

Standard Code Library

Radium

November 4, 2019

Contents

一切的开始	2
宏定义	2
读入挂	2
快速幂	3
计算几何	4
二维几何基础	4
距离	6
极角排序	7
Pick 定理	8
凸包	8
最小圆覆盖	9
扫描线	9
数据结构	12
树状数组	12
线段树	13

一切的开始

宏定义

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  #ifndef ONLINE_JUDGE
6  #define dbg(args...) \
7      do { \
8          cerr << "\033[32;1m" << #args << " -> "; \
9          err(args); \
10     } while (0)
11 #else
12 #define dbg(...)
13 #endif
14 void err() { cerr << "\033[39;0m" << endl; }
15 template <typename...> class T, typename t, typename... Args>
16 void err(T<t> a, Args... args) {
17     for (auto x : a) cerr << x << ' ';
18     err(args...);
19 }
20 template <typename T, typename... Args>
21 void err(T a, Args... args) {
22     cerr << a << ' ';
23     err(args...);
24 }
25 // -----
```

读入挂

```
1  inline char nc() {
2      static char buf[100000], *p1 = buf, *p2 = buf;
3      return p1 == p2 &&
4          (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2)
5          ? EOF
6          : *p1++;
7  }
8  template <typename T>
9  bool rn(T& v) {
10     static char ch;
11     while (ch != EOF && !isdigit(ch)) ch = nc();
12     if (ch == EOF) return false;
13     for (v = 0; isdigit(ch); ch = nc()) v = v * 10 + ch - '0';
14     return true;
15 }
16
17 template <typename T>
18 void o(T p) {
19     static int stk[70], tp;
```

```

20     if (p == 0) {
21         putchar('0');
22         return;
23     }
24     if (p < 0) {
25         p = -p;
26         putchar('-');
27     }
28     while (p) stk[++tp] = p % 10, p /= 10;
29     while (tp) putchar(stk[tp--] + '0');
30 }

```

快速幂

```

1  ll pow(ll x, ll n, ll mod) {
2      assert(n >= 0);
3      ll ret = mod != 1;
4      for (x %= mod; n; n >>= 1, x = x * x % mod)
5          if (n & 1) ret = ret * x % mod;
6      return ret;
7  }
8  ll inv(ll a, ll p) { return pow(a, p - 2, p); }
9  inline ll mul(ll a, ll b, ll mod) {
10     if (mod <= 1000000000)
11         return a * b % mod;
12     else if (mod <= 1000000000000011)
13         return (((a * (b >> 20) % mod) << 20) + (a * (b & ((1 << 20) - 1)))) % mod;
14     else {
15         ll d = (ll)floor(a * (long double)b / mod + 0.5);
16         ll ret = (a * b - d * mod) % mod;
17         if (ret < 0) ret += mod;
18         return ret;
19     }
20 }

```

计算几何

二维几何基础

```
1  /***** 基本定义 *****/
2
3  // 玄学 eps, 可手动调
4  const double eps = 1e-10;
5  // 精度较高的 PI
6  const double pi = acos(-1.0);
7  // 三态函数, 绝对值小于 eps 时认为是 0
8  int dcmp(double x) {
9      if (fabs(x) < eps) return 0;
10     else return x<0?-1:1;
11 }
12
13 /***** 点操作 *****/
14
15 // 点定义
16 struct Point {
17     double x, y;
18     Point(double x=0, double y=0):x(x),y(y) {} // 构造函数
19 };
20
21 // 点比较 (先比较 x 坐标, 再比较 y 坐标)
22 bool operator < (const Point &a, const Point &b) {
23     return a.x<b.x || (a.x==b.x && a.y<b.y);
24 }
25 // 点相等
26 bool operator == (const Point &a, const Point &b) {
27     return dcmp(a.x-b.x)==0 && dcmp(a.y-b.y)==0;
28 }
29
30 /***** 向量操作 *****/
31
32 // 程序实现上, Vector 只是 Point 的别名
33 typedef Point Vector;
34
35 // 向量加法
36 Vector operator + (Vector A, Vector B) {
37     return Vector(A.x+B.x, A.y+B.y);
38 }
39 // 向量减法
40 Vector operator - (Vector A, Vector B) {
41     return Vector(A.x-B.x, A.y-B.y);
42 }
43 // 向量数乘
44 Vector operator * (Vector A, double p) {
45     return Vector(A.x*p, A.y*p);
46 }
47 // 向量数除
```

```

48 Vector operator / (Vector A, double p) {
49     return Vector(A.x/p, A.y/p);
50 }
51 // 向量点积
52 double Dot(Vector A, Vector B) {
53     return A.x*B.x + A.y*B.y;
54 }
55 // 向量长度
56 double Length(Vector A) {
57     return sqrt(Dot(A, A));
58 }
59 // 向量夹角
60 double Angle(Vector A, Vector B) {
61     return acos(Dot(A, B) / Length(A) / Length(B));
62 }
63 // 向量叉积
64 double Cross(Vector A, Vector B) {
65     return A.x*B.y - A.y*B.x;
66 }
67 // 向量旋转
68 Vector Rotate(Vector A, double rad) {
69     return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
70 }
71
72 /***** 距离 *****/
73
74 // 两点间欧几里得距离
75 double Odistance(Point a, Point b) {
76     return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
77 }
78 // 两点间曼哈顿距离  $|a.x-b.x|+|a.y-b.y|$ 
79 double Mdistance(Point a, Point b) {
80     return fabs(a.x-b.x) + fabs(a.y-b.y);
81 }
82 // 两点间切比雪夫距离  $\max(|a.x-b.x|, |a.y-b.y|)$ 
83 double Cdistance(Point a, Point b) {
84     return max(fabs(a.x-b.x), fabs(a.y-b.y));
85 }
86 // 曼哈顿距离和切比雪夫距离的转换 (见下)
87
88 // 点到直线距离
89 double DistanceToLine(Point P, Point A, Point B) {
90     Vector v1 = B-A, v2 = P-A;
91     // 去掉绝对值得到有向线段
92     return fabs(Cross(v1, v2)) / Length(v1);
93 }
94 // 点到线段距离, 垂线不在线段上时距离为 PA 或 PB 的长度
95 double DistanceToSegment(Point P, Point A, Point B) {
96     if (A == B) return Length(P-A);
97     Vector v1 = B-A, v2 = P-A, v3 = P-B;
98     if (dcmp(Dot(v1, v2)) < 0) return Length(v2);
99     else if (dcmp(Dot(v1, v3)) < 0) return Length(v3);

```

```

100     else return fabs(Cross(v1, v2)) / Length(v1);
101 }
102
103 /***** 求面积 *****/
104
105 // 三角形两倍有向面积，可能为负！
106 double Area2(Point A, Point B, Point C) {
107     return Cross(B-A, C-A);
108 }
109 // 多边形有向面积，可能为负！
110 double PolygonArea(Point *p, int n) {
111     double area = 0.0;
112     for (int i = 1; i < n-1; i++)
113         area += Cross(p[i]-p[0], p[i+1]-p[0]);
114     return area/2.0;
115 }
116
117 /***** 相交判定 *****/
118 // 直线相交，注意 Cross(v,w) 不能为 0，为 0 两直线平行
119 Point GetLineInt(Point P, Vector v, Point Q, Vector w) {
120     Vector u = P - Q;
121     double t = Cross(w,u) / Cross(v,w);
122     return P+v*t;
123 }
124 // 点在线上
125 bool OnSegment(Point p, Point a1, Point a2) {
126     return dcmp(Cross(a1-p, a2-p))==0 && dcmp(Dot(a1-p, a2-p)) < 0;
127 }
128 // 线段相交，线段 a1a2, b1b2
129 // 注意不包含端点，端点需用 OnSegment 特判
130 bool SegProInt(Point a1, Point a2, Point b1, Point b2) {
131     double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1),
132     c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
133     return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
134 }
135 // 线段与直线相交，直线 a1a2, 线段 b1b2
136 bool SegLineInt(Point a1, Point a2, Point b1, Point b2) {
137     double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
138     return dcmp(c1)*dcmp(c2)<=0;
139 }

```

距离

欧氏距离：一般也称作欧几里得距离。在平面直角坐标系中，设点 A, B 的坐标分别为 $(x_1, y_1), (x_2, y_2)$ 则两点间的欧氏距离为：

$$|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

曼哈顿距离：在二维空间内，两个点之间的曼哈顿距离为它们横坐标之差的绝对值与纵坐标之差的绝对值之和。设点 $A(x_1, y_1), B(x_2, y_2)$ ，则 A, B 之间的曼哈顿距离用公式可以表示为：

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

切比雪夫距离：在二维空间内，两个点之间的切比雪夫距离为它们横坐标之差的绝对值与纵坐标之差的绝对值的最大值。设点 $A(x_1, y_1), B(x_2, y_2)$ ，则 A, B 之间的切比雪夫距离用公式可以表示为：

$$d(A, B) = \max(|x_1 - x_2|, |y_1 - y_2|)$$

曼哈顿距离与切比雪夫距离的转换：曼哈顿坐标系是通过切比雪夫坐标系旋转 45° 后，再缩小到原来的一半得到的。

- 将一个点 (x, y) 的坐标变为 $(x + y, x - y)$ 后，原坐标系中的曼哈顿距离等于新坐标系中的切比雪夫距离
- 将一个点 (x, y) 的坐标变为 $(\frac{x+y}{2}, \frac{x-y}{2})$ 后，原坐标系中的切比雪夫距离等于新坐标系中的曼哈顿距离

```
1 // 两点间欧几里得距离
2 double Odistance(Point a, Point b) {
3     return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
4 }
5 // 两点间曼哈顿距离 |a.x-b.x|+|a.y-b.y|
6 double Mdistance(Point a, Point b) {
7     return fabs(a.x-b.x) + fabs(a.y-b.y);
8 }
9 // 两点间切比雪夫距离 max(|a.x-b.x|, |a.y-b.y|)
10 double Cdistance(Point a, Point b) {
11     return max(fabs(a.x-b.x), fabs(a.y-b.y));
12 }
```

极角排序

```
1 // 利用 atan2, 精度低, 速度快
2 bool cmp1(Point p1, Point p2) {
3     if (atan2(p1.y,p1.x) == atan2(p2.y,p2.x))
4         return p1.x<p2.x;
5     return atan2(p1.y,p1.x) < atan2(p2.y,p2.x);
6 }
7
8 // 利用叉积, 精度高, 速度慢
9 bool cmp2(Point p1, Point p2) {
10     Point c(0,0);
11     if (dcmp(Cross(p1-c, p2-c))==0)
12         return p1.x<p2.x;
13     return dcmp(Cross(p1-c, p2-c))>0;
14 }
15
16 // 象限排序, 注意包含四个坐标轴
17 int Quadrant(Point p) {
18     if (p.x>0 && p.y>=0) return 1;
19     if (p.x<=0 && p.y>0) return 2;
20     if (p.x<0 && p.y<=0) return 3;
21     if (p.x>=0 && p.y<0) return 4;
22 }
23
24 // 先按象限排, 再按极角排
25 bool cmp3(Point p1, Point p2) {
```



```

26     if (Quadrant(p1)==Quadrant(p2))
27         return cmp1(p1, p2);
28     return Quadrant(p1)<Quadrant(p2);
29 }

```

Pick 定理

Pick 定理：给定顶点坐标均是整点（或正方形格子点）的简单多边形，皮克定理说明了其面积 A 和内部格点数目 i 、边上格点数目 b 的关系： $A = i + \frac{b}{2} - 1$ 。

它有以下推广：

- 取格点的组成图形的面积为一单位。在平行四边形格点，皮克定理依然成立。套用于任意三角形格点，皮克定理则是 $A = 2 \times i + b - 2$ 。
- 对于非简单的多边形 P ，皮克定理 $A = i + \frac{b}{2} - \chi(P)$ ，其中 $\chi(P)$ 表示 P 的 **欧拉特征数**。
- 高维推广：Ehrhart 多项式
- 皮克定理和 **欧拉公式** $V - E + F = 2$ 等价。

边上格点数目 b 求法：以格子点为顶点的线段，覆盖的点的个数为 $\gcd(dx, dy)$ ，其中， dx, dy 分别为线段横向占的点数和纵向占的点数。如果 dx 或 dy 为 0，则覆盖的点数为 dy 或 dx 。

```

1  struct Point {
2      int x, y;
3      Point(int x=0, int y=0):x(x),y(y) {} // 构造函数
4  };
5  typedef Point Vector;
6  Vector operator - (Vector A, Vector B) {
7      return Vector(A.x-B.x, A.y-B.y);
8  }
9  int Cross(Vector A, Vector B) {
10     return A.x*B.y - A.y*B.x;
11 }
12 int getb(Point A, Point B) {
13     int dx=abs(A.x-B.x),dy=abs(A.y-B.y);
14     if (dx==0) return dy;
15     if (dy==0) return dx;
16     return __gcd(dx,dy);
17 }
18 int Area2(Point A, Point B, Point C) {
19     return abs(Cross(B-A, C-A));
20 }
21 int calA(int i, int b) {
22     return i+b/2-1;
23 }
24 int cali(int A, int b) {
25     return A-b/2+1;
26 }

```

凸包

Andrew 算法：

```

1  int Andrew(Point *p, int n, Point *ch) {
2      sort(p, p+n); // 需要重载 < 号
3      int m = 0;
4      for (int i=0; i<n; i++) {
5          while(m>1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
6          ch[m++]=p[i];
7      }
8      int k=m;
9      for (int i=n-2; i>=0; i--) {
10         while(m>k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
11         ch[m++]=p[i];
12     }
13     if (n>1) m--;
14     return m;
15 }

```

最小圆覆盖

扫描线

扫描线一般运用在图形上面，它和它的字面意思十分相似，就是一条线在整个图上扫来扫去，它一般被用来解决图形面积，周长等问题。

例题：「HDU1542」 Atlantis

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <cstring>
4  #define left rt<<1
5  #define right rt<<1|1
6  #define MAXN 1005
7  using namespace std;
8
9  double s[MAXN<<2];
10
11 struct SegTree {
12     double l,r,sum;
13     int lazy;
14 }tree[MAXN<<2];
15
16 struct node {
17     double x,y1,y2;
18     int flag;
19     node(double x=0,double y1=0,double y2=0,int flag=0):x(x),y1(y1),y2(y2),flag(flag)
20     ↪ {}
21 }p[MAXN<<2];
22
23 bool cmp(node a, node b) {
24     return a.x<b.x;
25 }
26
27 void pushup(int rt) {

```

```

27     if (tree[rt].lazy) {
28         tree[rt].sum=tree[rt].r-tree[rt].l;
29     } else {
30         tree[rt].sum=tree[left].sum+tree[right].sum;
31     }
32 }
33
34
35 void build(int rt,int l,int r) {
36     tree[rt].lazy=0; tree[rt].sum=0;
37     tree[rt].l=s[l]; tree[rt].r=s[r];
38     if (l==r) {
39         return;
40     }
41     int m=(l+r)>>1;
42     build(left,l,m);
43     build(right,m,r);
44     pushup(rt);
45 }
46
47 void update(int rt,double y1,double y2,int flag) {
48     if (tree[rt].l==y1 && tree[rt].r==y2) {
49         tree[rt].lazy+=flag;
50         pushup(rt);
51         return;
52     }
53     if (tree[left].r>y1)
54         update(left,y1,min(tree[left].r,y2),flag);
55     if (tree[right].l<y2)
56         update(right,max(tree[right].l,y1),y2,flag);
57     pushup(rt);
58 }
59
60 int main() {
61     int cnt=0,n;
62     double x1,y1,x2,y2,ans;
63     while (scanf("%d",&n) && n) {
64         ans=0;
65         for (int i=0;i<n;i++) {
66             scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
67             p[i]=node(x1,y1,y2,1);
68             p[i+n]=node(x2,y1,y2,-1);
69             s[i+1]=y1;
70             s[i+n+1]=y2;
71         }
72         sort(s+1,s+1+2*n); //离散化
73         sort(p,p+2*n,cmp);
74         build(1,1,2*n);
75         update(1,p[0].y1,p[0].y2,p[0].flag);
76         for (int i=1;i<2*n;i++) {
77             ans+=(p[i].x-p[i-1].x)*tree[1].sum;
78             update(1,p[i].y1,p[i].y2,p[i].flag);

```

```
79         }
80         printf("Test case #%d\n",++cnt);
81         printf("Total explored area: %.21f\n\n",ans);
82     }
83     return 0;
84 }
```

数据结构

树状数组

单点修改，区间查询

```
1  #define MAXN 1000005
2  ll BIT[MAXN], A[MAXN];
3
4  inline int lowbit(int x) { return x&-x; }
5
6  void update(int p, ll x) {
7      while(p<=n) BIT[p]+=x, p+=lowbit(p);
8  }
9
10 // A[1]+A[2]...A[p] 的和
11 ll query(int p) {
12     ll ret=0;
13     while(p) ret+=BIT[p], p-=lowbit(p);
14     return ret;
15 }
16
17 // 读入
18 for (int i=1; i<=n; i++) {
19     scanf("%lld", &A[i]);
20     update(i, A[i]);
21 }
22
23 // 查询 [l, r]
24 query(r)-query(l-1)
```

区间修改，单点查询

```
1  #define MAXN 1000005
2  ll BIT[MAXN];
3  ll A[MAXN], D[MAXN];
4
5  inline int lowbit(int x) { return x&-x; }
6
7  void update(int p, ll x) {
8      while(p<=n) BIT[p]+=x, p+=lowbit(p);
9  }
10
11 // A[i]=D[1]+D[2]...+D[i]
12 ll query(int p) {
13     ll ret=0;
14     while(p) ret+=BIT[p], p-=lowbit(p);
15     return ret;
16 }
17
18 // 更新 [l, r], +x
```

```

19 update(l,x),update(r+1,-x);
20
21 // 查询点 A[p]
22 query(p);

```

区间修改，区间查询

```

1  #define MAXN 1000005
2  ll BIT[MAXN],BIT1[MAXN];
3  ll A[MAXN],D[MAXN];
4
5  inline int lowbit(int x) { return x&-x; }
6
7  void update(int p,ll x) {
8      for (int i=p;i<=n;i+=lowbit(i))
9          BIT[i]+=x,BIT1[i]+=x*p;
10 }
11
12 ll query(int p) {
13     ll ret=0;
14     for (int i=p;i>=1;i-=lowbit(i))
15         ret+=BIT[i]*(p+1)-BIT1[i];
16     return ret;
17 }
18
19 // 更新 [l,r],+x
20 update(l,x),update(r+1,-x);
21
22 // 查询 [l,r]
23 query(r)-query(l-1)

```

线段树

单点修改

```

1  #define left rt<<1
2  #define right rt<<1|1
3  #define MAXN 100005
4
5  struct SegTree {
6      int l,r;
7      ll sum;
8  }t[MAXN<<2];
9
10 void pushup(int rt) {
11     t[rt].sum=t[left].sum+t[right].sum;
12 }
13
14 void build(int l, int r, int rt) {
15     t[rt].sum=0;

```

```

16     t[rt].l=l; t[rt].r=r;
17     if (l==r) {
18         scanf("%lld",&t[rt].sum);
19         return;
20     }
21     int m=(l+r)>>1;
22     build(l,m,left);
23     build(m+1,r,right);
24     pushup(rt);
25 }
26
27 void update(int p, ll c, int rt) {
28     if (t[rt].l==t[rt].r) {
29         t[rt].sum+=c;
30         return;
31     }
32     int m=(t[rt].l+t[rt].r)>>1;
33     if (p<=m) update(p,c,left);
34     else update(p,c,right);
35     pushup(rt);
36 }
37
38 ll query(int L, int R, int rt) {
39     if (L<=t[rt].l && t[rt].r<=R) {
40         return t[rt].sum;
41     }
42     int m=(t[rt].l+t[rt].r)>>1;
43     ll ret=0;
44     if (L<=m) ret+=query(L,R,left);
45     if (m<R) ret+=query(L,R,right);
46     return ret;
47 }

```

区间修改

```

1  #define left rt<<1
2  #define right rt<<1|1
3  #define MAXN 100005
4
5  struct SegTree {
6      int l,r;
7      ll sum,lazy;
8  }t[MAXN<<2];
9
10 void pushup(int rt) {
11     t[rt].sum=t[left].sum+t[right].sum;
12 }
13
14 void pushdown(int rt) {
15     int m=t[rt].r-t[rt].l+1;
16     if (t[rt].lazy) {

```

```

17         t[left].sum+=t[rt].lazy*(m-(m>>1));
18         t[right].sum+=t[rt].lazy*(m>>1);
19         t[left].lazy+=t[rt].lazy;
20         t[right].lazy+=t[rt].lazy;
21         t[rt].lazy=0;
22     }
23 }
24
25 void build(int l, int r, int rt) {
26     t[rt].lazy=t[rt].sum=0;
27     t[rt].l=l; t[rt].r=r;
28     if (l==r) {
29         scanf("%lld",&t[rt].sum);
30         return;
31     }
32     int m=(l+r)>>1;
33     build(l,m,left);
34     build(m+1,r,right);
35     pushup(rt);
36 }
37
38 void update(int L, int R, ll c, int rt) {
39     if (L<=t[rt].l && t[rt].r<=R) {
40         t[rt].lazy+=c;
41         t[rt].sum+=c*(t[rt].r-t[rt].l+1);
42         return;
43     }
44     pushdown(rt);
45     int m=(t[rt].l+t[rt].r)>>1;
46     if (L<=m) update(L,R,c,left);
47     if (m<R) update(L,R,c,right);
48     pushup(rt);
49 }
50
51 ll query(int L, int R, int rt) {
52     if (L<=t[rt].l && t[rt].r<=R) {
53         return t[rt].sum;
54     }
55     pushdown(rt);
56     int m=(t[rt].l+t[rt].r)>>1;
57     ll ret=0;
58     if (L<=m) ret+=query(L,R,left);
59     if (m<R) ret+=query(L,R,right);
60     return ret;
61 }

```