2016 级算法第 6 次上机解题报告

15081070 张雨任

一、总结

主要考察知识点为计算几何、多项式与快速傅里叶变换 等新学算法知识以及 分治、贪心、动态规划等经典算法知识。

二、解题报告

A. Bamboo 之寻找小金刚

思路分析

本题考察叉积的应用，确定连续线段是向左转还是向右转。

算法分析

想要判断两条连续线段 p_0p_1 和 p_1p_2 是向左转还是向右转，那么需要找出一种方法确定∠$p_0p_1p_2$ 的转向。采用叉积运算来解决这个问题可以避免计算角度。我们只需要简单地判断一下有向线段 p_0p_2 是位于 p_0p_1 的顺时针还是逆时针方向。因此，我们计算出叉积$(p_2-p_0)\times(p_1-p_0)$。若结果为负，则 p_1p_2 在 p_0p_1 的逆时针方向，在 p_1 处左转。同理，若结果为正，则在顺时针方向，在 p_1 处右转。而叉积为 0 则意味着 p_0、p_1 和 p_2 三者共线。

这道题要注意细节。首先，左转和右转是分开计数的，在第 i 个左转 a+=i，错误的理解是在第 i 个转弯，是左转，则 a+=i。其次，在计算叉积的时候有 int 类型的乘法，两个 int 相乘可能会造成溢出，所以应该使用 long long。

时间复杂度为 O(n)。

参考代码

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#define ll long long
```

```
using namespace std;

struct Node {
    ll x;
    ll y;
}nd[10007],s;

ll n,a,cl,cr;

ll crossProduct(Node p1,Node p2,Node p3,Node p4) {
    return (p2.x-p1.x)*(p4.y-p3.y)-(p2.y-p1.y)*(p4.x-p3.x);
}

int main() {
    while(~scanf("%lld%lld",&n,&a)) {
        for(int i=0;i<n;i++) {
            scanf("%lld%lld",&nd[i].x,&nd[i].y);
        }
        s.x=nd[0].x;
        s.y=nd[0].y;
        cl=0;
        cr=0;
        for(int i=0;i<n;i++) {
            if(i!=0 && i!=n-1) {
                ll dr=crossProduct(s,nd[i],s,nd[i+1]);
                if(dr>0) {
                    cl++;
                    a+=cl;
                }
                else if(dr<0) {
                    cr++;
                    a-=cr;
                }
                s=nd[i];
            }
        }
        printf("%lld\n",a);

    }

}
```

B. ModricWang's FFT : EASY VERSION

思路分析

这道题考察 FFT 快速傅里叶变换，快速傅里叶变换可以用来做多项式乘法。这里延伸，用来做大数乘法。

算法分析

FFT 的基本思想是，通过 DFT 把多项式从系数表达转换为点值表达。

系数表达形如 $y=f(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots+a_nx^n$。

点值表达形如$\{(x_0,y_0),(x_1,y_1),(x_2,y_2),\ldots,(x_{n-1},y_{n-1})\}$，同时满足 $y_0=f(x_0)$, $y_1=f(x_1)$, $y_2=f(x_2)$, …, $y_{n-1}=f(x_{n-1})$。用这个点集就可以表示多项式。

点值表达的优点在于计算多项式乘法很方便。假设两个多项式为$\{(x_0,y_0),(x_1,y_1),(x_2,y_2),\ldots,(x_{n-1},y_{n-1})\}$和$\{(x_0,z_0),(x_1,z_1),(x_2,z_2),\ldots,(x_{n-1},z_{n-1})\}$，那么这两个多项式的乘积为，$\{(x_0,y_0z_0),(x_1,y_1z_1),(x_2,y_2 z_2),\ldots,(x_{n-1},y_{n-1}z_{n-1})\}$。

用系数表达来计算多项式乘积的时间复杂度是 $O(n^2)$，用点值表达来计算多项式乘积的时间复杂度是 $O(n)$。

计算出了$\{(x_0,y_0z_0),(x_1,y_1z_1),(x_2,y_2 z_2),\ldots,(x_{n-1},y_{n-1}z_{n-1})\}$就可以在通过 IDFT，把点值表达恢复到系数表达。

总结来说，快速 FFT 主要有以下四点：

1. 使次数界（上界）增加一倍。A(x)、B(x)的长度扩充到 2*n
2. 求值。主要是求点值表示 A(x)、B(x)的点值表示
3. 点乘。C(x)=A(x)*B(x)
4. 插值。对 C(x)进行插值，求出其系数表示。

对于大数乘法，再额外处理进位情况即可。

DFT 和 IDFT 是快速傅里叶变换的核心，具体的讲解可以参考一下题解以及《算法导论》：

http://www.cnblogs.com/lsx54321/archive/2012/07/20/2601632.html

http://blog.jobbole.com/58246/

http://blog.jobbole.com/70549/

FFT 的时间复杂度为 O(nlgn)。

参考代码

```
#include <cstdio>
#include <cmath>
#include <cstring>
#define N 205050

using namespace std;
const double PI=acos(-1.0);

/**
 * big number multiply
 **/

struct Complex {
    double real,image;
    Complex(double _real=0.0,double
_image=0.0):real(_real),image(_image) {}
    Complex operator*(Complex c) {
        return Complex(real*c.real-
image*c.image,real*c.image+image*c.real);
    }
    Complex operator+(Complex c) {
        return Complex(real+c.real,image+c.image);
    }
    Complex operator-(Complex c) {
        return Complex(real-c.real,image-c.image);
    }
};

void bit_rev(Complex *a,int loglen,int len) {
    for(int i=0;i<len;++i) {
        int t=i,p=0;
        for(int j=0;j<loglen;++j) {
            p<<=1;
            p=p | (t & 1);
            t>>=1;
```

```
        }
        if(p<i) {
            Complex temp=a[p];
            a[p]=a[i];
            a[i]=temp;
        }
    }
}

void FFT(Complex *a,int loglen,int len,int on) {
    bit_rev(a,loglen,len);
    for(int s=1,m=2;s<=loglen;++s,m<<=1) {
        Complex wn=Complex(cos(2*PI*on/m),sin(2*PI*on/m));
        for(int i=0;i<len;i+=m) {
            Complex w=Complex(1.0,0.0);
            for(int j=0;j<m/2;++j) {
                Complex u=a[i+j];
                Complex v=w*a[i+j+m/2];
                a[i+j]=u+v;
                a[i+j+m/2]=u-v;
                w=w*wn;
            }
        }
    }
    if(on==-1) {
        for(int i=0;i<len;++i)
            a[i].real/=len,a[i].image/=len;
    }
}

char a[N*2],b[N*2];
Complex pa[N*2],pb[N*2];
int ans[N*2];

int main () {
    while(~scanf("%s%s",a,b)) {
        int lena=strlen(a);
        int lenb=strlen(b);
        int n=1,loglen=0;
        while(n<lena+lenb) n<<=1,loglen++;
        for(int i=0,j=lena-1;i<n;++i,--j)
```

```
        pa[i]=Complex(j>=0?a[j]-'0':0.0,0.0);
      for(int i=0,j=lenb-1;i<n; ++i,--j)
        pb[i]=Complex(j>=0?b[j]-'0':0.0,0.0);
      for(int i=0;i<=n;++i) ans[i]=0;
      FFT(pa,loglen,n,1);
      FFT(pb,loglen,n,1);
      for(int i=0;i<n;++i) pa[i]=pa[i]*pb[i];
      FFT(pa,loglen,n,-1);
      for(int i=0;i<n;++i) ans[i]=pa[i].real+0.5;
      for(int i=0;i<n;++i) ans[i+1]+=ans[i]/10,ans[i]%=10; //8 jin
zhi
      int pos=lena+lenb-1;
      for(;pos>0 && ans[pos]<=0;--pos) ;
      for(;pos>=0;--pos) printf("%d",ans[pos]);
      printf("\n");
    }
    return 0;
}
```

C. AlvinZH 的学霸养成记 II

思路分析

本题考察贪心策略，是《算法导论》中活动调度问题的改版，难度有所提升。

算法分析

先看总体思路，对于每个课程，有两个参数：“持续时间”d，“结课时间”e，要求每个课程必须在“结课时间”前结束。求最多能学习多少门课程。

首先，要明确，“结课时间”越靠前的课程就越紧急，因此一定程度上要优先考虑这些课程。因此遍历课程之前，要按照课程的“结课时间”来排序。

然后，线性遍历每一门课程，由于使用贪心算法，因此应判断当前课程应不应该放在要学习的课程中，选取局部最优情况。那么什么样的课程应该被选取？

维护一个变量 cnt，表示已选取的课程的“总持续时间”。

对于遍历到的当前课程，如果选择该课程，得到的“总持续时间”cnt 要累加上该课程的“持续时间”d。这个新的“总持续时间”cnt 如果小于等于当前课程的“结课时间”e，说明加入这门课程能够保证在规定的“结课时间”之前完成，那么这门课程可以加入。

对于遍历到的当前课程，如果选择该课程，新的“总持续时间”cnt 要比当前课程的“结课时间”e 还要长，那么说明当前情况，这门课是暂时上不了了。但是，不代表这门课就要直接放弃。这种局面很有可能是被选中的某门课程“持续时间”d 太长造成的。因此，寻找被选中课程中的“持续时间”d 最长的那门，如果它的“持续时间”d 比当前课程还长，那么就把当前课程替换该课程，并且更新“总持续时间”cnt。由于更新后的“总持续时间”cnt 是低于新添加课程的“结课时间”e，并且能够保证课程之间不重叠，因此这样的更新总是合法的。

我们需要选择一个容器来盛放选中的课程。由于需要经常选择“持续时间”d 最长的课程，因此使用优先队列比较方便。该容器中存放选中课程的“持续时间”，每次取出最大值，因此维护一个大顶堆即可。

本算法时间复杂度为 O(nlgn)——每个课程遍历一遍，时间复杂度 O(n)；维护大顶堆的时间复杂度是 O(lgn)。

参考代码

```
#include <cstdio>
```

D. AlvinZH 的学霸养成记 V

思路分析

本题考查叉积的应用，考察通过叉积来判断极角的大小。

算法分析

首先对每个点到原点的向量进行排序，排序的依据是极角的大小，小的排在前面，大的排在前面，极角相等的时候，距离原点近的点排在前面。

那么可以写一个 cmp 函数，在调用 sort 函数进行排序。

对于被比较的两个点，我们只需要简单地判断一下有向线段 op_1 是位于 op_2 的顺时针还是逆时针方向（o 是原点）。因此，我们计算出叉积$(p_2o)\times(p_1o)$。若结果为负，则 op_1 在 op_2 的逆时针方向，说明 p_2 的极角小于 p_1。同理，若结果为正，则在顺时针方向，说明 p_2 的极角大于 p_1。当极角相同的时候，距离原点近的点排在前面。那么可以写出以下 cmp 函数：

```
bool cmpSortPolarAngle(Node a,Node b) {
    double tmp=crossProduct(od,a,od,b);
    if(tmp>0) return true;
    if(tmp==0 && dis(a,od)<dis(b,od)) return true;
    return false;
}
```

排序的时间复杂度为 O(nlgn)。

这道题要额外注意数据类型的使用，虽然输入数据在 int 范围内，但是处理叉积的时候由于两个 int 相乘可能会出现 int 溢出的情况，因此可以考虑使用 double 类型。最好不要使用 long long 类型，上机的时候发现使用 long long 会有一组数据超时。

参考代码

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#define ll long long
using namespace std;
```

```
struct Node {
    char s[107];
    double x;
    double y;
}nd[100007];

Node od;

double crossProduct(Node p1,Node p2,Node p3,Node p4) {
    return (p2.x-p1.x)*(p4.y-p3.y)-(p2.y-p1.y)*(p4.x-p3.x);
}

double dis(Node a,Node b) {
    return (b.x-a.x)*(b.x-a.x)+(b.y-a.y)*(b.y-a.y);
}

bool cmpSortPolarAngle(Node a,Node b) {
    double tmp=crossProduct(od,a,od,b);
    if(tmp>0) return true;
    if(tmp==0 && dis(a,od)<dis(b,od)) return true;
    return false;
}

int n;

int main() {
    while(~scanf("%d",&n)) {
        od.x=0;
        od.y=0;
        for(int i=0;i<n;i++) {
            scanf("%s%lf%lf",nd[i].s,&nd[i].x,&nd[i].y);
        }
        sort(nd,nd+n,cmpSortPolarAngle);
        for(int i=0;i<n;i++) {
            printf("%s\n",nd[i].s);
        }
        printf("\n");
    }
}
```

E. Bamboo 之吃我一拳

思路分析

本题计算最近点对距离，可以考虑使用分治策略来解决。

算法分析

对于给定点集，用一条直线将点集划分为左右两部分，使得左右点的数目相等，那么中间点的下标为 mid=(l+r)/2，初始状态下 l=0, r=N-1。按照点的横坐标进行排序，然后就很容易找到这条直线。

然后，采用分治策略，递归地求解这两部分。每次递归计算这部分的两点之间的最短距离。

假设分治策略计算出的两边的（也就是下标为 0~l 和 r~N-1 的点）最近距离，设为 d。先前计算的分别是下标小于 mid 的点集 S_1 的最近距离和下标大于 mid 的点集 S_2 的最近距离，因此接下来还要计算两点分别来自于 S_1、S_2 的最近距离，因此二重循环，外循环遍历 S_1 的点，内循环遍历 S_2 的点，维护一个最近距离。

这里有一个很重要的优化，也是本题的核心。就是在遍历点集 S_2 的时候，只需要遍历前 5 个点即可。

这道题要额外注意数据类型的选择，虽然坐标不超过|2e5|，但是由于 dis 函数计算距离的平方，因此能够达到 1e10 的量级，因此使用 double 更为保险。

本题时间复杂度为 O(nlgn)。

参考代码

```
#include <cstdio>
#include <algorithm>
#include <cmath>
#define ll long long
using namespace std;

const double inf=1e20;
double ans;
int N;

struct Node {
```

```
    double x;
    double y;
}nd[100007];

bool cmpMostLeft(Node a,Node b) {
    return a.x<b.x;
}

double dis(Node a,Node b) {
    return (b.x-a.x)*(b.x-a.x)+(b.y-a.y)*(b.y-a.y);
}

void FindCloeset(int l,int r) {
    if(l<r) {
        int mid=(l+r)/2;
        FindCloeset(l,mid);
        FindCloeset(mid+1,r);
        for(int i=l;i<=mid;i++) {
            for(int j=mid+1;j<=min(mid+5,r);j++) {
                double tmp=dis(nd[i],nd[j]);
                if(tmp<ans) ans=tmp;
            }
        }
    }
}

int main() {
    while(~scanf("%d",&N)) {
        for(int i=0;i<N;i++) {
            scanf("%lf%lf",&nd[i].x,&nd[i].y);
        }
        ans=inf;
        sort(nd,nd+N,cmpMostLeft);
        FindCloeset(0,N-1);
        double res;
        res=(double)ans;
        printf("%.2f\n",sqrt(res));

    }

}
```

G. ModricWang likes geometry

思路分析

本题考查计算几何，应用的几何知识是反演点。

算法分析

题目大意是，半径为 r 的圆，圆心在坐标系的原点上。给定圆内或圆上有两点 P、Q，满足 PO=QO，试求圆上一点 D，使得 PD+QD 为最小值。

计算线段长度相加的最小值，一般的思路就是通过“变换”使得两条线段共线，然后使用两点之间线段最短的公理，来确定 D 的坐标，从而计算 PD+QD 的大小。

本题使用的“变换”，叫做反演点。根据维基百科，反演点的定义如下：

反演是种几何变换。给定点 O、常数 k，点 P 的变换对应点就是在以 O 开始的射线 OP 上的一点 P'使得|OP||OP’|=k^2。此处，令常数 k 大小等于半径 r。那么有|OP||OP'|=r^2，根据相似三角形的知识，有△POD∽△DOP’。我们发现 PD 可以转化成求 PD 的值，即 PD=PD’×ros（ratio of similitude，相似比）。同理，做出 Q 的反演点 Q’，QD=Q’D×ros。那么 PD+QD=ros×(P’D+Q’D)，所以接下来的重点就是求 P’D+Q’D。使 P’Q’为一条线段，从而求得 D 的坐标。

接下来，可以分为两种情况：

P’Q’与圆交于 D。由于两点之间线段最短，所以 D 即为所求。那么使用点之间距离公式计算|P’Q’|，然后通过关系 PD+QD=ros×(P’D+Q’D）计算|PQ|。

P’Q’与圆没有交点。那么 D 点位于 P’Q’的中垂线与圆的焦点。通过重点坐标公式计算 P’Q’的中点 M，再通过点之间距离公式计算 MO，那么有|MO|/|DO|=|MO|/r=M.x/D.x=M.y/D.y（M.x 表示 M 点的横坐标）。通过这个比例可以计算出 D 点的横纵坐标 D.x 和 D.y。那么直接通过点之间距离公式计算 PD+QD 即可。

这道题由于大量使用了除法，如果 P、Q 都位于原点，再计算相似比 ros 时会导致除零异常，因此要特判 P、Q 位于原点的情况，DP=QD=OD=r，则结果为常值，值为 2*r。除此之位，还要注意浮点数精度问题，引入适当大小的 eps 可以解决。

参考代码

```
#include <cstdio>
```

```cpp
#include <cstring>
#include <algorithm>
#include <cmath>
#define ll long long
using namespace std;

const double eps=1e-7;

struct Node {
    double x;
    double y;
}P,Q,P_,Q_,midOfPQ,D;

double dis(Node a,Node b) {
    return sqrt((b.x-a.x)*(b.x-a.x)+(b.y-a.y)*(b.y-a.y));
}

double r,ros,d,midOfPQ_O,ans,P_Q_;

int T;

int main() {
    scanf("%d",&T);
    while(T--) {
        scanf("%lf%lf%lf%lf%lf",&r,&P.x,&P.y,&Q.x,&Q.y);
        d=sqrt(P.x*P.x+P.y*P.y);
        if(d<eps && d>-eps) {
            printf("%.3f\n",2*r);
        }
        else {
            ros=r*r/(d*d);
            P_.x=ros*P.x;
            P_.y=ros*P.y;
            Q_.x=ros*Q.x;
            Q_.y=ros*Q.y;
            midOfPQ.x=(P_.x+Q_.x)/2;
            midOfPQ.y=(P_.y+Q_.y)/2;
            midOfPQ_O=sqrt(midOfPQ.x*midOfPQ.x+midOfPQ.y*midOfPQ.y);
            if(midOfPQ_O>r) {
                D.x=midOfPQ.x/midOfPQ_O*r;
                D.y=midOfPQ.y/midOfPQ_O*r;
```

```
            ans=dis(D,P)+dis(D,Q);
//            printf(">:%.3f\n",ans);
        }
        else {
            P_Q_=dis(P_,Q_);
            ans=P_Q_*d/r;
//            printf("<=:%.3f\n",ans);
        }
        printf("%.3f\n",ans);
    }

}

}
```