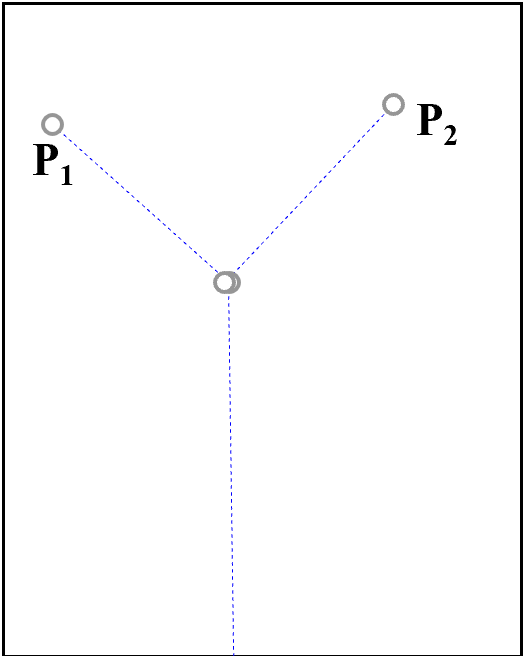
# 关于3D打印中的支撑问题求解

## 问题

### 问题：三维空间点到平面的最短支撑路径

**在三维空间中有互不重合的n个点Pi(其中i=0, 1, …, n-1)。 Pi的Z坐标值均大于0。定义有效支撑线段为与竖直方向夹角不大于a度的线段，其中a为在[0, 90)内的常数。请添加有效支撑线段使得每个点Pi通过有效支撑线段与Z=0平面连通。要求有效支撑线段总长度最小。**



### 子问题：考虑n=2的情况和n=3情况；

贡献认定: 由周海刚提供。

## 定义

### 定义一: 支撑圆锥

具体定义: 对于一个点P，考虑和它的连线与竖直方向夹角不大于a的，纵坐标大于0，小于P的纵坐标的点的集合称为点P的“角度a支撑圆锥”，当题给的a固定时，简记为“P的支撑圆锥”

贡献认定: 由周海刚提供。

### 定义二：结点，原始点，末端点

具体定义：我们称在一种支撑方案当中，两条支撑线段的交点称为一个“结点”，题给的一开始的n个点为“原始点”，若一个点只引出一条支撑线段，就称之为末端点；

贡献认定: 由周海刚提供。

### 定义三：支撑线段的“引出”

具体定义，我们称一条支撑线段AB由A引出，当且仅当A比B的纵坐标要大

贡献认定: 由周海刚提供。

### 定义四：支撑方案的对应图

具体定义：我们考虑一个有向图G，我们称其为一个支撑方案的对应图，如果在G当中每个点对应一个支撑方案中一个点，G中每条边对应支撑方案中一条支撑线段，且方向对应每一条支撑线段的引出方向

贡献认定: 由周海刚提供。

## 注：我们有了定义三和定义四，我们就会发现题目的不严谨之处，比如：我们对于以下的情况：

## 图像中A通过三条支撑线段AC、BC、以及B引出的向地面的垂直线段和地面相连，其中AB是原始点，C为产生的结点；但是这显然不符合3D打印当中的实际情况；因此我们可以对题目如此修改：

*对于每个原始点，在支撑方案当中必存在一条****有向路****使得它和地面相连，这一条路称之为A的支撑路*

特别的，如果一个原始点引出了一条支撑路，那么我们就称它是“被支撑的”

## 公式

### 公式一：

结论部分:

贡献认定: 由x提供。

证明部分

具体证明。



贡献认定: 由x提供。

## 定理

### 定理一：结点最少性原理

定理内容：如果一个支撑方案是最优的，那么不是原始点的结点处不可能只被一条支撑直线引向和引出一条支撑线段（我们把共线的两条支撑线段视为同一条）

定理的证明：如果在一个非原始点的结点处被引出和引出各一条支撑线段：



如图，AB，BC是两条支撑线段，但是B非原始点，我们来证明AC也是一条支撑线段，从而我们如果将AB，BC两条支撑线段换为AC，那么所有支撑线段长度之和减少了AB+BC，增加了AC，而由三角形的两边大于第三边，我们可以知道AC<AB+BC，从而支撑线段的长度和减小了，即我们得到了更好的方案，这和我们之前的假设矛盾；

我们设A，B两点引向地面的竖直直线分别为*l1，l2*，设B在*l1*的投影为D，C在*l1，l2*上的投影是E，F，我们有矩形的定义可以知道BDEF为一个矩形，从而EF=BD，FB=DE；

由于AB，BC为两条支撑线段，我们可以知道∠BAD，

∠CBF都不大于a，结合tan（x）在（0，）的单调递增性，我们就得到了：

即

从而：

结合BD=EF，BF=DE以及三角形当中的两边大于第三边

从而：

再由tan x的单调性，我们就得到了∠CAE< a，从而AC是一条支撑线段，定理得证；

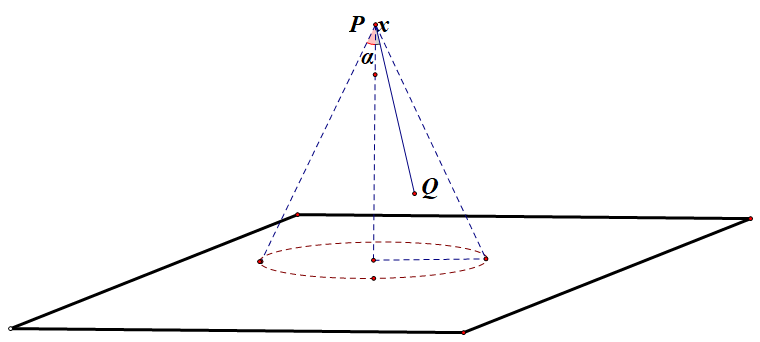
贡献认定: 由周海刚提供。

### 定理二: 对于点P，它的“支撑圆锥”是一个圆锥

结论部分： 对于纵坐标大于0的点Q，若PQ算作一条支撑线段，那么Q的集合便是一个以P为顶点的圆锥

贡献认定: 由周海刚提供。

证明部分：



设过P的竖直直线是*l*；我们找到和P连线与竖直方向夹角为a的点的集合，他们的集合恰是一条和以P为顶点，母线和高的夹角为a的圆锥的表面（通过旋转可知）；

而我们考虑Q点：在PQ和*l*形成的平面内（此处将Q点在P正下方的情况排除，因为在这种情况下，Q点确实在这个圆锥内）由于PQ和这个圆锥的高的夹角小于a，考虑这个平面内的圆锥的母线*t*和*s*,从而有PQ在*t*和*s*之间，从而我们可以知道Q点在两条母线的夹角内部，从而Q在圆锥内部；

另一方面，对于圆锥内部的任意一点Q，在PQ和*l*所形成的平面内，处于两条在这个平面内的两条母线之间，从而和这两条母线的夹角的角平分线的夹角小于两条母线夹角的一半，即a；

从而我们从两方面证明了“P的支撑圆锥”确实是一个以P为顶点圆锥，并且其母线和高的夹角是a；

贡献认定: 由周海刚提供。

### 定理三：支撑路最少原理

定理内容：在最优的支撑方案下，每个原始点引出的支撑路只有一条

定理证明：如果一个原始点P引出了至少两条支撑路，我们证明可以删除一些支撑线段，从而使得支撑线段的长度之和减少：我们考虑两条支撑路第一次分开的结点（有可能是原始点本身），那么这个结点引出了两条支撑线段，我们选取其中一条删去，我们由结点最少性原理可以知道这条支撑线段和底面相连或者另一个端点引出了其它的支撑线段；

总之，将这条支撑线段删去之后不会影响P“被支撑”的性质，而且也不会影响其他点的支撑性质。

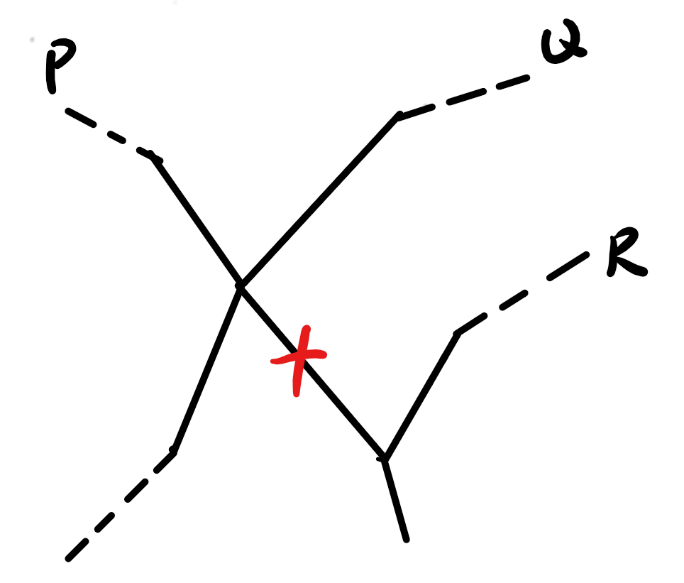
这是由于，在这条线段的引出顶点方面，如果这个线段的删去使得原来有的另外的一个原始点Q的支撑路消失了，我们可以让这个原始点的支撑路在这个结点处将原来的路径变为：在这个结点处的P的另外一条支撑路的剩余路线，那么这也是一条支撑路，并不会影响Q“被支撑”的性质；

而在删去的支撑线段的被引出端点处，我们注意这个端点是被引出的，从而也不会造成一条支撑路受影响的结果；

于是我们得到了：如果一个支撑方案当中有一个原始点引出了两条支撑路，那么我们可以通过删去其中一条支撑线段使得这个方案变得更加优秀，从反面来说也就是，如果一种方案是最优的，那么它的每个原始点都引出恰一条支撑路；

推论：由上述定理的证明，我们可以得出以下的结论：

在最优方案当中，结点引出的支撑线段不超过一条；

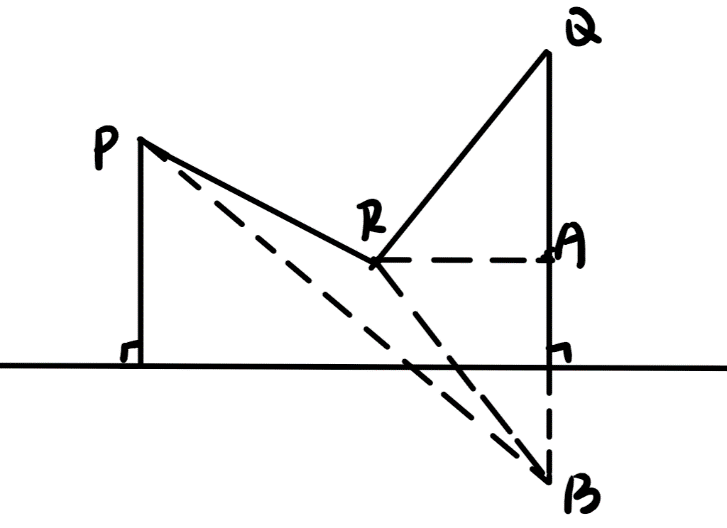
推论的证明：

事实上，如果有一个结点引出的支撑线段有两条，他们都是最后连接到地面上的，所以删除其中的一条，不会影响连接到这个结点的任意一个原始点（他们的支撑路可以通过另外一个支撑线段连接到地面上，因此我们重复这样的删除过程就可以到达每个结点引出的支撑线段不超过一条的近似最优情况。推论证明完毕。

贡献认定: 由周海刚提供。

### 定理四：Y字长度和最短判定定理

定理内容：在原先的n=2情况下，限制支撑方案为一个平面上的“Y”形（即一个结点R，两个原始点分别和这个节点相连，并且这个结点直接垂直引向地面；当然这个限制需要两个支撑圆锥有交点，否则这种Y的方案不可能存在），那么最优方案满足：

1. 两个原始点和这个结点的连线和竖直方向的夹角是相等的
2. 的取值会在下面的证明当中给出；

1. 我们考虑R在Q引向地面的垂线上的投影A，设B为Q关于点A的对称点，那么B在这条垂线上并且有：

（对称性）

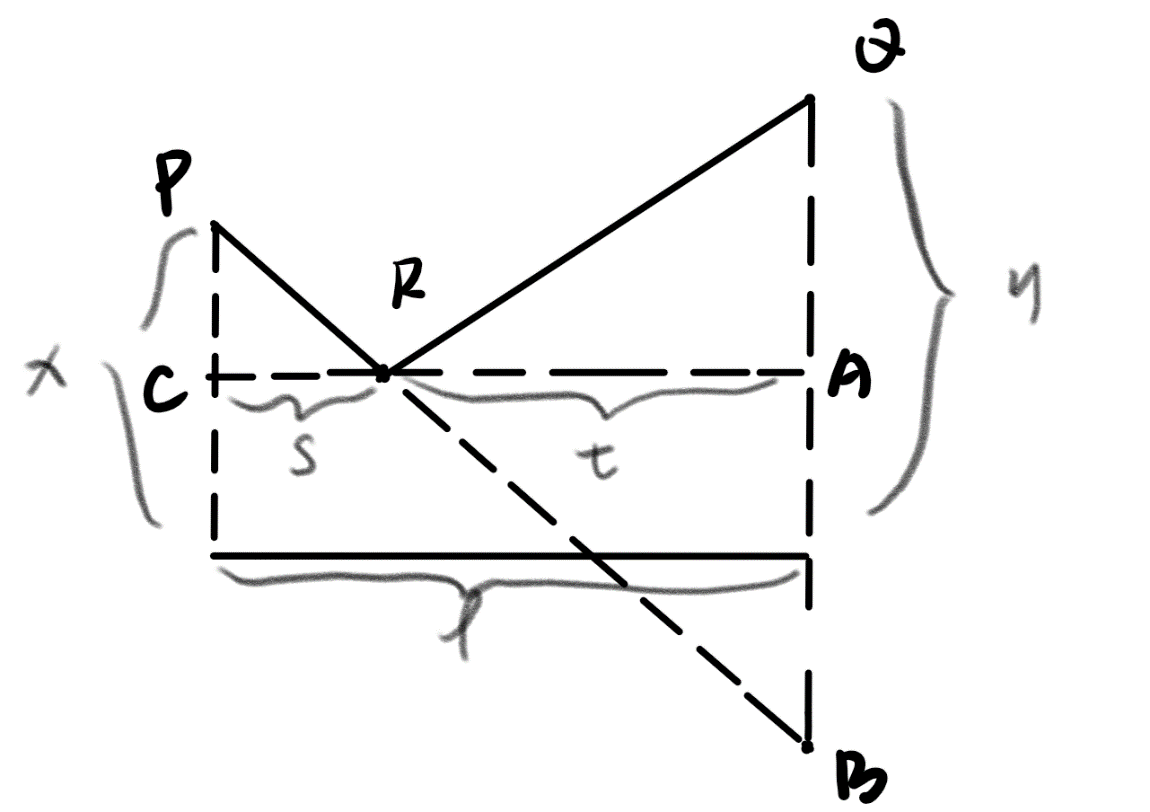
而在三角形（可能是退化的）PRB内由两边之和不小于第三边：

故

（联立）

所以在固定R点纵坐标不动的情况下，即类似于将R在水平方向“左右移动”，所得的不小于PB，此时PB与R在水平上的移动没有关系，从而的最小值为PB，且当且仅当R在PB上的时候取到，此时由对称性有：

∠RQB=∠RBQ

注意到由P引向地面的垂线和QB垂直，故有：PR，RQ和竖直方向的夹角相等，第一部分证毕；

1. 如题干中定义，接下来我们将探求当取何值的时候，所求的Y形支撑方案的总长度最小

记B点同1）证明中的定义，R分别在P，Q引出的地面的垂线上的投影是C，A，*l*为PQ在地面上的投影的距离，s=RC，t=RA，x为P的纵坐标，y为Q的纵坐标，我们先来求解s和t

由∠RPC和∠RQA等于可以知道：

而我们注意到R的纵坐标可以有两种方式表示（左右两边表示）:

x-PC=y-QA>0

联立，有：

而我们又有

从而将上述两式相加相减可以得到

此时我们可以求得Y形的总长度为：

PR+RQ+R的纵坐标，即：

注意此时需要满足R纵坐标不小于零，即

不小于0，所以满足：

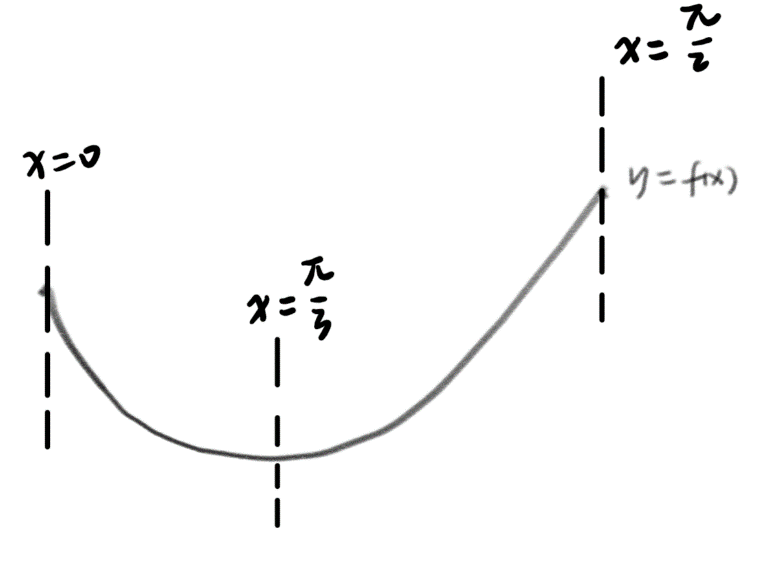
同时由于倾斜角的限制，

以及R在PQ两点之间的限制知道：

即

记b=min{}

我们考虑对于求导：

在限定在（0，）的时候（这是平凡的限制区间）有且只有一个根=，当大于这个值时大于零，当小于这个值时小于零，所以的图像草图如下所示：

那么我们就得到了在整个（0，）区间上的最小值在时取到，接下来需要讨论的取值范围

注意到θ的存在性（定理先设条件）可以得到

分三种情况：

第一种：

此时 最小值在时取到，这个最小值为

第二种：b<,此时*f*的最小值为b的时候取到，此时最小值：



第三种：>,此时的最小值为：

定理四的推论：

由上述证明可以知道对于一个给定的，我们可以给出所得到的R的坐标。

R在水平方向的坐标是可以由

给出，此即：

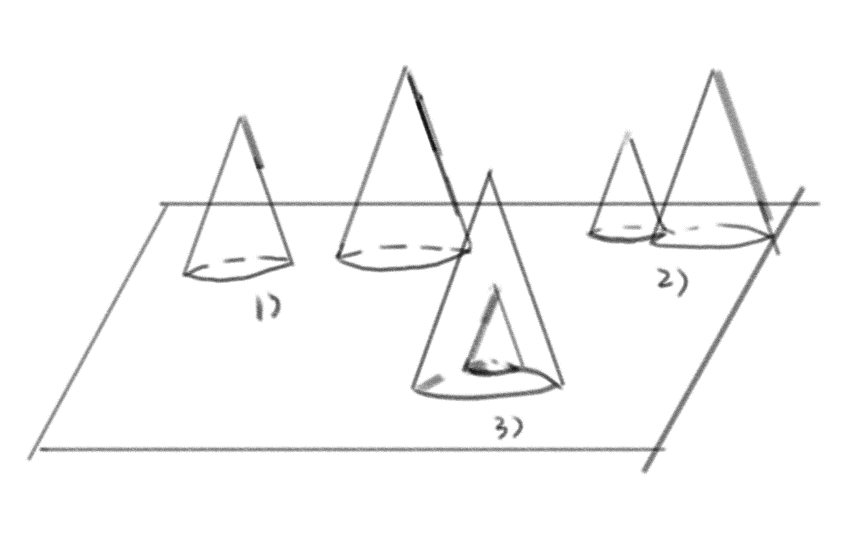
其中

R的纵坐标为

其中x，y为P，Q的纵坐标，即：

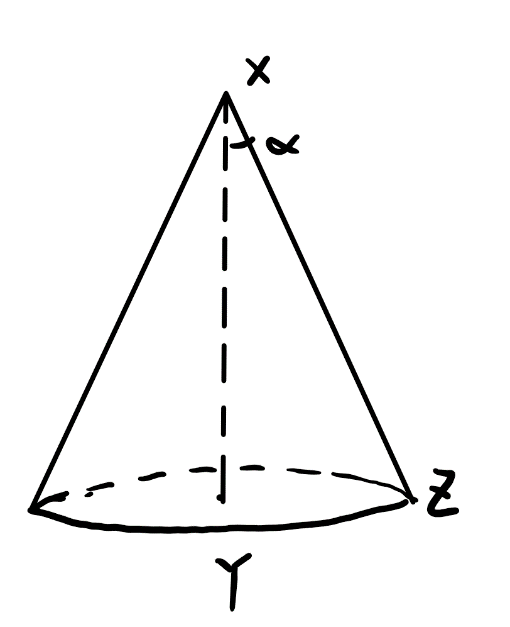
贡献认定: 由陈哲涵提供。

### 定理五：两个点P，Q的支撑圆锥的关系判定定理

定理的探究及内容：我们注意到两个支撑圆锥相交当且仅当两个圆锥的底面相交，两个圆锥有包含关系当且仅当两个底面包含；两个支撑圆锥相离当且仅当两个底面相离等等；

所以我们先来计算两个圆锥底面的半径，从而来讨论两个圆锥之间的关系。

对于两个圆锥中的任意一个（如图）

其中X为圆锥的顶点，XY为圆锥的高，Z点是底面边缘的一点。我们由高的定义，可以知道XY垂直于YZ，我们由三角函数可以知道底面的半径（YZ的长度）为

（其中z表示纵坐标）

而我们来观察两个圆的位置关系，只有相交，相离，包含，相切（这里只画出外切，内切同理）的关系，分别如下图所示：



可以看到，其中两个圆没有公共部分的情况有且只有相离，而在这种情况下，两个圆心的距离大于两个半径之和;

两个圆外切的情况下，两个圆心之间的距离为半径之和；

两个圆相交的时候，两个圆心之间的距离小于半径之和但是大于半径之差的绝对值（由图中的三角形当中的不等式可以知道）；

两个圆相内切的情况下，两个圆心之间的距离恰好等于两个半径之差的绝对值；

两个圆相包含的情况下，两个圆心之间的距离小于两个半径之差的绝对值；注意到顶点所引到地面的高的端点的x，y坐标是相同的，从而我们就得到了如下的判定定理：

I）若有：（其中P’，Q’为两个点在水平面的投影）

那么两个底面相离，即两圆锥相离；

II）若有：

那么两个底面相外切，即两个圆锥有且只有一个公共点；

III）若有：

那么两个底面相交，即两个圆锥有公共点但是不包含；

IV）若有：

那么两个底面相内切，即两个圆锥有内切关系；

V）若有

那么两个底面有包含关系，即两个圆锥有包含关系；

## 算法

### 算法一: 两个点的最短支撑算法（算法成熟度:最优算法）

**算法功能简述**: 如果P,Q两个点的支撑圆锥不相交，那么最短方案即为P，Q两点各自垂直引向地面；如果两个支撑圆锥相交，那么最优方案可以由定理四给出；

**算法输入**: *a*是支撑角度；

*（x1,y1,z1），（x2,y2,z2）­*是两个点P和Q坐标值

**算法输出**: 一种支撑方式

*数对三元数组(x,y,z) (a,b,c)表示一条支撑线段*

*支撑线段总长度*

**算法步骤开始**:

**步骤(1)**: 验算两个点的支撑圆锥的关系

**步骤(2)**: 若两个圆锥不相交，进入步骤2.1；若两个圆锥相交但是不包含（包括了外切的情况但是没有包括内切情况），进入2.2；若两个圆锥有包含关系（包括了内切情况），进入2.3

**步骤(2.1)**: 直接输出两个点分别垂直引向地面的线段，并计算并输出两个线段长度之和

**步骤(2.2)**:利用定理四找到“Y”形的方案中的最优方案与直接从两个原始点垂直引向地面的方案作比较，选取其中更好的方案输出；

**步骤(2.3)**: 直接输出两个点的连线的线段，以及较下方的点垂直引向地面的线段，并计算以及输出两个线段长度之和

**算法步骤结束。**

贡献认定: 由周海刚提供。

（算法的一些必要说明： 输出的线段由三元组的二元对表示，每次输出完线段换行。）

#### 算法原理

结论部分： 在只有两个点的情况下这种算法是最优的情况，也就是说，支撑线段的长度之和是最小的

证明部分： 我们分两个部分进行证明：

第一部分：两个支撑圆锥没有公共部分

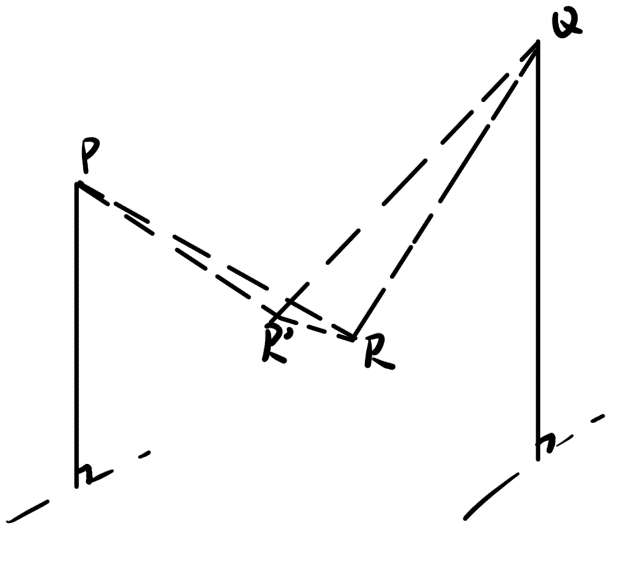
如果两个原始点P，Q在方案的对应图中相通，我们找到两个点支撑路的最早的公共点R（由相通可以知道R的存在性），我们由结点最小性原理可以知道PR,QR为两条支撑线段，从而R点在两个点的支撑圆锥内，而两个圆锥没有公共点，矛盾；故两个原始点不可能相通，故而两条支撑路不可能有公共点；而由PQ中任意一点引出的支撑路最短长度即为引向地面的竖直支撑线段，从而最佳方案为两个原始点各自引向地面的垂直支撑线段。

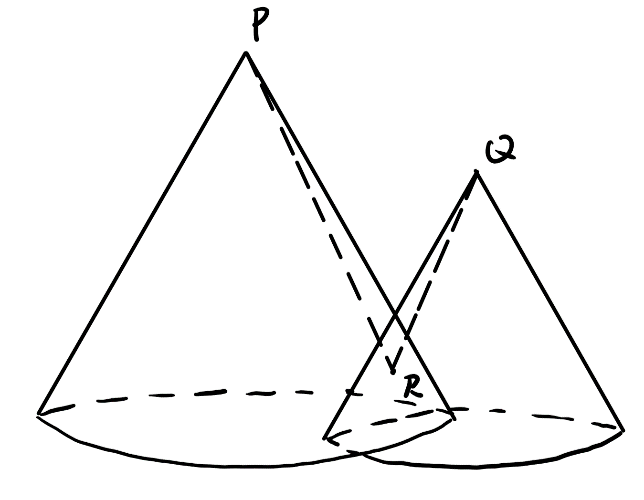
第二部分：两个圆锥有公共部分

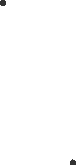
如果构造出来的支撑方案没有结点，那么最优的解决方案便为直接引向地面的垂直支撑线段；

如果构造出来的支撑方案存在结点R，那么由结点最小性原理我们可以知道PR，QR是两条支撑线段，即R在两个支撑圆锥的公共部分当中，而且R以下直接垂直引向地面（在R以下没有结点了，这是由支撑路最少性原理保证的）,我们接下来来讨论此时如果有最优方案，他应该是什么样子。

首先我们注意到在n=2的情况下，P，Q两个点的支撑路在R点之后直接就连向了地面（这是由结点最少性原理以及支撑路最少性原理保证的），所以在R点之下的最优支撑线段不管在什么情况下都是取直接垂直引向地面的支撑线段；有了这一层保证，我们考虑以下的调整，我们将R平移到R’，R之下还是直接垂直于地面的支撑线段，从而将支撑线段PR，QR变成了PR’，QR’，我们先来说明这两个也是支撑线段；



事实上，我们从支撑圆锥的角度来考虑：



由于R在两个支撑圆锥的交叉部分，注意到两个支撑圆锥的高所在的共同平面为交叉部分的对称轴，我们将R投影到这个平面上得到的R’必然也在交叉部分之内，从而我们由支撑圆锥的定义可以知道，PR’，PQ’都是支撑线段；

我们再证明，这种平移可以使得方案变得更优；

事实上由直角三角形内斜边大于直角边，注意到RR’垂直于对称平面，所以三角形RR’P和RR’Q都是直角三角形，同时我们得到：

PR’<PR； QR’<QR

从而此时得到的方案，支撑线段的总长度减少了（此处注意到R以下的支撑线段还是垂直于地面的支撑线段，从而长度仍然是R的纵坐标，这是由两个高所在平面垂直于地面保证的）从而原来包含R的方案不是最优方案，也就是说：

如果包含R的支撑方案是最优方案，那么R一定会在两个支撑圆锥的高的公共平面上；

第二方面我们考虑到定理四的存在，所以可以根据定理四来确定所需要的最好的方案，所输出的答案可以由所给的定理四的推论得出；

贡献认定: 由周海刚提供。

#### 算法复杂度分析

结论部分： x。

贡献认定: 由x提供。

证明部分： x。

贡献认定: 由x提供。

#### 算法理论验证

结论部分： 上述算法得出来的支撑方案确实符合“支撑角度”的限制。

贡献认定: 由周海刚提供。

证明部分： 分两种情况进行讨论：

第一种：两个点的支撑圆锥并不相交

此时得到的支撑方案为垂直的“柱子方案”，和竖直方向的夹角为0，从而是支撑线段；

第二种：两个点的支撑圆锥相交或外切

此时得到的方案为“Y字形方案”或者“柱子方案”，后者由之前的知道符合要求，我们考虑前者：其中“Y”的上面两条边由于是支撑圆锥的两条母线上的一部分，故和竖直方向的倾角为a，符合支撑线段的要求；下部分为竖直的线段，自然也是支撑线段；

贡献认定: 由周海刚提供。

算法使用Python3实现，在Python 3.7.3环境下运行。

import math

def two\_length(a1,b1,c1,a2,b2,c2):

return round(math.sqrt((a1-a2)\*\*2+(b1-b2)\*\*2+(c1-c2)\*\*2),3)

def two\_angle(a3,b3,c3):

return abs(math.acos(c3/math.sqrt(a3\*\*2+b3\*\*2+c3\*\*2)))

def check\_intersection():

if(r\_sum<center\_dis):#外离

return 0

if(r\_sum==center\_dis):#外切

return 1

if(r\_sum>center\_dis and center\_dis>r\_minus):#相交

return 1

if(r\_minus==center\_dis):#内切

return 1

if(r\_minus>center\_dis):#包含

return 2

def calculate\_a():

lenth=z1+z2

return lenth

def calculate\_b():

lenth=z3+two\_length(x1,y1,z1,x3,y3,z3)+two\_length(x2,y2,z2,x3,y3,z3)

return lenth

def calculate\_c():

if(two\_angle(x1-x2,y1-y2,z1-z2)>a):

return 0

if (z1>z2):

lenth=two\_length(x1,y1,z1,x2,y2,z2)+z2

else:

lenth=two\_length(x1,y1,z1,x2,y2,z2)+z1

return lenth

a=float(input("Angle:"))

a=math.radians(a)

x1,y1,z1=map(float,input("P:").split())

x2,y2,z2=map(float,input("Q:").split())

if(a==0):

lenth\_a=calculate\_a()

if(x1==x2 and y1==y2):

print(x1,y1,z1,x2,y2,z2)

if(z1<z2):

if(z1):print(x1,y1,z1,x1,y1,0)

print(z2)

else:

if(z2):print(x2,y2,z2,x2,y2,0)

print(z1)

else:

if(z1):print(x1,y1,z1,x1,y1,0)

if(z2):print(x2,y2,z2,x2,y2,0)

print(lenth\_a)

else:

if(a > math.pi/3):

a = math.pi/3

tan\_a=math.tan(a)

tan\_a=round(tan\_a,3)

r1=z1\*tan\_a

r2=z2\*tan\_a

r\_sum=r1+r2

r\_minus=abs(r1-r2)

center\_dis=math.sqrt((x1-x2)\*\*2+(y1-y2)\*\*2)

r1=round(r1,3)

r2=round(r2,3)

center\_dis=round(center\_dis,3)

circle\_type=check\_intersection()

lenth\_a=calculate\_a() #分别垂直引向地面

lenth\_b=lenth\_a

if(circle\_type==1):

z3=(z1+z2-center\_dis/tan\_a)/2

r1\_new=r1-z3\*tan\_a

r2\_new=r2-z3\*tan\_a

x3=x1+r1\_new\*(x2-x1)/(r1\_new+r2\_new)

y3=y1+r1\_new\*(y2-y1)/(r1\_new+r2\_new)

y3 = round(y3,3)

x3 = round(x3,3)

z3 = round(z3,3)

lenth\_b=calculate\_b() #三叉树

lenth\_c=calculate\_c() #高点连结低点再引向地面

if(lenth\_c == 0):

lenth\_c = lenth\_a

if(lenth\_a<=lenth\_b and lenth\_a<=lenth\_c):

if(z1):print(x1,y1,z1,x1,y1,0)

if(z2):print(x2,y2,z2,x2,y2,0)

print(lenth\_a)

elif(lenth\_b<=lenth\_a and lenth\_b<=lenth\_c):

print(x1,y1,z1,x3,y3,z3)

print(x2,y2,z2,x3,y3,z3)

if(z3):print(x3,y3,z3,x3,y3,0)

print(lenth\_b)

else:

print(x1,y1,z1,x2,y2,z2)

if(z1<=z2):

if(z1):print(x1,y1,z1,x1,y1,0)

else:

if(z2):print(x2,y2,z2,x2,y2,0)

print(lenth\_c)

贡献认定: 由汪子涵提供。

#### 算法数值验证正例: 标题

结论部分： x。

贡献认定: 由x提供。

证明部分： x。

贡献认定: 由x提供。

#### 算法数值验证反例: 标题

注：由多次数值验证以及理论上的证明，并没有数值反例的存在，故该部分略过；

贡献认定: 由周海刚提供。

## N=3的情况的讨论

因为不会排版了所以将n=3的问题放在n=2的后面进行讨论；在讨论n=3的时候我们先要给出另外一些定义，这些定义是前面没有用到的，在这里给出：

1. 定义
   * 1. 称上述给出的关于n=2的情况的算法称为算法1;

贡献认定: 由周海刚提供。

* + 1. 若干个点的凸包

定义内容：我们称有限个点的凸包为能够包含这些点的最小凸多边形。

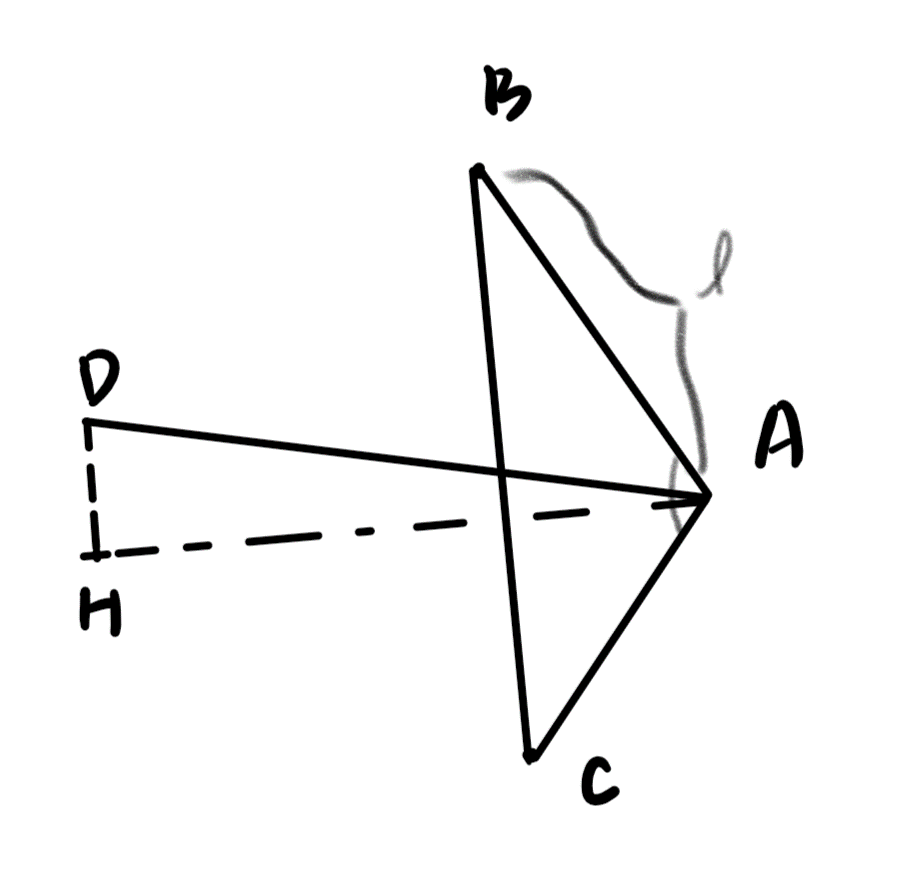
贡献认定: 由周海刚提供。

1. 公式
2. 定理
   1. 最优方案的节点限制定理

定理内容：在一种支撑方案当中，我们将所有的结点，原始点都在水平面上投影，那么所有的结点都不在这个水平面的凸包的顶点上，也就是说，所有的结点都在原始点的凸包内，因此我们命名为“限制定理”

定理证明：如果存在结点为所有的投影的凸包的顶点，那么它处于原始点的投影的凸包外部，我们将其向凸包内某一个水平方向移动一小段距离，我们来证明，支撑线段的长度减少了；

首先由于我们没有改变纵坐标的值，考虑到在空间当中两个点的欧氏距离为各坐标的差的平方和的开方，所以我们只需要证明每一个支撑线段在水平方向上的投影长度减少了（这便是x，y两坐标的差的平方的和的开方），也就是我们只需要考虑平面上的问题；



考虑这个结点A所被引出和被引出的诸支撑线段，考虑他们当中夹角最大的（由于这个点在凸包上那么这个最大角也会是不超过平角的角），考虑这个角的另外两个顶点B和C，我们将A向垂直于BC的方向移动我们证明每一条和A相连的支撑线段AD长度都会减少，事实上，从D向A移动的方向作垂线，垂足为H，那么AH自然会减少，而DH不变，所以直角三角形ADH的斜边AD也会减少。

我们重复这种操作，将A移动到BC边上；从而A点不再是凸包的一个顶点；我们重复这样的操作便可以知道，在最优情况下，所有的结点在原始点的投影的凸包内部，原定理证毕；

贡献认定: 由周海刚提供。

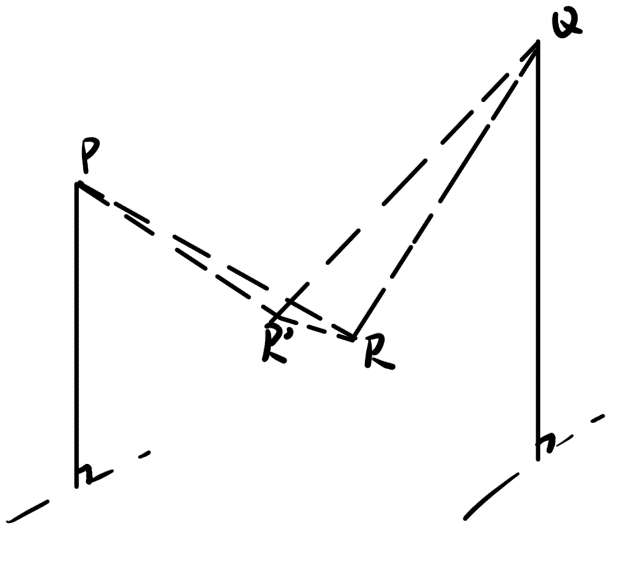
* 1. 两个点的最短支撑线段相交定理

定理内容：我们将在这个定理的讨论当中得出对于两个点PQ，用两条支撑线段使得他们相连的最短长度和的方案必然是使得结点一定是在两个支撑圆锥有公共部分但是不相包含的条件下的两个圆锥相交部分的最高点，或者是在包含关系条件下的较低的圆锥顶点处（首先，为了存在两个支撑线段相交的结点，两个点的支撑圆锥必然是有公共部分的）

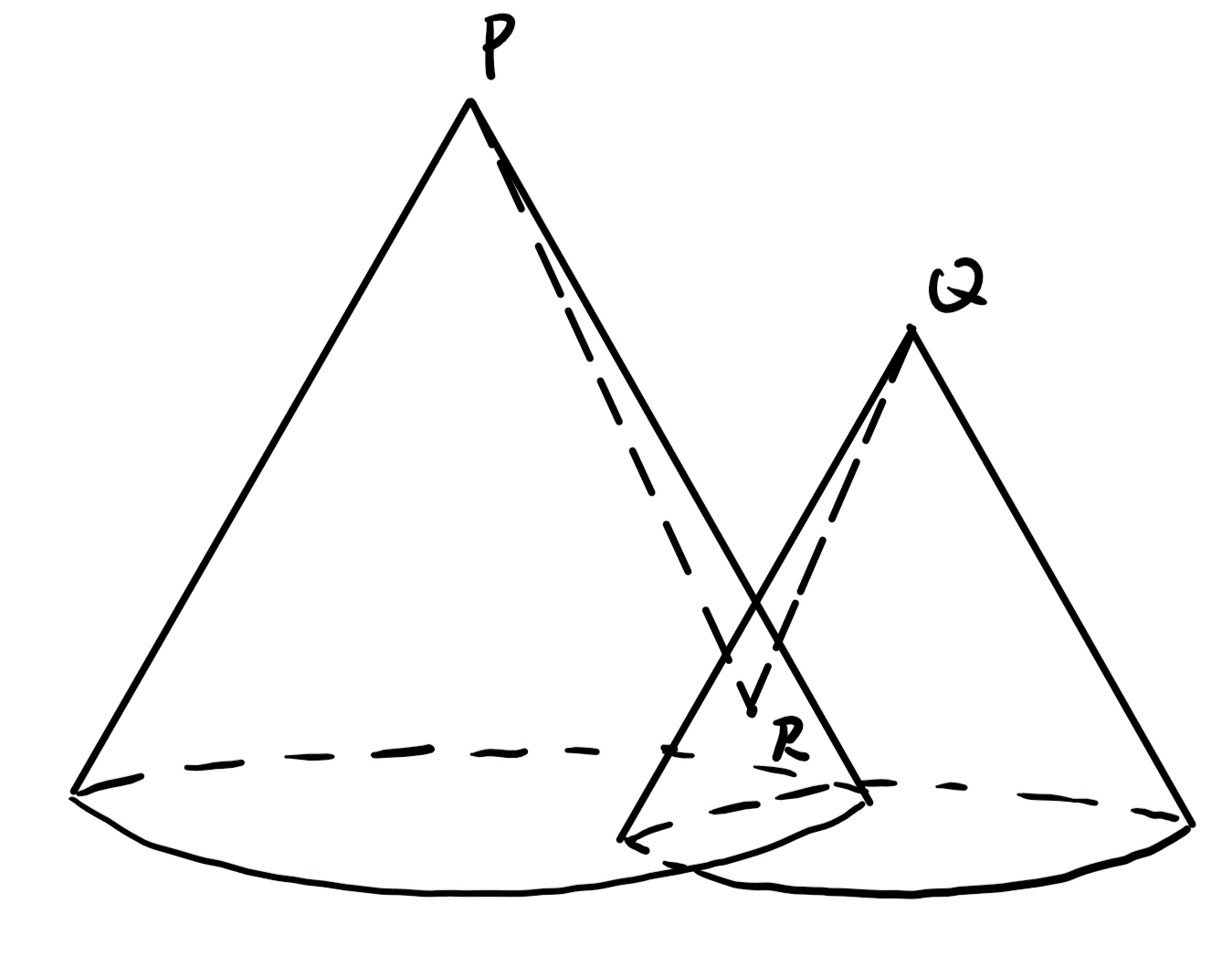
定理证明：我们分两个方面来证明：

第一部分，我们来证明在最短长度和的限制之下，这个相交的结点必然是在两个支撑圆锥的高的公共平面上。事实上我们可以使用在算法1的理论证明中的方法来证明（只需要做少许修改）：

我们考虑以下的调整，我们将R平移到R’，……从而将支撑线段PR，QR变成了PR’，QR’，我们先来说明这两个也是支撑线段；



事实上，我们从支撑圆锥的角度来考虑：





由于R在两个支撑圆锥的交叉部分，注意到两个支撑圆锥的高所在的共同平面为交叉部分的对称轴，我们将R投影到这个平面上得到的R’必然也在交叉部分之内，从而我们由支撑圆锥的定义可以知道，PR’，PQ’都是支撑线段；我们再证明，这种平移可以使得方案变得更优；

事实上由直角三角形内斜边大于直角边，注意到RR’垂直于对称平面，所以三角形RR’P和RR’Q都是直角三角形，同时我们得到：

PR’<PR； QR’<QR

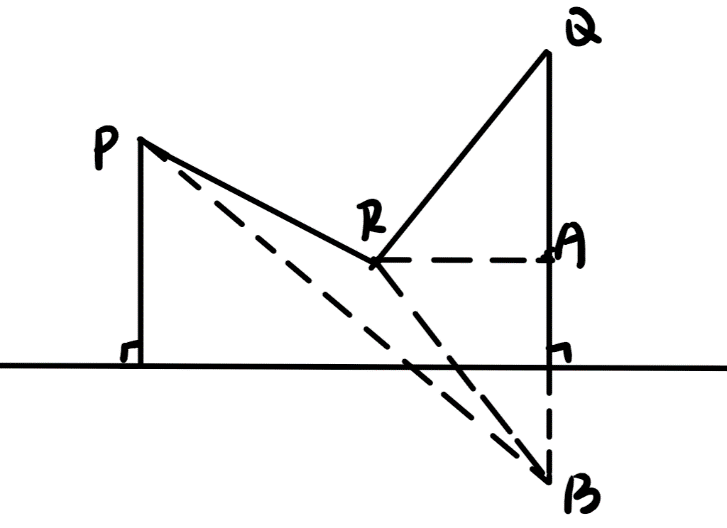
从而此时得到的方案，支撑线段的总长度减少了……从而原来包含R的方案不是最优方案，也就是说：

如果包含R的两条相交支撑线段之和是最短的，那么R一定会在两个支撑圆锥的高的公共平面上；

第二部分，我们证明它一定会处于两个圆锥相交部分的最高点或者是在包含关系条件下的较低的圆锥顶点处：（由于第一部分的证明我们可以将以下问题的讨论限制在一个平面之内）

我们同样由定理四的证明当中可以知道，如果两个支撑线段的长度和最短那么对于结点R，PR，QR和竖直方向的夹角相等：

两个原始点和这个结点的连线和竖直方向的夹角是相等的，的取值会在下面的证明当中给出；

证明如下：



我们考虑R在Q引向地面的垂线上的投影A，设B为Q关于点A的对称点，那么B在这条垂线上并且有：

（对称性）

而在三角形（可能是退化的）PRB内由两边之和不小于第三边：

故 （联立）

所以在固定R点纵坐标不动的情况下，即类似于将R在水平方向“左右移动”，所得的不小于PB，此时PB与R在水平上的移动没有关系，从而的最小值为PB，且当且仅当R在PB上的时候取到，此时由对称性有：∠RQB=∠RBQ

注意到由P引向地面的垂线和QB垂直，故有：PR，RQ和竖直方向的夹角相等，第一部分证毕；

且我们知道，当夹角为的时候，

其中*l*等于两个点PQ在地面上的投影的距离

所以它是关于的递减函数（sin 在定义区间上是递增函数），注意到的上界为

b=min{}

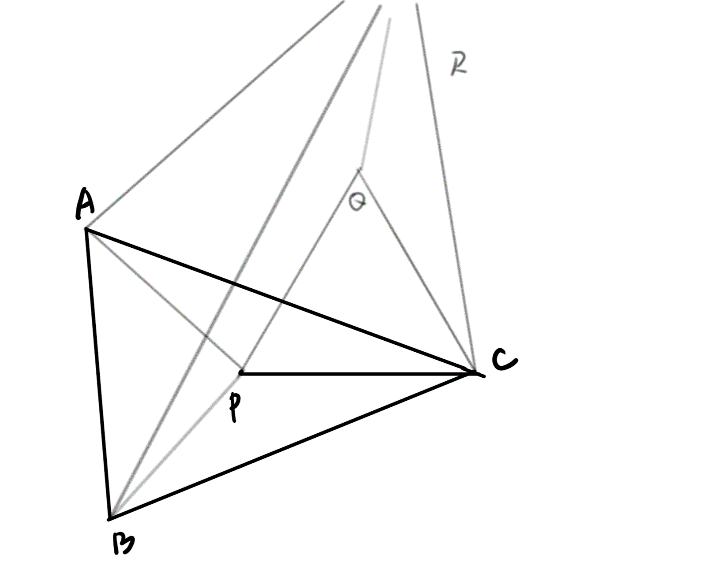
当min函数取到前面那个值时，给出的方案时直接连接两个给出的点；取到后者的时候，为两个圆锥相交的最高点；他们分别对应两个圆锥为包含关系和有交叉部分但是没有包含关系的情况，于是该定理证明完毕；

贡献认定: 由周海刚提供。

3.3. 费马点定理：

定理内容：在一个三个角都不大于120°的三角形当中，到三个顶点距离之和最短的点在三角形内，且满足到三个顶点连线的三条线段两两之间夹角为120°

定理的证明：

 如图：P为在三角形ABC当中任意一点，我们作两个等边三角形CPQ以及ACR，那么我们由几何知识可以知道三角形APC和三角形RQC是正相似的，因此我们可以得到：

AP=RQ

同时我们由等边三角形的三条边相等：

CP=PQ

从而我们可以得到点P到三角形三边的三条线段距离长度之和为：

AP+BQ+CP=BP+PQ+QR>=BR

其中最后一步是由两点之间线段最短得到的，不等式的等号成立的条件是BPQR四点共线，也就是：

∠BPC=180°-∠QPC=120°；

∠APC=∠RQC=180°-∠PQC=120°；

也就是P点处于三角形内满足与三个顶点的连线两两之间夹角为120°的点，这个点被称为三角形ABC的费马点。

贡献认定: 由周海刚提供。

1. 算法二：三个点的支撑算法（最初想法）
   1. 算法功能简述我们首先来利用支撑圆锥的位置
   2. 算法内容：
      1. 输入：三个点的三位笛卡尔坐标系内的坐标；最小支撑角a；
      2. 输出：用数个三维坐标对来表达出来所给出的支撑线段（支撑方案）以及所有支撑线段长度的和sum
      3. 算法具体操作：
         1. 首先利用支撑圆锥的关系判定定理判断三个所给的原始点的支撑圆锥的位置关系，如果存在一个点的支撑圆锥和其他两个均不相交，进入第一步；否则进入第二步；
         2. 第一步：将这个满足支撑圆锥和其他两个支撑圆锥都不相交的原始点直接垂直引向地面的线段记入支撑方案当中，同时对另外两个点运用算法1，输出这两个方案的并；
         3. 第二步：在这个地方，我们有如下几个方案，我们逐一实践，然后将他们进行比较，得出一个近似最优方案：
            1. 选取其中两个支撑圆锥有公共点的点运用算法1，得出一种支撑方案，然后另外一个点输出直接垂直引向地面的支撑线段，并且计算它们的支撑线段的和；比较至多三种选取方案的和，选取其中支撑线段和最短的方案记入到备选方案当中；
            2. 选取其中两个支撑圆锥有公共点的点根据之前给出的最短支撑圆锥相交定理，取出他们两个的最高交点，对这个点和另外的点运用算法一得出一个支撑方案；比较三种支撑方案，选取其中支撑线段和最短的方案记入到备选方案当中；
            3. 选取其中两个支撑圆锥有公共点的点，根据算法一得出一个新点，将这个新点和另外的那个点运用算法一，然后得出一个方案；比较至多三种支撑方案，选出其中支撑线段和最短的方案记入到备选方案当中；
         4. 选取三种备选方案当中线段长度之和最短的，然后将其当作最佳方案输出；算法结束

算法具体实现：

import math

accuracy=3

class point:

def \_\_init\_\_(self,xx,yy,zz):

self.x=xx

self.y=yy

self.z=zz

def print\_point(self):

print(self.x,self.y,self.z,end = ' ')

class line:

def \_\_init\_\_(self,p1,p2):

self.a=p1

self.b=p2

self.v=point(p1.x-p2.x,p1.y-p2.y,p1.z-p2.z)

def angle(self):#和竖直方向的夹角

return abs(math.acos(self.v.z/math.sqrt(self.v.x\*\*2+self.v.y\*\*2+self.v.z\*\*2)))

def lenth(self):

return round(math.sqrt(self.v.x\*\*2+self.v.y\*\*2+self.v.z\*\*2),accuracy)

def print\_line(self):

self.a.print\_point()

self.b.print\_point()

print()

def line\_to\_ground(p1):

p2 = point(p1.x,p1.y,0.0)

return line(p1,p2)

def lenth(\*args):

sum = 0

for i in args:

sum += i.lenth()

return sum

def calculate\_a(p1,p2,p3):

l1 = line(p1,p2)

l2 = line(p2,p3)

l3 = line\_to\_ground(p3)

l1.print\_line()

l2.print\_line()

l3.print\_line()

print(lenth(l1,l2,l3))

def calculate\_b(p1,p2,p3):

l1 = line(p1,p2)

l2 = line\_to\_ground(p2)

l3 = line\_to\_ground(p3)

l1.print\_line()

l2.print\_line()

l3.print\_line()

print(lenth(l1,l2,l3))

def calculate\_c(p1,p2,p3):

l1 = line\_to\_ground(p1)

l2 = line\_to\_ground(p2)

l3 = line\_to\_ground(p3)

l1.print\_line()

l2.print\_line()

l3.print\_line()

print(lenth(l1,l2,l3))

def calculate\_d(p1,p2,p3):

global a

if(a > math.pi/3):

a = math.pi/3

tan\_a=math.tan(a)

tan\_a=round(tan\_a,accuracy)

r1=p1.z\*tan\_a

r2=p2.z\*tan\_a

r\_sum=r1+r2

center\_dis=math.sqrt((p1.x-p2.x)\*\*2+(p1.y-p2.y)\*\*2)

zz=(p1.z+p2.z-center\_dis/tan\_a)/2

r1\_new=r1-zz\*tan\_a

r2\_new=r2-zz\*tan\_a

xx=p1.x+r1\_new\*(p2.x-p1.x)/(r1\_new+r2\_new)

yy=p1.y+r1\_new\*(p2.y-p1.y)/(r1\_new+r2\_new)

xx = round(xx,accuracy)

yy = round(yy,accuracy)

zz = round(zz,accuracy)

new\_p = point(xx,yy,zz)

l1 = line(p1,new\_p)

l2 = line(p2,new\_p)

l3 = line\_to\_ground(new\_p)

l4 = line\_to\_ground(p3)

l5 = line\_to\_ground(p1)

l6 = line\_to\_ground(p2)

flag = 1

l7 = line(p1,p2)

if(l7.angle()>tan\_a): flag = 0

if(p1.z > p2.z): l8 = line\_to\_ground(p2)

else: l8 = line\_to\_ground(p1)

lenth1 = lenth(l1,l2,l3,l4)

lenth2 = lenth(l4,l5,l6)

lenth3 = lenth(l7,l8,l4)

if (flag == 0): lenth3 = lenth1 + lenth2 +1

if(lenth1 <= lenth2 and lenth1 <= lenth3):

l1.print\_line()

l2.print\_line()

l3.print\_line()

l4.print\_line()

print(lenth1)

elif(lenth2 <= lenth1 and lenth2 <= lenth3):

l4.print\_line()

l5.print\_line()

l6.print\_line()

print(lenth2)

else:

l4.print\_line()

l7.print\_line()

l8.print\_line()

print(lenth3)

def check\_intersection(p1,p2,angle1,angle2):

tan\_a1=math.tan(angle1)

tan\_a2=math.tan(angle2)

tan\_a1=round(tan\_a1,accuracy)

tan\_a2=round(tan\_a2,accuracy)

r1=p1.z\*tan\_a1

r2=p2.z\*tan\_a2

r\_sum=r1+r2

r\_minus=abs(r1-r2)

center\_dis=math.sqrt((p1.x-p2.x)\*\*2+(p1.y-p2.y)\*\*2)

if(r\_sum<center\_dis):#外离

return 0

if(r\_sum==center\_dis):#外切

return 1

if(r\_sum>center\_dis and center\_dis>r\_minus):#相交

return 1

if(r\_minus==center\_dis):#内切

return 1

if(r\_minus>center\_dis):#包含

return 2

a=float(input("Angle:"))

x,y,z = map(float,input("P1:").split())

f1 = point(x,y,z)

x,y,z = map(float,input("P2:").split())

f2 = point(x,y,z)

x,y,z = map(float,input("P3:").split())

f3 = point(x,y,z)

if(f1.z<f2.z): f1,f2 = f2,f1

if(f1.z<f3.z): f1,f3 = f3,f1

if(f2.z<f3.z): f2,f3 = f3,f2

if(a==0):

if(f1.x == f2.x == f3.x and f1.y == f2.y == f3.y):

calculate\_a(f1,f2,f3)

elif (f1.x == f2.x and f1.y == f2.y):

calculate\_b(f1,f2,f3)

elif (f1.x == f3.x and f1.y == f3.y):

calculate\_b(f1,f3,f2)

elif (f2.x == f3.x and f2.y == f3.y):

calculate\_b(f2,f3,f1)

else:

calculate\_c(f1,f2,f3)

elif(a<0 or a>=90):

print("Invalid Syntax")

else:

a = math.radians(a)

po12 = check\_intersection(f1,f2,a,a)

po23 = check\_intersection(f2,f3,a,a)

po13 = check\_intersection(f1,f3,a,a)

if(po12 == po23 == po13 == 0):

calculate\_c(f1,f2,f3)

elif(po12 == 0 and po13 == 0):

calculate\_d(f2, f3, f1)

elif(po12 == 0 and po23 == 0):

calculate\_d(f1, f3, f2)

elif(po13 == 0 and po23 == 0):

calculate\_d(f1, f2, f3)

else:

print("Not slove yet")

贡献认定: 由汪子涵提供。

* 1. 算法说明
  2. 算法理论验证

下面逐一说明三种备选方案各自的合理性

第一种备选方案的合理性是考虑到以下情况：一个点的支撑圆锥和其他两个不相交，那么它不可能和任何一个结点相通，这样的话对两个支撑圆锥相交的两个点运用算法一是最好的；

第二种和第三种备选方案是考虑到贪心的算法：

先选取三个点当中的两个点，对他们选取一个节点，使得到结点的两条支撑线段的线段长度之和“最短”，但是此处的“最短”有很多理解方式。

第一种便是直观的理解：两个线段长度之和线段最短，这便是备选方案三的想法，通过两个点的支撑线段之和最短性定理（定理3.2）保证所取的两条支撑线段长度和确实是最短的；

第二种属于更加“优化”的一种想法，我们将这个相交的结点和地面的距离也加入进考虑（因为这个节点最终将连接到地面上，所以我们不太能脱离结点纵坐标来考虑问题），一个显然的想法便是直接运用算法一，得出一个近似解的结点；

通过这几种想法得出几种备选方案然后比较便可以得出一个较优的方案；

* 1. 算法数值验证
  2. 算法数值认证正例

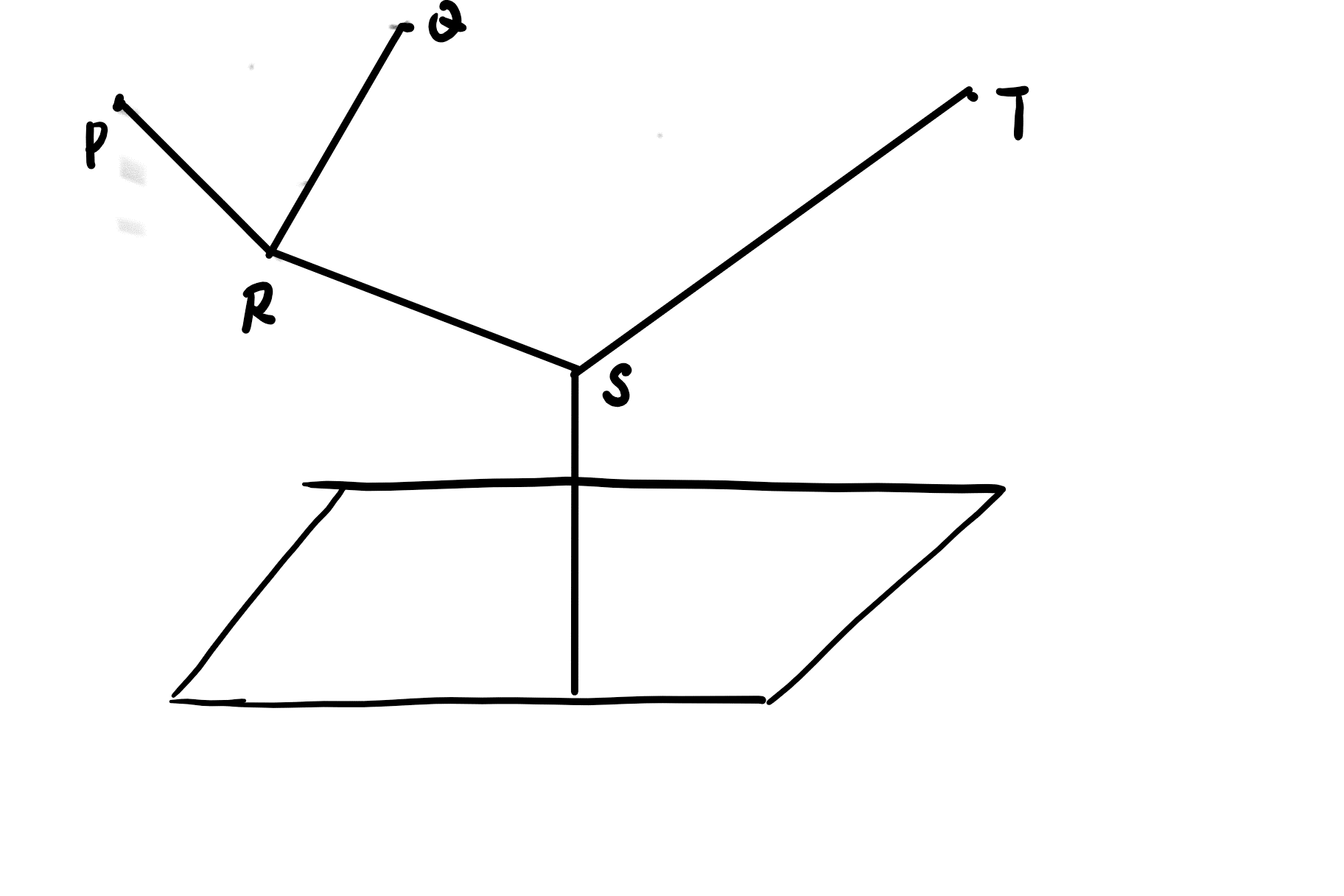
## 猜想

### 猜想: 在n=3的情况下如何考虑结点处于三个点当中的位置？

结论部分：直观来看，如果利用上述算法没有考虑到第一个确定的结点不在确定它的两个点所在的竖直平面内的情况，我们可以先研究三个原始点只有一个结点的情况。

贡献认定: 由周海刚提供。

对上面结论的分析或思考部分：对于最初想法当中这两种想法都有一个缺陷，便是在这两种方案当中都是使用算法1得出来的结点，我们由算法1的特征可以知道这个结点一定是和“计算出”它的两个点是处于同一个平面上的，但是在我们的想法当中，一直存在一个疑问，对于下图的情况是否可能更加优化？



也就是说，将图中的S从和PQ处于同一个竖直平面的位置靠近，那么ST的长度就会减少，那么总长度会不会因此减少呢？暂时放到猜想里面。

我们考虑图中的情况：我们对于PQ使用了算法一，得出了一个结点R，然后对于结点R和T使用了算法一，得到了一个新的结点S，我们考虑猜想当中的最初想法：在“大多数情况”下，R向靠近S的方向运动是“好的”。事实上，这是由于当我们从R点向PQS这个平面作垂线的时候，设垂足为H，那么我们由直角三角形的斜边大于直角边可以得到：PH、QH、SH分别小于PR、QR、SR，那么我们就有H点比R点更为优化；同时我们由费马点定理从而我们可以知道我们再调整H的位置直到H为三角形PQS的费马点——也就是说R如果是最优的，那么它“应该”满足以下条件：

1. R在平面PQS上并且R点为三角形PQS的费马点
2. S是RT两个点运用算法一得出来的结点

前文当中我们给“应该”打了引号是因为我们有角度的限制，而平面上的这个费马点R并不一定能够满足和P，Q两点的连线形成支撑线段，但是我们还是先研究上文中简单的情况：

还是从简单情况入手，我们考虑在算法一当中，所得到的结点和产生它的两个点的连线的夹角在“大部分情况下”都是120°，而在其他情况下情况显得比较好研究（这是因为我们注意到在其他情况下，情况是产生的结点在地面上或者是较低的那个原始点，这些都是可以进行讨论的），下面我们对于同时满足上述1）&2）以及使用算法一的时候取到夹角为120°的情况进行讨论，看是否能够求出R，S两个点的显性表达式：

贡献认定: 由周海刚提供。

## 联系方式：

## 姓名:周海刚

班级: 数93

电话: 13974948791

Email: zhouhg19@mails.tsinghua.edu.cn

## 参考文献

## 总结与感想

### 回顾：

在这次新生研讨课当中我选择了“最短支撑路径”问题，首先是因为对于三个选题当中的最长公共有序字符串的问题没有相关的了解和背景知识，所以觉得比较难然后没有选择；然后对于服装排料的问题觉得问题比较复杂，思索了一下觉得没有什么思路（服装布料的大小变量太多，不管是长宽或者是倾斜放置的角度），所以也没有选择，加上当时没有了解什么和算法有关的问题所以就选择了看起来和数学比较有关系的最短支撑路的问题。

拿到这个问题的时候首先查询了一下有关的文献，首先便确定了一个错误的方向（也就是n=3的时候的第二种选取结点的方法，有他的合理之处但是并不是n=2的情况时最优的情况），所以第一周基本上时间就被浪费了。这件事告诉我们两个道理：一个是只有真正将每个命题写出来的时候才能发现自己想法的错误之处；二是不要随便相信文献中的结论，特别是这种实际而又比较冷门的问题。

然后就花了一周的时间写完了基础的n=2的情况的大部分内容，然后发现很多结论还存在不严谨的证明，然后花了一周来补各种证明，然后基本上处于“发现trivial的结论，然后花好久证明”的状态。

再下一周就开始研究n=3的情况，然后发现其实之前提到的文章中的选取结点的方法也是有道理的，把它添加到了备选方案当中。

再下一周主要在忙思修的实践活动，所以没怎么研究这方面；再下一周开始思考n=3的情况的时候发现实际上这个问题非常困难，和n=2的情况完全不属于一个数量级，因为n=2的情况很容易归化到平面上的情况，而n=3的情况处于平面当中，直接就没办法考虑了orz，同时两个结点的自由度使得问题变得极其复杂，这也说明这个问题并没有看起来那么容易考虑。

### 现状：

现在在考虑n=3的情况，正在考虑从最优情况需要满足的条件反过来考虑最优的情况应该满足的一些条件，希望从费马点的有关性质考虑命题，但是支撑线段的角度限制实在是太过于强，所以估计想要有很大的进展比较难，只希望能够在一些比较trivial的情况当中得出一些结论。

### 感想：

报名这个新生研讨课的时候完全没有想到这门课是这种只研究一个课题的形式（以为是每周做一个比较困难的小作业），但是这种方式应该是比较接近于以后做学术研究写研究报告什么的时候的真实状况吧，觉得还是挺有意义同时也有很多感想。

首先是意识到了：那些看起来很简单的问题可能并没有想象的那么简单，这个道理的另一个表现形式也就是说，对于任何问题的最初想法都是粗糙的、需要打磨的、甚至不一定是正确的，只有当最后写在文档里面的时候才能发现逻辑上的哪怕一点点错误都有可能导致整个证明都不正确（这应该也是理论研究上的很常见的事件吧），所以一定在研究问题的时候需要严谨，自认为还做的不错，至少用到的结论都进行了证明而不管是否trivial。

第二点是心态的问题，在第六周第五周的时候被n=3的情况难到了，完全一点想法都没有，再加上各种事情以及写不完的作业都弄得心态有点爆炸；其实主要原因还是奢求自己想要找到某种方法一举解决吧，心里一直觉得n=3的时候，两个结点的自由度还是属于比较简单而且好控制的，但是实际上是存在比较大的困难的，至少在支撑线段的倾角限制之下很多操作的可行性都是不确定的，甚至在没有倾角的情况之下都感觉没有研究到很令人满意的程度，感觉立体的问题还是存在本质上的难度吧，心态也算调整好了，也算是一次磨练罢。

第三点是对于问题的思考思路问题，对于n=2的情况还好，对于n=3的情况思索的同时也就在思考这个问题了：也就是说，我其实一直都在寻找能够一下子或者是通过冗长的证明过程得到数学上严谨成立的最短的支撑方案，但是这样好像不是很可能，因为很多问题都是暂时只能找到近似解的罢，所以就需要我们设计某种算法——这也是这门课的设计初衷吧，想要我们学习设计算法的思维吧而不是要我们寻找某种可以在纸上就能够证明出来的“证明过程”，所以现在也在转变思路，不一定一定得强迫自己找到一种完全不靠电脑的方法了，希望通过电脑的计算来完成这个任务吧，一个很著名的例子就是四色定理（笑）。

总的来说，还是很有收获的，虽然可能在实质上的问题上面没有做到最好，但是在做题的思路和心态上面还是很大长进的，对于以后的学术研究上，虽然和这个研究的课题可能没有关系，但是在最基础的能力上已经有了很大提高了。