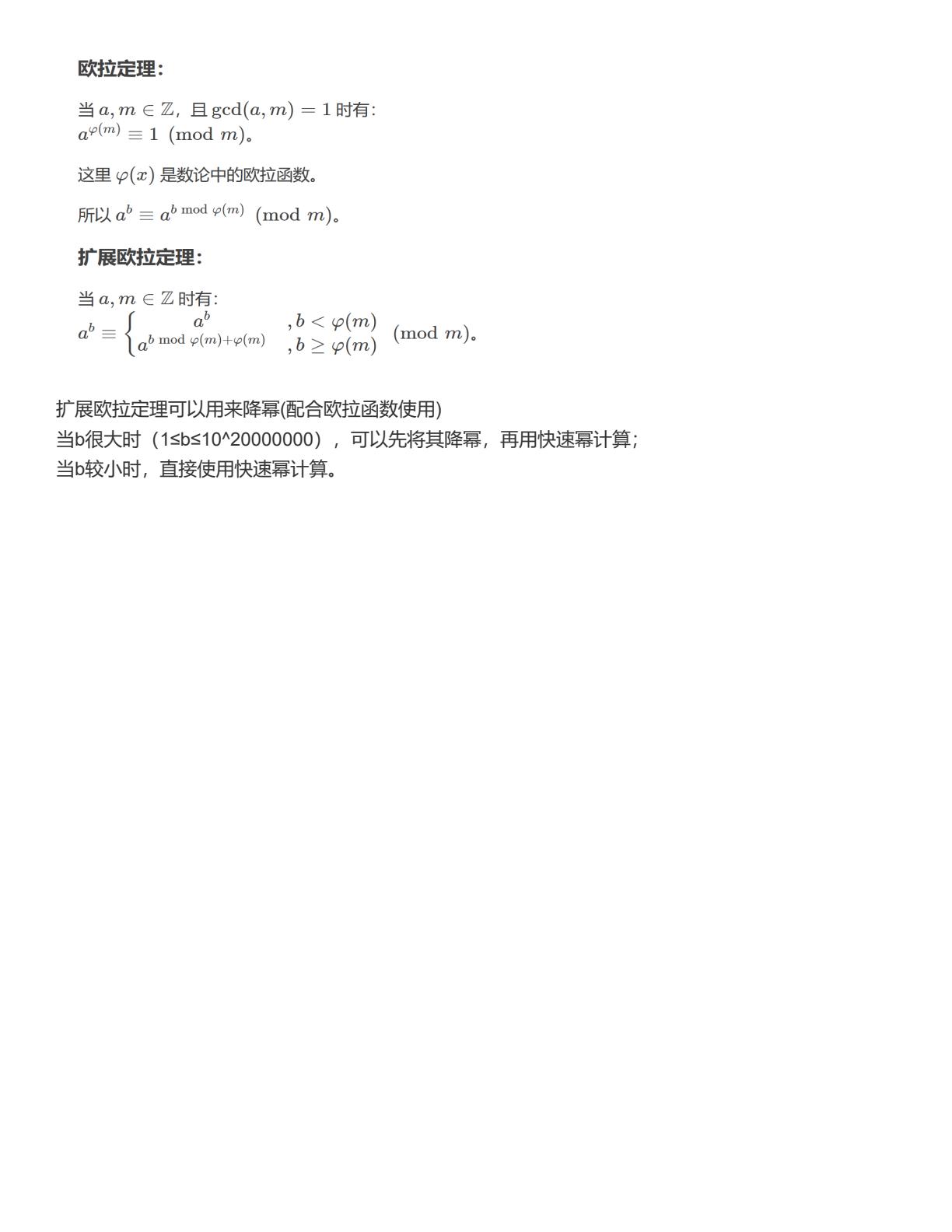
## 欧拉定理&费马小定理

****

****

## 扩展欧几里得

// ax+by = gcd(a,b)

ll exgcd(ll a, ll b, ll &x, ll &y)

{

if(b == 0)

{

x = 1, y = 0;

return a;

}

ll x1,y1,d;

d = exgcd(b, a%b, x1, y1);

x = y1, y = x1 - a/b\*y1;

return d;

}

x = (x%mod + mod) % mod; // 保证0 < x < mod

// 求 a 在模 mod 下的逆元, ax = 1(%mod)

exgcd(a,mod,x,y);

x = (x%mod + mod) % mod;

## 并查集

vector<int> fa,sz;

fa.resize(n+1),sz.resize(n+1,1);

iota(fa.begin(),fa.end(),0);

int find(int x) // 找根节点

{

if(fa[x]==x) return x;

return fa[x]=find(fa[x]);

}

void unionset(int x, int y) // 合并并查集

{

x=find(x),y=find(y);

if(x==y) return;

if(sz[x]>sz[y]) swap(x,y);

fa[x]=y;

sz[y]+=sz[x];

}

## 二分

int find(int q)

{

int l=0,r=n+1,mid;

while(l+1<r)

{

mid=l+r>>1;

if(a[mid]<=q)

l=mid;

else

r=mid;

}

return l;

}

## 树状数组

int n,m;

vector<ll> a;

ll lowbit(ll x)

{

return x&-x;

}

// 单点修改

void change(ll x, ll k)

{

while(x<=n)

{

a[x]+=k;

x+=lowbit(x);

}

}

// 查询区间[1,x]前缀和

ll query(ll x)

{

ll res=0;

while(x>0)

{

res+=a[x];

x-=lowbit(x);

}

return res;

}

int main()

{

cin>>n>>m;

a.resize(n+1);

for(i=1;i<=n;i++)

{

cin>>t;

change(i,t);

}

while(m--)

{

int opt,x,y;

cin>>opt>>x>>y;

if(opt==1)

change(x,y);

else

cout<<query(y)-query(x-1)<<endl;

}

}

## 线段树

// 线段树 version1

#define lc p<<1 // 左子节点

#define rc p<<1|1 // 右子节点

const ll N = 1e5+1;

vector<ll> w;

struct node

{

ll l,r,sum,add; // 区间，区间和，懒标记

}tr[4\*N];

void pushup(ll p)

{

tr[p].sum = tr[lc].sum + tr[rc].sum;

}

void pushdown(ll p)

{

if(tr[p].add)

{

tr[lc].sum += tr[p].add\*(tr[lc].r-tr[lc].l+1);

tr[rc].sum += tr[p].add\*(tr[rc].r-tr[rc].l+1);

tr[lc].add += tr[p].add;

tr[rc].add += tr[p].add;

tr[p].add = 0;

}

}

void build(ll p, ll x, ll y)

{

tr[p] = {x,y,w[x],0};

if(x==y) return;

ll mid = (tr[p].l+tr[p].r)/2;

build(lc,x,mid);

build(rc,mid+1,y);

pushup(p);

}

void update(ll p, ll x, ll y, ll k)

{

if(x<=tr[p].l && tr[p].r<=y)

{

tr[p].add += k;

tr[p].sum += k\*(tr[p].r-tr[p].l+1);

return;

}

ll mid = (tr[p].l+tr[p].r)/2;

pushdown(p);

if(x<=mid) update(lc,x,y,k);

if(y>mid) update(rc,x,y,k);

pushup(p);

}

ll query(ll p, ll x, ll y)

{

if(x<=tr[p].l && tr[p].r<=y)

return tr[p].sum;

ll res = 0;

pushdown(p);

ll mid = (tr[p].l+tr[p].r)/2;

if(x<=mid) res += query(lc,x,y);

if(y>mid) res += query(rc,x,y);

return res;

}

## 字符串哈希

// **字符串哈希+最长回文串**

// 自然溢出

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef unsigned long long ull;

#define endl '\n'

int n;

const ull P=131;

vector<ull> h1,h2,p;

ull geth(int l, int r, int f) // 1<=l<=r<=n

{

if(f==1) return (h1[r]-h1[l-1]\*p[r-l+1]);

else return (h2[r]-h2[l-1]\*p[r-l+1]);

}

bool OK(int m)

{

int i;

for(i=1;i+m-1<=n;i++)

if(geth(i,i+m-1,1)==geth(n-i-m+2,n-i+1,2))

return 1;

return 0;

}

int main()

{

ios::sync\_with\_stdio(0);

cin.tie(0),cout.tie(0);

string str;

getline(cin,str);

n=str.length();

str=' '+str;

int i,j;

h1.resize(n+1),h2.resize(n+1),p.resize(n+1);

p[0]=1;

for(i=1;i<=n;i++)

{

h1[i]=h1[i-1]\*P+ull(str[i]); // 下标范围 [1,n]

h2[i]=h2[i-1]\*P+ull(str[n-i+1]);

p[i]=p[i-1]\*P;

}

int l=0,r=n+1,mid;

while(l+1<r)

{

mid=l+r>>1;

if(OK(mid+1)) l=mid+1;

else if(OK(mid)) l=mid;

else r=mid;

}

cout<<l;

}

/\*---------------------------------------------------------------------------\*/

// **双哈希**

// 题目链接 https://oj.swjtu.edu.cn/p/SWJTU0061

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef pair<ll,ll> pll;

#define endl '\n'

ll mod[]={ll(1e9)+7,ll(1e9)+9}; // 类型转换

const ll P=131;

const ll N=3e6+10;

vector<ll> h[2],p[2];

pll geth(int l, int r)

{

ll res[2];

for(int i=0;i<2;i++)

{

res[i]=((h[i][r]-h[i][l-1]\*p[i][r-l+1])%mod[i]+mod[i])%mod[i];

// res[i]=(h[i][r]-h[i][l-1]\*p[i][r-l+1]%mod[i]+mod[i])%mod[i]; // another version

}

return make\_pair(res[0],res[1]);

}

int main()

{

ios::sync\_with\_stdio(0);

cin.tie(0),cout.tie(0);

int n,m;

string a,b;

cin>>n>>a>>m;

a=' '+a;

int i,j,k;

for(i=0;i<2;i++)

{

h[i].resize(N);

p[i].resize(N);

}

p[0][0]=p[1][0]=1;

for(i=1;i<=N;i++)

for(j=0;j<2;j++)

{

p[j][i]=(p[j][i-1]\*P)%mod[j];

}

pll ans={0,0}; // [1,n]

auto &[x,y]=ans;

for(i=1;i<=n;i++)

{

x=(x\*P+a[i])%mod[0];

y=(y\*P+a[i])%mod[1];

}

while(m--)

{

int f=0;

string str;

cin>>str;

b=' '+str+str;

for(i=1;i<b.size();i++)

{

for(j=0;j<2;j++)

{

h[j][i]=(h[j][i-1]\*P+b[i])%mod[j];

}

}

for(i=1;i+n-1<b.size();i++)

if(geth(i,i+n-1)==ans)

{

f=1;

break;

}

if(f)

{

cout<<"YES\n";

continue;

}

reverse(str.begin(),str.end());

b=' '+str+str;

for(i=1;i<b.size();i++)

{

for(j=0;j<2;j++)

{

h[j][i]=(h[j][i-1]\*P+b[i])%mod[j];

}

}

for(i=1;i+n-1<b.size();i++)

if(geth(i,i+n-1)==ans)

{

f=1;

break;

}

cout<<(f ? "YES" : "NO")<<endl;

}

}

/\*---------------------------------------------------------------------------\*/

// **大模数哈希**

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef \_\_int128\_t lll;

#define endl '\n'

const lll P=131;

const lll mod=lll(1e18)+9; // 类型转换

const lll N=2e5+10;

vector<lll> p,h;

lll geth(int l, int r) // 1<=l<=r<str.length()

{

return ((h[r]-h[l-1]\*p[r-l+1])%mod+mod)%mod;

}

int main()

{

ios::sync\_with\_stdio(0);

cin.tie(0),cout.tie(0);

int n;

cin>>n;

p.resize(N), h.resize(N);

int i,j;

p[0]=1;

for(i=1;i<N;i++)

p[i]=p[i-1]\*P%mod;

string str;

cin>>str;

lll t=0;

for(int i=0;i<str.length();i++)

h[i]=(h[i-1]\*P+str[i])%mod;

}

## 最近公共祖先

// 最近公共祖先

// version1

// **树链剖分**

const int N=5e5+10;

int fa[N],dep[N],sz[N],top[N],son[N];

vector<vector<int>> a;

void dfs1(int u, int fat)

{

fa[u]=fat,dep[u]=dep[fat]+1,sz[u]=1;

for(auto v:a[u])

{

if(v==fat)

continue;

dfs1(v,u);

sz[u]+=sz[v];

if(sz[son[u]]<sz[v])

son[u]=v;

}

}

void dfs2(int u, int topp)

{

top[u]=topp;

if(!son[u])

return;

dfs2(son[u],topp); //重儿子

for(auto v:a[u])

{

if(v==son[u] || v==fa[u])

continue;

dfs2(v,v);

}

}

int lca(int u, int v)

{

while(top[u]!=top[v])

{

if(dep[top[u]]<dep[top[v]])

swap(u,v);

u=fa[top[u]];

}

return (dep[u]<dep[v] ? u:v);

}

int main()

{

ios::sync\_with\_stdio(0);

cin.tie(0),cout.tie(0);

int n,m,s;

cin>>n>>m>>s;

a.resize(n+1);

for(int i=0;i<n-1;i++)

{

int x,y;

cin>>x>>y;

a[x].push\_back(y),a[y].push\_back(x);

}

dfs1(s,0);

dfs2(s,s);

for(int i=0;i<m;i++)

{

int a,b;

cin>>a>>b;

if(a==b)

cout<<a<<'\n';

else

cout<<lca(a,b)<<'\n';

}

}

// version 2

// **倍增**

vector<int> dep;

vector<vector<int>> a,fa;

void dfs(int u, int father)

{

int i;

fa[u][0]=father, dep[u]=dep[father]+1;

for(i=1;i<20;i++)

fa[u][i]=fa[fa[u][i-1]][i-1];

for(auto v:a[u])

if(v!=father)

dfs(v,u);

}

int lca(int u, int v)

{

int i;

if(dep[u]<dep[v])

swap(u,v);

for(i=19;i>=0;i--)

if(dep[fa[u][i]]>=dep[v])

u=fa[u][i];

if(u==v)

return u;

for(i=19;i>=0;i--)

if(fa[u][i]!=fa[v][i])

u=fa[u][i],v=fa[v][i];

return fa[u][0];

}

int main()

{

fa.resize(n+1),dep.resize(n+1);

a.resize(n+1);

for(i=0;i<=n;i++)

fa[i].resize(20); // 这里大小取决于n，保证2^(fa[i].size-1)>n

}

## 最小生成树

// kruskal

vector<int> fa(n+1),sz(n+1,1);

iota(fa.begin(),fa.end(),0);

auto find=[&](auto find, int x)

{

if(x==fa[x]) return x;

return (fa[x]=find(find,fa[x]));

};

for(auto [u,v,w]:a)

{

u=find(find,u),v=find(find,v);

if(sz[u]>sz[v]) swap(u,v);

if(u!=v)

{

fa[u]=v;

sz[v]+=sz[u];

a[u].push\_back({v,w});

a[v].push\_back({u,w});

}

}

## KMP

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

#define endl '\n'

int main()

{

ios::sync\_with\_stdio(0);

cin.tie(0),cout.tie(0);

string a,b; // a是目标串, b是模式串

cin>>a>>b;

a=' '+a;

b=' '+b;

int n=b.length()-1;

vector<int> ne(n+1); // ne数组从ne[1]开始用,ne[0]没用

int i,j,t;

for(i=2,j=0;i<=n;i++)

{

while(j && b[i]!=b[j+1])

j=ne[j];

if(b[i]==b[j+1])

j++;

ne[i]=j;

}

for(i=1,j=0;i<a.length();i++)

{

while(j && a[i]!=b[j+1])

j=ne[j];

if(a[i]==b[j+1])

j++;

if(j==n) // 匹配成功

cout<<i-n+1<<endl;

}

for(i=1;i<=n;i++)

cout<<ne[i]<<' ';

}

## 快速幂

ll quick\_power(ll base, ll power)

{

ll res = 1;

while(power)

{

if(power&1)

res = (res\*base) % mod;

power >>= 1;

base = (base\*base) % mod;

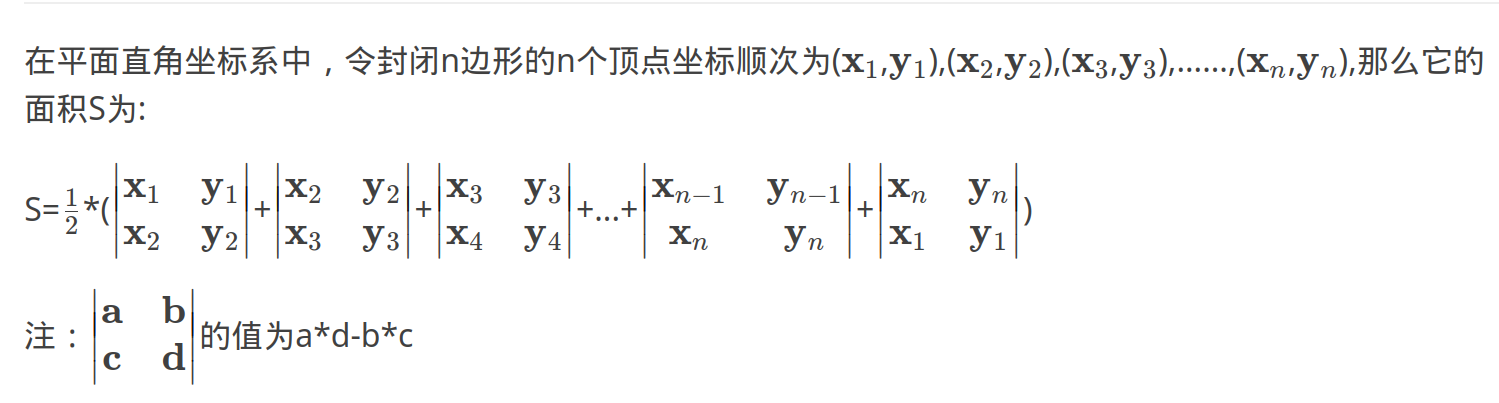
}

return res;

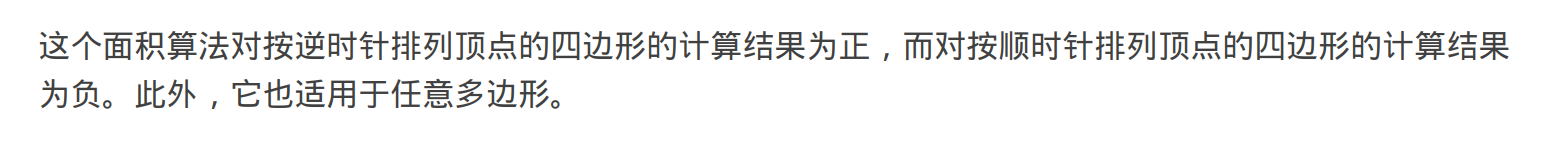
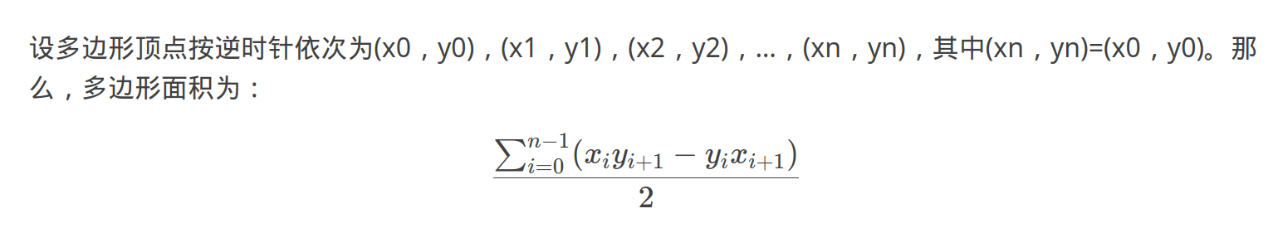
}

## 计算几何

### 多边形面积



也可表示为



### 皮克定理（单纯求面积可以用上面的公式）

给定顶点均是整点（或正方形格点）的简单多边形，皮克定理说明了其面积S和内部格点数目n、边上格点数目s的关系：

$S=a+ \frac {b}{2}-1$ 其中a表示多边形内部的点数，b表示多边形恰好落在边界上的点数，s表示多边形的面积。

**逆用皮克定理** 逆用pick定理时一定不能用a=（2s-b+2）/2，应该用a=s-b/2+1，不然会有精度问题 面积可以用之前的多边形面积公式求出

**皮克定理搭配（扩展）欧几里得**

（扩展）欧几里得可以求多边形的每条边上的点的个数 （扩展）欧几里得 可以放到计算几何中算线段AB上有多少整点, 毕竟x,y取整数 在两整点 A(x1,y1) 和 B(x2,y2) 的线段（含两端端点）中经过整点的个数为 gcd(∣x1−x2∣,∣y1−y2∣)+1

*[//](#_bookmark0) ax+by = gcd(a,b)*

[ll](#_bookmark1) exgcd(ll a, ll b, ll &x, ll &y)

[{](#_bookmark2)

if(b == 0)

{

x = 1, y = 0;

return a;

}

ll x1,y1,d;

d = exgcd(b, a%b, x1, y1); x = y1, y = x1 - a/b\*y1; return d;

[}](#_bookmark12)

[x](#_bookmark13) = (x%mod + mod) % mod; *// 保 证 0 < x < mod*

*[//](#_bookmark14) 求 a 在模 mod 下的逆元, ax = 1(%mod)*

[exgcd(a,mod,x,y);](#_bookmark15)

[x](#_bookmark16) = (x%mod + mod) % mod;

### 判断点与多边形的位置关系

这里以三角形为例。给定一个三角形ABC， 判断点P与其的位置关系。

cpp

int f1=sgn(cross(PA,AB)), f2=sgn(cross(PB,BC)), f3=sgn(cross(PC,CA)); if(f1f2 && f2f3) // 在三角形内部

cout<<1;

else if(f1\*f2<0 || f1\*f3<0 || f2\*f3<0) // 在三角形外部cout<<2;

else // 在三角形边上cout<<3;

1.所有结果的都大于0或都小于0，则点在三角形内部

2.如果既有大于0又有小于0的在外部

3.然后判断在边界上的，这种情况是有有一个叉积为0

不能先判断3再判断2， 因为有可能点P在三角形某条边的延长线上

### 凸包

[int](#_bookmark17) main()

[{](#_bookmark18)

ios::sync\_with\_stdio(0); cin.tie(0),cout.tie(0); int n;

cin>>n;

vector<Point> a(n+1), s(4\*n+1); *// s(模拟栈,大小开大点)*

int i,j,k,top=0; for(i=1;i<=n;i++)

cin>>a[i].x>>a[i].y; sort(a.begin()+1,a.end());

for(i=1;i<=n;i++) *// 下 凸 包*

{

while(top>1 && Line(s[top-1],s[top]).relation(a[i])!=1) *// 不在向量左边的就pop*

top--; s[++top]=a[i];

}

int t=top;

for(i=n-1;i>=1;i--) *// 上凸包*

{

while(top>t && Line(s[top-1],s[top]).relation(a[i])!=1) *// 不在向量左边的就pop*

top--; s[++top]=a[i];

}

double res=0;

for(i=1;i<top;i++) *// 求凸包周长, s[1]==s[top], 即第一个点和最后一个点相同*

res+=s[i].distance(s[i+1]); cout<<fixed<<setprecision(2)<<res<<endl;

[}](#_bookmark45)

### 计算几何(kuangbin 模板)

const double inf=1e20;

const double eps=1e-8;

const double pi=acos(-1.0);

const ll maxp=1010;

//判断正负

int sgn(double x) {

if (fabs(x)<eps) return 0;

if (x<0) return -1;

else return 1;

}

struct Point {

double x,y;

Point() {}

Point(double \_x, double \_y) {

x=\_x;

y=\_y;

}

void input() {

scanf("%lf%lf",&x,&y);

}

void output() {

printf("%.2f%.2f\n",x,y);

}

bool operator == (Point b)const {

return sgn(x-b.x)==0 && sgn(y-b.y)==0;

}

bool operator < (Point b)const {

return sgn(x-b.x)==0?sgn(y-b.y)<0:x<b.x;

}

Point operator -(const Point &b)const {

return Point(x-b.x,y-b.y);

}

// 向量叉积

double operator ^(const Point &b)const {

return x\*b.y-y\*b.x;

}

// 向量点积

double operator \*(const Point &b)const {

return x\*b.x+y\*b.y;

}

//返回长度

double len() {

return hypot(x,y);

}

//返回长度平方

double len2() {

return x\*x+y\*y;

}

//返回两点间距

double distance(Point p) {

return hypot(x-p.x,y-p.y);

}

Point operator +(const Point &b)const {

return Point(x+b.x,y+b.y);

}

Point operator \*(const double &k)const {

return Point(x\*k,y\*k);

}

Point operator /(const double &k)const {

return Point(x/k,y/k);

}

//pa和pb的夹角

double rad(Point a,Point b) {

Point p=\*this;

return fabs(atan2(fabs((a-p)^(b-p)),(a-p)\*(b-p)));

}

//化为长度为r的向量

Point trunc(double r) {

double l=len();

if (!sgn(l)) return \*this;

r/=l;

return Point(x\*r,y\*r);

}

//逆时针旋转 90 度

Point rotleft() {

return Point(-y,x);

}

//顺时针旋转 90 度

Point rotright() {

return Point(y,-x);

}

//绕着 p 点逆时针旋转 angle

Point rotate(Point p,double angle) {

Point v=(\*this)-p;

double c=cos(angle),s=sin(angle);

return Point(p.x+v.x\*c-v.y\*s,p.y+v.x\*s+v.y\*c);

}

};

struct Line {

Point s,e;

Line() {}

Line(Point \_s,Point \_e) {

s=\_s;

e=\_e;

}

bool operator ==(Line v) {

return (s==v.s) && (e==v.e);

}

//根据一个点和倾斜角 angle 确定直线,0<=angle<π

Line(Point p, double angle) {

s=p;

if (sgn(angle-pi/2)==0) {

e=(s+Point(0,1));

}

else {

e=(s+Point(1,tan(angle)));

}

}

//ax+by+c=0

Line(double a,double b,double c) {

if(sgn(a)==0) {

s=Point(0,-c/b);

e=Point(1,-c/b);

}

else if(sgn(b)==0) {

s=Point(-c/a,0);

e=Point(-c/a,1);

}

else {

s=Point(0,-c/b);

e=Point(1,(-c-a)/b);

}

}

void input() {

s.input();

e.input();

}

void adjust() {

if(e<s) swap(s,e);

}

//求线段长度

double length() {

return s.distance(e);

}

//返回直线倾斜角 0<=angle<π

double angle() {

double k=atan2(e.y-s.y,e.x-s.x);

if(sgn(k)<0) k+=pi;

if(sgn(k-pi)==0) k-= pi;

return k;

}

//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

//要保证两直线不平行或重合

Point crosspoint(Line v) {

double a1 = (v.e-v.s)^(s-v.s);

double a2 = (v.e-v.s)^(e-v.s);

return Point((s.x\*a2-e.x\*a1)/(a2-a1),(s.y\*a2-e.y\*a1)/(a2-a1

));

}

//点到直线的距离

double dispointtoline(Point p) {

return fabs((p-s)^(e-s))/length();

}

//点到线段的距离

double dispointtoseg(Point p) {

if(sgn((p-s)\*(e-s))<0 || sgn((p-e)\*(s-e))<0)

return min(p.distance(s),p.distance(e));

return dispointtoline(p);

}

//返回线段到线段的距离

//前提是两线段不相交，相交距离就是 0 了

double dissegtoseg(Line v) {

return min(min(dispointtoseg(v.s),dispointtoseg(v.e)),min(v

.dispointtoseg(s),v.dispointtoseg(e)));

}

//返回点 p 在直线上的投影

Point lineprog(Point p) {

return s + ( ((e-s)\*((e-s)\*(p-s)))/((e-s).len2()) );

}

//返回点 p 关于直线的对称点

Point symmetrypoint(Point p) {

Point q = lineprog(p);

return Point(2\*q.x-p.x,2\*q.y-p.y);

}

};

//AB X AC 向量叉乘

double cross(Point A,Point B,Point C) {

return (B-A)^(C-A);

}

// 向量叉乘

double cross(Line A, Line B) {

Point C(A.e-A.s), D(B.e-B.s);

return C^D;

}

//AB\*AC 向量点乘

double dot(Point A,Point B,Point C) {

return (B-A)\*(C-A);

}

// 向量点乘

double dot(Line A, Line B) {

Point C(A.e-A.s), D(B.e-B.s);

return C\*D;

}

struct circle {

Point p; //圆心

double r; //半径

circle() {}

circle(Point \_p,double \_r) {

p=\_p;

r=\_r;

}

circle(double x,double y,double \_r) {

p=Point(x,y);

r=\_r;

}

//三角形的外接圆

//需要 Point 的 + / rotate() 以及 Line 的 crosspoint()

//利用两条边的中垂线得到圆心

circle(Point a,Point b,Point c) {

Line u=Line((a+b)/2,((a+b)/2)+((b-a).rotleft()));

Line v=Line((b+c)/2,((b+c)/2)+((c-b).rotleft()));

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

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);

}

void input() {

p.input();

scanf("%lf",&r);

}

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);

if (sgn(dst-r)<0) return 2;

else if (sgn(dst-r)==0) return 1;

return 0;

}

//直线和圆的关系

//比较的是圆心到直线的距离和半径的关系

int relationline(Line v) {

double dst=v.dispointtoline(p);

if (sgn(dst-r)<0) return 2;

else if (sgn(dst-r)==0) return 1;

return 0;

}

//两圆的关系

//5 相离

//4 外切

//3 相交

//2 内切

//1 内含

//需要 Point 的 distance

int relationcircle(circle v) {

double d=p.distance(v.p);

if (sgn(d-r-v.r)>0) return 5;

if (sgn(d-r-v.r)==0) return 4;

double l=fabs(r-v.r);

if (sgn(d-r-v.r)<0 && sgn(d-l)>0) return 3;

if (sgn(d-l)==0) return 2;

if (sgn(d-l)<0) return 1;

}

//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

//需要 relationcircle

int pointcrosscircle(circle v,Point &p1,Point &p2) {

int rel=relationcircle(v);

if (rel==1 || rel==5)return 0;

double d=p.distance(v.p);

double l=(d\*d+r\*r-v.r\*v.r)/(2\*d);

double h=sqrt(r\*r-l\*l);

Point tmp=p+(v.p-p).trunc(l);

p1=tmp+((v.p-p).rotleft().trunc(h));

p2=tmp+((v.p-p).rotright().trunc(h));

if(rel==2 || rel==4)

return 1;

return 2;

}

//求直线和圆的交点，返回交点个数

int pointcrossline(Line v,Point &p1,Point &p2) {

if (!(\*this).relationline(v)) return 0;

Point a=v.lineprog(p);

double d=v.dispointtoline(p);

d=sqrt(r\*r-d\*d);

if (sgn(d)==0) {

p1=a;

p2=a;

return 1;

}

p1=a+(v.e-v.s).trunc(d);

p2=a-(v.e-v.s).trunc(d);

return 2;

}

//得到过 a,b 两点，半径为 r1 的两个圆

int gercircle(Point a,Point b,double r1,circle &c1,circle &c2) {

circle x(a,r1),y(b,r1);

int t=x.pointcrosscircle(y,c1.p,c2.p);

if (!t) return 0;

c1.r=c2.r=r;

return t;

}

//得到与直线 u 相切，过点 q, 半径为 r1 的圆

int getcircle(Line u,Point q,double r1,circle &c1,circle &c2) {

double dis = u.dispointtoline(q);

if (sgn(dis-r1\*2)>0) return 0;

if (sgn(dis)==0) {

c1.p=q+((u.e-u.s).rotleft().trunc(r1));

c2.p=q+((u.e-u.s).rotright().trunc(r1));

c1.r=c2.r=r1;

return 2;

}

Line u1=Line((u.s+(u.e-u.s).rotleft().trunc(r1)),(u.e+(u.e-u.s).rotleft().trunc(r1)));

Line u2=Line((u.s + (u.e-u.s).rotright().trunc(r1)),(u.e+(u.e-u.s).rotright().trunc(r1)));

circle cc=circle(q,r1);

Point p1,p2;

if (!cc.pointcrossline(u1,p1,p2))

cc.pointcrossline(u2,p1,p2);

c1=circle(p1,r1);

if (p1==p2) {

c2=c1;

return 1;

}

c2=circle(p2,r1);

return 2;

}

//同时与直线 u,v 相切，半径为 r1 的圆

int getcircle (Line u,Line v,double r1,circle &c1,circle &c2,circle &c3,circle &c4) {

if(u.parallel(v))return 0;//两直线平行

Line u1=Line(u.s+(u.e-u.s).rotleft().trunc(r1),u.e+(u.e-u.s).rotleft().trunc(r1));

Line u2=Line(u.s+(u.e-u.s).rotright().trunc(r1),u.e+(u.e-u.s).rotright().trunc(r1));

Line v1=Line(v.s+(v.e-v.s).rotleft().trunc(r1),v.e+(v.e-v.s).rotleft().trunc(r1));

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

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)

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;

}

//求圆和三角形 pab 的相交面积

double areatriangle(Point a,Point b) {

if (sgn((p-a)^(p-b))==0) return 0.0;

Point q[5];

int len=0;

q[len++]=a;

Line l(a,b);

Point p1,p2;

if (pointcrossline(l,q[1],q[2])==2) {

if (sgn((a-q[1])\*(b-q[1]))<0) q[len++]=q[1];

if (sgn((a-q[2])\*(b-q[2]))<0) q[len++]=q[2];

}

q[len++]=b;

if (len==4 && sgn((q[0]-q[1])\*(q[2]-q[1]))>0) swap(q[1],q[2]);

double res=0;

for (int i=0; i<len-1; i++) {

if (relation(q[i])==0||relation(q[i+1])==0) {

double arg=p.rad(q[i],q[i+1]);

res+=r\*r\*arg/2.0;

}

else {

res+=fabs((q[i]-p)^(q[i+1]-p))/2.0;

}

}

return res;

}

};

// 最小面积覆盖

// A 必须是凸包 (而且是逆时针顺序)

double minRectangleCover(polygon A) {

//要特判 A.n < 3 的情况

if (A.n<3) return 0.0;

A.p[A.n]=A.p[0];

double ans=-1;

int r=1,p=1,q;

for(int i=0; i<A.n; i++) {

//卡出离边 A.p[i] - A.p[i+1] 最远的点

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)

r=(r+1)%A.n;

//卡出 A.p[i] - A.p[i+1] 方向上正向 n 最远的点

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)

p=(p+1)%A.n;

if (i==0) q=p;

//卡出 A.p[i] - A.p[i+1] 方向上负向最远的点

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)

q=(q+1)%A.n;

double d=(A.p[i]-A.p[i+1]).len2();

double tmp=cross(A.p[i],A.p[i+1],A.p[r])\*(dot(A.p[i],A.p[i+1],A.p[p])-dot(A.p[i],A.p[i+1],A.p[q]))/d;

if (ans<0 || ans>tmp) ans=tmp;

}

return ans;

}

// 直线切凸多边形

//多边形是逆时针的，在 q1q2 的左侧

vector<Point> convexCut(const vector<Point> &ps,Point q1,Point q2) {

vector<Point>qs;

int n=ps.size();

for (int i=0; i<n; i++) {

Point p1=ps[i],p2=ps[(i+1)%n];

int d1=sgn((q2-q1)^(p1-q1)),d2=sgn((q2-q1)^(p2-q1));

if (d1>=0)

qs.push\_back(p1);

if (d1\*d2<0)

qs.push\_back(Line(p1,p2).crosspoint(Line(q1,q2)));

}

return qs;

}

// 半平面交

struct halfplane:public Line {

double angle;

halfplane() {}

//表示向量 s->e 逆时针 (左侧) 的半平面

halfplane(Point \_s,Point \_e) {

s=\_s;

e=\_e;

}

halfplane(Line v) {

s=v.s;

e=v.e;

}

void calcangle() {

angle=atan2(e.y-s.y,e.x-s.x);

}

bool operator <(const halfplane &b)const {

return angle<b.angle;

}

};

struct halfplanes {

int n;

halfplane hp[maxp];

Point p[maxp];

int que[maxp];

int st,ed;

void push(halfplane tmp) {

hp[n++]=tmp;

}

//去重

void unique() {

int m=1;

for (int i=1; i<n; i++) {

if (sgn(hp[i].angle-hp[i-1].angle)!=0)

hp[m++]=hp[i];

else if(sgn((hp[m-1].e-hp[m-1].s)^(hp[i].s-hp[m-1].s))>0)

hp[m-1]=hp[i];

}

n=m;

}

bool halfplaneinsert() {

for (int i=0; i<n; i++) hp[i].calcangle();

sort(hp,hp+n);

unique();

que[st=0]=0;

que[ed=1]=1;

p[1]=hp[0].crosspoint(hp[1]);

for (int i=2; i<n; i++) {

while (st<ed && sgn((hp[i].e-hp[i].s)^(p[ed]-hp[i].s))<0)

ed--;

while (st<ed && sgn((hp[i].e-hp[i].s)^(p[st+1]-hp[i].s))<0)

st++;

que[++ed]=i;

if (hp[i].parallel(hp[que[ed-1]])) return false;

p[ed]=hp[i].crosspoint(hp[que[ed-1]]);

}

while (st<ed && sgn((hp[que[st]].e-hp[que[st]].s)^(p[ed]-hp[que[st]].s))<0)

ed--;

while (st<ed && sgn((hp[que[ed]].e-hp[que[ed]].s)^(p[st+1]-hp[que[ed]].s))<0)

st++;

if (st+1>=ed) return false;

return true;

}

//得到最后半平面交得到的凸多边形

//需要先调用 halfplaneinsert() 且返回 true

void getconvex(polygon &con) {

p[st]=hp[que[st]].crosspoint(hp[que[ed]]);

con.n=ed-st+1;

for(int j=st,i=0; j<=ed; i++,j++)

con.p[i]=p[j];

}

};

// 多圆问题

const int maxn = 1010;

struct circles {

circle c[maxn];

double ans[maxn];//ans[i] 表示被覆盖了 i 次的面积

double pre[maxn];

int n;

circles() {}

void add(circle cc) {

c[n++]=cc;

}

//x 包含在 y 中

bool inner(circle x,circle y) {

if (x.relationcircle(y) != 1) return 0;

return sgn(x.r-y.r)<=0?1:0;

}

//圆的面积并去掉内含的圆

void init\_or() {

bool mark[maxn]= {0};

int i,j,k=0;

for (i=0; i<n; i++) {

for (j=0; j<n; j++)

if (i!=j && !mark[j]) {

if ((c[i]==c[j]) || inner(c[i],c[j]))

break;

}

if (j<n) mark[i]=1;

}

for (i=0; i<n; i++)

if(!mark[i])

c[k++]=c[i];

n=k;

}

//圆的面积交去掉内含的圆

void init\_add() {

int i,j,k;

bool mark[maxn] = {0};

for (i=0; i<n; i++) {

for (j=0; j<n; j++)

if (i!=j && !mark[j]) {

if ((c[i]==c[j]) || inner(c[j],c[i])) break;

}

if (j<n) mark[i]=1;

}

for (i=0; i<n; i++)

if (!mark[i])

c[k++]=c[i];

n=k;

}

//半径为 r 的圆，弧度为 th 对应的弓形的面积

double areaarc(double th,double r) {

return 0.5\*r\*r\*(th-sin(th));

}

//测试 SPOJVCIRCLES SPOJCIRUT

//SPOJVCIRCLES 求 n 个圆并的面积，需要加上 init\_or() 去掉重复圆（否则WA）

//SPOJCIRUT 是求被覆盖 k 次的面积，不能加 init\_or()

//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()

//求多圆面积并，需要 init\_or, 其中一个目的就是去掉相同圆

void getarea() {

memset(ans,0,sizeof(ans));

vector<pair<double,int> >v;

for(int i=0; i<n; i++) {

v.clear();

v.push\_back(make\_pair(-pi,1));

v.push\_back(make\_pair(pi,-1));

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];

}

};

## Beat\_match

### data.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

#define endl '\n'

mt19937 mt(time(0));

uniform\_int\_distribution<int> dist1(3,20), dist2(3,20), dist3(3,20); // 数据建议给小一点

int ran(int f)

{

if(f==1) return dist1(mt);

if(f==2) return dist2(mt);

if(f==3) return dist3(mt);

}

int main()

{

ios::sync\_with\_stdio(0);

cin.tie(0),cout.tie(0);

cout<<ran(1)<<endl;

}

**Correct.cpp**

**To\_be\_tested.cpp**

### Beat\_match.cpp

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

#define endl '\n'

string str[4];

// #define lin // linux

#define win // windows

int main()

{

#ifdef lin

str[0]="./data > ./data.in"; // 数据生成

str[1]="./correct < ./data.in > ./std.out"; // 正确代码

str[2]="./To\_be\_tested < ./data.in > ./solve.out"; // 错误/待测试 代码

str[3]="diff ./std.out ./solve.out > ./diff.log";

#endif

#ifdef win

str[0]="data.exe > data.in"; // 数据生成

str[1]="correct.exe < data.in > std.out"; // 正确代码

str[2]="To\_be\_tested.exe < data.in > solve.out"; // 错误/待测试 代码

str[3]="fc std.out solve.out > diff.log";

#endif

int T=0;

while(1)

{

cout<<"test case: "<<++T<<" ";

for(int i=0;i<3;i++)

system(str[i].c\_str());

if(system(str[3].c\_str()))

{

cout<<"WA\n";

break;

}

cout<<"AC\n";

// system("pause");

}

}

## Tips

### 注意在pop的时候判断非空

cpp priority\_queue<pii> pq;

while(!pq.empty() && cnt>pq.top().first) pq.pop();

### priority\_queue-比较函数

cpp

typedef pair<int,int> pii; struct cmp // 仿函数

{

bool operator() (const pii &a, const pii &b) // 相当于重载 '<' 运算符

{

if(a.first!=b.first) return a.first>b.first;

return a.second<b.second;

}

};

priority\_queue<pii,vector<pii>,cmp> pq; // 优先队列用仿函数

### 数字+字符混合输入

如果输入 192.168.88.0/24 , 可以用scanf输入

cpp

int a,b,c,d,e; scanf("%d.%d.%d.%d/%d",&a,&b,&c,&d,&e);

### 数组下标越界

运行程序没有报错但是没输出, 可能是下标越界. 下标越界可能是 ***数组开小了*** 导致的.