ACM Template

Rien

October 2, 2021

rien_zhu@163.com

Contents

1 字符串

1.1 KMP

```
/*
Next[] 的含义:x[i-Next[i]...i-1]=x[0...Next[i]-1]
Next[i] 为满足:x[i-z...i-1]=x[0...z-1] 的最大值
*/
void kmp_pre(char x[],int m,int next[]){
    int i,j;
    j=next[0]=-1;
    i=0;
    while(i<m){
        while(-1!=j && x[i]!=x[j])
            j=next[j];
        next[++i]=++j;
    }
}
const int N=1000005;
int Next[N];
/*
  返回 x 在 y 中出现的次数, 可以重叠
  y 是主串,x 是模式串
*/
int KMP_Count(char x[],int m,char y[], int n){
    int i,j;
    int ans=0;
    kmp_pre(x,m,Next);
    i=j=0;
    while(i<n){
        while(-1!=j && y[i]!=x[j])
            j=Next[j];
        i++;j++;
        if(j>=m){
            ans++;
            j=Next[j];
        }
    }
    return ans;
}
```

1.2 AC 自动机

```
/*
题意：多组数据, 每组给出一个文本和 n 个模式
    0 代表能重叠, 1 代表不能重叠, 求每个模版串在文本串中出现的次数
样例输入:
abababac
2
0 aba
1 aba
样例输出:
3
2
```

```
12  */
13  #define INF 0x3f3f3f3f
14  #define LL long long
15  const int MOD=10007;
16  const int N=500000+5;
17  using namespace std;
18  struct AC_Automata{
19      int tire[N][26];//字典树
20      int val[N];//字符串结尾标记
21      int fail[N];//失配指针
22      int last[N];//last[i]=j 表 j 节点表示的单词是 i 节点单词的后缀，且 j 节
        ↪ 点是单词节点
23      int tot;//编号
24      int time[N];//上次的单词出现在文本串的位置
25      int len[N];//单词节点的长度
26      int cnt[N][2];//计数
27      void init(){//初始化 0 号点
28          tot=1;
29          val[0]=0;
30          last[0]=0;
31          fail[0]=0;
32          memset(tire[0],0,sizeof(tire[0]));
33      }
34      void insert(char *s){//构造 trie
35          int sLen=strlen(s);
36          int root=0;
37          for(int i=0;i<sLen;i++){
38              int id=s[i]-'a';
39              if(tire[root][id]==0){
40                  tire[root][id]=tot;
41                  memset(tire[tot],0,sizeof(tire[tot]));
42                  val[tot++]=0;
43              }
44              root=tire[root][id];
45          }
46          val[root]=1;
47          time[root]=0;
48          len[root]=sLen;
49          cnt[root][0]=0;
50          cnt[root][1]=0;
51      }
52      void build(){//构造 fail 与 last
53          queue<int> q;
54          for(int i=0;i<26;i++){
55              int root=tire[0][i];
56              if(root!=0){
57                  fail[root]=0;
58                  last[root]=0;
59                  q.push(root);
60              }
61          }
62          while(!q.empty()){//bfs 求 fail
63              int k=q.front();
```

```
            q.pop();
            for(int i=0;i<26; i++){
                int u=tire[k][i];
                if(u==0)
                    continue;
                q.push(u);

                int v=fail[k];
                while(v && tire[v][i]==0)
                    v=fail[v];
                fail[u]=tire[v][i];
                last[u]=val[fail[u]]?fail[u]:last[fail[u]];
            }
        }
    }
    void query(char *s){//匹配
        int len=strlen(s);
        int j=0;
        for(int i=0;i<len;i++){
            int id=s[i]-'a';
            while(j && tire[j][id]==0)
                j=fail[j];
            j=tire[j][id];
            if(val[j])
                print(j,i+1);
            else if(last[j])
                print(last[j],i+1);
        }
    }
    void print(int i,int pos){
        if(val[i]){
            cnt[i][0]++;
            if(time[i]+len[i]<=pos){//判断是否有重叠
                time[i]=pos;
                cnt[i][1]++;
            }
            print(last[i],pos);
        }
    }
    int queryT(char *s,int op){//匹配单个文本
        int len=strlen(s);
        int root=0;
        for(int i=0;i<len;i++){
            int id=s[i]-'a';
            root=tire[root][id];
        }
        return cnt[root][op];
    }
}ac;
char P[N][10];
char T[N];
int op[N];
int main(){
```

```
117     int Case=1;
118     while(scanf("%s",T)!=EOF){
119         ac.init();
120         int n;
121         scanf("%d",&n);
122         for(int i=0;i<n;i++){
123             scanf("%d%s",&op[i],P[i]);
124             ac.insert(P[i]);
125         }
126         ac.build();
127         ac.query(T);
128         printf("Case %d\n",Case++);
129         for(int i=0;i<n;i++)
130             printf("%d\n",ac.queryT(P[i],op[i]));
131         printf("\n");
132     }
133     return 0;
134 }
```

2 数据结构

2.1 并查集

2.1.1 优化并查集

```cpp
/*
路径压缩并查集，
*/
const int N=1e5+5;
int fa[N];
void init(){
    for(int i=1;i<=N;++i)
        fa[i]=i;
}
int find(int x){
    int temp=x;
    while(temp!=fa[temp])
        temp=fa[temp];
    while(x!=fa[x]){
        x=fa[x];
        fa[x]=temp;
    }
    return temp;
}
void union(int x,int y){
    fa[find(x)]=find(y);
}
/*
按秩合并并查集，不破坏树形结构
连通块数量为 block，大小为 size
*/
int f[N],size[N],block;
void Init(){
    for(int i=1;i<=N;++i)
        f[i]=0;
    block=n;
}
int Find(int x){
    if(!f[x]) f[x]=x,size[x]=1;
    if(f[x]==x) return x;
    return f[x]=Find(f[x]);
}
void Union(int x,int y){
    x=Find(x);
    y=Find(y);
    if(x=y) return ;
    if(size[x]>size[y]) swap(x,y);
    f[x]=y;
    size[y]+=size[x];
    block--;
}
```

2.2 01 字典树

```cpp
/*
01trie 树
在一组数中找跟某个数异或结果最大的数
*/
const int MAXN=100005;
//MAXN 右移需根据题目判断，如数据范围为 int 则 *32 即右移 5 位
int val[MAXN<<5];       //val[i]=j 表示编号为 i 的节点的值为 j
int cnt;                //节点数量
int tree[MAXN<<5][2];   //tree[i][0]=j 表示编号为 i 的节点的 0 子节点的编号为 j
void init(){
    cnt=1;
    tree[0][0]=tree[0][1]=0;
}
void insert(int x){
    int v,u=0;
    for(int i=31;i>=0;--i){
        v=(x>>i)&1;
        if(!tree[u][v]){
            tree[cnt][0]=tree[cnt][1]=0;//初始化新节点的子节点
            val[cnt]=0;                 //节点值为 0，表示到此不是一个数
            tree[u][v]=cnt++;           //指向该节点
        }
        u=tree[u][v];//到下一节点
    }
    val[u]=x;
}
int query(int x){
    int v,u=0;
    for(int i=31;i>=0;i--){
        v=(x>>i)&1;
        if(tree[u][v^1])//利用贪心策略，优先寻找和当前位不同的数
            u=tree[u][v^1];
        else
            u=tree[u][v];
    }
    return val[u];
}
int  main(){
    int n,m;scanf("%d%d",&n,&m);
    init();
    int tmp;
    for(int i=0;i<n;++i){
        scanf("%d",&tmp);
        insert(tmp);
    }
    for(int i=0;i<m;++i){
        scanf("%d",&tmp);
        printf("%d\n",query(tmp));
    }
    return 0;
}
```

2.3 树状数组

2.3.1 一维树状数组

```cpp
/*
BIT 一维树状数组
*/
const int N=1000005;
ll tree[N],a[N],b[N];
int n,m,op,l,r,v;
//单点修改区间查询
void add(int p,int x){
    for(int i=p;i<=n;i+=i&-i)
        tree[i]+=x;
}
ll query(int p){
    ll ret=0;
    for(int i=p;i;i-=i&-i)
        ret+=tree[i];
    return ret;
}
int main(){
    //读入数据
    for(int i=1;i<=n;++i){
        cin>>v;
        add(i,v);
    }
    add(l,v);//将第 l 个数加上 v
    cout<<query(r)-query(l-1)<<endl;//询问 [l-r] 区间的和
    return 0;
}

//区间修改单点查询
void add(int p,int x){
    for(int i=p;i<=n;i+=i&-i)
        tree[i]+=x;
}
ll query(int p){
    ll ret=0;
    for(int i=p;i;i-=i&-i)
        ret+=tree[i];
    return ret;
}
int main(){
    //读入数据，维护差分数组
    for(int i=1;i<=n;++i){
        cin>>a[i];
        add(i,a[i]-a[i-1]);
    }
    add(l,v),add(r+1,-v);//将 [l-r] 加上 v
    cout<<query(l)<<endl//询问第 l 个数的值
    return 0;
}

```

```cpp
//区间修改区间查询
void add(int p,int x){
    for(int i=p;i<=n;i+=i&-i){
        b[i]+=(ll)x;
        tree[i]+=(ll)p*x;
    }
}
ll query(int p){
    ll ret=0;
    for(int i=p;i;i-=i&-i)
        ret+=1ll*(p+1)*b[i]-tree[i];
    return ret;
}
int main(){
    //读入数据，维护差分数组
    for(int i=1;i<=n;++i){
        cin>>a[i];
        add(i,a[i]-a[i-1]);
    }
    add(l,v),add(r+1,-v);//将 [l-r] 加上 v
    cout<<query(r)-query(l-1)<<endl;//询问 [l-r] 区间的和
    return 0;
}
```

2.3.2 求逆序对

```cpp
//树状数组求逆序对
void modify(int x, int d){
        for(int i = x; i <= n; i += i&-i)c[i] += d;
}
int getsum(int x){
        int sum = 0;
        for(int i = x; i >= 1; i -= i&-i)sum += c[i];
        return sum;
}
int main(){
        scanf("%d", &n);
        for(int i = 1; i <= n; i ++){
                scanf("%d", &x);
                ans += getsum(n) - getsum(x);
                modify(x, 1);
        }
}
```

3 数学

3.1 质数

3.1.1 质数的判定

```
bool is_prime(int n){
    if(n<2) return false;
    for(int i=2;i<=sqrt(n);i++)
        if(n%i) return false;
    return true;
}
```

3.1.2 质数筛

```
/*
Eratosthenes 筛法
*/
const int N=1e5+5;
bool v[N];//值为 false 表示质数，为 true 为合数
int prime[N];//存储素数
int cnt;
void primes(int n){//筛选小于 n 的素数
    memset(v,0,sizeof(v));
    for(int i=2;i<=n;++i){
        if(v[i]==0){
            prime[++cnt]=i;
            for(int j=i; i*j<=n; j++)
                v[i*j] = true;
        }
    }
}

/*
线性筛
*/
const int N=1e8+5;
bool v[N];//值为 false 表示质数，为 true 为合数
int prime[N/10];//存储素数
int cnt;
void primes(int n){//筛选小于 n 的素数
    memset(v,0,sizeof(v));
    for(int i=2; i<=n; i++){
                if(v[i]==false)
                        prime[++cnt] = i ;
                for(int j=1; j<=cnt&&i*prime[j]<=n; j++){
                        v[i*prime[j]] = true ;
                        if(i%prime[j]==0)
                                break;
                }
        }
}
```

3.1.3 大区间质数筛

```cpp
/*
POJ 2689
给出区间 [L,U], 找出区间中距离最近和最远的质数
1<=L,U<=2147483647 , 区间长度 <=1000000
不能直接筛质数, 先求出 sqrt(2147483647) 约 5W 内的质数, 因为给定区间长度少于 1e6, 所以直接用 5w 内质数筛出来区间内质数就可以。
*/
const int MAXN=100010;
int prime[MAXN+5];
void getpPrime(){
    memset(prime,0,sizeof(prime));
    for(int i=2;i<MAXN;++i){
        if(!prime[i])
            prime[++prime[0]]=i;
        for(int j=1;j<=prime[0] && prime[j]<=MAXN/i;++j){
            prime[prime[j]*i]=1;
            if(i%prime[j]==0)
                break;
        }
    }
}
bool notprime[1000005];
int prime2[1000005];
void getPrime2(int L,int R){
    memset(notprime,0,sizeof(notprime));
    if(L<2) L=2;
    for(int i=1;i<=prime[0] && (ll)prime[i]*prime[i]<=R;++i){
        int s=L/prime[i]+(L%prime[i]>0);
        if(s==1) s=2;
        for(int j=s;(ll)j*prime[i]<=R;++j){
            if((ll)j*prime[i]>=L)
                notprime[j*prime[i]-L]=1;
        }
    }
    prime2[0]=0;
    for(int i=0;i<=R-L;++i){
        if(!notprime[i])
            prime2[++prime2[0]]=i+L;
    }
}
int main(){
    getpPrime();
    int L,U;
    while(scanf("%d%d",&L,&U)==2){
        getPrime2(L,U);
        if(prime2[0]<2)
            printf("There are no adjacent primes.\n");
        else {
            int x1=0,x2=1000005,y1=0,y2=0;
            for(int i=1;i<prime2[0];++i){
                if(prime2[i+1]-prime2[i]<x2-x1){
                    x1=prime2[i];
```

```
                    x2=prime2[i+1];
                }
                if(prime2[i+1]-prime2[i]>y2-y1){
                    y1=prime2[i];
                    y2=prime2[i+1];
                }
            }
            printf("%d,%d are closest, %d,%d are most distant.\n",x1,x2,y1,y2);
        }
    }
}
```

3.2 约数

3.2.1 最大公约数

```
int gcd(int a,int b){
    return b?gcd(b,a%b):a;
}
```

3.3 快速幂

3.3.1 快速乘

```
/*
一种想法是用类似类似 2 进制加法的方法实现乘法，复杂度为 O(log)
另一种就是用 long double 来进行优化取模运算。因为它其实很巧妙的运用了自动溢出这个操作，我们的代码中的 z 就表示 x*y/p ，所以我们要求的就变成了 x*y-x*y/p*p ，虽然这两个部分都是会溢出的，但（unsigned）保证了它们溢出后的差值基本不变，所以即使它会溢出也不会影响最终结果的！
*/
inline ll ksc(ll x,ll y,ll p){
        ll z=(ld)x/p*y;
        ll res=(ull)x*y-(ull)z*p;
        return (res+p)%p;
}
// ll 表示 long long
// ld 表示 long double
// ull 表示 unsigned long long
// 一种自动溢出的数据类型（存满了就会自动变为 0）
```

3.3.2 数字快速幂

```
//a 的 b 次方对 p 取余
//可能 a,b 都比较大，所以需要自己另写乘法
ll Mul(ll x,ll y,ll P){
    ll tmp=(x*y-(ll)((long double)x/P*y+1.0e-8)*P);
    return (tmp+P)%P;
}
ll ksm(ll a, ll b, ll p){
    ll ret=1;
    while(b){
        if(b&1) ret=Mul(ret,a,p);
        a=Mul(a,a,p);
```

```
        b>>=1;
    }
    return ret;
}
```

3.3.3 矩阵快速幂

```
const int mod = 1e9 + 7;
struct Matrix{
  int n, m;
  int num[100][100];//矩阵规模最大为 100*100
  Matrix(){}
  Matrix(int n, int m): n(n), m(m){
    memset(num, 0, sizeof num);
  }
  friend Matrix operator * (const Matrix &a, const Matrix &b){
    Matrix c(a.n, b.m);
    for(int i = 0; i < c.n; i++){
      for(int j = 0; j < c.m; j++){
        for(int k = 0; k < a.m; k++){
          c.num[i][j] = (c.num[i][j] + (long long)a.num[i][k] *
          ↪  b.num[k][j] % mod) % mod;
        }
      }
    }
    return c;
  }
};
Matrix ksm(Matrix a, long long b){
  Matrix s(a.n, a.n);
  for(int i = 0; i < a.n; i++) s.num[i][i] = 1;
  while(b){
    if(b & 1) s = s * a;
    a = a * a;
    b >>= 1;
  }
  return s;
}
int main(){
  int n;long long k;//n*n 的矩阵,k 次方
  scanf("%d%lld",&n,&k);
  Matrix M(n, n);
  for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
      scanf("%d",&M.num[i][j]);
    }
  }
  M = ksm(M, k);
  for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
      printf("%d ",M.num[i][j]);
    }
    printf("\n");
  }
```

```
  return 0;
}
```

3.4 组合数学

```
/*
组合数学求法:
1. 杨辉三角打表
所需空间:O(nm), 时间: 预处理 O(nm), 查询 O(1)
*/
const int mod = 1e9+7;
int c[maxn][maxn];
void init(int n){
        c[0][0]=1;
        rep(i,1,n){
                c[i][0]=1;
                rep(j,1,n+1)c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
        }
}

/*
2. 直接用公式
Cnm=n!/(m!(n-m)!)
不能取模, 时间: 预处理 O(1), 查询 O(n)
*/
ll C(ll n, ll m)
{
    if (n < m) return 0;
    ll ret = 1;
    rep(i, n - m + 1, n)
        ret *= i;
    rep(i, 2, m)
        ret /= i;
    return ret;
}

/*
3. 乘法逆元
可以取模, 适用于 P 比较大的题目
所需空间 O(n), 时间: 预处理 O(n), 查询 O(1)
*/
//N 的范围需要根据题目确定
int fac[N+5],inv[N+5];
ll qpow(ll bsc,ll y){
        ll ret = 1;
        while(y){
                if(y&1) ret = ret*bsc%mod;
                bsc = bsc*bsc%mod;
                y >>= 1;
        }
        return ret;
}
void init(){
        fac[0] = 1;
```

```
        for(int i=1;i<=N;i++)
                fac[i] = (ll)fac[i-1]*i%mod;
        inv[N] = qpow(fac[N],mod-2);
        for(int i=N-1;i>=0;i--)
                inv[i] = (ll)inv[i+1]*(i+1)%mod;
}
int C(int n,int m){
        if(m>n) return 0;
        return (ll)fac[n]*inv[m]%mod*inv[n-m]%mod;
}

/*
4.Lucas 定理
模数必须是质数,n 比较大,p 比较小, 不能通过预处理阶乘和逆元来计算
lucas(n,m,p) 返回 C(n,m) 对 p 取模的结果
*/
const ll p = (ll)1e9 + 7;
ll pow(ll a, ll b, ll m){
    ll ans = 1;
    a %= m;
    while(b)
    {
        if(b & 1)ans = (ans % m) * (a % m) % m;
        b /= 2;
        a = (a % m) * (a % m) % m;
    }
    ans %= m;
    return ans;
}
ll inv(ll x, ll p)//x 关于 p 的逆元, p 为素数
{
    return pow(x, p - 2, p);
}
ll C(ll n, ll m, ll p)//组合数 C(n, m) % p
{
    if(m > n)return 0;
    ll up = 1, down = 1;//分子分母;
    for(int i = n - m + 1; i <= n; i++)up = up * i % p;
    for(int i = 1; i <= m; i++)down = down * i % p;
    return up * inv(down, p) % p;
}
ll Lucas(ll n, ll m, ll p)
{
    if(m == 0)return 1;
    return C(n % p, m % p, p) * Lucas(n / p, m / p, p) % p;
}
int main()
{
        long long n,m;
        while(~scanf("%lld%lld",&n,&m))
        {
                printf("%lld\n",Lucas(n,m,p));
        }
```

```
        return 0;
}
```

3.5 莫比乌斯反演

3.5.1 整除分块

可以用到整除分块的形式，大致是这样的：

$$\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$$

用整除分块可以在 $O(\sqrt{n})$ 求出，代码如下：

```
for(int l=1,r;l<=n;l=r+1){
        r=n/(n/l);
        ans+=(r-l+1)*(n/l);
}
```

3.5.2 莫比乌斯函数

```
const int MAXN=1000000;
bool check[MAXN+10];
int prime[MAXN+10];
int mu[MAXN+10];
void Moblus(){
    memset(check,false,sizeof(check));
    mu[1]=1;
    int tot=0;
    for(int i=2;i<=MAXN;i++){
        if(!check[i]){
            prime[tot++]=i;
            mu[i]=-1;
        }
        for(int j=0;j<tot;j++){
            if(i*prime[j]>MAXN) break;
            check[i*prime[j]]=true;
            if(i%prime[j]==0){
                mu[i*prime[j]]=0;
                break;
            }
            else mu[i*prime[j]]=-mu[i];
        }
    }
}
```

3.6 BSGS

```
//a^x = b (mod n) n 是素数和不是素数都行
//求解上式 0 <= x < n
//调用函数的时候记得 b=b%n
#define MOD 76543
int hs[MOD],head[MOD],Next[MOD],id[MOD],top;
void insert(int x,int y){
    int k=x%MOD;
    hs[top]=x,id[top]=y,Next[top]=head[k],head[k]=top++;
```

```
}
int find(int x){
    int k=x%MOD;
    for(int i=head[k];i!=-1;i=Next[i])
        if(hs[i]==x) return id[i];
    return -1;
}
int BSGS(int a,int b,int n){
    memset(head,-1,sizeof(head));
    top=1;
    if(b==1) return 0;
    int m=sqrt(n*1.0),j;
    long long x=1,p=1;
    for(int i=0;i<m;++i,p=p*a%n) insert(p*b%n,i);
    for(long long i=m;;i+=m){
        if((j=find(x=x*p%n))!=-1) return i-j;
        if(i>n) break;
    }
    return -1;
}
```

3.7 中国剩余定理

中国剩余定理 (Chinese Remainder Theorem, CRT) 可求解如下形式的一元线性同余方程组（其中 $n_1, n_2, \cdots, n_k$ 两两互质）：

$$\begin{cases} x & \equiv a_1 \pmod{n_1} \\ x & \equiv a_2 \pmod{n_2} \\ & \vdots \\ x & \equiv a_k \pmod{n_k} \end{cases}$$

算法流程

1. 计算所有模数的积 n；
2. 对于第 i 个方程：
   (a) 计算 $m_i = \frac{n}{n_i}$；
   (b) 计算 m_i 在模 n_i 意义下的 [逆元](./inverse.md) m_i^{-1}；
   (c) 计算 $c_i = m_i m_i^{-1}$（不要对 n_i 取模）。
3. 方程组的唯一解为：$x = \sum_{i=1}^{k} a_i c_i \pmod{n}$。

代码如下：

```
LL CRT(int k, LL* a, LL* r) {
  LL n = 1, ans = 0;
  for (int i = 1; i <= k; i++) n = n * r[i];
  for (int i = 1; i <= k; i++) {
    LL m = n / r[i], b, y;
    exgcd(m, r[i], b, y);  // b * m mod r[i] = 1
    ans = (ans + a[i] * m * b % mod) % mod;
  }
  return (ans % mod + mod) % mod;
}
```

4 计算几何

4.1 二维几何

```cpp
// 计算几何模板
const double eps = 1e-8;
const double pi = acos(-1.0);
const double inf = 1e20;
const int maxp = 1010;
//Compares a double to zero
int sgn(double x){
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
//square of a double
inline double sqr(double x){return x*x;}
/*
 * Point
 * Point()                - Empty constructor
 * Point(double _x,double _y)  - constructor
 * input()              - double input
 * output()             - %.2f output
 * operator ==          - compares x and y
 * operator <           - compares first by x, then by y
 * operator -           - return new Point after subtracting curresponging x and y
 * operator ^           - cross product of 2d points
 * operator *           - dot product
 * len()                - gives length from origin
 * len2()               - gives square of length from origin
 * distance(Point p)    - gives distance from p
 * operator + Point b   - returns new Point after adding curresponging x and y
 * operator * double k - returns new Point after multiplieing x and y by k
 * operator / double k - returns new Point after divideing x and y by k
 * rad(Point a,Point b)- returns the angle of Point a and Point b from this Point
 * trunc(double r)      - return Point that if truncated the distance from center to r
 * rotleft()            - returns 90 degree ccw rotated point
 * rotright()           - returns 90 degree cw rotated point
 * rotate(Point p,double angle) - returns Point after rotateing the Point centering at p by angle radian ccw
 */
struct Point{
    double x,y;
    Point(){}
    Point(double _x,double _y){
        x = _x;
        y = _y;
    }
    void input(){
```

```
45          scanf("%lf%lf",&x,&y);
46      }
47      void output(){
48          printf("%.2f %.2f\n",x,y);
49      }
50      bool operator == (Point b)const{
51          return sgn(x-b.x) == 0 && sgn(y-b.y) == 0;
52      }
53      bool operator < (Point b)const{
54          return sgn(x-b.x)== 0?sgn(y-b.y)<0:x<b.x;
55      }
56      Point operator -(const Point &b)const{
57          return Point(x-b.x,y-b.y);
58      }
59      //叉积
60      double operator ^(const Point &b)const{
61          return x*b.y - y*b.x;
62      }
63      //点积
64      double operator *(const Point &b)const{
65          return x*b.x + y*b.y;
66      }
67      //返回长度
68      double len(){
69          return hypot(x,y);//库函数
70      }
71      //返回长度的平方
72      double len2(){
73          return x*x + y*y;
74      }
75      //返回两点的距离
76      double distance(Point p){
77          return hypot(x-p.x,y-p.y);
78      }
79      Point operator +(const Point &b)const{
80          return Point(x+b.x,y+b.y);
81      }
82      Point operator *(const double &k)const{
83          return Point(x*k,y*k);
84      }
85      Point operator /(const double &k)const{
86          return Point(x/k,y/k);
87      }
88      //`计算 pa 和 pb 的夹角`
89      //`就是求这个点看 a,b 所成的夹角`
90      //`测试 LightOJ1203`
91      double rad(Point a,Point b){
92          Point p = *this;
93          return fabs(atan2( fabs((a-p)^(b-p)) , (a-p)*(b-p) ));
94      }
95      //`化为长度为 r 的向量`
96      Point trunc(double r){
97          double l = len();
```

```
98          if(!sgn(l))return *this;
99          r /= l;
100         return Point(x*r,y*r);
101     }
102     //`逆时针旋转 90 度`
103     Point rotleft(){
104         return Point(-y,x);
105     }
106     //`顺时针旋转 90 度`
107     Point rotright(){
108         return Point(y,-x);
109     }
110     //`绕着 p 点逆时针旋转 angle`
111     Point rotate(Point p,double angle){
112         Point v = (*this) - p;
113         double c = cos(angle), s = sin(angle);
114         return Point(p.x + v.x*c - v.y*s,p.y + v.x*s + v.y*c);
115     }
116 };
117 /*
118  * Stores two points
119  * Line()                          - Empty constructor
120  * Line(Point _s,Point _e)         - Line through _s and _e
121  * operator ==                     - checks if two points are same
122  * Line(Point p,double angle)      - one end p , another end at angle
    ↪  degree
123  * Line(double a,double b,double c) - Line of equation ax + by + c = 0
124  * input()                         - inputs s and e
125  * adjust()                        - orders in such a way that s < e
126  * length()                        - distance of se
127  * angle()                         - return 0 <= angle < pi
128  * relation(Point p)               - 3 if point is on line
129  *                                   1 if point on the left of line
130  *                                   2 if point on the right of line
131  * pointonseg(double p)            - return true if point on segment
132  * parallel(Line v)                - return true if they are parallel
133  * segcrossseg(Line v)             - returns 0 if does not intersect
134  *                                   returns 1 if non-standard
    ↪  intersection
135  *                                   returns 2 if intersects
136  * linecrossseg(Line v)            - line and seg
137  * linecrossline(Line v)           - 0 if parallel
138  *                                   1 if coincides
139  *                                   2 if intersects
140  * crosspoint(Line v)              - returns intersection point
141  * dispointtoline(Point p)         - distance from point p to the line
142  * dispointtoseg(Point p)          - distance from p to the segment
143  * dissegtoseg(Line v)             - distance of two segment
144  * lineprog(Point p)               - returns projected point p on se line
145  * symmetrypoint(Point p)          - returns reflection point of p over se
146  *
147  */
148 struct Line{
```

```
149     Point s,e;
150     Line(){}
151     Line(Point _s,Point _e){
152         s = _s;
153         e = _e;
154     }
155     bool operator ==(Line v){
156         return (s == v.s)&&(e == v.e);
157     }
158     //`根据一个点和倾斜角 angle 确定直线,0<=angle<pi`
159     Line(Point p,double angle){
160         s = p;
161         if(sgn(angle-pi/2) == 0){
162             e = (s + Point(0,1));
163         }
164         else{
165             e = (s + Point(1,tan(angle)));
166         }
167     }
168     //ax+by+c=0
169     Line(double a,double b,double c){
170         if(sgn(a) == 0){
171             s = Point(0,-c/b);
172             e = Point(1,-c/b);
173         }
174         else if(sgn(b) == 0){
175             s = Point(-c/a,0);
176             e = Point(-c/a,1);
177         }
178         else{
179             s = Point(0,-c/b);
180             e = Point(1,(-c-a)/b);
181         }
182     }
183     void input(){
184         s.input();
185         e.input();
186     }
187     void adjust(){
188         if(e < s)swap(s,e);
189     }
190     //求线段长度
191     double length(){
192         return s.distance(e);
193     }
194     //`返回直线倾斜角 0<=angle<pi`
195     double angle(){
196         double k = atan2(e.y-s.y,e.x-s.x);
197         if(sgn(k) < 0)k += pi;
198         if(sgn(k-pi) == 0)k -= pi;
199         return k;
200     }
201     //`点和直线关系`
```

```
    //`1  在左侧`
    //`2  在右侧`
    //`3  在直线上`
    int relation(Point p){
        int c = sgn((p-s)^(e-s));
        if(c < 0)return 1;
        else if(c > 0)return 2;
        else return 3;
    }
    // 点在线段上的判断
    bool pointonseg(Point p){
        return sgn((p-s)^(e-s)) == 0 && sgn((p-s)*(p-e)) <= 0;
    }
    //`两向量平行 (对应直线平行或重合)`
    bool parallel(Line v){
        return sgn((e-s)^(v.e-v.s)) == 0;
    }
    //`两线段相交判断`
    //`2 规范相交`
    //`1 非规范相交`
    //`0 不相交`
    int segcrossseg(Line v){
        int d1 = sgn((e-s)^(v.s-s));
        int d2 = sgn((e-s)^(v.e-s));
        int d3 = sgn((v.e-v.s)^(s-v.s));
        int d4 = sgn((v.e-v.s)^(e-v.s));
        if( (d1^d2)==-2 && (d3^d4)==-2 )return 2;
        return (d1==0 && sgn((v.s-s)*(v.s-e))<=0) ||
            (d2==0 && sgn((v.e-s)*(v.e-e))<=0) ||
            (d3==0 && sgn((s-v.s)*(s-v.e))<=0) ||
            (d4==0 && sgn((e-v.s)*(e-v.e))<=0);
    }
    //`直线和线段相交判断`
    //`-*this line   -v seg`
    //`2 规范相交`
    //`1 非规范相交`
    //`0 不相交`
    int linecrossseg(Line v){
        int d1 = sgn((e-s)^(v.s-s));
        int d2 = sgn((e-s)^(v.e-s));
        if((d1^d2)==-2) return 2;
        return (d1==0||d2==0);
    }
    //`两直线关系`
    //`0 平行`
    //`1 重合`
    //`2 相交`
    int linecrossline(Line v){
        if((*this).parallel(v))
            return v.relation(s)==3;
        return 2;
    }
    //`求两直线的交点`
```

```
255     //`要保证两直线不平行或重合`
256     Point crosspoint(Line v){
257         double a1 = (v.e-v.s)^(s-v.s);
258         double a2 = (v.e-v.s)^(e-v.s);
259         return Point((s.x*a2-e.x*a1)/(a2-a1) , (s.y*a2-e.y*a1)/(a2-a1));
260     }
261     //点到直线的距离
262     double dispointtoline(Point p){
263         return fabs((p-s)^(e-s))/length();
264     }
265     //点到线段的距离
266     double dispointtoseg(Point p){
267         if(sgn((p-s)*(e-s))<0 || sgn((p-e)*(s-e))<0)
268             return min(p.distance(s),p.distance(e));
269         return dispointtoline(p);
270     }
271     //`返回线段到线段的距离`
272     //`前提是两线段不相交，相交距离就是 0 了`
273     double dissegtoseg(Line v){
274         return min(min(dispointtoseg(v.s) , dispointtoseg(v.e)),
275         min(v.dispointtoseg(s),v.dispointtoseg(e)));
276     }
277     //`返回点 p 在直线上的投影`
278     Point lineprog(Point p){
279         return s + ( ((e-s)*((e-s)*(p-s))) / ((e-s).len2()) );
280     }
281     //`返回点 p 关于直线的对称点`
282     Point symmetrypoint(Point p){
283         Point q = lineprog(p);
284         return Point(2*q.x-p.x,2*q.y-p.y);
285     }
286 };
287 //圆
288 struct circle{
289     Point p;//圆心
290     double r;//半径
291     circle(){}
292     circle(Point _p,double _r){
293         p = _p;
294         r = _r;
295     }
296     circle(double x,double y,double _r){
297         p = Point(x,y);
298         r = _r;
299     }
300     //`三角形的外接圆`
301     //`需要 Point 的 + / rotate() 以及 Line 的 crosspoint()`
302     //`利用两条边的中垂线得到圆心`
303     //`测试：UVA12304`
304     circle(Point a,Point b,Point c){
305         Line u = Line((a+b)/2 , ((a+b)/2)+((b-a).rotleft()));
306         Line v = Line((b+c)/2 , ((b+c)/2)+((c-b).rotleft()));
307         p = u.crosspoint(v);
```

```
        r = p.distance(a);
    }
    //`三角形的内切圆`
    //`参数 bool t 没有作用，只是为了和上面外接圆函数区别`
    //`测试：UVA12304`
    circle(Point a,Point b,Point c,bool t){
        Line u,v;
        double m = atan2(b.y-a.y,b.x-a.x), n = atan2(c.y-a.y,c.x-a.x);
        u.s = a;
        u.e = u.s + Point(cos((n+m)/2),sin((n+m)/2));
        v.s = b;
        m = atan2(a.y-b.y,a.x-b.x) , n = atan2(c.y-b.y,c.x-b.x);
        v.e = v.s + Point(cos((n+m)/2) , sin((n+m)/2));
        p = u.crosspoint(v);
        r = Line(a,b).dispointtoseg(p);
    }
    //输入
    void input(){
        p.input();
        scanf("%lf",&r);
    }
    //输出
    void output(){
        printf("%.2lf %.2lf %.2lf\n",p.x,p.y,r);
    }
    bool operator == (circle v){
        return (p==v.p) && sgn(r-v.r)==0;
    }
    bool operator < (circle v)const{
        return ((p<v.p)||((p==v.p) && sgn(r-v.r)<0));
    }
    //面积
    double area(){
        return pi*r*r;
    }
    //周长
    double circumference(){
        return 2*pi*r;
    }
    //`点和圆的关系`
    //`0 圆外`
    //`1 圆上`
    //`2 圆内`
    int relation(Point b){
        double dst = b.distance(p);
        if(sgn(dst-r) < 0)return 2;
        else if(sgn(dst-r)==0)return 1;
        return 0;
    }
    //`线段和圆的关系`
    //`比较的是圆心到线段的距离和半径的关系`
    int relationseg(Line v){
        double dst = v.dispointtoseg(p);
```

```
361         if(sgn(dst-r) < 0)return 2;
362         else if(sgn(dst-r) == 0)return 1;
363         return 0;
364     }
365     //`直线和圆的关系`
366     //`比较的是圆心到直线的距离和半径的关系`
367     int relationline(Line v){
368         double dst = v.dispointtoline(p);
369         if(sgn(dst-r) < 0)return 2;
370         else if(sgn(dst-r) == 0)return 1;
371         return 0;
372     }
373     //`两圆的关系`
374     //`5 相离`
375     //`4 外切`
376     //`3 相交`
377     //`2 内切`
378     //`1 内含`
379     //`需要 Point 的 distance`
380     //`测试：UVA12304`
381     int relationcircle(circle v){
382         double d = p.distance(v.p);
383         if(sgn(d-r-v.r) > 0)return 5;
384         if(sgn(d-r-v.r) == 0)return 4;
385         double l = fabs(r-v.r);
386         if(sgn(d-r-v.r)<0 && sgn(d-l)>0)return 3;
387         if(sgn(d-l)==0)return 2;
388         if(sgn(d-l)<0)return 1;
389     }
390     //`求两个圆的交点，返回 0 表示没有交点，返回 1 是一个交点，2 是两个交点`
391     //`需要 relationcircle`
392     //`测试：UVA12304`
393     int pointcrosscircle(circle v,Point &p1,Point &p2){
394         int rel = relationcircle(v);
395         if(rel == 1 || rel == 5)return 0;
396         double d = p.distance(v.p);
397         double l = (d*d+r*r-v.r*v.r)/(2*d);
398         double h = sqrt(r*r-l*l);
399         Point tmp = p + (v.p-p).trunc(l);
400         p1 = tmp + ((v.p-p).rotleft().trunc(h));
401         p2 = tmp + ((v.p-p).rotright().trunc(h));
402         if(rel == 2 || rel == 4)
403             return 1;
404         return 2;
405     }
406     //`求直线和圆的交点，返回交点个数`
407     int pointcrossline(Line v,Point &p1,Point &p2){
408         if(!(*this).relationline(v))return 0;
409         Point a = v.lineprog(p);
410         double d = v.dispointtoline(p);
411         d = sqrt(r*r-d*d);
412         if(sgn(d) == 0){
413             p1 = a;
```

```
414             p2 = a;
415             return 1;
416         }
417         p1 = a + (v.e-v.s).trunc(d);
418         p2 = a - (v.e-v.s).trunc(d);
419         return 2;
420     }
421     //`得到过 a,b 两点，半径为 r1 的两个圆`
422     int gercircle(Point a,Point b,double r1,circle &c1,circle &c2){
423         circle x(a,r1),y(b,r1);
424         int t = x.pointcrosscircle(y,c1.p,c2.p);
425         if(!t)return 0;
426         c1.r = c2.r = r1;
427         return t;
428     }
429     //`得到与直线 u 相切，过点 q，半径为 r1 的圆`
430     //`测试：UVA12304`
431     int getcircle(Line u,Point q,double r1,circle &c1,circle &c2){
432         double dis = u.dispointtoline(q);
433         if(sgn(dis-r1*2)>0)return 0;
434         if(sgn(dis) == 0){
435             c1.p = q + ((u.e-u.s).rotleft().trunc(r1));
436             c2.p = q + ((u.e-u.s).rotright().trunc(r1));
437             c1.r = c2.r = r1;
438             return 2;
439         }
440         Line u1 = Line((u.s + (u.e-u.s).rotleft().trunc(r1)) , (u.e +
            ↪  (u.e-u.s).rotleft().trunc(r1)));
441         Line u2 = Line((u.s + (u.e-u.s).rotright().trunc(r1)) , (u.e +
            ↪  (u.e-u.s).rotright().trunc(r1)));
442         circle cc = circle(q,r1);
443         Point p1,p2;
444         if(!cc.pointcrossline(u1,p1,p2)) cc.pointcrossline(u2,p1,p2);
445         c1 = circle(p1,r1);
446         if(p1 == p2){
447             c2 = c1;
448             return 1;
449         }
450         c2 = circle(p2,r1);
451         return 2;
452     }
453     //`同时与直线 u,v 相切，半径为 r1 的圆`
454     //`测试：UVA12304`
455     int getcircle(Line u,Line v,double r1,circle &c1,circle &c2,circle
        ↪  &c3,circle &c4){
456         if(u.parallel(v))return 0;//两直线平行
457         Line u1 = Line(u.s + (u.e-u.s).rotleft().trunc(r1),u.e +
            ↪  (u.e-u.s).rotleft().trunc(r1));
458         Line u2 = Line(u.s + (u.e-u.s).rotright().trunc(r1),u.e +
            ↪  (u.e-u.s).rotright().trunc(r1));
459         Line v1 = Line(v.s + (v.e-v.s).rotleft().trunc(r1),v.e +
            ↪  (v.e-v.s).rotleft().trunc(r1));
460         Line v2 = Line(v.s + (v.e-v.s).rotright().trunc(r1),v.e +
            ↪  (v.e-v.s).rotright().trunc(r1));
```

```
        c1.r = c2.r = c3.r = c4.r = r1;
        c1.p = u1.crosspoint(v1);
        c2.p = u1.crosspoint(v2);
        c3.p = u2.crosspoint(v1);
        c4.p = u2.crosspoint(v2);
        return 4;
    }
    //`同时与不相交圆 cx,cy 相切, 半径为 r1 的圆`
    //`测试：UVA12304`
    int getcircle(circle cx,circle cy,double r1,circle &c1,circle &c2){
        circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);
        int t = x.pointcrosscircle(y,c1.p,c2.p);
        if(!t)return 0;
        c1.r = c2.r = r1;
        return t;
    }

    //`过一点作圆的切线 (先判断点和圆的关系)`
    //`测试：UVA12304`
    int tangentline(Point q,Line &u,Line &v){
        int x = relation(q);
        if(x == 2)return 0;
        if(x == 1){
            u = Line(q,q + (q-p).rotleft());
            v = u;
            return 1;
        }
        double d = p.distance(q);
        double l = r*r/d;
        double h = sqrt(r*r-l*l);
        u = Line(q,p + ((q-p).trunc(l) + (q-p).rotleft().trunc(h)));
        v = Line(q,p + ((q-p).trunc(l) + (q-p).rotright().trunc(h)));
        return 2;
    }
    //`求两圆相交的面积`
    double areacircle(circle v){
        int rel = relationcircle(v);
        if(rel >= 4)return 0.0;
        if(rel <= 2)return min(area(),v.area());
        double d = p.distance(v.p);
        double hf = (r+v.r+d)/2.0;
        double ss = 2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
        double a1 = acos((r*r+d*d-v.r*v.r)/(2.0*r*d));
        a1 = a1*r*r;
        double a2 = acos((v.r*v.r+d*d-r*r)/(2.0*v.r*d));
        a2 = a2*v.r*v.r;
        return a1+a2-ss;
    }
    //`求两圆相交的面积 (精度更高)(需要 long double)`
    double areacircle2(circle v)
    {
        double a=hypot(p.x-v.p.x,p.y-v.p.y),b=r,c=v.r;
        double s1=pi*r*r,s2=pi*v.r*v.r;
```

```
514         if(sgn(a-b-c)>=0)
515             return 0;
516         if(sgn(a+min(b,c)-max(b,c))<=0)
517             return min(s1,s2);
518         else
519         {
520             double cta1=2*acos((a*a+b*b-c*c)/(2*a*b));
521             double cta2=2*acos((a*a+c*c-b*b)/(2*a*c));
522             return cta1/(2*pi)*s1-0.5*sin(cta1)*b*b +
                ↪  cta2/(2*pi)*s2-0.5*sin(cta2)*c*c;
523         }
524     }
525     //`求圆和三角形 pab 的相交面积`
526     //`测试：POJ3675 HDU3982 HDU2892`
527     double areatriangle(Point a,Point b){
528         if(sgn((p-a)^(p-b)) == 0)return 0.0;
529         Point q[5];
530         int len = 0;
531         q[len++] = a;
532         Line l(a,b);
533         Point p1,p2;
534         if(pointcrossline(l,q[1],q[2])==2){
535             if(sgn((a-q[1])*(b-q[1]))<0)q[len++] = q[1];
536             if(sgn((a-q[2])*(b-q[2]))<0)q[len++] = q[2];
537         }
538         q[len++] = b;
539         if(len == 4 && sgn((q[0]-q[1])*(q[2]-q[1]))>0)swap(q[1],q[2]);
540         double res = 0;
541         for(int i = 0;i < len-1;i++){
542             if(relation(q[i])==0||relation(q[i+1])==0){
543                 double arg = p.rad(q[i],q[i+1]);
544                 res += r*r*arg/2.0;
545             }
546             else{
547                 res += fabs((q[i]-p)^(q[i+1]-p))/2.0;
548             }
549         }
550         return res;
551     }
552 };
553
554 /*
555  * n,p  Line l for each side
556  * input(int _n)                          - inputs _n size polygon
557  * add(Point q)                           - adds a point at end of the list
558  * getline()                              - populates line array
559  * cmp                                    - comparision in convex_hull
    ↪  order
560  * norm()                                 - sorting in convex_hull order
561  * getconvex(polygon &convex)             - returns convex hull in convex
562  * Graham(polygon &convex)                - returns convex hull in convex
563  * isconvex()                             - checks if convex
564  * relationpoint(Point q)                 - returns 3 if q is a vertex
```

```
565  *                                                2 if on a side
566  *                                                1 if inside
567  *                                                0 if outside
568  * convexcut(Line u,polygon &po)          - left side of u in po
569  * gercircumference()                     - returns side length
570  * getarea()                              - returns area
571  * getdir()                               - returns 0 for cw, 1 for ccw
572  * getbarycentre()                        - returns barycenter
573  *
574  */
575 struct polygon{
576     int n;
577     Point p[maxp];
578     Line l[maxp];
579     void input(int _n){
580         n = _n;
581         for(int i = 0;i < n;i++)
582             p[i].input();
583     }
584     void add(Point q){
585         p[n++] = q;
586     }
587     void getline(){
588         for(int i = 0;i < n;i++){
589             l[i] = Line(p[i],p[(i+1)%n]);
590         }
591     }
592     struct cmp{
593         Point p;
594         cmp(const Point &p0){p = p0;}
595         bool operator()(const Point &aa,const Point &bb){
596             Point a = aa, b = bb;
597             int d = sgn((a-p)^(b-p));
598             if(d == 0){
599                 return sgn(a.distance(p)-b.distance(p)) < 0;
600             }
601             return d > 0;
602         }
603     };
604     //`进行极角排序`
605     //`首先需要找到最左下角的点`
606     //`需要重载号好 Point 的 < 操作符 (min 函数要用)`
607     void norm(){
608         Point mi = p[0];
609         for(int i = 1;i < n;i++)mi = min(mi,p[i]);
610         sort(p,p+n,cmp(mi));
611     }
612     //`得到凸包`
613     //`得到的凸包里面的点编号是 0$\sim$n-1 的`
614     //`两种凸包的方法`
615     //`注意如果有影响，要特判下所有点共点，或者共线的特殊情况`
616     //`测试 LightOJ1203  LightOJ1239`
617     void getconvex(polygon &convex){
```

```
618         sort(p,p+n);
619         convex.n = n;
620         for(int i = 0;i < min(n,2);i++){
621             convex.p[i] = p[i];
622         }
623         if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n--;//特
            ↪ 判
624         if(n <= 2)return;
625         int &top = convex.n;
626         top = 1;
627         for(int i = 2;i < n;i++){
628             while(top && sgn((convex.p[top]-p[i])^(convex.p[top-1]-p[i]))
                ↪ <= 0)
629                 top--;
630             convex.p[++top] = p[i];
631         }
632         int temp = top;
633         convex.p[++top] = p[n-2];
634         for(int i = n-3;i >= 0;i--){
635             while(top != temp &&
                ↪ sgn((convex.p[top]-p[i])^(convex.p[top-1]-p[i])) <= 0)
636                 top--;
637             convex.p[++top] = p[i];
638         }
639         if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n--;//特
            ↪ 判
640         convex.norm();//`原来得到的是顺时针的点，排序后逆时针`
641     }
642     //`得到凸包的另外一种方法`
643     //`测试 LightOJ1203  LightOJ1239`
644     void Graham(polygon &convex){
645         norm();
646         int &top = convex.n;
647         top = 0;
648         if(n == 1){
649             top = 1;
650             convex.p[0] = p[0];
651             return;
652         }
653         if(n == 2){
654             top = 2;
655             convex.p[0] = p[0];
656             convex.p[1] = p[1];
657             if(convex.p[0] == convex.p[1])top--;
658             return;
659         }
660         convex.p[0] = p[0];
661         convex.p[1] = p[1];
662         top = 2;
663         for(int i = 2;i < n;i++){
664             while( top > 1 &&
                ↪ sgn((convex.p[top-1]-convex.p[top-2])^(p[i]-convex.p[top-2]))
                ↪ <= 0 )
```

```
665                 top--;
666             convex.p[top++] = p[i];
667         }
668         if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n--;//特
            ↪ 判
669     }
670     //`判断是不是凸的`
671     bool isconvex(){
672         bool s[3];
673         memset(s,false,sizeof(s));
674         for(int i = 0;i < n;i++){
675             int j = (i+1)%n;
676             int k = (j+1)%n;
677             s[sgn((p[j]-p[i])^(p[k]-p[i]))+1] = true;
678             if(s[0] && s[2])return false;
679         }
680         return true;
681     }
682     //`判断点和任意多边形的关系`
683     //` 3 点上`
684     //` 2 边上`
685     //` 1 内部`
686     //` 0 外部`
687     int relationpoint(Point q){
688         for(int i = 0;i < n;i++){
689             if(p[i] == q)return 3;
690         }
691         getline();
692         for(int i = 0;i < n;i++){
693             if(l[i].pointonseg(q))return 2;
694         }
695         int cnt = 0;
696         for(int i = 0;i < n;i++){
697             int j = (i+1)%n;
698             int k = sgn((q-p[j])^(p[i]-p[j]));
699             int u = sgn(p[i].y-q.y);
700             int v = sgn(p[j].y-q.y);
701             if(k > 0 && u < 0 && v >= 0)cnt++;
702             if(k < 0 && v < 0 && u >= 0)cnt--;
703         }
704         return cnt != 0;
705     }
706     //`直线 u 切割凸多边形左侧`
707     //`注意直线方向`
708     //`测试：HDU3982`
709     void convexcut(Line u,polygon &po){
710         int &top = po.n;//注意引用
711         top = 0;
712         for(int i = 0;i < n;i++){
713             int d1 = sgn((u.e-u.s)^(p[i]-u.s));
714             int d2 = sgn((u.e-u.s)^(p[(i+1)%n]-u.s));
715             if(d1 >= 0)po.p[top++] = p[i];
716             if(d1*d2 < 0)po.p[top++] =
                ↪ u.crosspoint(Line(p[i],p[(i+1)%n]));
```

```
        }
    }
    //`得到周长`
    //`测试 LightOJ1239`
    double getcircumference(){
        double sum = 0;
        for(int i = 0;i < n;i++){
            sum += p[i].distance(p[(i+1)%n]);
        }
        return sum;
    }
    //`得到面积`
    double getarea(){
        double sum = 0;
        for(int i = 0;i < n;i++){
            sum += (p[i]^p[(i+1)%n]);
        }
        return fabs(sum)/2;
    }
    //`得到方向`
    //` 1 表示逆时针, 0 表示顺时针`
    bool getdir(){
        double sum = 0;
        for(int i = 0;i < n;i++)
            sum += (p[i]^p[(i+1)%n]);
        if(sgn(sum) > 0)return 1;
        return 0;
    }
    //`得到重心`
    Point getbarycentre(){
        Point ret(0,0);
        double area = 0;
        for(int i = 1;i < n-1;i++){
            double tmp = (p[i]-p[0])^(p[i+1]-p[0]);
            if(sgn(tmp) == 0)continue;
            area += tmp;
            ret.x += (p[0].x+p[i].x+p[i+1].x)/3*tmp;
            ret.y += (p[0].y+p[i].y+p[i+1].y)/3*tmp;
        }
        if(sgn(area)) ret = ret/area;
        return ret;
    }
    //`多边形和圆交的面积`
    //`测试：POJ3675 HDU3982 HDU2892`
    double areacircle(circle c){
        double ans = 0;
        for(int i = 0;i < n;i++){
            int j = (i+1)%n;
            if(sgn( (p[j]-c.p)^(p[i]-c.p) ) >= 0)
                ans += c.areatriangle(p[i],p[j]);
            else ans -= c.areatriangle(p[i],p[j]);
        }
        return fabs(ans);
```

```
770     }
771     //`多边形和圆关系`
772     //` 2 圆完全在多边形内`
773     //` 1 圆在多边形里面，碰到了多边形边界`
774     //` 0 其它`
775     int relationcircle(circle c){
776         getline();
777         int x = 2;
778         if(relationpoint(c.p) != 1)return 0;//圆心不在内部
779         for(int i = 0;i < n;i++){
780             if(c.relationseg(l[i])==2)return 0;
781             if(c.relationseg(l[i])==1)x = 1;
782         }
783         return x;
784     }
785 };
786 //`AB X AC`
787 double cross(Point A,Point B,Point C){
788     return (B-A)^(C-A);
789 }
790 //`AB*AC`
791 double dot(Point A,Point B,Point C){
792     return (B-A)*(C-A);
793 }
794 //`最小矩形面积覆盖`
795 //` A 必须是凸包 (而且是逆时针顺序)`
796 //` 测试 UVA 10173`
797 double minRectangleCover(polygon A){
798     //`要特判 A.n < 3 的情况`
799     if(A.n < 3)return 0.0;
800     A.p[A.n] = A.p[0];
801     double ans = -1;
802     int r = 1, p = 1, q;
803     for(int i = 0;i < A.n;i++){
804         //`卡出离边 A.p[i] - A.p[i+1] 最远的点`
805         while( sgn( cross(A.p[i],A.p[i+1],A.p[r+1]) -
            ↪  cross(A.p[i],A.p[i+1],A.p[r]) ) >= 0 )
806             r = (r+1)%A.n;
807         //`卡出 A.p[i] - A.p[i+1] 方向上正向 n 最远的点`
808         while(sgn( dot(A.p[i],A.p[i+1],A.p[p+1]) -
            ↪  dot(A.p[i],A.p[i+1],A.p[p]) ) >= 0 )
809             p = (p+1)%A.n;
810         if(i == 0)q = p;
811         //`卡出 A.p[i] - A.p[i+1] 方向上负向最远的点`
812         while(sgn(dot(A.p[i],A.p[i+1],A.p[q+1]) -
            ↪  dot(A.p[i],A.p[i+1],A.p[q])) <= 0)
813             q = (q+1)%A.n;
814         double d = (A.p[i] - A.p[i+1]).len2();
815         double tmp = cross(A.p[i],A.p[i+1],A.p[r]) *
816             (dot(A.p[i],A.p[i+1],A.p[p]) -
                ↪  dot(A.p[i],A.p[i+1],A.p[q]))/d;
817         if(ans < 0 || ans > tmp)ans = tmp;
818     }
```

```
819     return ans;
820 }
821
822 //`直线切凸多边形`
823 //`多边形是逆时针的, 在 q1q2 的左侧`
824 //`测试:HDU3982`
825 vector<Point> convexCut(const vector<Point> &ps,Point q1,Point q2){
826     vector<Point>qs;
827     int n = ps.size();
828     for(int i = 0;i < n;i++){
829         Point p1 = ps[i], p2 = ps[(i+1)%n];
830         int d1 = sgn((q2-q1)^(p1-q1)), d2 = sgn((q2-q1)^(p2-q1));
831         if(d1 >= 0)
832             qs.push_back(p1);
833         if(d1 * d2 < 0)
834             qs.push_back(Line(p1,p2).crosspoint(Line(q1,q2)));
835     }
836     return qs;
837 }
838 //`半平面交`
839 //`测试 POJ3335 POJ1474 POJ1279`
840 //**************************
841 struct halfplane:public Line{
842     double angle;
843     halfplane(){}
844     //`表示向量 s->e 逆时针 (左侧) 的半平面`
845     halfplane(Point _s,Point _e){
846         s = _s;
847         e = _e;
848     }
849     halfplane(Line v){
850         s = v.s;
851         e = v.e;
852     }
853     void calcangle(){
854         angle = atan2(e.y-s.y,e.x-s.x);
855     }
856     bool operator <(const halfplane &b)const{
857         return angle < b.angle;
858     }
859 };
860 struct halfplanes{
861     int n;
862     halfplane hp[maxp];
863     Point p[maxp];
864     int que[maxp];
865     int st,ed;
866     void push(halfplane tmp){
867         hp[n++] = tmp;
868     }
869     //去重
870     void unique(){
871         int m = 1;
```

```
872         for(int i = 1;i < n;i++){
873             if(sgn(hp[i].angle-hp[i-1].angle) != 0)
874                 hp[m++] = hp[i];
875             else if(sgn( (hp[m-1].e-hp[m-1].s)^(hp[i].s-hp[m-1].s) ) > 0)
876                 hp[m-1] = hp[i];
877         }
878         n = m;
879     }
880     bool halfplaneinsert(){
881         for(int i = 0;i < n;i++)hp[i].calcangle();
882         sort(hp,hp+n);
883         unique();
884         que[st=0] = 0;
885         que[ed=1] = 1;
886         p[1] = hp[0].crosspoint(hp[1]);
887         for(int i = 2;i < n;i++){
888             while(st<ed && sgn((hp[i].e-hp[i].s)^(p[ed]-hp[i].s))<0)ed--;
889             while(st<ed &&
                ↪  sgn((hp[i].e-hp[i].s)^(p[st+1]-hp[i].s))<0)st++;
890             que[++ed] = i;
891             if(hp[i].parallel(hp[que[ed-1]]))return false;
892             p[ed]=hp[i].crosspoint(hp[que[ed-1]]);
893         }
894         while(st<ed &&
            ↪  sgn((hp[que[st]].e-hp[que[st]].s)^(p[ed]-hp[que[st]].s))<0)ed--;
895         while(st<ed &&
            ↪  sgn((hp[que[ed]].e-hp[que[ed]].s)^(p[st+1]-hp[que[ed]].s))<0)st++;
896         if(st+1>=ed)return false;
897         return true;
898     }
899     //`得到最后半平面交得到的凸多边形`
900     //`需要先调用 halfplaneinsert() 且返回 true`
901     void getconvex(polygon &con){
902         p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
903         con.n = ed-st+1;
904         for(int j = st,i = 0;j <= ed;i++,j++)
905             con.p[i] = p[j];
906     }
907 };
908 //***************************
909
910 const int maxn = 1010;
911 struct circles{
912     circle c[maxn];
913     double ans[maxn];//`ans[i] 表示被覆盖了 i 次的面积`
914     double pre[maxn];
915     int n;
916     circles(){}
917     void add(circle cc){
918         c[n++] = cc;
919     }
920     //`x 包含在 y 中`
921     bool inner(circle x,circle y){
```

```
922         if(x.relationcircle(y) != 1)return 0;
923         return sgn(x.r-y.r)<=0?1:0;
924     }
925     //圆的面积并去掉内含的圆
926     void init_or(){
927         bool mark[maxn] = {0};
928         int i,j,k=0;
929         for(i = 0;i < n;i++){
930             for(j = 0;j < n;j++)
931                 if(i != j && !mark[j]){
932                     if( (c[i]==c[j])||inner(c[i],c[j]) )break;
933                 }
934             if(j < n)mark[i] = 1;
935         }
936         for(i = 0;i < n;i++)
937             if(!mark[i])
938                 c[k++] = c[i];
939         n = k;
940     }
941     //`圆的面积交去掉内含的圆`
942     void init_add(){
943         int i,j,k;
944         bool mark[maxn] = {0};
945         for(i = 0;i < n;i++){
946             for(j = 0;j < n;j++)
947                 if(i != j && !mark[j]){
948                     if( (c[i]==c[j])||inner(c[j],c[i]) )break;
949                 }
950             if(j < n)mark[i] = 1;
951         }
952         for(i = 0;i < n;i++)
953             if(!mark[i])
954                 c[k++] = c[i];
955         n = k;
956     }
957     //`半径为 r 的圆，弧度为 th 对应的弓形的面积`
958     double areaarc(double th,double r){
959         return 0.5*r*r*(th-sin(th));
960     }
961     //`测试 SPOJVCIRCLES SPOJCIRUT`
962     //`SPOJVCIRCLES 求 n 个圆并的面积，需要加上 init\_or() 去掉重复圆（否则
        ↪ WA）`
963     //`SPOJCIRUT 是求被覆盖 k 次的面积，不能加 init\_or()`
964     //`对于求覆盖多少次面积的问题，不能解决相同圆，而且不能 init\_or()`
965     //`求多圆面积并，需要 init\_or，其中一个目的就是去掉相同圆`
966     void getarea(){
967         memset(ans,0,sizeof(ans));
968         vector<pair<double,int> >v;
969         for(int i = 0;i < n;i++){
970             v.clear();
971             v.push_back(make_pair(-pi,1));
972             v.push_back(make_pair(pi,-1));
973             for(int j = 0;j < n;j++)
```

```
                if(i != j){
                    Point q = (c[j].p - c[i].p);
                    double ab = q.len(),ac = c[i].r, bc = c[j].r;
                    if(sgn(ab+ac-bc)<=0){
                        v.push_back(make_pair(-pi,1));
                        v.push_back(make_pair(pi,-1));
                        continue;
                    }
                    if(sgn(ab+bc-ac)<=0)continue;
                    if(sgn(ab-ac-bc)>0)continue;
                    double th = atan2(q.y,q.x), fai =
                    ↪  acos((ac*ac+ab*ab-bc*bc)/(2.0*ac*ab));
                    double a0 = th-fai;
                    if(sgn(a0+pi)<0)a0+=2*pi;
                    double a1 = th+fai;
                    if(sgn(a1-pi)>0)a1-=2*pi;
                    if(sgn(a0-a1)>0){
                        v.push_back(make_pair(a0,1));
                        v.push_back(make_pair(pi,-1));
                        v.push_back(make_pair(-pi,1));
                        v.push_back(make_pair(a1,-1));
                    }
                    else{
                        v.push_back(make_pair(a0,1));
                        v.push_back(make_pair(a1,-1));
                    }
                }
            sort(v.begin(),v.end());
            int cur = 0;
            for(int j = 0;j < v.size();j++){
                if(cur && sgn(v[j].first-pre[cur])){
                    ans[cur] += areaarc(v[j].first-pre[cur],c[i].r);
                    ans[cur] += 0.5*(Point(c[i].p.x+c[i].r*cos(pre[cur])
                    ↪  , c[i].p.y+c[i].r * sin(pre[cur])) ^
                    ↪  Point(c[i].p.x+c[i].r *
                    ↪  cos(v[j].first),c[i].p.y+c[i].r *
                    ↪  sin(v[j].first)));
                }
                cur += v[j].second;
                pre[cur] = v[j].first;
            }
        }
        for(int i = 1;i < n;i++)
            ans[i] -= ans[i+1];
    }
};
```

4.2 三维几何

```
const double eps = 1e-8;
int sgn(double x){
        if(fabs(x) < eps)return 0;
        if(x < 0)return -1;
        else return 1;
```

```
6   }
7   struct Point3{
8           double x,y,z;
9           Point3(double _x = 0,double _y = 0,double _z = 0){
10                  x = _x;
11                  y = _y;
12                  z = _z;
13          }
14          void input(){
15                  scanf("%lf%lf%lf",&x,&y,&z);
16          }
17          void output(){
18                  printf("%.2lf %.2lf %.2lf\n",x,y,z);
19          }
20          bool operator ==(const Point3 &b)const{
21                  return sgn(x-b.x) == 0 && sgn(y-b.y) == 0 && sgn(z-b.z)
                    ↪  == 0;
22          }
23          bool operator <(const Point3 &b)const{
24                  return sgn(x-b.x)==0 ?
                    ↪  (sgn(y-b.y)==0?sgn(z-b.z)<0:y<b.y):x<b.x;
25          }
26          double len(){
27                  return sqrt(x*x+y*y+z*z);
28          }
29          double len2(){
30                  return x*x+y*y+z*z;
31          }
32          double distance(const Point3 &b)const{
33                  return sqrt((x-b.x)*(x-b.x) + (y-b.y)*(y-b.y) +
                    ↪  (z-b.z)*(z-b.z));
34          }
35          Point3 operator -(const Point3 &b)const{
36                  return Point3(x-b.x,y-b.y,z-b.z);
37          }
38          Point3 operator +(const Point3 &b)const{
39                  return Point3(x+b.x,y+b.y,z+b.z);
40          }
41          Point3 operator *(const double &k)const{
42                  return Point3(x*k,y*k,z*k);
43          }
44          Point3 operator /(const double &k)const{
45                  return Point3(x/k,y/k,z/k);
46          }
47          //点乘
48          double operator *(const Point3 &b)const{
49                  return x*b.x+y*b.y+z*b.z;
50          }
51          //叉乘
52          Point3 operator ^(const Point3 &b)const{
53                  return Point3(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
54          }
55          double rad(Point3 a,Point3 b){
```

```
56                  Point3 p = (*this);
57                  return acos( ( (a-p)*(b-p) ) /
                    ↪  (a.distance(p)*b.distance(p)) );
58          }
59          //变换长度
60          Point3 trunc(double r){
61                  double l = len();
62                  if(!sgn(l))return *this;
63                  r /= l;
64                  return Point3(x*r,y*r,z*r);
65          }
66  };
67  struct Line3
68  {
69          Point3 s,e;
70          Line3(){}
71          Line3(Point3 _s,Point3 _e)
72          {
73                  s = _s;
74                  e = _e;
75          }
76          bool operator ==(const Line3 v)
77          {
78                  return (s==v.s)&&(e==v.e);
79          }
80          void input()
81          {
82                  s.input();
83                  e.input();
84          }
85          double length()
86          {
87                  return s.distance(e);
88          }
89          //点到直线距离
90          double dispointtoline(Point3 p)
91          {
92                  return ((e-s)^(p-s)).len()/s.distance(e);
93          }
94          //点到线段距离
95          double dispointtoseg(Point3 p)
96          {
97                  if(sgn((p-s)*(e-s)) < 0 || sgn((p-e)*(s-e)) < 0)
98                          return min(p.distance(s),e.distance(p));
99                  return dispointtoline(p);
100         }
101         //`返回点 p 在直线上的投影`
102         Point3 lineprog(Point3 p)
103         {
104                 return s+(((e-s)*((e-s)*(p-s))) / ((e-s).len2()));
105         }
106         //`p 绕此向量逆时针 arg 角度`
107         Point3 rotate(Point3 p,double ang)
```

```
108         {
109                 if(sgn(((s-p)^(e-p)).len()) == 0)return p;
110                 Point3 f1 = (e-s)^(p-s);
111                 Point3 f2 = (e-s)^(f1);
112                 double len = ((s-p)^(e-p)).len()/s.distance(e);
113                 f1 = f1.trunc(len); f2 = f2.trunc(len);
114                 Point3 h = p+f2;
115                 Point3 pp = h+f1;
116                 return h + ((p-h)*cos(ang)) + ((pp-h)*sin(ang));
117         }
118         //`点在直线上`
119         bool pointonseg(Point3 p)
120         {
121                 return sgn( ((s-p)^(e-p)).len() ) == 0 &&
                    ↪  sgn((s-p)*(e-p)) == 0;
122         }
123 };
124 struct Plane
125 {
126         Point3 a,b,c,o;//`平面上的三个点，以及法向量`
127         Plane(){}
128         Plane(Point3 _a,Point3 _b,Point3 _c)
129         {
130                 a = _a;
131                 b = _b;
132                 c = _c;
133                 o = pvec();
134         }
135         Point3 pvec()
136         {
137                 return (b-a)^(c-a);
138         }
139         //`ax+by+cz+d = 0`
140         Plane(double _a,double _b,double _c,double _d)
141         {
142                 o = Point3(_a,_b,_c);
143                 if(sgn(_a) != 0)
144                         a = Point3((-_d-_c-_b)/_a,1,1);
145                 else if(sgn(_b) != 0)
146                         a = Point3(1,(-_d-_c-_a)/_b,1);
147                 else if(sgn(_c) != 0)
148                         a = Point3(1,1,(-_d-_a-_b)/_c);
149         }
150         //`点在平面上的判断`
151         bool pointonplane(Point3 p)
152         {
153                 return sgn((p-a)*o) == 0;
154         }
155         //`两平面夹角`
156         double angleplane(Plane f)
157         {
158                 return acos(o*f.o)/(o.len()*f.o.len());
159         }
```

```
        //`平面和直线的交点，返回值是交点个数`
        int crossline(Line3 u,Point3 &p)
        {
                double x = o*(u.e-a);
                double y = o*(u.s-a);
                double d = x-y;
                if(sgn(d) == 0)return 0;
                p = ((u.s*x)-(u.e*y))/d;
                return 1;
        }
        //`点到平面最近点（也就是投影）`
        Point3 pointtoplane(Point3 p)
        {
                Line3 u = Line3(p,p+o);
                crossline(u,p);
                return p;
        }
        //`平面和平面的交线`
        int crossplane(Plane f,Line3 &u)
        {
                Point3 oo = o^f.o;
                Point3 v = o^oo;
                double d = fabs(f.o*v);
                if(sgn(d) == 0)return 0;
                Point3 q = a + (v*(f.o*(f.a-a))/d);
                u = Line3(q,q+oo);
                return 1;
        }
};
```

4.3 平面最近点对

```
const int MAXN = 100010;
const double eps = 1e-8;
const double INF = 1e20;
struct Point{
        double x,y;
        void input(){
                scanf("%lf%lf",&x,&y);
        }
};
double dist(Point a,Point b){
        return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
Point p[MAXN];
Point tmpt[MAXN];
bool cmpx(Point a,Point b){
        return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool cmpy(Point a,Point b){
        return a.y < b.y || (a.y == b.y && a.x < b.x);
}
double Closest_Pair(int left,int right){
        double d = INF;
```

```
        if(left == right)return d;
        if(left+1 == right)return dist(p[left],p[right]);
        int mid = (left+right)/2;
        double d1 = Closest_Pair(left,mid);
        double d2 = Closest_Pair(mid+1,right);
        d = min(d1,d2);
        int cnt = 0;
        for(int i = left;i <= right;i++){
                if(fabs(p[mid].x - p[i].x) <= d)
                        tmpt[cnt++] = p[i];
        }
        sort(tmpt,tmpt+cnt,cmpy);
        for(int i = 0;i < cnt;i++){
                for(int j = i+1;j < cnt && tmpt[j].y - tmpt[i].y < d;j++)
                        d = min(d,dist(tmpt[i],tmpt[j]));
        }
        return d;
}
int main(){
        int n;
        while(scanf("%d",&n) == 1 && n){
                for(int i = 0;i < n;i++)p[i].input(); //输入点的坐标
                sort(p,p+n,cmpx);
                printf("%.2lf\n",Closest_Pair(0,n-1)); //返回平面最近点对
                ↪ 之间的距离
        }
    return 0;
}
```

4.4 三维凸包

```
const double eps = 1e-8;
const int MAXN = 550;
int sgn(double x){
        if(fabs(x) < eps)return 0;
        if(x < 0)return -1;
        else return 1;
}
struct Point3{
        double x,y,z;
        Point3(double _x = 0, double _y = 0, double _z = 0){
                x = _x;
                y = _y;
                z = _z;
        }
        void input(){
                scanf("%lf%lf%lf",&x,&y,&z);
        }
        bool operator ==(const Point3 &b)const{
                return sgn(x-b.x) == 0 && sgn(y-b.y) == 0 && sgn(z-b.z)
                ↪ == 0;
        }
        double len(){
                return sqrt(x*x+y*y+z*z);
```

```
23          }
24          double len2(){
25                  return x*x+y*y+z*z;
26          }
27          double distance(const Point3 &b)const{
28                  return
                    ↪  sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y)+(z-b.z)*(z-b.z));
29          }
30          Point3 operator -(const Point3 &b)const{
31                  return Point3(x-b.x,y-b.y,z-b.z);
32          }
33          Point3 operator +(const Point3 &b)const{
34                  return Point3(x+b.x,y+b.y,z+b.z);
35          }
36          Point3 operator *(const double &k)const{
37                  return Point3(x*k,y*k,z*k);
38          }
39          Point3 operator /(const double &k)const{
40                  return Point3(x/k,y/k,z/k);
41          }
42          //点乘
43          double operator *(const Point3 &b)const{
44                  return x*b.x + y*b.y + z*b.z;
45          }
46          //叉乘
47          Point3 operator ^(const Point3 &b)const{
48                  return Point3(y*b.z-z*b.y , z*b.x-x*b.z , x*b.y-y*b.x);
49          }
50  };
51  struct CH3D{
52          struct face{
53                  //表示凸包一个面上的三个点的编号
54                  int a,b,c;
55                  //表示该面是否属于最终的凸包上的面
56                  bool ok;
57          };
58          //初始顶点数
59          int n;
60          Point3 P[MAXN];
61          //凸包表面的三角形数
62          int num;
63          //凸包表面的三角形
64          face F[8*MAXN];
65          int g[MAXN][MAXN];
66          //叉乘
67          Point3 cross(const Point3 &a,const Point3 &b,const Point3 &c){
68                  return (b-a)^(c-a);
69          }
70          //`三角形面积 *2`
71          double area(Point3 a,Point3 b,Point3 c){
72                  return ((b-a)^(c-a)).len();
73          }
74          //`四面体有向面积 *6`
```

```
75          double volume(Point3 a,Point3 b,Point3 c,Point3 d){
76                  return ((b-a)^(c-a))*(d-a);
77          }
78          //`正：点在面同向`
79          double dblcmp(Point3 &p,face &f){
80                  Point3 p1 = P[f.b] - P[f.a];
81                  Point3 p2 = P[f.c] - P[f.a];
82                  Point3 p3 = p - P[f.a];
83                  return (p1^p2)*p3;
84          }
85          void deal(int p,int a,int b){
86                  int f = g[a][b];
87                  face add;
88                  if(F[f].ok){
89                          if(dblcmp(P[p],F[f]) > eps)
90                                  dfs(p,f);
91                          else {
92                                  add.a = b;
93                                  add.b = a;
94                                  add.c = p;
95                                  add.ok = true;
96                                  g[p][b] = g[a][p] = g[b][a] = num;
97                                  F[num++] = add;
98                          }
99                  }
100         }
101         //递归搜索所有应该从凸包内删除的面
102         void dfs(int p,int now){
103                 F[now].ok = false;
104                 deal(p,F[now].b,F[now].a);
105                 deal(p,F[now].c,F[now].b);
106                 deal(p,F[now].a,F[now].c);
107         }
108         bool same(int s,int t){
109                 Point3 &a = P[F[s].a];
110                 Point3 &b = P[F[s].b];
111                 Point3 &c = P[F[s].c];
112                 return fabs(volume(a,b,c,P[F[t].a])) < eps &&
113                         fabs(volume(a,b,c,P[F[t].b])) < eps &&
114                         fabs(volume(a,b,c,P[F[t].c])) < eps;
115         }
116         //构建三维凸包
117         void create(){
118                 num = 0;
119                 face add;
120
121                 //*********************************
122                 //此段是为了保证前四个点不共面
123                 bool flag = true;
124                 for(int i = 1;i < n;i++){
125                         if(!(P[0] == P[i])){
126                                 swap(P[1],P[i]);
127                                 flag = false;
```

```
                                break;
                        }
                }
                if(flag)return;
                flag = true;
                for(int i = 2;i < n;i++){
                        if( ((P[1]-P[0])^(P[i]-P[0])).len() > eps ){
                                swap(P[2],P[i]);
                                flag = false;
                                break;
                        }
                }
                if(flag)return;
                flag = true;
                for(int i = 3;i < n;i++){
                        if(fabs( ((P[1]-P[0])^(P[2]-P[0]))*(P[i]-P[0]) )
                        ↪  > eps){
                                swap(P[3],P[i]);
                                flag = false;
                                break;
                        }
                }
                if(flag)return;
                //**********************************

                for(int i = 0;i < 4;i++){
                        add.a = (i+1)%4;
                        add.b = (i+2)%4;
                        add.c = (i+3)%4;
                        add.ok = true;
                        if(dblcmp(P[i],add) > 0)swap(add.b,add.c);
                        g[add.a][add.b] = g[add.b][add.c] =
                        ↪  g[add.c][add.a] = num;
                        F[num++] = add;
                }
                for(int i = 4;i < n;i++)
                        for(int j = 0;j < num;j++)
                                if(F[j].ok && dblcmp(P[i],F[j]) > eps){
                                        dfs(i,j);
                                        break;
                                }
                int tmp = num;
                num = 0;
                for(int i = 0;i < tmp;i++)
                        if(F[i].ok)
                                F[num++] = F[i];
        }
        //表面积
        //`测试：HDU3528`
        double area(){
                double res = 0;
                if(n == 3){
                        Point3 p = cross(P[0],P[1],P[2]);
```

```
179                     return p.len()/2;
180             }
181             for(int i = 0;i < num;i++)
182                     res += area(P[F[i].a],P[F[i].b],P[F[i].c]);
183             return res/2.0;
184     }
185     double volume(){
186             double res = 0;
187             Point3 tmp = Point3(0,0,0);
188             for(int i = 0;i < num;i++)
189                     res += volume(tmp,P[F[i].a],P[F[i].b],P[F[i].c]);
190             return fabs(res/6);
191     }
192     //表面三角形个数
193     int triangle(){
194             return num;
195     }
196     //表面多边形个数
197     //`测试：HDU3662`
198     int polygon(){
199             int res = 0;
200             for(int i = 0;i < num;i++){
201                     bool flag = true;
202                     for(int j = 0;j < i;j++)
203                             if(same(i,j)){
204                                     flag = 0;
205                                     break;
206                             }
207                     res += flag;
208             }
209             return res;
210     }
211     //重心
212     //`测试：HDU4273`
213     Point3 barycenter(){
214             Point3 ans = Point3(0,0,0);
215             Point3 o = Point3(0,0,0);
216             double all = 0;
217             for(int i = 0;i < num;i++){
218                     double vol =
                        ↪  volume(o,P[F[i].a],P[F[i].b],P[F[i].c]);
219                     ans = ans +
                        ↪  (((o+P[F[i].a]+P[F[i].b]+P[F[i].c])/4.0)*vol);
220                     all += vol;
221             }
222             ans = ans/all;
223             return ans;
224     }
225     //点到面的距离
226     //`测试：HDU4273`
227     double ptoface(Point3 p,int i){
228             double tmp1 =
                ↪  fabs(volume(P[F[i].a],P[F[i].b],P[F[i].c],p));
```

```
229                 double tmp2 =
                    ↪  ((P[F[i].b]-P[F[i].a])^(P[F[i].c]-P[F[i].a])).len();
230                 return tmp1/tmp2;
231         }
232 };
233 CH3D hull;
234 //hdu4273 给一个三维凸包，求重心到表面的最短距离
235 int main()
236 {
237     while(scanf("%d",&hull.n) == 1){
238                 for(int i = 0;i < hull.n;i++) hull.P[i].input();
239                 hull.create();
240                 Point3 p = hull.barycenter();
241                 double ans = 1e20;
242                 for(int i = 0;i < hull.num;i++)
243                         ans = min(ans,hull.ptoface(p,i));
244                 printf("%.3lf\n",ans);
245         }
246     return 0;
247 }
```

5 图论

5.1 BFS

5.1.1 伪代码

```
/*
宽度优先，就是每次都尝试访问同一层的节点。如果同一层都访问完了，再访问下一层。
这样做的结果是，BFS 算法找到的路径是从起点开始的最短合法路径。换言之，这条路
↪ 所包含的边数最小。
在 BFS 结束时，每个节点都是通过从起点到该点的最短路径访问的。
*/
bfs(s) {
  q = new queue()
  q.push(s), visited[s] = true
  while (!q.empty()) {
    u = q.pop()
    for each edge(u, v) {
      if (!visited[v]) {
        q.push(v)
        visited[v] = true
      }
    }
  }
}
```

5.1.2 BFS 记录距离和路径

```
/*
队列 Q 记录要处理的节点，vis 数组来标记某个节点是否已经访问过。
d 数组记录某个点到起点的距离，可以得到起点到一个点的距离。
p 数组是记录从起点到这个点的最短路上的上一个点，可以方便地还原出起点到一个点
↪ 的最短路径。
restore(x) 输出的是从起点到 x 这个点所经过的点。

开始的时候，我们把起点 s 以外的节点的 vis 值设为 0，意思是没有访问过。然后把
↪ 起点 s 放入队列 Q 中。
之后，我们每次从队列 Q 中取出队首的点 u，把 u 相邻的所有点 v 标记为已经访问过
↪ 了并放入队列 Q。
直到某一时刻，队列 Q 为空，这时 BFS 结束。

时间复杂度 O(n + m)
空间复杂度 O(n)（vis 数组和队列）
*/
void bfs(int u) {
  while (!Q.empty()) Q.pop();
  Q.push(u);
  vis[u] = 1;
  d[u] = 0;
  p[u] = -1;
  while (!Q.empty()) {
    u = Q.front();
    Q.pop();
    for (int i = head[u]; i; i = e[i].x) {
      if (!vis[e[i].t]) {
```

```
        Q.push(e[i].t);
        vis[e[i].t] = 1;
        d[e[i].t] = d[u] + 1;
        p[e[i].t] = u;
      }
    }
  }
}
void restore(int x) {
  vector<int> res;
  for (int v = x; v != -1; v = p[v]) {
    res.push_back(v);
  }
  std::reverse(res.begin(), res.end());
  for (int i = 0; i < res.size(); ++i) printf("%d", res[i]);
  puts("");
}
```

5.2 最短路

5.2.1 Dijkstra 算法单源最短路

```
/*
Dijkstra 算法 + 堆优化，复杂度为 O(ElogE)
注意初始化
权值必须是非负
*/
const int N=100010,M=1000010;
int head[N],ver[M],edge[M],Next[M],d[N];
bool v[N];
int n,m,tot;
//大根堆 (优先队列),pair 的第二维为节点编号
//pair 的第一维为 dist 的相反数 (利用相反数变成小根堆)
priority_queue<pair<int,int> > q;
void add(int x,int y,int z){
    ver[++tot]=y,edge[tot]=z,Next[tot]=head[x],head[x]=tot;
}
void dijkstra(){
    memset(d,0x3f,sizeof(d));//dist 数组
    memset(v,0,sizeof(v));//节点标记
    //起始点为 1，若更改要改两处
    d[1]=0;
    q.push(make_pair(0,1));
    while(q.size()){
        int x=q.top().second;q.pop();
        if(v[x]) continue;
        v[x]=1;
        for(int i=head[x];i;i=Next[i]){
            int y=ver[i],z=edge[i];
            if(d[y]>d[x]+z){
                d[y]=d[x]+z;
                q.push(make_pair(-d[y],y));
            }
        }
```

```
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;++i){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        //如果是无向图还要 add(y,x,z)
        add(x,y,z);
    }
    dijkstra();
    //求单源最短路
    for(int i=1;i<=n;++i)
        printf("%d\n",d[i]);
    return 0;
}
```

5.2.2 bellman_ford 算法单源最短路

```
/*
单源最短路 bellman_ford 算法, 复杂度 O(VE)
可以处理负边权图
可以判断是否存在负环回路.
返回 true: 当且仅当图中不包含从源点可达的负权回路
vector<Edge> E; 先 E.clear() 初始化, 然后加入所有边
点的编号从 1 开始
*/
const int INF=0x3f3f3f3f;
const int N=2550;
int dist[N];
struct Edge
{
    int u,v;
    int cost;
    Edge(int _u=0,int _v=0,int _cost=0):u(_u),v(_v),cost(_cost){}
};
vector<Edge>E;
//点的编号从 1 开始
void addedge(int u,int v,int w){
    E.push_back(Edge(u,v,w));
}
bool bellman_ford(int start,int n){
    for(int i=1;i<=n;++i)   dist[i]=INF;
    dist[start]=0;
    for(int i=1;i<n;++i){
        bool flag=false;
        for(int j=0;j<E.size();j++){
            int u=E[j].u;
            int v=E[j].v;
            int cost=E[j].cost;
            if(dist[v]>dist[u]+cost){
                dist[v]=dist[u]+cost;
                flag=true;
```

```
            }
        }
        if(!flag) return true;//没有负环回路
    }
    for(int j=0;j<E.size();j++)
        if(dist[E[j].v]>dist[E[j].u]+E[j].cost)
            return false;//有负环回路
    return true;//没有负环回路
}
int main()
{
    int n,m;cin>>n>>m;
    E.clear();
    for(int i=1;i<=m;++i){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        addedge(x,y,z);
        addedge(y,x,z);
    }
    bellman_ford(s,n);
    for(int i=1;i<=n;++i)
        printf("%d\n",dist[i]);
    return 0;
}
```

5.2.3 Floyd 任意两点最短路径

```
/*
Floyd 算法 计算任意两点间的距离 复杂度为 O(N^3)
*/
#define inf 0x3f3f3f3f
int dist[1000][1000];
int main()
{
    int k,i,j,n,m;///n 表示顶点个数, m 表示边的条数
    scanf("%d %d",&n,&m);
    for(i=1; i<=n; i++){//初始化
        for(j=1; j<=n; j++){
            if(i==j) dist[i][j]=0;
            else dist[i][j]=inf;
        }
    }
    int a,b,c;
    for(i=1; i<=m; i++){//有向图
        scanf("%d %d %d",&a,&b,&c);
        dist[a][b]=min(dist[a][b],c);
    }
    for(k=1; k<=n; k++){    //Floyd-Warshall 算法核心语句
        for(i=1; i<=n; i++){
            for(j=1; j<=n; j++){
                dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
    for(i=1; i<=n; i++){//输出最终的结果, 最终二维数组中存的即使两点之间的
    ↪ 最短距离
        for(j=1; j<=n; j++)
```

```
            printf("%10d",dist[i][j]);
        printf("\n");
    }
    return 0;
}
```

5.3 最小生成树

无向图中，其某个子图中任意两个顶点都互相连通并且是一棵树，称之为生成树；若每个顶点有权值，则其权值和最小的生成树为最小生成树。

5.3.1 Prim 算法

```
/*
Prim 求 MST
耗费矩阵 cost[][], 标号从 0 开始,0~n-1 注意注意!!!
返回最小生成树的权值, 返回-1 则表示原图不连通
*/
const int INF=0x3f3f3f3f;
const int MAXN=110;
bool vis[MAXN];
int lowc[MAXN];
//点从 0 到 n-1
int Prim(int cost[][MAXN],int n){
    int ans=0;
    memset(vis,0,sizeof vis);
    vis[0]=true;
    for(int i=1;i<n;++i)   lowc[i]=cost[0][i];
    for(int i=1;i<n;++i){
        int minc=INF;
        int p=-1;
        for(int j=0;j<n;++j){
            if(!vis[j]&&minc>lowc[j]){
                minc=lowc[j];
                p=j;
            }
        }
        if(minc==INF) return -1;//原图不连通
        ans+=minc;
        vis[p]=true;
        for(int j=0;j<n;++j){
            if(!vis[j] && lowc[j]>cost[p][j])
                lowc[j]=cost[p][j];
        }
    }
    return ans;
}
int main()
{
    int cost[MAXN][MAXN];
    //注意对耗费矩阵赋初值,memset 只能读 1 个字符, 这样和都赋给 INF 结果一样
    memset(cost,0x3f,sizeof(cost));
    while (k--){
        int u,v,w;
```

```
        cin>>u>>v>>w;
        //读入数据，注意题目给出的节点是否从 0 开始且无向边要两次赋值
        u--;
        v--;
        cost[u][v]=w;
        cost[v][u]=w;
    }
    cout<<Prim(cost,n)<<endl;
    return 0;
}
```

5.3.2 Kruskal 算法

```
/*
Kruskal 算法求 MST
*/
//根据题目调试最大点数和边数
const int MAXN=1100;//最大点数
const int MAXM=200005;//最大边数
int F[MAXN];//并查集
struct Edge {
    int u,v,w;
}edge[MAXM];//储存边的信息：起点，终点，权值
int tol=0;//边数，记得赋值为 0
void addedge(int u,int v,int w){//加边
    edge[tol].u=u;
    edge[tol].v=v;
    edge[tol++].w=w;
}
bool cmp(Edge a,Edge b){//排序
    return a.w<b.w;
}
int find(int x){
    return F[x]==x?x:F[x]=find(F[x]);
}
//传入点数，返回最小生成树权值，如果不连通返回-1
int Kruskal(int n){
    //根据点的编号（从 0 或从 1 开始）初始并查集
    for(int i=1;i<=n;++i) F[i]=i;
    sort(edge,edge+tol,cmp);
    int cnt=0;//计算加入边数
    int ans=0;
    for(int i=0;i<tol;++i){
        int u=edge[i].u;
        int v=edge[i].v;
        int w=edge[i].w;
        int t1=find(u);
        int t2=find(v);
        if(t1!=t2){
            ans+=w;
            F[t1]=t2;
            cnt++;
        }
        if(cnt==n-1) break;
```

```
42      }
43      if(cnt<n-1) return -1;//不连通
44      else return ans;
45  }
46  int main()
47  {
48      int n,m;cin>>n>>m;//点数和边数
49      for(int i=0;i<m;++i){
50          int u,v,w;cin>>u>>v>>w;
51          addedge(u,v,w);//只用读一次就行
52      }
53      cout<<Kruskal(n);//返回最小生成树权值
54      return 0;
55  }
```

6 动态规划

6.1 悬线法

悬线法的用途：针对求给定矩阵中满足某条件的极大矩阵，比如“面积最大的长方形、正方形”“周长最长的矩形等等”。

悬线法的基本思路：维护三个二维数组，Left,Right,Up 数组。

Left 数组存储从 map[i][j] 这个点出发，满足条件能到达的最左边地方。

Right 数组存储从 map[i][j] 这个点出发，满足条件能到达的最右边地方。

Up 数组存储从这点以上满足条件的能到达的最大长度。

递推公式：

Up：Up[i][j] = Up[i-1][j] + 1

Right：min(Right[i][j]，Right[i-1],[j])

Left::：max(Left[i][j]，Left[i-1][j])

```
/*
悬线法求最大全 1 子矩阵
*/
const int MAXN=2005;
char x[MAXN][MAXN];
int y[MAXN][MAXN],l[MAXN][MAXN],r[MAXN][MAXN],up[MAXN][MAXN];
int main(){
    int n,m; cin>>n>>m;
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j){
            cin>>x[i][j];
            //将输入处理成 01 矩阵
            if(x[i][j]=='F'){
                //此处赋值需注意，只有 1 处才赋值，否则全 0 矩阵也会输出面积为 1
                y[i][j]=1;
                l[i][j]=j;
                r[i][j]=j;
                up[i][j]=1;
            }
            else
                y[i][j]=0;
        }
    }
    //按行处理
    for(int i=1;i<=n;++i){
        for(int j=2;j<=m;++j){
            if(y[i][j-1] && y[i][j])
                l[i][j]=l[i][j-1];
        }
        for(int j=m-1;j>=1;--j){
            if(y[i][j+1] && y[i][j])
                r[i][j]=r[i][j+1];
        }
    }
    int len,ans=0;
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j){
            //i>1 很关键，用于处理只有 1 行的情况
            if(i>1 && y[i-1][j] && y[i][j]){
```

```
                up[i][j]=up[i-1][j]+1;
                l[i][j]=max(l[i][j],l[i-1][j]);
                r[i][j]=min(r[i][j],r[i-1][j]);
            }
            len=r[i][j]-l[i][j]+1;
            ans=max(ans,len*up[i][j]);
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

6.2 LIS 最长上升子序列

```
/*
动态规划求最长上升子序列，复杂度为 O(nlogn)
*/
int DP(int n){
    int i,len,pos;
    b[1]=a[1];
    len=1;
    for(i=2;i<=n;++i){
        //a[i] 大则可直接插到后面
        if(a[i]>=b[len]){
            len=len+1;
            b[len]=a[i];
        }
        //二分法在 b 数组中找出第一个比 a[i] 大的位置并让 a[i] 代替
        else {
            pos=lower_bound(b+1,b+1+len,a[i])-b;
            b[pos]=a[i];
        }
    }
    return len;
}
```

6.3 背包

6.3.1 基础背包

```
int nValue,nKind;
const int N=100005;
int dp[N],cost[N],value[N],num[N];
//0-1 背包，代价为 cost，价值为 value
void ZeroOnePack(int cost,int value){
    for(int i=nValue;i>=cost;i--)
        dp[i]=max(dp[i],dp[i-cost]+value);
}
//完全背包，代价为 cost，价值为 value
void CompletePack(int cost,int value){
    for(int i=cost;i<=nValue;++i)
        dp[i]=max(dp[i],dp[i-cost]+value);
}
//多重背包，代价为 cost，价值为 value，数量为 amount
```

```
15  void MultiplePack(int cost,int value,int amount){
16      if(cost*amount>=nValue)
17          CompletePack(cost,value);
18      else {
19          int k=1;
20          while(k<amount){
21              ZeroOnePack(k*cost,k*value);
22              amount-=k;
23              k<<=1;
24          }
25          ZeroOnePack(amount*cost,amount*value);
26      }
27  }
28  int main(){
29      //记得清空数组
30      memset(dp,0,sizeof dp);
31      //0-1 背包
32      rep(i,1,nKind)
33          ZeroOnePack(cost[i],value[i]);
34      printf("%d\n",dp[nValue]);
35      //完全背包
36      rep(i,1,nKind)
37          CompletePack(cost[i],value[i]);
38      printf("%d\n",dp[nValue]);
39      //多重背包
40      rep(i,1,nKind)
41          MultiplePack(cost[i],value[i],num[i]);
42      printf("%d\n",dp[nValue]);
43      //分组背包
44      rep(k,1,ts)//循环每一组
45          for(int i=nValue;i>=0;--i)//循环背包容量
46              rep(j,1,cnt[k])//循环该组的每一个物品
47                  if (i >= cost[t[k][j]])
48                  dp[i]=max(dp[i],dp[i-cost[t[k][j]]]+value[t[k][j]]);
49                  //像 0-1 背包一样状态转移
50      return 0;
51  }
```

7 杂项

7.1 输入输出

7.1.1 scanf 和 prinf 的用法

```
/*printf 一些用法:
%d: 输出 int
%lld: 输出 long long
%lf: 输出 double
%.3lf: 输出四舍五入保留 3 位小数
%3d: 输出 3 位整型数, 不够 3 位右对齐
%-3d: 输出 3 位整型数, 不够 3 位左对齐
%9.2f: 输出场宽为 9 的浮点数, 其中小数位为 2, 整数位为 7, 小数点占一位, 不够 9 位右对齐。
%8s: 输出 8 个字符的字符串, 不够 8 个字符右对齐。
*/
/*
例题: 题目的输入形式为
1
00:00:00,498 -> 00:00:02,827
- Here's what I love most
about food and diet.

2
00:00:02,827 -> 00:00:06,383
We all eat several times a day,

3
00:00:06,383 -> 00:00:09,427
of what goes on our plate
and what stays off.
*/

/*
要求 1: 是把每个输入的第二行加上一个数
对于第二行的输入和输入可以使用以下形式:
*/
scanf("%d:%d:%d,%d",&a,&b,&c,&d);
solve(a,b,c,d);//计算加上一个数
printf("%02d:%02d:%02d,%03d --> ",a,b,c,d);
scanf(" --> %d:%d:%d,%d",&a,&b,&c,&d);
solve(a,b,c,d);
printf("%02d:%02d:%02d,%03d",a,b,c,d);

/*
要求 2: 之后的输入可能有多行, 按输入输出即可
用如下方式处理可能的输入
*/
while(getline(cin,s)){
    if(s==kk)//预先处理终止条件
        break;
    else
        cout<<s<<endl;
```

```
}
```

7.1.2 快读

```
//快读
template <typename T> T &read(T &r) {
    r = 0; bool w = 0; char ch = getchar();
    while(ch < '0' || ch > '9') w = ch == '-' ? 1 : 0, ch = getchar();
    while(ch >= '0' && ch <= '9') r = (r << 3) + (r <<1) + (ch ^ 48), ch
    ↪  = getchar();
    return r = w ? -r : r;
}
//用法:
read(n);
```

7.1.3 关闭同步

```
//关闭同步
ios::sync_with_stdio(0);
cin.tie(0);
```

7.1.4 快读快写

```
#include <bits/stdc++.h>
using namespace std;
//快读
inline int read() {
    int num=0, w=0;
    char ch=0;
    while (!isdigit(ch)) {
        w|=ch=='-';
        ch = getchar();
    }
    while (isdigit(ch)) {
        num = (num<<3) + (num<<1) + (ch^48);
        ch = getchar();
    }
    return w? -num: num;
}
//快写
inline void write(int x)
{
    if(x<0) {
        putchar('-');
        x = -x;
    }
    if(x>9) write(x / 10);
    putchar(x % 10 + '0');
}


int main(){
    int a;
    a = read();      //读入到 t 中
    write(t);        //输出 t
```

```
    putchar('\n');
}
```

7.1.5 8.11 所用板子

```
#include<bits/stdc++.h>
using namespace std;
#define ll  long long
#define ull unsigned long long
#define ld long double
#define rep(i, s, t) for (int i = (int)(s); i <= (int)(t); ++ i)
#define debug(x) cerr<<#x<<" = "<<(x)<<endl
template <typename T> T &read(T &r) {
    r = 0; bool w = 0; char ch = getchar();
    while(ch < '0' || ch > '9') w = ch == '-' ? 1 : 0, ch = getchar();
    while(ch >= '0' && ch <= '9') r = (r << 3) + (r <<1) + (ch ^ 48), ch
    ↪  = getchar();
    return r = w ? -r : r;
}
ll mod=1e9+7;
ll INF=1e15;
ll Inf=0x3f3f3f3f;
double pi=acos(-1.0);


int  main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    return 0;
}
```

7.2 高精度

7.2.1 int128

```
//int128 是 ll 的两倍，需要自己写输入输出
#include <bits/stdc++.h>
using namespace std;
inline __int128 read(){
    __int128 x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-')
            f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9'){
        x=x*10+ch-'0';
        ch=getchar();
    }
    return x*f;
}
inline void print(__int128 x){
```

```
    if(x<0){
        putchar('-');
        x=-x;
    }
    if(x>9)
        print(x/10);
    putchar(x%10+'0');
}
int main(){
    __int128 a = read();
    __int128 b = read();
    print(a + b);
    cout<<endl;
    return 0;
}
```

7.2.2 简单高精度

```
static const int LEN = 1004;
int a[LEN], b[LEN], c[LEN], d[LEN];

void clear(int a[]) {
  for (int i = 0; i < LEN; ++i) a[i] = 0;
}

void read(int a[]) {
  static char s[LEN + 1];
  scanf("%s", s);
  clear(a);
  int len = strlen(s);
  for (int i = 0; i < len; ++i) a[len - i - 1] = s[i] - '0';
}

void print(int a[]) {
  int i;
  for (i = LEN - 1; i >= 1; --i)
    if (a[i] != 0) break;
  for (; i >= 0; --i) putchar(a[i] + '0');
  putchar('\n');
}

void add(int a[], int b[], int c[]) {
  clear(c);
  for (int i = 0; i < LEN - 1; ++i) {
    c[i] += a[i] + b[i];
    if (c[i] >= 10) {
      c[i + 1] += 1;
      c[i] -= 10;
    }
  }
}

void sub(int a[], int b[], int c[]) {
    clear(c);
```

```
    for (int i = 0; i < LEN - 1; ++i) {
        c[i] += a[i] - b[i];
        if (c[i] < 0) {
            c[i + 1] -= 1;
            c[i] += 10;
        }
    }
}

void mul(int a[], int b[], int c[]) {
  clear(c);
    for (int i = 0; i < LEN - 1; ++i) {
    for (int j = 0; j <= i; ++j) c[i] += a[j] * b[i - j];
    if (c[i] >= 10) {
        c[i + 1] += c[i] / 10;
        c[i] %= 10;
    }
    }
}

inline bool greater_eq(int a[], int b[], int last_dg, int len) {
    if (a[last_dg + len] != 0) return true;
    for (int i = len - 1; i >= 0; --i) {
        if (a[last_dg + i] > b[i]) return true;
        if (a[last_dg + i] < b[i]) return false;
    }
    return true;
}

void div(int a[], int b[], int c[], int d[]) {
    clear(c);clear(d);
    int la, lb;
    for (la = LEN - 1; la > 0; --la)
        if (a[la - 1] != 0) break;
    for (lb = LEN - 1; lb > 0; --lb)
        if (b[lb - 1] != 0) break;
    if (lb == 0) {
        puts("> <");
        return;
    }
    for (int i = 0; i < la; ++i) d[i] = a[i];
    for (int i = la - lb; i >= 0; --i) {
        while (greater_eq(d, b, i, lb)) {
            for (int j = 0; j < lb; ++j) {
                d[i + j] -= b[j];
                if (d[i + j] < 0) {
                    d[i + j + 1] -= 1;
                    d[i + j] += 10;
                }
            }
            c[i] += 1;
        }
    }
```

```
}

int main() {
  read(a);
  char op[4],scanf("%s", op);
  read(b);
  switch (op[0]) {
    case '+':
      add(a, b, c);print(c);
      break;
    case '-':
      sub(a, b, c);print(c);
      break;
    case '*':
      mul(a, b, c);print(c);
      break;
    case '/':
      div(a, b, c, d);
      print(c);print(d);
      break;
  }
  return 0;
}
```

7.2.3 压位高精度

```
//高精度，支持乘法和加法但只支持正数
struct BigInt{
    const static int mod = 10000;
    const static int DLEN = 4;
    //根据题目要求可对 a 数组大小进行修改
    int a[6000],len;
    BigInt(){
        memset(a,0,sizeof(a));
        len=1;
    }
    BigInt(int v){
        memset(a,0,sizeof(a));
        len=0;
        do{
            a[len++]=v%mod;
            v/=mod;
        }while(v);
    }
    BigInt(const char s[]){
        memset(a,0,sizeof(a));
        int L=strlen(s);
        len=L/DLEN;
        if(L%DLEN)  len++;
        int index = 0;
        for(int i=L-1;i>=0;i-=DLEN){
            int t=0;
            int k=i-DLEN+1;
            if(k<0) k=0;
```

```
            for(int j=k;j<=i;++j)
                t=t*10+s[j]-'0';
            a[index++]=t;
        }
    }
    BigInt operator +(const BigInt &b)const {
        BigInt res;
        res.len=max(len,b.len);
        for(int i=0;i<=res.len;++i)
            res.a[i]=0;
        for(int i=0;i<res.len;++i){
            res.a[i]+=((i<len)?a[i]:0)+((i<b.len)?b.a[i]:0);
            res.a[i+1]+=res.a[i]/mod;
            res.a[i]%=mod;
        }
        if(res.a[res.len]>0) res.len++;
        return res;
    }
    BigInt operator *(const BigInt &b)const {
        BigInt res;
        for(int i=0;i<len;++i){
            int up= 0;
            for(int j=0;j<b.len;++j){
                int temp=a[i]*b.a[j]+res.a[i+j]+up;
                res.a[i+j]=temp%mod;
                up=temp/mod;
            }
            if(up!=0)
                res.a[i+b.len]=up;
        }
        res.len=len+b.len;
        while(res.a[res.len-1]==0 && res.len>1) res.len--;
        return res;
    }
    void output(){
        printf("%d",a[len-1]);
        for(int i=len-2;i>=0;--i)
            printf("%04d",a[i]);
        printf("\n");
    }
};
int  main()
{
    //字符串读入
    char a[2005],b[2005];
    cin>>a>>b;
    BigInt A,B;
    A=BigInt(a),B=BigInt(b);
    //可以直接用 cout 输出 char 数组内容
    cout<<a<<" "<<b<<endl;
    (A+B).output();//加法
    (A*B).output();//乘法
    return 0;
```

```
}
```

7.3 离散化

```
/*
对于包含重复元素，并且相同元素离散化后也要相同
例：
a:7 4 7 3 4 10
b:3 2 3 1 2 4
m=4
*/
const int maxn=1e5+10;
int a[maxn], t[maxn], b[maxn];
int main(){
        int n;scanf("%d",&n);
        for(int i=1; i<=n; i++)
                scanf("%d",&a[i]),t[i]=a[i];
        sort(t+1,t+n+1);
        int m=unique(t+1,t+1+n)-t-1;//求出的 m 为不重复的元素的个数
        for(int i=1; i<=n; i++)
                b[i]=lower_bound(t+1,t+1+m,a[i])-t;
        return 0;
}

/*
较上一种复杂度低
对于：
1. 包含重复元素，并且相同元素离散化后不相同
2. 不包含重复元素，并且不同元素离散化后不同
例：
a:7 4 7 3 4 10
b:4 2 5 1 3 6
*/
struct A
{
    int x, idx;
    bool operator < (const A &rhs) const
    {
        return x < rhs.x;
    }
};
A a[N];
int b[N];
int main()
{
        int n;
        scanf("%d",&n);
        for(int i = 1; i <= n; ++i){
                scanf("%d", &a[i].x);
                a[i].idx = i;
        }
        sort(a + 1, a + n + 1);
        for(int i = 1; i <= n; ++i)
            b[a[i].idx] = i;
```

```
51          return 0;
52  }
```

7.4 常用公式

- $\sum_{i=1}^{n} i = \frac{n*(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n*(n+1)*(2n+1)}{6}$
- $\sum_{i=1}^{n} i^3 = \left[\frac{n*(n+1)}{2}\right]^2$

8 经典例题

8.1 某天是星期几

```
/*
求某天是星期几
*/
char *name[] = { "monday", "tuesday", "wednesday",
"thursday", "friday", "saturday", "sunday" };
int main(void)
{
    int d, m, y, a;
    printf("Day: "); scanf("%d",&d);
    printf("Month: "); scanf("%d",&m);
    printf("Year: "); scanf("%d",&y);
    // 1 月 2 月当作前一年的 13,14 月
    if (m == 1 || m == 2) { m += 12; y--; }
    // 判断是否在 1752 年 9 月 3 日之前
    if ((y < 1752) || (y == 1752 && m < 9) ||(y == 1752 && m == 9 && d <
    ↪  3))
        a = (d + 2*m + 3*(m+1)/5 + y + y/4 +5) % 7;
    else
        a = (d + 2*m + 3*(m+1)/5 + y + y/4 - y/100 + y/400)%7;
    printf("it's a %s\n", name[a]);
return 0;
```

8.2 动态区间最大和

在数列 p_n 中, 找出一个字段 $[l,r](r-l+1<=k)$, 最大化 $\sum_{i=l}^{r} p_i$

```
/*
example:
input:
5 2
1 2 3 4 5
ouput:
9
solution:
单调队列维护前缀和
*/
const int N=5e5+5;
int a[N],dequeue[N],s[N];
void ac(){
        int n,k;scanf("%d%d",&n,&k);
        int ans=-1;
        for(int i=1;i<=n;++i)
                read(a[i]),s[i]=s[i-1]+a[i],ans=max(ans,a[i]);
        int tail=0,head=1;
        dequeue[++tail]=0;//便于处理前缀和
        for(int i=1;i<=n;++i){
                while(head<=tail && dequeue[head]<i-k)
                        head++;
        /*ans 和队尾元素出队需要根据题意决定，因为之前已经算过
        只有一个元素的情况，所以不能让所有元素出队 */
                ans=max(ans,s[i]-s[dequeue[head]]);
```

```
                while(head<=tail && s[dequeue[tail]]>=s[i])
                        tail--;
                dequeue[++tail]=i;
        }
        printf("%d",ans);
}
```

8.3 二分查找

```
/*
几种二分方法整理
元素可以重复
*/

//lower_bound(num, num+size, x)-num: 大于等于 x 的第一个数的下标
//upper_bound(num, num+size, x)-num: 大于 x 的第一个数的下标

//1. 求等于 x 的最小的 index, 不存在返回-1
int binary (int *num, int start, int end, int x) {
        int l = start, r = end, ans=-1;
        while(l <= r) {
                int mid = (l+r) >> 1;
                if(num[mid] == x) {
                        ans = mid;
                        r = mid - 1;
                }
                else if(num[mid] > x)
                        r = mid - 1;
                else
                        l = mid + 1;
        }
        return ans;
}

//2. 求等于 x 的最大的 index, 不存在返回-1
int binary (int *num, int start, int end, int x) {
        int l = start, r = end, ans=-1;
        while(l <= r) {
                int mid = (l+r) >> 1;
                if(num[mid] == x) {
                        ans = mid;
                        l = mid + 1;
                }
                else if(num[mid] > x)
                        r = mid - 1;
                else
                        l = mid + 1;
        }
        return ans;
}

//3. 求小于 x 的最大的 index
int binary (int *num, int start, int end, int x) {
        int l = start, r = end;
```

```
        while(l <= r) {
                int mid = (l+r) >> 1;
                if(num[mid] >= x)
                        r = mid - 1;
                else
                        l = mid + 1;
        }
        return r;
}

//4. 求大于 x 的最小的 index
int binary (int *num, int start, int end, int x) {
        int l = start, r = end;
        while(l <= r) {
                int mid = (l+r) >> 1;
                if(num[mid] <= x)
                        l = mid + 1;
                else
                        r = mid - 1;
        }
        return l;
}

//5. 求大于等于 x 的最小的 index
int binary (int *num, int start, int end, int x) {
        int l = start, r = end;
        while(l <= r) {
                int mid = (l+r) >> 1;
                if(num[mid] >= x)
                        r = mid - 1;
                else
                        l = mid + 1;
        }
        return l;
}

//6. 求小于等于 x 的最大的 index
int binary (int *num, int start, int end, int x) {
        int l = start, r = end;
        while(l <= r) {
                int mid = (l+r) >> 1;
                if(num[mid] <= x)
                        l = mid + 1;
                else
                        r = mid - 1;
        }
        return r;
}
```

9 注意事项

- 注意范围!!! 爆 int 多少回了? 有时候 ans 定义为 ll 也是不够的, 注意改为 ll 后 scanf 和 printf, 不行就 signed main,#define int ll
- 注意初始点的设置, 初始点是否有效是否得到正确的更新
- 注意边界条件如 $<$ 和 <=, 注意特判如 $n = 1$