]) ) top--;
                //对栈顶要同样的操作
                while( bot < top && judgein(dq[bot+1] , dq[bot] , ord[i]) ) bot++;
                dq[++top] = ord[i];
        }
        //最后还要处理一下栈里面存在的栈顶的线在栈底交点末尾位置，或者栈顶在栈尾两条线的右边
        while( bot < top && judgein(dq[top-1] , dq[top] , dq[bot]) ) top--;
```

```
        while( bot < top && judgein(dq[bot+1] , dq[bot] , dq[top]) ) bot++;
        //最后一条线是重合的
        dq[--bot] = dq[top];
        //求多边形
        pg.n = 0;
        for(i = bot + 1;i <= top ; i++) //求相邻两条线的交点
        pg.p[pg.n++] = isIntersected(ls[dq[i-1]].a, ls[dq[i-1]].b, ls[dq[i]].a,ls[dq[i]].b);
}
inline void add(double a,double b,double c,double d){//添加线段
        ls[lnum].a.x = a; ls[lnum].a.y = b;
        ls[lnum].b.x = c; ls[lnum].b.y = d;
        lnum++;
}
int main(){
        int n,i;
        scanf("%d",&n);
        double a,b,c,d;
        for(i = 0 ;i < n ; i ++) {
                //输入代表一条向量(x = (c - a),y = (d - b));
                scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
                add(a,b,c,d);
        }
        //下面是构造一个大矩形边界
        add(0,0,big,0);//down
        add(big,0,big,big);//right
        add(big,big,0,big);//up
        add(0,big,0,0);//left
        HalfPlaneIntersection(); //求半平面交/对 pg 求
        double area = 0;
        n = pg.n;
        ///最后多边形的各个点保存在 pg 里面
        for(i = 0 ; i < n ; i ++)
        area += pg.p[i].x * pg.p[(i+1)%n].y - pg.p[(i+1)%n].x * pg.p[i].y;
        //x1 * y2 - x2 * y1 用叉积求多边形面积
        area=fabs(area)/2.0; //所有面积应该是三角形面积之和，而叉积求出来的是四边形的面积和，所以要除 2
        printf("%.1f\n",area);
        return 0;
}
```

5.6 圆的面积并

- 复杂度 $O(N^2 \, log \, N)$

```
typedef complex<double> point;
typedef pair<point, double> circle;
const double PI = acos(-1);
int n;
circle c[MXN];
pair<double, int> s[MXN << 1];
```

```
int tot, cnt;
inline bool cmp(const circle &a, const circle &b){
        return a.second > b.second;
}
inline double cross(const point &a, const point &b) { return imag(conj(a) * b); }
inline void circleIntersect(const circle &a, const circle &b){
        point o1 = a.first, o2 = b.first;
        double r1 = a.second, r2 = b.second;
        double d = abs(o1 - o2);
        if (d >= r1 + r2) return;
        double alpha = acos((d * d + r1 * r1 - r2 * r2) / (2 * d * r1));
        double l = arg((o2 - o1) * exp(point(0, -alpha)));
        double r = arg((o2 - o1) * exp(point(0, +alpha)));
        if (l > r) --cnt;
        s[tot++] = make_pair(l, -1);
        s[tot++] = make_pair(r, +1);
}
inline double archArea(const point &o, double r, double t1, double t2){
        point p1 = o + point(r, 0) * exp(point(0, t1));
        point p2 = o + point(r, 0) * exp(point(0, t2));
        double alpha = t2 - t1;
        return 0.5 * cross(p1, p2) + 0.5 * r * r * (alpha - sin(alpha));
}
double calc(circle c[], int n){
        sort(c, c + n, cmp);
        int N = 0;
        for (int i = 0, j; i < n; ++i) {
                for (j = 0; j < N; ++j)
                if (abs(c[i].first - c[j].first) <= c[j].second - c[i].second)
                break;
                if (j == N) c[N++] = c[i];
        }
        n = N;
        double ret = 0;
        for (int i = 0; i < n; ++i) {
                tot = cnt = 0;
                s[tot++] = make_pair(-PI, +1);
                s[tot++] = make_pair(+PI, -1);
                for (int j = 0; j < n; ++j)
                if (i != j) circleIntersect(c[i], c[j]);
                sort(s, s + tot);
                double now = - PI;
                for (int j = 0; j < tot; ++j) {
                        cnt += s[j].second;
                        if (cnt == 0 && s[j].second == -1)
                        ret += archArea(c[i].first, c[i].second, now, s[j].first);
                        now = s[j].first;
```

```
			}
		}
		return ret;
}
```

5.7 Delaunay 三角形剖分

```
#define OTHER(e, p) ((e)->oi == p ? (e)->dt : (e)->oi)
#define NEXT(e, p) ((e)->oi == p ? (e)->on : (e)->dn)
#define PREV(e, p) ((e)->oi == p ? (e)->op : (e)->dp)
#define V(p1, p2, u, v) (u = p2->x - p1->x, v = p2->y - p1->y)
#define C2(u1, v1, u2, v2) ((u1) * (v2) - (v1) * (u2))
#define C3(p1, p2, p3) ((p2->x - p1->x) * (p3->y - p1->y) - (p2->y - p1->y) * (p3->x -
p1->x))
#define DOT(u1, v1, u2, v2) ((u1) * (u2) + (v1) * (v2))
#define SQR(x) ((x) * (x))
#define MXN 100007
struct point {
	long long x, y;
	int id;
	struct edge *in;
	bool operator < (const point &a) const {
		return x < a.x || (x == a.x && y < a.y);
	}
};
struct edge {
	point *oi, *dt;
	edge *on, *op, *dn, *dp;
};
struct graphEdge {
	int u, v;
} gE[3 * MXN];
int n, m;
point p[MXN], *q[MXN];
edge Mem[3 * MXN], *elist[3 * MXN];
int nfree;
void allocMemory(){
	nfree = 3 * n;
	edge *e = Mem;
	for (int i = 0; i < nfree; ++i) elist[i] = e++;
}
void splice(edge *a, edge *b, point *v){
	edge *next;
	if (a->oi == v) next = a->on, a->on = b;
	else next = a->dn, a->dn = b;
	if (next->oi == v) next->op = b;
	else next->dp = b;
	if (b->oi == v) b->on = next, b->op = a;
	else b->dn = next, b->dp = a;
```

```
}
edge *makeEdge(point *u, point *v){
        edge *e = elist[--nfree];
        e->on = e->op = e->dn = e->dp = e;
        e->oi = u; e->dt = v;
        if (u->in == NULL) u->in = e;
        if (v->in == NULL) v->in = e;
        return e;
}
edge *join(edge *a, point *u, edge *b, point *v, bool side){
        edge *e = makeEdge(u, v);
        if (side) {
                if (a->oi == u) splice(a->op, e, u);
                else splice(a->dp, e, u);
                splice(b, e, v);
                } else {
                splice(a, e, u);
                if (b->oi == v) splice(b->op, e, v);
                else splice(b->dp, e, v);
        }
        return e;
}
void remove(edge *e){
        point *u = e->oi, *v = e->dt;
        if (u->in == e) u->in = e->on;
        if (v->in == e) v->in = e->dn;
        if (e->on->oi == u) e->on->op = e->op;
        else e->on->dp = e->op;
        if (e->op->oi == u) e->op->on = e->on;
        else e->op->dn = e->on;
        if (e->dn->oi == v) e->dn->op = e->dp;
        else e->dn->dp = e->dp;
        if (e->dp->oi == v) e->dp->on = e->dn;
        else e->dp->dn = e->dn;
        elist[nfree++] = e;
}
void makeGraph(){
        for (int i = 0; i < n; ++i) {
                point *u = p + i;
                edge *start = u->in, *e = u->in;
                do {
                        point *v = OTHER(e, u);
                        if (u < v) {
                                gE[m].u = u - p;
                                gE[m++].v = v - p;
                        }
                        e = NEXT(e, u);
```

```
            } while (e != start);
        }
}
void lowTan(edge *eL, point *oL, edge *eR, point *oR, edge **lLow, point **OL, edge **rLow, point **OR){
        point *dL = OTHER(eL, oL), *dR = OTHER(eR, oR);
        while (true) {
                if (C3(oL, oR, dL) < 0) {
                        eL = PREV(eL, dL);
                        oL = dL; dL = OTHER(eL, oL);
                }
                else if (C3(oL, oR, dR) < 0) {
                        eR = NEXT(eR, dR);
                        oR = dR; dR = OTHER(eR, oR);
                }
                else break;
        }
        *OL = oL; *OR = oR;
        *lLow = eL; *rLow = eR;
}
void merge(edge *lr, point *s, edge *rl, point *u, edge **tan){
        point *O, *D, *OR, *OL;
        edge *B, *L, *R;
        lowTan(lr, s, rl, u, &L, &OL, &R, &OR);
        *tan = B = join(L, OL, R, OR, false);
        O = OL; D = OR;
        do {
                edge *El = NEXT(B, O), *Er = PREV(B, D), *next, *prev;
                point *l = OTHER(El, O), *r = OTHER(Er, D);
                double l1, l2, l3, l4, r1, r2, r3, r4;
                V(l, O, l1, l2); V(l, D, l3, l4); V(r, O, r1, r2); V(r, D, r3, r4);
                double cl = C2(l1, l2, l3, l4), cr = C2(r1, r2, r3, r4);
                bool BL = cl > 0, BR = cr > 0;
                if (!BL && !BR) break;
                double cotL, cotR, u1, v1, u2, v2, N1, P1, cotN, cotP;
                if (BL) {
                        double dl = DOT(l1, l2, l3, l4);
                        cotL = dl / cl;
                        do {
                                next = NEXT(El, O);
                                V(OTHER(next, O), O, u1, v1); V(OTHER(next, O), D, u2, v2);
                                N1 = C2(u1, v1, u2, v2);
                                if (!(N1 > 0)) break;
                                cotN = DOT(u1, v1, u2, v2) / N1;
                                if (cotN > cotL) break;
                                remove(El);
                                El = next;
```

```
                cotL = cotN;
            } while (true);
        }
        if (BR) {
            double dr = DOT(r1, r2, r3, r4);
            cotR = dr / cr;
            do {
                prev = PREV(Er, D);
                V(OTHER(prev, D), O, u1, v1); V(OTHER(prev, D), D, u2, v2);
                P1 = C2(u1, v1, u2, v2);
                if (!(P1 > 0)) break;
                cotP = DOT(u1, v1, u2, v2) / P1;
                if (cotP > cotR) break;
                remove(Er);
                Er = prev;
                cotR = cotP;
            } while (true);
        }
        l = OTHER(El, O); r = OTHER(Er, D);
        if (!BL || (BL && BR && cotR < cotL)) {
            B = join(B, O, Er, r, false);
            D = r;
            } else {
            B = join(El, l, B, D, false);
            O = l;
        }
    } while (true);
}
void divide(int s, int t, edge **L, edge **R){
    int n = t - s + 1;
    if (n == 2) {
        *L = *R = makeEdge(q[s], q[t]);
    }
    else if (n == 3) {
        edge *a = makeEdge(q[s], q[s + 1]), *b = makeEdge(q[s + 1], q[t]);
        splice(a, b, q[s + 1]);
        double v = C3(q[s], q[s + 1], q[t]);
        if (v > 0.0) {
            join(a, q[s], b, q[t], false);
            *L = a; *R = b;
        }
        else if (v < 0.0) {
            *L = *R = join(a, q[s], b, q[t], true);
        }
        else { *L = a; *R = b; }
    }
    else if (n > 3) {
```

```
            edge *ll, *lr, *rl, *rr, *tan;
            int mid = (s + t) / 2;
            divide(s, mid, &ll, &lr); divide(mid + 1, t, &rl, &rr);
            merge(lr, q[mid], rl, q[mid + 1], &tan);
            if (tan->oi == q[s]) ll = tan;
            if (tan->dt == q[t]) rr = tan;
            *L = ll; *R = rr;
        }
}
long long dist[MXN];
void work(){
        for (int i = 0; i < n; ++i)
        dist[i] = (long long)2e18;
        for (int i = 0; i < m; ++i) {
                long long d = SQR(p[gE[i].u].x - p[gE[i].v].x) + SQR(p[gE[i].u].y - p[gE[i].v
                ].y);
                dist[p[gE[i].u].id] = min(dist[p[gE[i].u].id], d);
                dist[p[gE[i].v].id] = min(dist[p[gE[i].v].id], d);
        }
}
int main(){
        int T;
        scanf("%d", &T);
        while (T--) {
                scanf("%d", &n);
                allocMemory();
                for (int i = 0; i < n; ++i) {
                        scanf("%lld%lld", &p[i].x, &p[i].y);
                        p[i].id = i;
                        p[i].in = NULL;
                }
                sort(p, p + n);
                for (int i = 0; i < n; ++i) q[i] = p + i;
                edge *L, *R;
                divide(0, n - 1, &L, &R);
                m = 0;
                makeGraph();
                work();
                for (int i = 0; i < n; ++i)
                printf("%lld\n", dist[i]);
        }
}
```

5.8 最小圆覆盖

```
const int MXN=1000;
const double EPS=1e-7;
using namespace std;
struct Point{
```

```
        double x, y;
};
template < typename T >
T sqr(T x){
        return x*x;
}
double dist(Point A,Point B){
        return sqrt(sqr(A.x-B.x)+sqr(A.y-B.y));
}
Point p[MXN], center;
double minR;
bool inside(const Point &p){
        return dist(p, center) < minR + EPS;
}
void getPoint(const Point &a, const Point &b){
        center.x = (a.x + b.x) * 0.5;
        center.y = (a.y + b.y) * 0.5;
        minR = dist(a, b) * 0.5;
}
double area(const Point &a, const Point &b, const Point &c){
        return fabs((b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y)) * 0.5;
}
void calc(const Point &a, const Point &b, double sinA, double cosA, double k){
        // counter-clock wise
        double x = (b.x - a.x) * k;
        double y = (b.y - a.y) * k;
        center.x = a.x + cosA * x - sinA * y;
        center.y = a.y + cosA * y + sinA * x;
}
void getCenter(const Point &a, const Point &b, const Point &c){
        // outer circle radius = abc / 4s
        minR = dist(a, b) * dist(a, c) * dist(b, c) / (4 * area(a, b, c));
        double len = dist(a, b) * 0.5;
        double tmpL = sqrt(minR * minR - len * len);
        double k = minR / (len * 2);
        calc(a, b, tmpL / minR, len / minR, k);
        if (inside(c)) return;
        calc(a, b, -tmpL / minR, len / minR, k);
}
void work(){
        int n;
        scanf("%d", &n);
        for (int i = 0; i < n; ++i)
        scanf("%lf%lf", &p[i].x, &p[i].y);
        random_shuffle(p, p + n);
        getPoint(p[0], p[1]);
        for (int i = 2; i < n; ++i) {
```

```
			if (inside(p[i])) continue;
			getPoint(p[0], p[1]);
			for (int j = 0; j < i; ++j) {
				if (inside(p[j])) continue;
				getPoint(p[i], p[j]);
				for (int k = 0; k < j; ++k)
				if (!inside(p[k]))
				getCenter(p[i], p[j], p[k]);
			}
		}
		printf("%.2f\n", minR);//半径
		printf("%.2f %.2f\n", center.x, center.y);//圆心
}
```

5.9 圆与多边形交

```
const int MAXN=1000;
const double EPS=1e-7;
using namespace std;
struct point{
		double first,second;
		point(){
			first=0;
			second=0;
		}
		point(double x,double y){
			first=x;
			second=y;
		}
		bool operator ==(point b) const{
			return first==b.first && second==b.second;
		}
};
struct polygon{
		int n;
		point points[MAXN];
		int size(){
			return n;
		}
		point& operator [](int i){
			return points[i];
		}
		void resize(int x){
			n=x;
		}
};
const double eps=1e-7;
const double pi=acos(-1.0);
inline double length(point a){
```

```
	return sqrt(a.first*a.first+a.second*a.second);
}
inline point operator +(point a,point b){
	return point(a.first+b.first,a.second+b.second);
}
inline point operator -(point a,point b){
	return point(a.first-b.first,a.second-b.second);
}
inline point operator *(point a,double t){
	return point(a.first*t,a.second*t);
}
inline point operator /(point a,double t){
	return point(a.first/t,a.second/t);
}
inline double operator *(point a,point b){
	return a.first*b.first+a.second*b.second;
}
inline double operator ^(point a,point b){
	return a.first*b.second-a.second*b.first;
}
inline double angle(point a,point b){
	double ans=fabs(atan2(a.second,a.first)-atan2(b.second,b.first));
	if (ans>pi) ans=pi*2-ans;
	return ans;
}
const point no_solution(0,0);
point intersection_to_circle(point p,point q,double r){
	point u=q-p;
	double A=u*u;
	double B=2*(p*u);
	double C=p*p-r*r;
	double delta=B*B-4*A*C;
	if (delta<-eps) return no_solution;
	else if (fabs(delta)<eps){
		double t=-B/(2*A);
		if (t<-eps || t>1.0+eps)
		return no_solution;
		return p+u*t;
	}
	else{
		double t1=(-B-sqrt(delta))/(2.0*A);
		double t2=(-B+sqrt(delta))/(2.0*A);
		double t;
		bool flag1=(t1>-eps && t1<1.0+eps);
		bool flag2=(t2>-eps && t2<1.0+eps);
		if (!flag1 && !flag2) return no_solution;
		else if (!flag1) t=t2;
```

```
        else t=t1;
        return p+u*t;
    }
}
point point_rotate(point a,point o,double theta){
    double x=(a-o).first;
    double y=(a-o).second;
    point ans(x*cos(theta)-y*sin(theta),x*sin(theta)+y*cos(theta));
    return ans+o;
}
polygon polygon_rotate(polygon p,point o,double theta){
    for (int i=0;i<=p.size();i++)
    p[i]=point_rotate(p[i],o,theta);
    return p;
}
double intersection_area(polygon p,point c,double r){
    double ans=0;
    for (int i=0;i<p.size();i++){
        point a=p[i]-c;
        point b=p[i+1]-c;
        double la=length(a);
        double lb=length(b);
        double now_area;
        int now_sign;
        if ((a^b)>0) now_sign=1;
        else now_sign=-1;
        double phi=angle(a,b);
        if (la<r+eps && lb<r+eps){
            now_area=fabs(a^b);
        }
        else if (la<r+eps){
            point c=intersection_to_circle(b,a,r);
            double alpha=angle(a,c);
            double beta=phi-alpha;
            now_area=la*r*sin(alpha)+r*r*beta;
        }
        else if (lb<r+eps){
            point c=intersection_to_circle(a,b,r);
            double alpha=angle(b,c);
            double beta=phi-alpha;
            now_area=lb*r*sin(alpha)+r*r*beta;
        }
        else{
            point c=intersection_to_circle(a,b,r);
            point d=intersection_to_circle(b,a,r);
            if (c==no_solution)
            now_area=r*r*phi;
```

```
			else{
				double alpha=angle(c,d);
				double beta=phi-alpha;
				now_area=r*r*sin(alpha)+r*r*beta;
			}
		}
		now_area*=now_sign;
		ans+=now_area;
	}
	return ans*0.5;
}
double xx0,yy0,v0,ang,t,g,r;
int main(){
	while (scanf("%lf%lf%lf", &xx0, &yy0, &r)!=EOF){
		int n;
		scanf("%d",&n);
		polygon pol;
		pol.n=n;
		for (int i=0;i<n;i++){
			double x,y;
			scanf("%lf%lf",&x,&y);
			pol[i]=point(x,y);
		}
		pol[n]=pol[0];
		double ans=intersection_area(pol,point(xx0,yy0),r);
		printf("%.2f\n",fabs(ans));
	}
}
```

6 Chapter 6 Miscellaneous 杂题

6.1 树的分治

- $O(N\ log\ N)$
- 给一棵树，求距离<=m 的点对数

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int MXN = 30007;
struct arc {
	int v, d, l;
	arc *next;
	arc() {}
	arc(int v, int d, int l, arc *next) : v(v), d(d), l(l), next(next) {}
} *adj[MXN], mem[MXN << 1];
int memCnt;
inline void addEdge(int u, int v, int d, int l){
```

```
        adj[u] = &(mem[memCnt++] = arc(v, d, l, adj[u]));
        adj[v] = &(mem[memCnt++] = arc(u, d, l, adj[v]));
}
int n, m;
int size[MXN], maxSize[MXN];
int d[MXN], l[MXN];
int cnt, tot;
int list[MXN], ss[MXN], tt[MXN];
int c[MXN << 1];
int data[MXN << 1];
bool vis[MXN];
inline void update(int x, int v){
        x = lower_bound(data, data + cnt, x) - data + 1;
        for (int i = x; i <= cnt; i += i & -i)
        c[i] = max(c[i], v);
}
inline int query(int x){
        int ret = 0;
        x = lower_bound(data, data + cnt, x) - data + 1;
        for (int i = x; i > 0; i -= i & -i)
        ret = max(ret, c[i]);
        return ret;
}
void dfs1(int u, int fa, int n, int &r){
        size[u] = 1; maxSize[u] = 0;
        for (arc *p = adj[u]; p; p = p->next)
        if (!vis[p->v] && p->v != fa) {
                dfs1(p->v, u, n, r);
                size[u] += size[p->v];
                maxSize[u] = max(maxSize[u], size[p->v]);
        }
        maxSize[u] = max(maxSize[u], n - size[u]);
        if (r < 0 || maxSize[u] < maxSize[r]) r = u;
}
void dfs2(int u, int fa){
        size[u] = 1;
        ss[u] = tot;
        list[tot++] = u;
        for (arc *p = adj[u]; p; p = p->next)
        if (!vis[p->v] && p->v != fa) {
                d[p->v] = d[u] + p->d;
                l[p->v] = l[u] + p->l;
                dfs2(p->v, u);
                size[u] += size[p->v];
        }
        tt[u] = tot;
        data[cnt++] = d[u];
```

```
        if (m - d[u] >= 0) data[cnt++] = m - d[u];
}
int solve(int u, int n){
        int r = -1;
        dfs1(u, -1, n, r);
        u = r;
        vis[u] = true;
        tot = cnt = 0;
        d[u] = l[u] = 0;
        dfs2(u, -1);
        sort(data, data + cnt);
        cnt = unique(data, data + cnt) - data;
        memset(c, 0, (cnt + 1) << 2);
        int ret = 0;
        for (arc *p = adj[u]; p; p = p->next) {
                int v = p->v;
                if (!vis[v]) {
                        for (int i = ss[v]; i < tt[v]; ++i)
                        if (m - d[list[i]] >= 0)
                        ret = max(ret, query(m - d[list[i]]) + l[list[i]]);
                        for (int i = ss[v]; i < tt[v]; ++i)
                        update(d[list[i]], l[list[i]]);
                }
        }
        for (arc *p = adj[u]; p; p = p->next)
        if (!vis[p->v])
        ret = max(ret, solve(p->v, size[p->v]));
        return ret;
}
int main(){
        int T;
        scanf("%d", &T);
        while (T--) {
                memset(adj, 0, sizeof adj);
                memCnt = 0;
                scanf("%d%d", &n, &m);
                for (int i = 1; i < n; ++i) {
                        int u, v, d, l;
                        scanf("%d%d%d%d", &u, &v, &d, &l);
                        addEdge(u, v, d, l);
                }
                memset(vis, 0, sizeof vis);
                printf("%d\n", solve(1, n));
        }
}
```

6.2 矩形面积并

- 线段树以线段为端点而不是以点为端点

- TT　　矩形坐标类型
- MaxN　　矩形个数
- tree[]　　线段覆盖长度
- add[]　　覆盖次数
- s[]　　离散后对应的 y 值
- p[]　　矩形变线段的插入
- a[]　　矩形的左下角(x,y)和右上角(xx,yy)点的坐标
- n　　矩形个数
- M　　矩形变线段后线段条数
- N　　离散后 y 轴点的个数

```cpp
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <string>
#include <cstring>
#include <map>
#define TT double
const int MaxN=1000+50;
using namespace std;
struct node{
	TT x,y,xx,yy;
};
struct Node{
	int l,r,k;
	TT x;
};
node a[MaxN];
int N,add[MaxN<<3],M,n;
TT tree[MaxN<<3],s[MaxN<<1];
Node p[MaxN<<1];
map < TT , int > Map;
int cmp(Node A,Node B){
	return A.x<B.x;
}
void Ins(int y,int yy,TT x,int k){
	p[M].l=y;
	p[M].r=yy;
	p[M].x=x;
	p[M++].k=k;
}
void precess(){
	Map.clear();
	for (int i=0;i<n;i++)
	Map[a[i].y]=1,Map[a[i].yy]=1;
	map < TT , int > :: iterator it;
	it=Map.begin(),s[N=0]=0;
	while (it!=Map.end()){
```

```
                s[++N]=it->first;
                it->second=N;
                it++;
        }
        M=0;
        for (int i=0;i<n;i++){
                Ins(Map[a[i].y],Map[a[i].yy],a[i].x,1);
                Ins(Map[a[i].y],Map[a[i].yy],a[i].xx,-1);
        }
}
void build(int k,int l,int r){
        tree[k]=add[k]=0;
        if (l+1==r) return;
        int Mid=(l+r)>>1;
        build(k+k,l,Mid);
        build(k+k+1,Mid,r);
}
void Add(int k,int L,int R,int l,int r,int v){
        if (l<=L && R<=r){
                add[k]+=v;
                if (add[k]) tree[k]=s[R]-s[L];else
                if (L+1==R) tree[k]=0;else
                tree[k]=tree[k+k]+tree[k+k+1];
                return;
        }
        int Mid=(L+R)>>1;
        if (l<Mid) Add(k+k,L,Mid,l,r,v);
        if (r>Mid) Add(k+k+1,Mid,R,l,r,v);
        if (add[k]) tree[k]=s[R]-s[L];else
        tree[k]=tree[k+k]+tree[k+k+1];
}
double Calc(){
        precess();
        if (N<2) return 0.0;
        build(1,1,N);
        sort(p,p+M,cmp);
        TT ret=0;
        TT now;
        for (int i=0;i<M;i++){
                if (i) ret+=tree[1]*(p[i].x-now);
                now=p[i].x;
                if (p[i].l<p[i].r) Add(1,1,N,p[i].l,p[i].r,p[i].k);
        }
        return ret;
}
int main(){
        int tt=0;
```

```
        while (cin >> n && n){
                for (int i=0;i<n;i++)
                cin >> a[i].x >> a[i].y >> a[i].xx >> a[i].yy;
                printf("Test case #%d\n", ++tt);
                printf("Total explored area: %.2lf\n\n", Calc());
        }
        return 0;
}
```

6.3 最长回文子串

```
int Cal(char s[]){
        int i,j,k,n,p[100000];
        char str[100000];
        n=strlen(s);
        str[0]='$',str[1]='#';
        for (int i=0;i<n;i++){
                str[i*2+2]=s[i];
                str[i*2+3]='#';
        }
        n=n*2+2,str[n]=0;
        int mx=0,id;
        for (int i=1;i<n;i++){
                if (mx>i) p[i]=min(p[2*id-i],p[id]+id-i);else
                p[i]=1;
                for (;str[i+p[i]]==str[i-p[i]];p[i]++);
                if (p[i]+i>mx) mx=p[i]+i,id=i;
        }
        int ret=0;
        for (int i=0;i<n;i++)
        ret=max(ret,p[i]);
        return ret-1;
}
```

Part III
Ohter

1 Chapter 1 STL

1.1 map

- begin() 返回指向 map 头部的迭代器
- clear() 删除所有元素
- count() 返回指定元素出现的次数
- empty() 如果 map 为空返回 true
- end() 返回指向 map 末尾的迭代器
- erase() 删除一个元素
- find() 查找一个元素
- insert() 插入元素
- lower_bound() 返回键值>=给定元素的第一个位置
- size() 返回 map 中元素个数
- swap() 交换两个 map
- upper_bound() 返回键值>给定元素的第一个位置

1.2 vector

- push_back(t)在容器的最后添加一个值为 t 的数据，容器的 size 变大
- size()返回容器中数据的个数
- empty()判断 vector 是否为空
- insert(pointer,number, content)向 v 中 pointer 指向的位置插入 number 个 content 的内容
- pop_back()删除容器的末元素，并不返回该元素
- erase(pointer1,pointer2)删除 pointer1 到 pointer2 中间（包括 pointer1 所指）的元素
- clear()删除容器中的所有元素

1.3 sstream

```
int main(){
        char ch[1000];
        gets(ch);
        stringstream ssin(ch);
        string st;
        while (ssin >> st) cout << st << endl;
        return 0;
}
```

2 Chapter 2 JAVA

2.1 import java.math.*

2.1.1 BigInteger 类

- abs() 返回其值是此 BigInteger 的绝对值的 BigInteger
- add(BigInteger val) 返回其值为 (this + val) 的 BigInteger
- and(BigInteger val) 返回其值为 (this & val) 的 BigInteger
- andNot(BigInteger val) 返回其值为 (this & ~val) 的 BigInteger
- clearBit(int n) 返回其值与清除了指定位的此 BigInteger 等效的 BigInteger
- compareTo(BigInteger val) 将此 BigInteger 与指定的 BigInteger 进行比较
- divide(BigInteger val) 返回其值为 (this / val) 的 BigInteger
- doubleValue() 将此 BigInteger 转换为 double
- equals(Object x) 比较此 BigInteger 与指定的 Object 的相等性
- gcd(BigInteger val) 返回其值是 this 和 val 的最大公约数

- intValue()　　　　　　　将此 BigInteger 转换为 int
- isProbablePrime(int p)　　若 this 可能为素数返回 true，若一定为合数返回 false
- max(BigInteger val)　　　返回此 BigInteger 和 val 的最大值
- min(BigInteger val)　　　返回此 BigInteger 和 val 的最小值
- mod(BigInteger m)　　　　返回其值为 (this mod m) 的 BigInteger
- modInverse(BigInteger m)　　返回其值为 ($this^{-1}$ mod m) 的 BigInteger
- modPow(BigInteger exponent, BigInteger m) 返回其值为 ($this^{exponent}$ mod m) 的 BigInteger
- multiply(BigInteger val)　　返回其值为 (this * val) 的 BigInteger
- negate()　　　　　　　　返回其值是 (-this) 的 BigInteger
- nextProbablePrime()　　　返回大于此 BigInteger 的可能为素数的第一个整数
- not()　　　　　　　　　返回其值为 (~this) 的 BigInteger
- or(BigInteger val)　　　返回其值为 (this | val) 的 BigInteger
- pow(int exponent)　　　　返回其值为 ($this^{exponent}$) 的 BigInteger
- probablePrime(int bitLength, Random rnd) 返回指定长度可能是素数的正 BigInteger
- remainder(BigInteger val)　　返回其值为 (this % val) 的 BigInteger
- setBit(int n)　　　　　　返回其值与设置了指定位的此 BigInteger 等效的 BigInteger
- shiftLeft(int n)　　　　　返回其值为 (this << n) 的 BigInteger
- shiftRight(int n)　　　　返回其值为 (this >> n) 的 BigInteger
- signum()　　　　　　　　返回此 BigInteger 的正负号函数
- subtract(BigInteger val)　　返回其值为 (this - val) 的 BigInteger
- toString()　　　　　　　返回此 BigInteger 的十进制字符串表示形式
- xor(BigInteger val)　　　返回其值为 (this ^ val) 的 BigInteger

2.1.2 BigDecimal 类

- abs()　　　　　　　　　返回 BigDecimal，其值为此 BigDecimal 的绝对值
- add(BigDecimal augend)　　返回一个 BigDecimal，其值为 (this + augend)
- compareTo(BigDecimal val)　　将此 BigDecimal 与指定的 BigDecimal 比较。
- divide(BigDecimal divisor)　　返回一个 BigDecimal，其值为 (this / divisor)
- doubleValue()　　　　　　将此 BigDecimal 转换为 double
- equals(Object x)　　　　　比较此 BigDecimal 与指定的 Object 的相等性
- intValue()　　　　　　　将此 BigDecimal 转换为 int
- max(BigDecimal val)　　　返回此 BigDecimal 和 val 的最大值
- min(BigDecimal val)　　　返回此 BigDecimal 和 val 的最小值
- movePointLeft(int n)　　　返回一个 BigDecimal，它等效于将该值的小数点向左移动 n 位
- movePointRight(int n)　　返回一个 BigDecimal，它等效于将该值的小数点向右移动 n 位
- multiply(BigDecimal x)　　返回一个 BigDecimal，其值为 (this ×x)
- negate()　　　　　　　　返回 BigDecimal，其值为 (-this)
- plus()　　　　　　　　　返回 BigDecimal，其值为 (+this)
- pow(int n)　　　　　　　返回其值为 ($this^n$) 的 BigDecimal
- precision()　　　　　　　返回此 BigDecimal 的精度。
- remainder(BigDecimal d)　　返回其值为 (this % d) 的 BigDecimal
- round(MathContext mc)　　返回根据 MathContext 设置进行舍入后的 BigDecimal
- scale()　　　　　　　　　返回此 BigDecimal 的标度
- scaleByPowerOfTen(int n)　　返回其数值等于 (this * 10^n) 的 BigDecimal
- stripTrailingZeros()　　　返回尾部没有零的 BigDecimal
- subtract(BigDecimal x)　　返回一个 BigDecimal，其值为 (this - x)
- toBigInteger()　　　　　　将此 BigDecimal 转换为 BigInteger
- toEngineeringString()　　返回此 BigDecimal 的字符串表示形式，有指数时使用工程计数法

- toPlainString()　返回不带指数字段的此 BigDecimal 的字符串表示形式
- toString()　返回此 BigDecimal 的字符串表示形式，需要指数使用科学记数法
- ulp()　返回此 BigDecimal 的 ulp（最后一位的单位）的大小
- valueOf(double val)　将 double 转换为 BigDecimal
- valueOf(long val)　将 long 值转换为具有零标度的 BigDecimal

2.2 import java.lang.*

2.2.1 String 类

- charAt(int index)　返回指定索引处的 char 值
- compareTo(String anotherString)　按字典顺序比较两个字符串
- compareToIgnoreCase(String str)　按字典顺序比较两个字符串，不考虑大小写
- concat(String str)　将指定字符串连接到此字符串的结尾
- contains(CharSequence s)　当且仅当此 this 包含指定 char 值序列时返回 true
- endsWith(String suffix)　测试此字符串是否以指定的后缀结束
- isEmpty()　当且仅当 length() 为 0 时返回 true
- length()　返回此字符串的长度
- substring(int beginIndex)　返回一个新的字符串，它是此字符串的一个子字符串
- substring(int begin, int end)　返回一个新字符串，它是此字符串的一个子字符串
- toCharArray()　将此字符串转换为一个新的字符数组
- toLowerCase()　将此 String 中的所有字符都转换为小写
- toString()　返回此对象本身（它已经是一个字符串!）
- toUpperCase()　将此 String 中的所有字符都转换为大写
- trim()　返回字符串的副本，忽略前导空白和尾部空白
- valueOf(char[] data)　返回 char 数组参数的字符串表示形式
- valueOf(char[] c,int offset,int count)　返回 char 数组的特定子数组的字符串表示形式
- valueOf(double d)　返回 double 参数的字符串表示形式
- valueOf(float f)　返回 float 参数的字符串表示形式
- valueOf(int i)　返回 int 参数的字符串表示形式

2.3 JAVA 程序示例

```
import java.io.*;
import java.math.*;
import java.util.*;
public class Main{
        public static void main(String[] args){
                Scanner in = new Scanner(System.in);
                int n = in.nextInt();//读入
                int t[] = new int[n];//开数组
                BigInteger a[] = new BigInteger[100];//开数组
                for (int i = 0; i < n; i++) t[i] = in.nextInt();
                Arrays.sort(t, 0, n);//排序函数
                String st;
                st=in.next();
                while (in.hasNext()){}//读入
                System.out.println(x + " " + y);
        }
}
```

3 Chapter 2 Precautions 注意事项

- 重边
- 重点
- 自环
- 有向无向
- 边界
- 输出的字符串
- 输入是否有无关字符
- 行尾多余空格
- yes 和 no 是不是反的
- 尽量用 long long，或者 BigInteger
- memset 大数组容易 TLE
- 数组尽量开大
- 判断数字字符串大小是否出错
- 二分答案