# 第8章 计算几何

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | const double pi = acos(-1.0); //圆周率，精确到15位小数：3.141592653589793  const double eps = 1e-8; //偏差值，有时用1e-10，但是要注意精度  int sgn(double x){ //判断x的大小  if(fabs(x) < eps) return 0; //x==0，返回0  else return x<0?-1:1; //x<0返回-1，x>0返回1  }  int dcmp(double x, double y){ //比较两个浮点数  if(fabs(x - y) < eps) return 0; //x==y，返回0  else return x<y ?-1:1; //x<y返回-1，x>y返回1  } |

8.1 二维几何

8.1.1 点和向量

**1. 点**

二维平面中的点，用坐标(*x, y*)来表示。

|  |  |
| --- | --- |
| 1  2  3  4  5 | struct Point{  double x,y;  Point(){}  Point(double x,double y):x(x),y(y){}  }; |

**2. 两点之间的距离**

把两点看成直角三角形的两个顶点，斜边就是两点的距离。

（1）用库函数hypot()计算直角三角形的斜边长。

|  |
| --- |
| double Distance(Point A, Point B){ return hypot(A.x-B.x,A.y-B.y); } |

（2）或者用sqrt()函数计算。

|  |
| --- |
| double Dist(Point A,Point B){ return sqrt((A.x-B.x)\*(A.x-B.x) + (A.y-B.y)\*(A.y-B.y)); } |

**3. 向量**

|  |
| --- |
| typedef Point Vector; |

* 提示：向量并不是一个有向线段，只是表示方向和大小。所以向量平移后仍然不变。

**4. 向量的运算**

（1）加：点与点的加法运算没有意义；点与向量相加得到另一个点；向量与向量相加得到另外一个向量。如图8.2所示。

|  |
| --- |
| Point operator + (Point B){return Point(x+B.x,y+B.y);} |

（2）减：两个点的差是一个向量；向量A减B，得到由B指向A的向量。

|  |
| --- |
| Point operator - (Point B){return Point(x-B.x,y-B.y);} |

（3）乘：向量与实数相乘得到等比例放大的向量。

|  |
| --- |
| Point operator \* (double k){return Point(x\*k,y\*k);} |

（4）除：向量与实数相除得到等比例缩小的向量。

|  |
| --- |
| Point operator / (double k){return Point(x/k,y/k);} |

（5）等于：

|  |
| --- |
| bool operator == (Point B){return sgn(x-B.x)==0 && sgn(y-B.y)==0;} |

8.1.2 点积和叉积

向量的基本运算是点积和叉积，它们定义了向量的大小、方向的基本关系，计算几何的各种操作几乎都基于这两种运算。

**1. 点积（Dot product）**

|  |
| --- |
| double Dot(Vector A,Vector B){ return A.x\*B.x + A.y\*B.y; } |

**2. 点积的应用**

（2）求向量A的长度

|  |
| --- |
| double Len(Vector A){return sqrt(Dot(A,A));} |

由于开方运算可能导致小数，可以改为求长度的平方，避免开方运算：

|  |
| --- |
| double Len2(Vector A){return Dot(A,A);} |

（3）求向量A与B的夹角大小

|  |
| --- |
| double Angle(Vector A,Vector B){return acos(Dot(A,B)/Len(A)/Len(B));} |

**3. 叉积（****Cross product）**

计算叉积A×B时，也不需要用到夹角θ，而是用下面的简单代码计算：

|  |
| --- |
| double Cross(Vector A,Vector B){return A.x\*B.y – A.y\*B.x;} |

A×B与B×A相反。

叉积有正负，这个性质使得叉积能用于很多重要的场合。

**4. 叉积的基本应用**

2）计算两向量构成的平行四边形有向面积

三个点A、B、C，以A为公共点，得到2个向量B-A和C-A，它们构成的平行四边形，面积是：

|  |
| --- |
| double Area2(Point A,Point B,Point C){ return Cross(B-A, C-A);} |

4）向量旋转

|  |  |
| --- | --- |
| 1  2  3 | Vector Rotate(Vector A, double rad){  return Vector(A.x\*cos(rad)-A.y\*sin(rad), A.x\*sin(rad)+A.y\*cos(rad));  } |

特殊情况是旋转90度：

逆时针旋转90度：Rotate(A, pi/2)，返回Vector(-A.y, A.x)；

顺时针旋转90度：Rotate(A, -pi/2)，返回Vector(A.y, - A.x)。

有时需要求单位法向量，即逆时针转90度，然后取单位值。

|  |
| --- |
| Vector Normal(Vector A){return Vector(-A.y/Len(A), A.x/Len(A));} |

5）用叉积检查两个向量是否平行或重合

|  |
| --- |
| bool Parallel(Vector A, Vector B){return sgn(Cross(A,B)) == 0;} //返回true表示平行或重合 |

8.1.3 点和线

**1. 直线的表示**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | struct Line{  Point p1,p2; //（1）线上的两个点  Line(){}  Line(Point p1,Point p2):p1(p1),p2(p2){}  Line(Point p,double angle){ //（4）根据一个点和倾斜角 angle 确定直线,0<=angle<pi  p1 = p;  if(sgn(angle – pi/2) == 0){p2 = (p1 + Point(0,1));}  else{p2 = (p1 + Point(1,tan(angle)));}  }  Line(double a,double b,double c){ //（2）ax+by+c=0  if(sgn(a) == 0){  p1 = Point(0,-c/b);  p2 = Point(1,-c/b);  }  else if(sgn(b) == 0){  p1 = Point(-c/a,0);  p2 = Point(-c/a,1);  }  else{  p1 = Point(0,-c/b);  p2 = Point(1,(-c-a)/b);  }  }  }; |

**2. 线段的表示**

|  |
| --- |
| typedef Line Segment; |

**3. 点和直线的位置关系**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int Point\_line\_relation(Point p, Line v){  int c = sgn(Cross(p-v.p1,v.p2-v.p1));  if(c < 0)return 1; //1：p在v的左边  if(c > 0)return 2; //2：p在v的右边  return 0; //0：p在v上  } |

**4. 点和线段的位置关系**

|  |  |
| --- | --- |
| 1  2  3 | bool Point\_on\_seg(Point p, Line v){ //点和线段：0 点不在线段v上；1 点在线段v上  return sgn(Cross(p-v.p1, v.p2-v.p1)) == 0 && sgn(Dot(p – v.p1,p- v.p2)) <= 0;  } |

**5. 点到直线的距离**

|  |  |
| --- | --- |
| 1  2  3 | double Dis\_point\_line(Point p, Line v){  return fabs(Cross(p-v.p1,v.p2-v.p1))/Distance(v.p1,v.p2);  } |

**6. 点在直线上的投影**

|  |  |
| --- | --- |
| 1  2  3  4 | Point Point\_line\_proj(Point p, Line v){  double k = Dot(v.p2-v.p1,p-v.p1)/Len2(v.p2-v.p1);  return v.p1+(v.p2-v.p1)\*k;  } |

**7. 点关于直线的对称点**

|  |  |
| --- | --- |
| 1  2  3  4 | Point Point\_line\_symmetry(Point p, Line v){  Point q = Point\_line\_proj(p,v);  return Point(2\*q.x-p.x,2\*q.y-p.y);  } |

**8. 点到线段的距离**

|  |  |
| --- | --- |
| 1  2  3  4  5 | double Dis\_point\_seg(Point p, Segment v){  if(sgn(Dot(p- v.p1,v.p2-v.p1))<0 || sgn(Dot(p- v.p2,v.p1-v.p2))<0)  return min(Distance(p,v.p1),Distance(p,v.p2));  return Dis\_point\_line(p,v); //点的投影在线段上  } |

**9. 两条直线的位置关系**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int Line\_relation(Line v1, Line v2){  if(sgn(Cross(v1.p2-v1.p1,v2.p2-v2.p1)) == 0){  if(Point\_line\_relation(v1.p1,v2)==0) return 1; //1 重合  else return 0; //0 平行  }  return 2; //2 相交  } |

**10. 两条直线的交点**

|  |  |
| --- | --- |
| 1  2  3  4  5 | Point Cross\_point(Point a,Point b,Point c,Point d){ //Line1:ab, Line2:cd  double s1 = Cross(b-a,c-a);  double s2 = Cross(b-a,d-a); //叉积有正负  return Point(c.x\*s2-d.x\*s1,c.y\*s2-d.y\*s1)/(s2-s1);  } |

**11. 两个线段是否相交**

|  |  |
| --- | --- |
| 1  2  3  4  5 | bool Cross\_segment(Point a,Point b,Point c,Point d){ //Line1:ab, Line2:cd  double c1 = Cross(b-a,c-a),c2=Cross(b-a,d-a);  double d1 = Cross(d-c,a-c),d2=Cross(d-c,b-c);  return sgn(c1)\*sgn(c2) < 0 && sgn(d1)\*sgn(d2) < 0; //1相交；0不相交  } |

|  |  |
| --- | --- |
| 例8-1. 神秘大三角 洛谷P1355 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #include <bits/stdc++.h>  using namespace std;  struct Point{  int x,y;  Point (){}  Point (int x,int y):x(x),y(y){}  Point operator + (Point B){return Point(x+B.x,y+B.y);}  Point operator - (Point B){return Point(x-B.x,y-B.y);}  }v[3],p; //v:三角形， p:点  typedef Point Vector; //定义向量  double Cross(Vector A,Vector B){return A.x\*B.y - A.y\*B.x;} //叉积  int main(){  int left=0,right=0;  for(int i=0;i<3;i++) scanf(" (%d,%d)",&v[i].x,&v[i].y);  scanf(" (%d,%d)",&p.x,&p.y);  for(int i=0;i<3;i++) {  int relation = Cross(p-v[i],p-v[(i+1)%3]);  if(relation > 0) right++; //p在直线v右边  if(relation < 0) left++; //p在直线v左边  }  if(right==3 || left==3) puts("1"); //在三角内  else if(right>0 && left>0) puts("2"); //在三角外  else if(right+left==1) puts("4"); //在顶点上  else puts("3"); //在边界上，不含顶点  return 0;  } |

8.1.4 多边形

**1. 点和多边形的关系**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | int Point\_in\_polygon(Point pt,Point \*p,int n){ //点pt，多边形Point \*p  for(int i = 0;i < n;i++){ //3：点在多边形的顶点上  if(p[i] == pt) return 3;  }  for(int i = 0;i < n;i++){ //2：点在多边形的边上  Line v=Line(p[i],p[(i+1)%n]);  if(Point\_on\_seg(pt,v)) return 2;  }  int num = 0;  for(int i = 0;i < n;i++){  int j = (i+1)% n;  int c = sgn(Cross(pt-p[j],p[i]-p[j]));  int u = sgn(p[i].y – pt.y);  int v = sgn(p[j].y – pt.y);  if(c > 0 && u < 0 && v >=0) num++;  if(c < 0 && u >=0 && v < 0) num--;  }  return num != 0; //1：点在内部; 0：点在外部  } |

**2. 多边形的面积**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | double Polygon\_area(Point \*p, int n){ //Point \*p表示多边形  double area = 0;  for(int i = 0;i < n;i++)  area += Cross(p[i],p[(i+1)%n]);  return area/2; //面积有正负，返回时不能简单地取绝对值  } |

**3. 多边形的重心**

|  |
| --- |
| 例8-2. Lifting the Stone poj 1385 |

下面代码中的函数Polygon\_center()返回多边形的重心。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | #include <stdio.h>  struct Point{  double x,y;  Point(double X=0,double Y=0){x=X,y=Y;}  Point operator + (Point B){return Point (x+B.x,y+B.y);}  Point operator - (Point B){return Point (x-B.x,y-B.y);}  Point operator \* (double k){return Point (x\*k,y\*k);}  Point operator / (double k){return Point (x/k,y/k);}  };  typedef Point Vector;  double Cross(Vector A,Vector B){return A.x\*B.y - A.y\*B.x;}  double Polygon\_area(Point \*p, int n){ //求多边形面积  double area = 0;  for(int i = 0;i < n;i++) area += Cross(p[i],p[(i+1)%n]);  return area/2; //面积有正负，不能取绝对值  }  Point Polygon\_center(Point \*p, int n){ //求多边形重心  Point ans(0,0);  if(Polygon\_area(p,n)==0) return ans;  for(int i = 0;i < n;i++)  ans = ans+(p[i]+p[(i+1)%n])\*Cross(p[i],p[(i+1)%n]);  return ans/Polygon\_area(p,n)/6;  }  Point p[100000];  int main(){  int t; scanf("%d",&t);  while(t--){  int n; scanf("%d",&n);  for(int i=0;i<n;i++) scanf("%lf%lf",&p[i].x,&p[i].y);  Point c = Polygon\_center(p,n); //重心坐标  printf("%.2f %.2f\n",c.x,c.y); //注意这里输出用%f，不是%lf  }  return 0;  } |

8.1.5 凸包

|  |  |
| --- | --- |
| 例8-3. 圈奶牛 洛谷 P2742 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  37  48  49  50  51  52 | #include <bits/stdc++.h>  using namespace std;  const int N = 1e5+1;  const double eps = 1e-6;  int sgn(double x){ //判断x是否等于0  if(fabs(x) < eps) return 0;  else return x<0?-1:1;  }  struct Point{  double x,y;  Point(){}  Point(double x, double y):x(x),y(y){}  Point operator + (Point B){return Point(x+B.x,y+B.y);}  Point operator - (Point B){return Point(x-B.x,y-B.y);}  bool operator == (Point B){return sgn(x-B.x) == 0 && sgn(y-B.y) == 0;}  bool operator < (Point B){ //用于sort()排序，先按x排序，再按y排序  return sgn(x-B.x)<0 || (sgn(x-B.x)==0 && sgn(y-B.y)<0);}  };  typedef Point Vector;  double Cross(Vector A,Vector B){return A.x\*B.y - A.y\*B.x;} //叉积  double Distance(Point A,Point B){return hypot(A.x-B.x,A.y-B.y);}  //Convex\_hull()求凸包。凸包顶点放在ch中，返回值是凸包的顶点数  int Convex\_hull(Point \*p,int n,Point \*ch){  n = unique(p,p+n)-p; //去除重复点  sort(p,p+n); //对点排序：按x从小到大排序，如果x相同，按y排序  int v=0;  //求下凸包。如果p[i]是右拐弯的，这个点不在凸包上，往回退  for(int i=0;i<n;i++){  while(v>1 && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-1]))<=0) //把后面ch[v-1]改成ch[v-2]也行  v--;  ch[v++]=p[i];  }  int j=v;  //求上凸包  for(int i=n-2;i>=0;i--){  while(v>j && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-1]))<=0) //把后面ch[v-1]改成ch[v-2]也行  v--;  ch[v++]=p[i];  }  if(n>1) v--;  return v; //返回值v是凸包的顶点数  }  Point p[N],ch[N]; //输入点是p[]，计算得到的凸包顶点放在ch[]中  int main(){  int n; cin >> n;  for(int i=0;i<n;i++) scanf("%lf%lf",&p[i].x,&p[i].y);  int v = Convex\_hull(p,n,ch); //返回凸包的顶点数v  double ans=0;  for(int i=0;i<v;i++) ans += Distance(ch[i],ch[(i+1)%v]); //计算凸包周长  printf("%.2f\n",ans);  return 0;  } | |

8.1.6 最近点对

|  |  |  |
| --- | --- | --- |
| 例8-4. 平面上的最接近点对 洛谷 P1257 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | #include <bits/stdc++.h>  using namespace std;  const double eps = 1e-8;  const int N = 100010;  const double INF = 1e20;  int sgn(double x){  if(fabs(x) < eps) return 0;  else return x<0?-1:1;  }  struct Point{double x,y;};  double Distance(Point A, Point B){return hypot(A.x-B.x,A.y-B.y);}  bool cmpxy(Point A,Point B){ //排序：先对横坐标x排序，再对y排序  return sgn(A.x-B.x)<0 || (sgn(A.x-B.x)==0 && sgn(A.y-B.y)<0);  }  bool cmpy (Point A,Point B){return sgn(A.y-B.y)<0;} //只对y坐标排序  Point p[N],tmp\_p[N];  double Closest\_Pair(int left,int right){  double dis = INF;  if(left == right) return dis; //只剩1个点  if(left + 1 == right) return Distance(p[left], p[right]);//只剩2个点  int mid = (left+right)/2; //分治  double d1 = Closest\_Pair(left,mid); //求s1内的最近点对  double d2 = Closest\_Pair(mid+1,right); //求s2内的最近点对  dis = min(d1,d2);  int k = 0;  for(int i=left;i<=right;i++) //在s1和s2中间附近找可能的最小点对  if(fabs(p[mid].x - p[i].x) <= dis) //按x坐标来找  tmp\_p[k++] = p[i];  sort(tmp\_p,tmp\_p+k,cmpy); //按y坐标排序，用于剪枝。这里不能按x坐标排序  for(int i=0;i<k;i++)  for(int j=i+1;j<k;j++){  if(tmp\_p[j].y - tmp\_p[i].y >= dis) break; //剪枝  dis = min(dis,Distance(tmp\_p[i],tmp\_p[j]));  }  return dis; //返回最小距离  }  int main(){  int n; cin >>n;  for(int i=0;i<n;i++) scanf("%lf%lf",&p[i].x,&p[i].y);  sort(p,p+n,cmpxy); //先排序  printf("%.4f\n",Closest\_Pair(0,n-1)); //输出最短距离  return 0;  } |

8.1.8 半平面交

**1. 半平面的表示**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | struct Line{  Point p; //直线上一个点  Vector v; //方向向量，它的左边是半平面  double ang; //极角，从x正半轴旋转到v的角度  Line(){};  Line(Point p, Vector v):p(p),v(v){ang = atan2(v.y, v.x);}  bool operator < (Line &L){return ang < L.ang;} //用于排序  }; |

**2. 半平面交算法**

|  |  |  |
| --- | --- | --- |
| 例8-6. Run hdu 2297 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | #include <bits/stdc++.h>  using namespace std;  const double INF = 1e12;  const double pi = acos(-1.0); //圆周率，精确到15位小数：3.141592653589793  const double eps = 1e-8;  int sgn(double x){  if(fabs(x) < eps) return 0;  else return x<0?-1:1;  }  struct Point{  double x,y;  Point(){}  Point(double x,double y):x(x),y(y){}  Point operator + (Point B){return Point(x+B.x,y+B.y);}  Point operator – (Point B){return Point(x-B.x,y-B.y);}  Point operator \* (double k){return Point(x\*k,y\*k);}  };  typedef Point Vector;  double Cross(Vector A,Vector B){return A.x\*B.y – A.y\*B.x;} //叉积  struct Line{  Point p;  Vector v;  double ang;  Line(){};  Line(Point p,Vector v):p(p),v(v){ang=atan2(v.y,v.x);}  bool operator < (Line &L){return ang<L.ang;} //用于极角排序  };  //点p在线L左边，即点p在线L在外面：  bool OnLeft(Line L,Point p){return sgn(Cross(L.v,p-L.p))>0;}  Point Cross\_point(Line a,Line b){ //两直线交点  Vector u=a.p-b.p;  double t=Cross(b.v,u)/Cross(a.v,b.v);  return a.p+a.v\*t;  }  vector<Point> HPI(vector<Line> L){ //求半平面交，返回凸多边形  int n=L.size();  sort(L.begin(),L.end()); //将所有半平面按照极角排序。  int first,last; //指向双端队列的第一个和最后一个元素  vector<Point> p(n); //两个相邻半平面的交点  vector<Line> q(n); //双端队列  vector<Point> ans; //半平面交形成的凸包  q[first=last=0]=L[0];  for(int i=1;i<n;i++){  //情况1：删除尾部的半平面  while(first<last && !OnLeft(L[i], p[last-1])) last--;  //情况2：删除首部的半平面：  while(first<last && !OnLeft(L[i], p[first])) first++;  q[++last]=L[i]; //将当前的半平面加入双端队列尾部  //极角相同的两个半平面，保留左边：  if(fabs(Cross(q[last].v,q[last-1].v)) < eps){  last--;  if(OnLeft(q[last],L[i].p)) q[last]=L[i];  }  //计算队列尾部半平面交点：  if(first<last) p[last-1]=Cross\_point(q[last-1],q[last]);  }  //情况3：删除队列尾部的无用半平面  while(first<last && !OnLeft(q[first],p[last-1])) last--;  if(last-first<=1) return ans; //空集  p[last]=Cross\_point(q[last],q[first]); //计算队列首尾部的交点。  for(int i=first;i<=last;i++) ans.push\_back(p[i]); //复制。  return ans; //返回凸多边形  }  int main(){  int T,n; cin>>T;  while(T--){  cin>>n;  vector<Line> L;  L.push\_back(Line(Point(0,0),Vector(0,-1))); //加一个半平面F:反向y轴  L.push\_back(Line(Point(0,INF),Vector(-1,0))); //加一个半平面E:y极大的向左的直线  while(n--){  double a,b; scanf(“%lf%lf”,&a,&b);  L.push\_back(Line(Point(0,a),Vector(1,b)));  }  vector<Point> ans=HPI(L); //得到凸多边形  printf(“%d\n”,ans.size()-2); //去掉人为加的两个点  }  return 0;  } |

8.2.1 基本的定义和计算

**1. 圆的定义**

用圆心和半径表示圆。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | struct Circle{  Point c; //圆心  double r; //半径  Circle(){}  Circle(Point c,double r):c(c),r(r){}  Circle(double x,double y,double \_r){c=Point(x,y);r = \_r;}  }; |

**2. 点和圆的关系**

点和圆的关系，根据点到圆心的距离判断。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int Point\_circle\_relation(Point p, Circle C){  double dst = Distance(p,C.c);  if(sgn(dst – C.r) < 0) return 0; //0 点在圆内  if(sgn(dst – C.r) ==0) return 1; //1 圆上  return 2; //2 圆外  } |

**3. 直线和圆的关系**

直线和圆的关系，根据圆心到直线的距离判断。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int Line\_circle\_relation(Line v,Circle C){  double dst = Dis\_point\_line(C.c,v);  if(sgn(dst-C.r) < 0) return 0; //0 直线和圆相交  if(sgn(dst-C.r) ==0) return 1; //1 直线和圆相切  return 2; //2 直线在圆外  } |

**4. 线段和圆的关系**

线段和圆的关系，根据圆心到线段的距离判断。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int Seg\_circle\_relation(Segment v,Circle C){  double dst = Dis\_point\_seg(C.c,v);  if(sgn(dst-C.r) < 0) return 0; //0线段在圆内  if(sgn(dst-C.r) ==0) return 1; //1线段和圆相切  return 2; //2线段在圆外  } |

**5. 直线和圆的交点**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | //pa, pb是交点。返回值是交点个数  int Line\_cross\_circle(Line v,Circle C,Point &pa,Point &pb){  if(Line\_circle\_relation(v, C)==2) return 0;//无交点  Point q = Point\_line\_proj(C.c,v); //圆心在直线上的投影点  double d = Dis\_point\_line(C.c,v); //圆心到直线的距离  double k = sqrt(C.r\*C.r-d\*d);  if(sgn(k) == 0){ //1个交点，直线和圆相切  pa = q; pb = q; return 1;  }  Point n=(v.p2-v.p1)/ Len(v.p2-v.p1); //单位向量  pa = q + n\*k; pb = q - n\*k;  return 2; //2个交点  } |

8.2.2 最小圆覆盖

|  |  |  |
| --- | --- | --- |
| 例8-7. 最小圆覆盖 洛谷P1742 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | #include <bits/stdc++.h>  using namespace std;  #define eps 1e-8  const int N = 1e5+1;  int sgn(double x){  if(fabs(x) < eps) return 0;  else return x<0?-1:1;  }  struct Point{ double x, y; };  double Distance(Point A, Point B){return hypot(A.x-B.x,A.y-B.y);}  Point circle\_center(const Point a, const Point b, const Point c){  Point center;  double a1=b.x-a.x, b1=b.y-a.y, c1=(a1\*a1+b1\*b1)/2;  double a2=c.x-a.x, b2=c.y-a.y, c2=(a2\*a2+b2\*b2)/2;  double d =a1\*b2-a2\*b1;  center.x =a.x+(c1\*b2-c2\*b1)/d;  center.y =a.y+(a1\*c2-a2\*c1)/d;  return center;  }  void min\_cover\_circle(Point \*p, int n, Point &c, double &r){  random\_shuffle(p, p + n); //随机函数，打乱所有点。这一步很重要  c=p[0]; r=0; //从第1个点p0开始。圆心为p0，半径为0  for(int i=1;i<n;i++) //扩展所有点  if(sgn(Distance(p[i],c)-r)>0){ //点pi在圆外部  c=p[i]; r=0; //重新设置圆心为pi，半径为0  for(int j=0;j<i;j++) //重新检查前面所有的点。  if(sgn(Distance(p[j],c)-r)>0){ //两点定圆  c.x=(p[i].x + p[j].x)/2;  c.y=(p[i].y + p[j].y)/2;  r=Distance(p[j],c);  for(int k=0;k<j;k++)  if (sgn(Distance(p[k],c)-r)>0){ //两点不能定圆，就三点定圆  c=circle\_center(p[i],p[j],p[k]);  r=Distance(p[i], c);  }  }  }  }  Point p[N];  int main(){  int n; cin >> n;  for(int i=0;i<n;i++) scanf("%lf%lf",&p[i].x,&p[i].y);  Point c; double r; //最小覆盖圆的圆心和半径  min\_cover\_circle(p,n,c,r);  printf("%.10f\n%.10f %.10f\n",r,c.x,c.y);  return 0;  } |

8.3.1 三维点和线

**1. 点和向量**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | struct Point3{ //三维点  double x,y,z;  Point3(){}  Point3(double x,double y,double z):x(x),y(y),z(z){}  Point3 operator + (Point3 B){return Point3(x+B.x,y+B.y,z+B.z);}  Point3 operator – (Point3 B){return Point3(x-B.x,y-B.y,z-B.z);}  Point3 operator \* (double k){return Point3(x\*k,y\*k,z\*k);}  Point3 operator / (double k){return Point3(x/k,y/k,z/k);}  bool operator == (Point3 B){ return sgn(x-B.x)==0 && sgn(y-B.y)==0 && sgn(z-B.z)==0;}  };  typedef Point3 Vector3; //三维向量 |

两点之间的距离：

|  |  |
| --- | --- |
| 1  2  3 | double Distance(Vector3 A,Vector3 B){  return sqrt((A.x-B.x)\*(A.x-B.x)+(A.y-B.y)\*(A.y-B.y)+ (A.z-B.z)\*(A.z-B.z));  } |

**2. 线和线段**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | struct Line3{  Point3 p1,p2;  Line3(){}  Line3(Point3 p1,Point3 p2):p1(p1),p2(p2){}  };  typedef Line3 Segment3; //定义线段，两端点是Point p1,p2 |

8.3.2 三维点积

**1. 三维点积的定义**

求向量A、B点积的代码：

|  |
| --- |
| double Dot(Vector3 A,Vector3 B){return A.x\*B.x+A.y\*B.y+A.z\*B.z;} |

**2. 三维点积的基本应用**

2）求向量A的长度

|  |
| --- |
| double Len(Vector3 A){ return sqrt(Dot(A, A));} |

或者是求长度的平方，避免开方运算：

|  |
| --- |
| double Len2(Vector3 A){ return Dot(A, A);} |

3）求向量A与B的夹角

|  |
| --- |
| double Angle(Vector3 A,Vector3 B){return acos(Dot(A,B)/Len(A)/Len(B));} |

8.3.3 三维叉积

**1. 三维叉积的定义**

|  |  |
| --- | --- |
| 1  2  3 | Vector3 Cross(Vector3 A,Vector3 B){  return Point3(A.y\*B.z-A.z\*B.y, A.z\*B.x-A.x\*B.z, A.x\*B.y-A.y\*B.x);  } |

**2. 三维叉积的基本应用**

1） 求三角形面积

|  |  |
| --- | --- |
| 1  2 | //三角形面积的2倍  double Area2(Point3 A,Point3 B,Point3 C){return Len(Cross(B-A, C-A));} |

2）点和线的有关问题

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | //三维：点在直线上  bool Point\_line\_relation(Point3 p,Line3 v){  return sgn( Len(Cross(v.p1-p,v.p2-p))) == 0 && sgn(Dot(v.p1-p,v.p2-p))== 0;  }  //三维：点到线段距离。  double Dis\_point\_seg(Point3 p, Segment3 v){  if(sgn(Dot(p- v.p1,v.p2-v.p1)) < 0 || sgn(Dot(p- v.p2,v.p1-v.p2)) < 0)  return min(Distance(p,v.p1),Distance(p,v.p2));  return Dis\_point\_line(p,v);  }  //三维：点 p 在直线上的投影  Point3 Point\_line\_proj(Point3 p, Line3 v){  double k=Dot(v.p2-v.p1,p-v.p1)/Len2(v.p2-v.p1);  return v.p1+(v.p2-v.p1)\*k;  } |

3）平面

|  |  |
| --- | --- |
| 1  2  3  4  5 | struct Plane{  Point3 p1,p2,p3; //平面上的三个点  Plane(){}  Plane(Point3 p1,Point3 p2,Point3 p3):p1(p1),p2(p2),p3(p3){}  }; |

4）平面法向量

|  |
| --- |
| oint3 Pvec(Point3 A, Point3 B, Point3 C){return Cross(B-A,C-A);} |

或者：

|  |
| --- |
| Point3 Pvec(Plane f){return Cross(f.p2-f.p1,f.p3-f.p1);} |

5）平面的有关问题

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | //四点共平面  bool Point\_on\_plane(Point3 A,Point3 B,Point3 C,Point3 D){  return sgn(Dot(Pvec(A,B,C),D-A)) == 0;  }  //两平面平行  int Parallel(Plane f1, Plane f2){ return Len(Cross(Pvec(f1),Pvec(f2))) < eps; }  //两平面垂直  int Vertical (Plane f1, Plane f2){ return sgn(Dot(Pvec(f1),Pvec(f2)))==0; } |

6）直线和平面的交点

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | int Line\_cross\_plane(Line3 u,Plane f,Point3 &p){  Point3 v = Pvec(f); //平面的法向量  double x = Dot(v, u.p2-f.p1);  double y = Dot(v, u.p1-f.p1);  double d = x-y;  if(sgn(x) == 0 && sgn(y) == 0) return -1; //-1：v在f上  if(sgn(d) == 0) return 0; //0：v与f平行  p = ((u.p1 \* x)-(u.p2 \* y))/d; //v与f相交  return 1;  } |

7）四面体的有向体积

|  |  |
| --- | --- |
| 1  2  3 | //四面体有向体积\*6  double volume4(Point3 a,Point3 b,Point3 c,Point3 d){  return Dot(Cross(b-a,c-a),d-a); } |

8.3.4 最小球覆盖

|  |
| --- |
| 例8-8. super star poj 2069 |

下面是最小球覆盖的模拟退火代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | #include<algorithm>  #include<cmath>  using namespace std;  const double eps = 1e-7;  struct Point3{double x,y,z;} p[35];  int n;  double Distance(Point3 A,Point3 B){  return sqrt((A.x-B.x)\*(A.x-B.x)+(A.y-B.y)\*(A.y-B.y)+ (A.z-B.z)\*(A.z-B.z));  }  double solve(){  double T=100.0; //初始温度  double delta = 0.98; //降温系数  Point3 c = p[0]; //球心  int pos;  double r; //半径  while(T>eps) { //eps是终止温度  pos = 0; r = 0; //初始：p[0]是球心，半径是0  for(int i=0; i<n; i++) //迭代：找距离球心最远的点  if(Distance(c,p[i])>r){  r = Distance(c,p[i]); //距离球心最远的点肯定在球周上  pos = i;  }  c.x += (p[pos].x-c.x)/r\*T; //逼近最后的解  c.y += (p[pos].y-c.y)/r\*T;  c.z += (p[pos].z-c.z)/r\*T;  T \*= delta; //降温  }  return r;  }  int main(){  double ans;  while(~scanf("%d",&n),n) {  for(int i=0;i<n;i++) scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);  ans = solve();  printf("%.5f\n",ans);  }  return 0;  } |

8.3.6 三维几何例题

**1. 化球为圆**

|  |  |  |
| --- | --- | --- |
| 例8-9. Ghost Busters poj 2177 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87 | #include <cmath>  #include <vector>  #include <algorithm>  using namespace std;  const int N=105;  const double eps = 1e-7;  int sgn(double x){ //判断x是否等于0  if(fabs(x)<eps) return 0;  else return x<0?-1:1;  }  struct Point3{  double x,y,z;  Point3(){}  Point3(double x,double y,double z):x(x),y(y),z(z){}  Point3 operator + (Point3 B){return Point3(x+B.x,y+B.y,z+B.z);}  Point3 operator - (Point3 B){return Point3(x-B.x,y-B.y,z-B.z);}  Point3 operator \* (double k){return Point3(x\*k,y\*k,z\*k);}  Point3 operator / (double k){return Point3(x/k,y/k,z/k);}  Point3 adjust (double L){ //调整长度  double len=sqrt(x\*x+y\*y+z\*z);  L/=len;  return Point3(x\*L,y\*L,z\*L);  }  };  typedef Point3 Vector3;  double Dot(Vector3 A,Vector3 B){return A.x\*B.x+A.y\*B.y+A.z\*B.z;} //点积  Vector3 Cross(Vector3 A,Vector3 B)  { return Point3(A.y\*B.z-A.z\*B.y,A.z\*B.x-A.x\*B.z,A.x\*B.y-A.y\*B.x);} //叉积  double Len(Vector3 A){return sqrt(Dot(A,A));} //向量的长度  double Distance(Point3 A,Point3 B) //两点的距离  { return sqrt((A.x-B.x)\*(A.x-B.x)+(A.y-B.y)\*(A.y-B.y)+(A.z-B.z)\*(A.z-B.z)); }  struct Line3{ //三维：线  Point3 p1,p2;  Line3(){}  Line3(Point3 p1,Point3 p2):p1(p1),p2(p2){}  };  double Dis\_point\_line(Point3 p, Line3 v) //三维：点到直线距离  { return Len(Cross(v.p2-v.p1,p-v.p1))/Distance(v.p1,v.p2); }  vector<Point3> P; //储存球的交点  void intersect(Point3 c1,double r1,Point3 c2,double r2){ //计算球的交点  double d1=Len(c1),d2=Len(c2); //球心到原点距离  c2=c2/d2\*d1; //调整球2，让两个球心与原点等距  r2=r2/d2\*d1;  double d=Len(c1-c2); //连心线长度  if(sgn(d-r1-r2)==0){P.push\_back(c1+(c2-c1)/d\*r1);return;} //(1)相切，存相切的点  if(sgn(d-r1-r2)>0) return; //(2)相离，没有交点  if(sgn(d-fabs(r1-r2))<=0) return; //(3)包含，没有交点  //(4)下面处理两个圆相交，有两个交点  double b=(r1\*r1+d\*d-r2\*r2)/(2\*d); //余弦定理  double h =sqrt(r1\*r1-b\*b);  Point3 M=c1+(c2-c1)/d\*b; //两交点中点所在位置  Point3 v=Cross(c1,M); //叉积求得两交点所在直线的向量  v=v.adjust(h)+M;  P.push\_back(v);  P.push\_back(M\*2-v);  }  int check(Point3 p,Point3 c,double r){ //检查交点p是否在球内或球面上  Line3 v(Point3(0,0,0),p);  double x = Dis\_point\_line(c,v);  return sgn(x-r) <= 0;  }  Point3 c[N]; //球心  double r[N]; //球半径  int main (){  int n; scanf("%d",&n);  for(int i=1;i<=n;i++) {  scanf("%lf%lf%lf",&(c[i].x),&(c[i].y),&(c[i].z));  scanf("%lf",&r[i]);  P.push\_back(c[i]);  }  for(int i=1;i<=n;i++) //求任意两个球的交点，记录在vector P中  for(int j=i+1;j<=n;j++)  intersect(c[i],r[i],c[j],r[j]);  int ans=0,temp,w; //w记录选中的交点  for(int i=0;i<P.size();i++) { //检查每条射线，记录穿过最多球体的数量  temp=0;  for(int j=1;j<=n;j++) temp += check(P[i],c[j],r[j]);  if(temp>ans) ans=temp, w=i;  }  printf("%d\n",ans);  for(int i=1;i<=n && ans;i++) //枚举所有球，判断与选定的最佳射线是否相交  if(check(P[w],c[i],r[i])) {  ans--;  printf(ans!=0?"%d ":"%d\n",i);  }  return 0;  } |

**2. 仿射变换**

|  |  |  |
| --- | --- | --- |
| 例8-10. A Letter to Programmers https://vjudge.net/problem/UVALive-5719 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 | //改写自：www.cppblog.com/hanfei19910905/archive/2012/06/24/180035.html  #include<bits/stdc++.h>  using namespace std;  const double eps = 1e-6;  const double pi = acos(-1.0); //圆周率，精确到15位小数：3.141592653589793  const int N = 4;  struct matrix {  double num[N][N];  matrix (double a){ //单位矩阵乘以a  memset(num,0,sizeof(num));  for(int i=0;i<N;i++) num[i][i] = a;  }  matrix(double x,double y,double z){ //平移变换  memset(num,0,sizeof(num));  for(int i=0;i<N;i++) num[i][i] = 1; //单位矩阵  num[3][0] = x; num[3][1] = y; num[3][2] = z;  }  matrix(double x,double y,double z, int X){ //缩放变换  memset(num,0,sizeof(num));  for(int i=0;i<N;i++) num[i][i] = 1; //单位矩阵  num[0][0] = x; num[1][1] = y; num[2][2] = z;  }  matrix(double P[3],double ang){ //旋转变换  memset(num,0,sizeof(num));  for(int i=0;i<N;i++) num[i][i] = 1; //单位矩阵  double flag [3][3] = {0,1,-1, -1,0,1, 1,-1,0};  double sum = P[0] + P[1] + P[2];  for(int i=0;i<3;i++)  for(int j=0;j<3;j++)  if(i==j) num[i][j] = P[i]\*P[i] + (1-P[i]\*P[i]) \* cos(ang);  else num[i][j] = P[i]\*P[j]\*(1-cos(ang))+(sum-P[i]-P[j])\*sin(ang)\*flag[i][j];  }  };  matrix operator \* (const matrix& a, const matrix& b){ //普通矩阵乘法。注意const  matrix c(0);  for(int i=0; i<N; i++)  for(int j=0; j<N; j++)  for(int k = 0; k<N; k++)  c.num[i][j] += a.num[i][k] \* b.num[k][j];  return c;  }  matrix pow\_matrix(matrix a, int n){ //普通矩阵快速幂  matrix ans(1);  while(n) {  if(n&1) ans = ans \* a;  a = a \* a;  n>>=1;  }  return ans;  }  matrix dfs(){  matrix ans(1);  while(1){  string cmd; cin >> cmd;  if(cmd=="end") return ans;  if(cmd=="repeat"){  int k; scanf("%d",&k);  matrix temp = dfs();  ans = ans \* pow\_matrix(temp, k);  }  else {  double x,y,z; scanf("%lf%lf%lf",&x,&y,&z);  if(cmd == "translate") {matrix temp(x,y,z); ans=ans\*temp;}  else if(cmd == "scale"){matrix temp(x,y,z,0); ans=ans\*temp;}  else if(cmd == "rotate"){  double a; scanf("%lf",&a);  a = a/180\*pi;  double sum = sqrt(x\*x + y\*y +z\*z);  double p[3] = {x/sum, y/sum, z/sum};  matrix temp(p,a);  ans = ans \* temp;  }  }  }  }  int main(){  int n;  while(~scanf("%d",&n) && n){  matrix t = dfs();  while(n--){  double x,y,z,px,py,pz; scanf("%lf%lf%lf",&x,&y,&z);  px = x\*t.num[0][0] + y\*t.num[1][0] + z\*t.num[2][0] + t.num[3][0];  py = x\*t.num[0][1] + y\*t.num[1][1] + z\*t.num[2][1] + t.num[3][1];  pz = x\*t.num[0][2] + y\*t.num[1][2] + z\*t.num[2][2] + t.num[3][2];  printf("%.2f %.2f %.2f\n",px+eps,py+eps,pz+eps);  }  puts("");  }  } |