

PumpkinYing的模板

喵喵喵

目录

[字符串 2](#_Toc525571218)

[KMP 2](#_Toc525571219)

[Manacher算法 3](#_Toc525571220)

[最小表示法 4](#_Toc525571221)

[exKmp 5](#_Toc525571222)

[AC自动机 6](#_Toc525571223)

[后缀数组 9](#_Toc525571224)

[后缀自动机 11](#_Toc525571225)

[回文树 15](#_Toc525571226)

[计算几何 18](#_Toc525571227)

[二维计算几何 18](#_Toc525571228)

[三维计算几何+分数模拟除法 24](#_Toc525571229)

[最小圆覆盖 29](#_Toc525571230)

[三维凸包 31](#_Toc525571231)

# 字符串

## KMP

const int maxn = 1000010;

char s[maxn],t[maxn];

int nxt[maxn];

void getnxt(char\* t) {

int n = strlen(s);

int i = 0,j = -1;

nxt[0] = -1;

while(i < n) {

while(j != -1 && t[i] != t[j]) j = nxt[j];

++i,++j;

nxt[i] = j;

}

}

void kmp(char\* s,char\* t) {

int n = strlen(s),m = strlen(t);

int i = 0,j = 0;

while(i < n) {

while(j != -1 && s[i] != t[j]) j = nxt[j];

++i,++j;

if(j == m) j = nxt[j];

}

}

## Manacher算法

const int N=1e6+10;

char s[N<<1],t[N];

int r[N<<1];

int manacher(const char\* a){

int id=0,mx=0,len=0;

s[len++]='!',s[len++]='#';

for(int i=0;a[i];i++) s[len++]=a[i],s[len++]='#';

s[len]=0;

int ans=0;

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

if(mx>i) r[i]=min(r[2\*id-i],mx-i);

else r[i]=1;

while(s[i-r[i]]==s[i+r[i]]) r[i]++;

if(i+r[i]>mx) id=i,mx=i+r[i];

ans=max(ans,r[i]-1);

}

return ans;

}

## 最小表示法

int getmin(char\* s) {

int n = strlen(s);

int i = 0,j = 1,k = 0;

while(i < n && j < n && k < n) {

int t = s[(i+k)%n]-s[(j+k)%n];

if(t == 0) k++;

else {

if(t > 0) i += k+1;

else j += k+1;

if(i == j) j++;

k = 0;

}

}

return min(i,j);

}

## exKmp

const int maxn = 1000010;

int f[maxn],ex[maxn];

void getFail(char\* t) {

int p = 0,mx = 0,n = strlen(t);

f[0] = n;

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

if(mx > i+f[i-p]) {

f[i] = f[i-p];

continue;

}

f[i] = max(mx-i,0);

while(t[i+f[i]] == t[f[i]]) f[i]++;

if(i+f[i] > mx) mx = i+f[i],p = i;

}

}

void ex\_kmp(char\* s,char\* t) {

getFail(t);

int p = 0,mx = 0,n = strlen(s);

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

if(mx > i+f[i-p]) {

ex[i] = ex[i-p];

continue;

}

ex[i] = max(mx-i,0);

while((i+ex[i] < n) && s[i+ex[i]] == t[ex[i]]) ex[i]++;

if(i+ex[i] > mx) mx = i+ex[i],p = i;

}

}

## AC自动机

const int maxn = 1010;

const int up = 26;

int cnt = 0,root = 0;

struct Node {

int num,fail,last;

int nxt[up];

void clear() {

mem(nxt,-1);

num = fail = last = 0;

}

}ns[maxn];

int newNode() {

ns[cnt].clear();

return cnt++;

}

void Insert(char\* s) {

int len = strlen(s);

int p = root;

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

int pos = s[i]-'A';

if(ns[p].nxt[pos] == -1) ns[p].nxt[pos] = newNode();

p = ns[p].nxt[pos];

}

ns[p].num++;

}

void getFail() {

queue<int> q;

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

int x = ns[root].nxt[i];

if(x == -1) ns[root].nxt[i] = root;

else {

ns[x].fail = ns[x].last = root;

q.push(x);

}

}

while(!q.empty()) {

int t = q.front();

q.pop();

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

int x = ns[t].nxt[i];

if(x == -1) {

ns[t].nxt[i] = ns[ns[t].fail].nxt[i];

}

else {

ns[x].fail = ns[ns[t].fail].nxt[i];

q.push(x);

if(ns[ns[x].fail].num > 0) ns[x].last = ns[x].fail;

else ns[x].last = ns[ns[x].fail].last;

}

}

}

}

void init() {

cnt = 0;

root = newNode();

}

## 后缀数组

const int maxn = 100005;

int rk[maxn],sa[maxn],height[maxn],w[maxn],cnt[maxn],res[maxn];

int n;

//sa[i]是排名为 i 的后缀在原串中的位置

//rank[i]是 i 位置的后缀排名

void getSa (int up) {

int \*k = rk,\*id = height,\*r = res;

for(int i = 0;i < up;i++) cnt[i] = 0;

for(int i = 0;i < n;i++) cnt[k[i] = w[i]]++;

for(int i = 0;i < up;i++) cnt[i+1] += cnt[i];

for(int i = n - 1; i >= 0; i--) sa[--cnt[k[i]]] = i;

for (int d = 1;d <= n;d <<= 1) {

int p = 0;

for(int i = n - d; i < n; i++) id[p++] = i;

for(int i = 0;i < n;i++) if(sa[i] >= d) id[p++] = sa[i] - d;

for(int i = 0;i < n;i++) r[i] = k[id[i]];

for(int i = 0;i < up;i++) cnt[i] = 0;

for(int i = 0;i < n;i++) cnt[r[i]]++;

for(int i = 0;i < up;i++) cnt[i+1] += cnt[i];

for(int i = n - 1; i >= 0; i--) sa[--cnt[r[i]]] = id[i];

swap(k,r);

p = 0;

k[sa[0]] = p++;

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

if(sa[i]+d < n && sa[i+1]+d <n &&r[sa[i]] == r[sa[i+1]]&& r[sa[i]+d] == r[sa[i+1]+d])

k[sa[i+1]] = p - 1;

else k[sa[i+1]] = p++;

}

if(p >= n) return ;

up = p;

}

}

//height[i]是排名 i 的后缀与排名 i-1 的后缀的公共部分

void getHeight() {

for(int i = 0;i < n;i++) rk[sa[i]] = i;

height[0] = 0;

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

if(rk[i] == 0) continue;

int j = sa[rk[i]-1];

while(i+p < n && j+p < n && w[i+p] == w[j+p]) p++;

height[rk[i]] = p;

p = max(0,p - 1);

}

}

//s[] 是原串

void getSuffix(char s[]) {

n = strlen(s);

int up = 0;

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

w[i] = s[i];

up = max(up,w[i]);

}

getSa(up+1);

getHeight();

}

后缀自动机

const int maxn = 250010;

const int up = 30;

struct state {

int len,pre;

int right;

int nxt[up];

}ns[maxn\*2];

int cnt = 0;

int newnode(int l) {

ns[cnt].len = l;

ns[cnt].pre = -1;

mem(ns[cnt].nxt,-1);

return cnt++;

}

int root = 0;

int last = 0;

void init() {

cnt = 0;

last = 0;

root = newnode(0);

}

void extend(int x) {

int p = last;

int np = newnode(ns[p].len+1);

while(p != -1 && ns[p].nxt[x] == -1)

ns[p].nxt[x] = np,p = ns[p].pre;

if(p == -1)

ns[np].pre = root;

else {

int q = ns[p].nxt[x];

if(ns[p].len+1 == ns[q].len) {

ns[np].pre = q;

}

else {

int clone = newnode(ns[p].len+1);

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

ns[clone].nxt[i] = ns[q].nxt[i];

ns[clone].pre = ns[q].pre;

for(;p != -1 && ns[p].nxt[x] == q;p = ns[p].pre)

ns[p].nxt[x] = clone;

ns[q].pre = ns[np].pre = clone;

ns[clone].right = 0;

}

}

last = np;

ns[np].right = 1;

}

int work(char\* s) {

int p = root;

int len = 0;

int ret = 0;

int n = strlen(s);

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

int pos = s[i]-'a';

while(p != -1 && ns[p].nxt[pos] == -1) {

p = ns[p].pre,len = ns[p].len;

}

if(p == -1) {

p = 0;

continue;

}

p = ns[p].nxt[pos];

len++;

ret = max(ret,len);

}

return ret;

}

char a[maxn];

int c[maxn];

int sa[maxn\*2];

int main() {

init();

scanf("%s",a);

int n = strlen(a);

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

extend(a[i]-'a');

for(int i = 1;i < cnt;i++) c[ns[i].len]++;

for(int i = 1;i <= n;i++) c[i] += c[i-1];

for(int i = cnt-1;i > 0;i--) sa[c[ns[i].len]--] = i;

for(int i = cnt-1;i > 0;i--) {

ns[ns[sa[i]].pre].right += ns[sa[i]].right;

}

return 0;

}

## 回文树

const int maxn = 200010;

const int up = 30;

struct node {

int nxt[up];

int fail;

int num,cnt;

int len;

}ns[maxn];

int last,len;

int cnt = 0;

int newnode(int l) {

mem(ns[cnt].nxt,0);

ns[cnt].fail = -1;

ns[cnt].num = ns[cnt].cnt = 0;

ns[cnt].len = l;

return cnt++;

}

void init() {

cnt = 0;

newnode(0);

newnode(-1);

last = 0;

len = 0;

ns[0].fail = 1;

}

int getFail(int x) {

while(s1[len-ns[x].len-1] != s1[len]) x = ns[x].fail;

return x;

}

void extend(int x) {

++len;

int cur = getFail(last);

if(ns[cur].nxt[x] == 0) {

int p = newnode(ns[cur].len+2);

ns[p].fail = ns[getFail(ns[cur].fail)].nxt[x];

ns[cur].nxt[x] = p;

ns[p].num = ns[ns[p].fail].num+1;

}

last = ns[cur].nxt[x];

ns[last].cnt++;

}

# 计算几何

## 二维计算几何

//two dimention

struct Point {

double x,y;

Point(double x = 0,double y = 0) : x(x),y(y) {}

};

typedef Point Vector;

Vector operator + (Vector a,Vector b) {

return Vector(a.x+b.x,a.y+b.y);

}

Vector operator - (Point a,Point b) {

return Vector(a.x-b.x,a.y-b.y);

}

Vector operator \* (Vector a,double k) {

return Vector(a.x\*k,a.y\*k);

}

Vector operator / (Vector a,double k) {

return Vector(a.x/k,a.y/k);

}

const double eps = 1e-7;

int dcmp(double x) {

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

else return x < 0 ? -1 : 1;

}

double dot(Vector a,Vector b) {

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

}

double length(Vector a) {

return sqrt(dot(a,a));

}

double angle(Vector a,Vector b) {

return acos(dot(a,b)/length(a)/length(b));

}

double cross(Vector a,Vector b) {

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

}

Vector rotate(Vector a,double rad) {

return Vector(a.x\*cos(rad)-a.y\*sin(rad),a.x\*sin(rad)+a.y\*cos(rad));

}

struct Line {

Point a,b;

Line(Point a = Point(0,0),Point b = Point(0,0)) : a(a),b(b) {}

};

//a,b intersect guaranteed

Point getIntersection(Line a,Line b) {

Point P = a.a;

Point Q = b.a;

Vector v = a.b-a.a;

Vector w = b.b-b.a;

Vector u = P-Q;

double t = cross(w,u)/cross(v,w);

return P+v\*t;

}

double pointLineDis(Point p,Line l) {

Vector v1 = l.b-l.a;

Vector v2 = p-l.a;

return fabs(cross(v1,v2))/length(v1);

}

typedef Line Seg;

bool onSeg(Point a,Seg s) {

return dcmp(cross(s.a-a,s.b-a)) == 0 && dcmp(dot(s.a-a,s.b-a)) <= 0;

}

bool lineIntersect(Line a,Line b) {

Vector v1 = a.b-a.a;

Vector v2 = b.b-b.a;

return dcmp(cross(v1,v2)) != 0;

}

bool segIntersect(Seg a,Seg b) {

if(onSeg(a.a,b) || onSeg(a.b,b) || onSeg(b.a,a) || onSeg(b.b,a)) return true;

int d1,d2;

d1 = dcmp(cross(a.a-b.a,b.b-b.a))\*dcmp(cross(a.b-b.a,b.b-b.a));

d2 = dcmp(cross(b.a-a.a,a.b-a.a))\*dcmp(cross(b.b-a.a,a.b-a.a));

if(d1 == -1 && d2 == -1) return true;

return false;

}

//Half plane intersection

bool cmp(Line a,Line b) {

Vector v1 = a.b-a.a;

Vector v2 = b.b-b.a;

double ang1 = atan2(v1.y,v1.x);

double ang2 = atan2(v2.y,v2.x);

return ang1 < ang2;

}

const int maxn = 100010;

Line lines[maxn];

Point p[maxn];

Point poly[maxn];

Line q[maxn];

bool onLeft(Point a,Line l) {

return cross(l.b-l.a,a-l.a) > 0;

}

int halfPlaneIntersection(int n) {

sort(lines,lines+n,cmp);

int l,r;

l = r = 0;

q[0] = lines[0];

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

Line cur = lines[i];

while(l < r && dcmp(cross(cur.b-cur.a,p[r-1]-cur.a)) <= 0) r--;

while(l < r && dcmp(cross(cur.b-cur.a,p[l]-cur.a)) <= 0) l++;

q[++r] = cur;

if(dcmp(cross(q[r].b-q[r].a,q[r-1].b-q[r-1].a)) == 0) {

r--;

if(dcmp(cross(q[r].b-q[r].a,cur.a-q[r].a)) == 1) q[r] = cur;

}

if(l < r) p[r-1] = getIntersection(q[r-1],q[r]);

}

while(l < r && dcmp(cross(q[l].b-q[l].a,p[r-1]-q[l].a)) <= 0) r--;

if(!(l < r)) return -1;

p[r] = getIntersection(q[r],q[l]);

for(int i = l;i <= r;i++) poly[i-l] = p[i];

return r-l+1;

}

## 三维计算几何+分数模拟除法

const double eps = 1e-10;

struct Point {

ll x,y,z;

Point(ll \_x = 0,ll \_y = 0,ll \_z = 0) : x(\_x),y(\_y),z(\_z) {}

bool operator == (const Point& b) const {

return x == b.x && y == b.y && z == b.z;

}

};

typedef Point Vector;

Vector operator + (Vector a,Vector b) {

return Vector(a.x+b.x,a.y+b.y,a.z+b.z);

}

Vector operator - (Point a,Point b) {

return Vector(a.x-b.x,a.y-b.y,a.z-b.z);

}

ll operator \* (Vector a,Vector b) {

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

}

Vector cross(Vector a,Vector b) {

return Vector(a.y\*b.z-a.z\*b.y,a.z\*b.x-a.x\*b.z,a.x\*b.y-a.y\*b.x);

}

Vector operator \* (double a,Vector b) {

return Vector(a\*b.x,a\*b.y,a\*b.z);

}

//平方之后的长度

ll vlen(Vector a) {

return a\*a;

}

struct Frac {

ll p,q;

Frac(ll \_p = 0,ll \_q = 1) : p(\_p),q(\_q) {}

bool operator < (const Frac& b) const {

return p\*b.q < q\*b.p;

}

void print() {

ll s[35];

for(int i = 0;i <= 32;i++) {

s[i] = p/q;

p %= q;

p \*= 10;

}

s[31] += 5;

for(int i = 31;i > 0;i--) {

if(s[i] > 9) {

s[i] -= 10;

s[i-1]++;

}

else break;

}

printf("%lld.",s[0]);

for(int i = 1;i <= 30;i++) printf("%lld",s[i]);

printf("\n");

}

};

struct Segment {

Point a,b;

Segment(Point \_a = Point(0,0,0),Point \_b = Point(0,0,0)) : a(\_a),b(\_b) {}

};

typedef Segment Line;

ll pointDis(Point a,Point b) {

Vector v = a-b;

return vlen(v);

}

//距离均为平方之后

Frac pointLineDis(Point p,Line l) {

return Frac(vlen(cross(p-l.a,l.b-l.a)),pointDis(l.a,l.b));

}

//不平行直线距离

Frac lineDis(Line a,Line b) {

Vector n = cross(a.a-a.b,b.a-b.b);

return Frac(((a.a-b.a)\*n)\*((a.a-b.a)\*n),vlen(n));

}

//-1在a远离b方向 1在b远离a方向

int getDir(Point p,Segment l) {

if((p-l.a)\*(l.b-l.a) < 0) return -1;

if((p-l.b)\*(l.a-l.b) < 0) return 1;

return 0;

}

Frac pointSegDis(Point p,Segment l) {

if(l.a == l.b) return Frac(pointDis(p,l.a),1);

if(getDir(p,l) == -1) return Frac(pointDis(p,l.a),1);

if(getDir(p,l) == 1) return Frac(pointDis(p,l.b),1);

return pointLineDis(p,l);

}

Frac segDis(Segment a,Segment b) {

Frac ans(1e9,1);

Vector n = cross(a.a-a.b,b.a-b.b);

Vector p1 = cross(a.b-a.a,n);

Vector p2 = cross(b.b-b.a,n);

//判断是否平移之后交叉

if(((b.a-a.a)\*p1)\*((b.b-a.a)\*p1) < 0 && ((a.a-b.a)\*p2)\*((a.b-b.a)\*p2) < 0) {

if(vlen(n) != 0) ans = lineDis(a,b);

else ans = Frac(0,1);

}

ans = min(ans,pointSegDis(a.a,b));

ans = min(ans,pointSegDis(a.b,b));

ans = min(ans,pointSegDis(b.a,a));

ans = min(ans,pointSegDis(b.b,a));

return ans;

}

## 最小圆覆盖

Point ps[510];

void shuffle(Point\* p,int n) {

srand(23333);

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

swap(p[i],p[rand()%n]);

}

}

Point midPoint(Point a,Point b) {

return Point((a.x+b.x)/2,(a.y+b.y)/2);

}

Vector verticalVec(Vector x) {

return Vector(x.y,-x.x);

}

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

Line u,v;

u.a = midPoint(a,b);

v.a = midPoint(b,c);

u.b = u.a+verticalVec(b-u.a);

v.b = v.a+verticalVec(b-v.a);

return getIntersection(u,v);

}

void minCir(int n) {

shuffle(ps,n);

Point O = ps[0];

double r = 0;

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

if(dcmp(dis(O,ps[i])-r) <= 0) continue;

O = ps[i];

r = 0;

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

if(dcmp(dis(O,ps[j])-r) <= 0) continue;

O = midPoint(ps[i],ps[j]);

r = dis(O,ps[j]);

for(int k = 0;k < j;k++) {

if(dcmp(dis(O,ps[k])-r) <= 0) continue;

O = getCenter(ps[i],ps[j],ps[k]);

r = dis(O,ps[i]);

}

}

}

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

}

## 三维凸包

const int maxn = 510;

const double eps = 1e-7;

//三维点

struct Point {

double x,y,z;

Point(){}

Point(double \_x, double \_y, double \_z): x(\_x), y(\_y), z(\_z){}

Point operator - (const Point p1){return Point(x - p1.x, y - p1.y, z - p1.z);}

Point operator \* (Point p) {return Point(y\*p.z-z\*p.y, z\*p.x-x\*p.z, x\*p.y-y\*p.x);} //叉乘

double operator ^ (Point p) {return x\*p.x+y\*p.y+z\*p.z;} //点乘

};

struct \_3DCH {

struct fac {

int a, b, c; //表示凸包一个面上三个点的编号

bool ok; //表示该面是否属于最终凸包中的面

};

int n; //初始点数

Point P[maxn]; //初始点

int cnt; //凸包表面的三角形数

fac F[maxn\*8]; //凸包表面的三角形

int to[maxn][maxn];

double vlen(Point a) {return sqrt(a.x\*a.x+a.y\*a.y+a.z\*a.z);} //向量长度

double area(Point a, Point b, Point c) {return vlen((b-a)\*(c-a));} //三角形面积\*2

double volume(Point a, Point b, Point c, Point d) {return (b-a)\*(c-a)^(d-a);} //四面体有向体积\*6

//正：点在面同向

double pointOf(Point &p, fac &f) {

Point m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];

return (m \* n) ^ t;

}

void deal(int p,int a,int b) {

int f = to[a][b];

fac add;

if (F[f].ok) {

if (pointOf(P[p], F[f]) > eps)

dfs(p, f);

else {

add.a = b, add.b = a, add.c = p, add.ok = 1;

to[p][b] = to[a][p] = to[b][a] = cnt;

F[cnt++] = add;

}

}

}

void dfs(int p,int cur) {

F[cur].ok = 0;

deal(p, F[cur].b, F[cur].a);

deal(p, F[cur].c, F[cur].b);

deal(p, F[cur].a, F[cur].c);

}

bool same(int s, int t) {

Point &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];

return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b, c, P[F[t].b])) < eps && fabs(volume(a, b, c, P[F[t].c])) < eps;

}

//构建三维凸包

void construct() {

cnt = 0;

if (n < 4)

return;

/\*\*\*\*\*\*\*\*\*此段是为了保证前四个点不公面，若已保证，可去掉\*\*\*\*\*\*\*\*/

bool sb = 1;

//使前两点不公点

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

if (vlen(P[0] - P[i]) > eps){

swap(P[1], P[i]);

sb = 0;

break;

}

}

if(sb) return;

sb = 1;

//使前三点不公线

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

if (vlen((P[0] - P[1]) \* (P[1] - P[i])) > eps){

swap(P[2], P[i]);

sb = 0;

break;

}

}

if(sb) return;

sb = 1;

//使前四点不共面

for (int i = 3; i < n; i++){

if (fabs((P[0] - P[1]) \* (P[1] - P[2]) ^ (P[0] - P[i])) > eps){

swap(P[3], P[i]);

sb = 0;

break;

}

}

if(sb) return;

/\*\*\*\*\*\*\*\*\*此段是为了保证前四个点不公面\*\*\*\*\*\*\*\*/

fac add;

for (int i = 0; i < 4; i++){

add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = 1;

if (pointOf(P[i], add) > 0)

swap(add.b, add.c);

to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;

F[cnt++] = add;

}

for (int i = 4; i < n; i++){

for (int j = 0; j < cnt; j++){

if (F[j].ok && pointOf(P[i], F[j]) > eps){

dfs(i, j);

break;

}

}

}

int tmp = cnt;

cnt = 0;

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

if (F[i].ok){

F[cnt++] = F[i];

}

}

}

//表面积

double area(){

double ret = 0.0;

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

ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);

}

return ret / 2.0;

}

//体积

double volume(){

Point O(0, 0, 0);

double ret = 0.0;

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

ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);

}

return fabs(ret / 6.0);

}

//表面三角形数

int facetCnt\_tri(){

return cnt;

}

//表面多边形数

int facetCnt(){

int ans = 0;

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

bool nb = 1;

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

if (same(i, j)){

nb = 0;

break;

}

}

ans += nb;

}

return ans;

}

};

\_3DCH hull; //内有大数组，不易放在函数内

int main() {

while (~scanf("%d", &hull.n)){

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

scanf("%lf%lf%lf", &hull.P[i].x, &hull.P[i].y, &hull.P[i].z);

hull.construct();

printf("%d\n", hull.facetCnt());

}

return 0;

}

### 最小球覆盖

const int maxn = 10010;

const double eps = 1e-8;

struct Point {

double x,y,z;

void read() {

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

}

Point(double \_x = 0,double \_y = 0,double \_z = 0) : x(\_x),y(\_y),z(\_z) {}

Point operator - (const Point& b) const {

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

}

double len() {

return x\*x+y\*y+z\*z;

}

void print() {

cerr<<x<<' '<<y<<' '<<z<<endl;

}

}ps[maxn];

Point cross(Point a,Point b) {

return Point(a.y\*b.z-a.z\*b.y,a.z\*b.x-a.x\*b.z,a.x\*b.y-a.y\*b.x);

}

double dis(Point a,Point b) {

return (a-b).len();

}

Point Gauss(double a[][5],int n) {

int m = 4;

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

int l = i;

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

if (abs(a[j][i]) > abs(a[l][i])) l = j;

if (l != i) {

for (int j = 1;j <= m;j++) swap(a[l][j],a[i][j]);

}

double t = a[i][i];

for (int j = 1;j <= m;j++)

a[i][j] /= t;

for (int j = 1;j <= n;j++)

if (j != i && a[j][i] != 0) {

double t = a[j][i];

for (int k = 1;k <= m;k++)

a[j][k] -= t\*a[i][k];

}

}

return Point(a[1][4],a[2][4],a[3][4]);

}

Point getMid\_2(Point a,Point b) {

return Point((a.x+b.x)/2,(a.y+b.y)/2,(a.z+b.z)/2);

}

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

return cross(a-b,a-c);

}

double sqr(double x) {

return x\*x;

}

int dcmp(double x) {

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

return x < 0 ? -1 : 1;

}

Point getMid\_3(Point a,Point b,Point c) {

Point n = getVertic(a,b,c);

double para[5][5];

para[1][1] = 2\*(a.x-b.x);

para[1][2] = 2\*(a.y-b.y);

para[1][3] = 2\*(a.z-b.z);

para[1][4] = sqr(a.x)+sqr(a.y)+sqr(a.z)-sqr(b.x)-sqr(b.y)-sqr(b.z);

para[2][1] = 2\*(a.x-c.x);

para[2][2] = 2\*(a.y-c.y);

para[2][3] = 2\*(a.z-c.z);

para[2][4] = sqr(a.x)+sqr(a.y)+sqr(a.z)-sqr(c.x)-sqr(c.y)-sqr(c.z);

para[3][1] = n.x;

para[3][2] = n.y;

para[3][3] = n.z;

para[3][4] = n.x\*a.x+n.y\*a.y+n.z\*a.z;

Point ret = Gauss(para,3);

return ret;

}

Point getMid\_4(Point a,Point b,Point c,Point d) {

double para[5][5];

para[1][1] = 2\*(a.x-b.x);

para[1][2] = 2\*(a.y-b.y);

para[1][3] = 2\*(a.z-b.z);

para[1][4] = sqr(a.x)+sqr(a.y)+sqr(a.z)-sqr(b.x)-sqr(b.y)-sqr(b.z);

para[2][1] = 2\*(a.x-c.x);

para[2][2] = 2\*(a.y-c.y);

para[2][3] = 2\*(a.z-c.z);

para[2][4] = sqr(a.x)+sqr(a.y)+sqr(a.z)-sqr(c.x)-sqr(c.y)-sqr(c.z);

para[3][1] = 2\*(a.x-d.x);

para[3][2] = 2\*(a.y-d.y);

para[3][3] = 2\*(a.z-d.z);

para[3][4] = sqr(a.x)+sqr(a.y)+sqr(a.z)-sqr(d.x)-sqr(d.y)-sqr(d.z);

Point ret = Gauss(para,3);

return ret;

}

void shuffle(int n) {

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

swap(ps[i],ps[rand()%n]);

}

}

double solve(int n) {

shuffle(n);

Point O = ps[0];

double r = 0;

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

if(dcmp(dis(O,ps[i])-r) <= 0) continue;

O = ps[i];

r = 0;

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

if(dcmp(dis(O,ps[j])-r) <= 0) continue;

O = getMid\_2(ps[i],ps[j]);

r = dis(O,ps[j]);

for(int k = 0;k < j;k++) {

if(dcmp(dis(O,ps[k])-r) <= 0) continue;

O = getMid\_3(ps[i],ps[j],ps[k]);

r = dis(O,ps[k]);

for(int l = 0;l < k;l++) {

if(dcmp(dis(O,ps[l])-r) <= 0) continue;

O = getMid\_4(ps[i],ps[j],ps[k],ps[l]);

r = dis(O,ps[l]);

}

}

}

}

return r;

}

int main() {

srand(1019);

int n;

while(~scanf("%d",&n) && n) {

for(int i = 0;i < n;i++) ps[i].read();

double ans = solve(n);

printf("%.3f\n",sqrt(ans));

}

return 0;

}