Standard Code Library

Radium

December 7, 2019

Contents

计算几何

二维几何基础

```cpp
/*************** 基本定义 ***************/

// 玄学 eps，可手动调
const double eps=1e-10;
// 精度较高的 PI
const double pi=acos(-1.0);
// 三态函数，绝对值小于 eps 时认为是 0
int dcmp(double x) {
    if (fabs(x)<eps) return 0;
    else return x<0?-1:1;
}

/*************** 点操作 ***************/

// 点定义
struct Point {
    double x, y;
    Point(double x=0, double y=0):x(x),y(y) {} // 构造函数
};

// 点比较（先比较 x 坐标，再比较 y 坐标）
bool operator < (const Point &a, const Point &b) {
    return a.x<b.x || (a.x==b.x && a.y<b.y);
}
// 点相等
bool operator == (const Point &a, const Point &b) {
    return dcmp(a.x-b.x)==0 && dcmp(a.y-b.y)==0;
}

/*************** 向量操作 ***************/

// 程序实现上，Vector 只是 Point 的别名
typedef Point Vector;

// 向量加法
Vector operator + (Vector A, Vector B) {
    return Vector(A.x+B.x, A.y+B.y);
}
// 向量减法
Vector operator - (Vector A, Vector B) {
    return Vector(A.x-B.x, A.y-B.y);
}
// 向量数乘
Vector operator * (Vector A, double p) {
    return Vector(A.x*p, A.y*p);
}
// 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  int Odistance2(Point a, Point b) {
79      return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y)
80  }
81  // 两点间曼哈顿距离 |a.x-b.x|+|a.y-b.y|
82  double Mdistance(Point a, Point b) {
83      return fabs(a.x-b.x) + fabs(a.y-b.y);
84  }
85  // 两点间切比雪夫距离 max(|a.x-b.x|,|a.y-b.y|)
86  double Cdistance(Point a, Point b) {
87      return max(fabs(a.x-b.x), fabs(a.y-b.y));
88  }
89  // 曼哈顿距离和切比雪夫距离的转换（见下）
90
91  // 点到直线距离
92  double DistanceToLine(Point P, Point A, Point B) {
93      Vector v1=B-A, v2=P-A;
94      // 去掉绝对值得到有向线段
95      return fabs(Cross(v1, v2)) / Length(v1);
96  }
97  // 点到线段距离，垂线不在线段上时距离为 PA 或 PB 的长度
98  double DistanceToSegment(Point P, Point A, Point B) {
99      if (A==B) return Length(P-A);
```

```
    Vector v1=B-A, v2=P-A, v3=P-B;
    if (dcmp(Dot(v1, v2))<0) return Length(v2);
    else if (dcmp(Dot(v1, v3))<0) return Length(v3);
    else return fabs(Cross(v1, v2)) / Length(v1);
}

/*************** 求面积 ***************/

// 三角形两倍有向面积，可能为负!
double Area2(Point A, Point B, Point C) {
    return Cross(B-A, C-A);
}
// 多边形有向面积，可能为负!
double PolygonArea(Point *p, int n) {
    double area=0.0;
    for (int i=1; i<n-1; i++)
        area+=Cross(p[i]-p[0], p[i+1]-p[0]);
    return area/2.0;
}

/*************** 相交判定 ***************/
// 直线相交，注意 Cross(v,w) 不能为 0，为 0 两直线平行
Point GetLineInt(Point P, Vector v, Point Q, Vector w) {
    Vector u=P-Q;
    double t=Cross(w,u) / Cross(v,w);
    return P+v*t;
}
// 点在线上
bool OnSegment(Point p, Point a1, Point a2) {
    return dcmp(Cross(a1-p, a2-p))==0 && dcmp(Dot(a1-p, a2-p)) < 0;
}
// 线段相交，线段 a1a2, b1b2
// 注意不包含端点，端点需用 OnSegment 特判
bool SegProInt(Point a1, Point a2, Point b1, Point b2) {
    double c1=Cross(a2-a1, b1-a1), c2=Cross(a2-a1, b2-a1),
           c3=Cross(b2-b1, a1-b1), c4=Cross(b2-b1, a2-b1);
    return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
}
// 线段与直线相交，直线 a1a2, 线段 b1b2
bool SegLineInt(Point a1, Point a2, Point b1, Point b2) {
    double c1 = Cross(a2-a1,b1-a1), c2 = Cross(a2-a1,b2-a1);
    return dcmp(c1)*dcmp(c2)<=0;
}
```

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 $(x1, y1), (x2, y2)$ 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 $A(x1,y1),B(x2,y2)$ ，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 $A(x1,y1),B(x2,y2)$ ，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

```cpp
// 两点间欧几里得距离
double Odistance(Point a, Point b) {
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
// 两点间曼哈顿距离 |a.x-b.x|+|a.y-b.y|
double Mdistance(Point a, Point b) {
    return fabs(a.x-b.x) + fabs(a.y-b.y);
}
// 两点间切比雪夫距离 max(|a.x-b.x|,|a.y-b.y|)
double Cdistance(Point a, Point b) {
    return max(fabs(a.x-b.x), fabs(a.y-b.y));
}
```

极角排序

```cpp
// 利用 atan2, 精度低, 速度快
bool cmp1(Point a, Point b) {
    if (atan2(a.y, a.x) == atan2(b.y, b.x))
        return a.x<b.x;
    return atan2(a.y, a.x) < atan2(b.y, b.x);
}

// 利用叉积, 精度高, 速度慢
bool cmp2(Point a, Point b) {
    Point c(0, 0);
    if (dcmp(Cross(a-c, b-c))==0)
        return a.x<b.x;
    return dcmp(Cross(a-c, b-c))>0;
}

// 象限排序, 注意包含四个坐标轴
int Quadrant(Point p) {
    if (p.x>0 && p.y>=0)  return 1;
    if (p.x<=0 && p.y>0)  return 2;
    if (p.x<0 && p.y<=0)  return 3;
    if (p.x>=0 && p.y<0)  return 4;
}
```

```
23
24  // 先按象限排，再按极角排
25  bool cmp3(Point a, Point b) {
26      if (Quadrant(a)==Quadrant(b))
27          return cmp1(a, b);
28      return Quadrant(a)<Quadrant(b);
29  }
```

Pick 定理

Pick 定理：给定顶点座标均是整点（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：

```
struct Point {
    int x, y;
    Point(int x=0, int y=0):x(x),y(y) {}
}p[MAXN],ch[MAXN];

typedef Point Vector;

bool operator < (const Point &a, const Point &b) {
    return a.x<b.x || (a.x==b.x && a.y<b.y);
}

Vector operator - (Vector A, Vector B) {
    return Vector(A.x-B.x, A.y-B.y);
}

int Cross(Vector A, Vector B) {
    return A.x*B.y - A.y*B.x;
}

// 核心代码
int Andrew(Point *p, int n, Point *ch) {
    sort(p, p+n); // 需要重载 < 号
    int m=0;
    for (int i=0; i<n; i++) {
        while(m>1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++]=p[i];
    }
    int k=m;
    for (int i=n-2; i>=0; i--) {
        while(m>k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++]=p[i];
    }
    if (n>1) m--;
    return m;
}
```

平面最近点对

给定 n 个二维平面上的点，求一组欧几里得距离最近的点对。分治算法，时间复杂度 $O(n \log n)$。

```
struct Point {
    int x, y, id;
};

bool cmpx(Point a, Point b) {
    if (a.x==b.x) return a.y<b.y;
    return a.x<b.x;
}
```

```

bool cmpy(Point a, Point b) {
    return a.y<b.y;
}

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

int n;
Point p[MAXN];
double minDis;
int ansa, ansb;

void updateAns(Point a, Point b) {
    double Dis=Odistance(a, b);
    if (Dis<minDis) minDis=Dis,ansa=a.id,ansb=b.id;
}

void rec(int l, int r) {
    if (r-l<=3) {
        for (int i=l; i<=r; i++)
            for (int j=i+1; j<=r; j++)
                updateAns(p[i], p[j]);
        sort(p+l, p+r+1, cmpy);
        return;
    }

    int m=(l+r)>>1;
    int midx=p[m].x;
    rec(l,m), rec(m+1,r);
    static Point t[MAXN];
    merge(p+l, p+m+1, p+m+1, p+r+1, t, cmpy);
    copy(t, t+r-l+1, p+l);

    int tsz=0;
    for (int i=l; i<=r; i++) {
        if (abs(p[i].x-midx) < minDis) {
            for (int j=tsz-1; j>=0 && p[i].y-t[j].y<minDis; j--)
                updateAns(p[i], t[j]);
            t[tsz++]=p[i];
        }
    }
}

int main() {
    scanf("%d",&n);
    for (int i=0; i<n; i++) {
        scanf("%d %d",&p[i].x, &p[i].y);
        p[i].id=i;
    }
    // 计算答案
```

```
    sort(p, p+n, cmpx);
    minDis=1E20;
    rec(0, n-1);

    if (ansa>ansb) swap(ansa, ansb);
    printf("%d %d %.6f\n",ansa, ansb, minDis);
    return 0;
}
```

旋转卡壳

也能用来求平面最远 (近) 点对，时间复杂度 $O(n)$，需先求出凸包。

```
// 先抄一遍凸包

// 这里求的是距离的平方
int Odistance2(Point a, Point b) {
    return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
}

// 传入凸包，返回答案
int RotCal(Point* ch, int m) {
    ch[m]=ch[0]; int j=1;
    int ret=0;
    for(int i=0 ;i<m; i++) {

        while(abs(Cross(ch[i]-ch[j],ch[i+1]-ch[j]))<abs(Cross(ch[i]-ch[j+1],ch[i+1]-ch[j+1])))
        {
            j=(j+1)%m;
        }
        ret=max(ret,Odistance2(ch[i], ch[j]));
    }
    return ret;
}
```

半平面交

```
// 点定义
struct Point {
    double x, y;
    Point(double x=0, double y=0):x(x),y(y) {} // 构造函数
}poly[MAXN];

typedef Point Vector;

// 向量加法
Vector operator + (Vector A, Vector B) {
    return Vector(A.x+B.x, A.y+B.y);
}
// 向量减法
Vector operator - (Vector A, Vector B) {
```

```
    return Vector(A.x-B.x, A.y-B.y);
}
// 向量数乘
Vector operator * (Vector A, double p) {
    return Vector(A.x*p, A.y*p);
}

// 向量叉积
double Cross(Vector A, Vector B) {
    return A.x*B.y - A.y*B.x;
}

// 有向线段
struct Line {
    Point P;
    Vector v;
    double ang;
    Line() {}
    Line (Point P, Vector v):P(P),v(v) {
        ang=atan2(v.y, v.x); // 求极角
    }
    bool operator < (const Line &L) const {
        return ang < L.ang; // 用于极角排序
    }
};

// 点 p 在有向线段 L 的左边 (线上不算)
bool OnLeft(Line L, Point p) {
    return Cross(L.v, p-L.P) > 0;
}

// 两直线交点
Point GetInt(Line a, Line b) {
    Vector u = a.P - b.P;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.P+a.v*t;
}

// 求半平面交
int HalfPlaneInt(Line *L, int n, Point *poly) {
    sort(L, L+n); // 极角排序
    int first, last;
    Point *p = new Point[n];
    Line *q = new Line[n];
    q[first=last=0] = L[0];
    for (int i=1; i<n; i++) {
        while (first < last && !OnLeft(L[i], p[last-1])) last--;
        while (first < last && !OnLeft(L[i], p[first])) first++;
        q[++last] = L[i];
        if (fabs(Cross(q[last].v, q[last-1].v)) < eps) {
            last--;
            if (OnLeft(q[last], L[i].P)) q[last] = L[i];
```

```
        }
        if (first < last) p[last-1] = GetInt(q[last-1], q[last]);
    }
    while (first < last && !OnLeft(q[first], p[last-1])) last--;
    if (last - first <= 1) return 0;
    p[last] = GetInt(q[last], q[first]);

    int m=0;
    for (int i=first; i<=last; i++) poly[m++] = p[i];
    return m;
}
```

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

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#define left rt<<1
#define right rt<<1|1
#define MAXN 1005
using namespace std;

double s[MAXN<<2];

struct SegTree {
    double l, r, sum;
    int lazy;
}tree[MAXN<<2];

struct node {
    double x, y1, y2;
    int flag;
    node(double x=0,double y1=0,double y2=0,int flag=0):x(x),y1(y1),y2(y2),flag(flag)
    ↪ {}
}p[MAXN<<2];

bool cmp(node a, node b) {
    return a.x<b.x;
}

void pushup(int rt) {
    if (tree[rt].lazy) {
        tree[rt].sum=tree[rt].r-tree[rt].l;
    } else {
        tree[rt].sum=tree[left].sum+tree[right].sum;
    }
}
```

```
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 (r-l>1) {
39          int m=(l+r)>>1;
40          build(left, l, m);
41          build(right, m, r);
42          pushup(rt);
43      }
44  }
45
46  void update(int rt, double y1, double y2, int flag) {
47      if (tree[rt].l==y1 && tree[rt].r==y2) {
48          tree[rt].lazy+=flag;
49          pushup(rt);
50          return;
51      }
52      if (tree[left].r>y1)
53          update(left, y1, min(tree[left].r, y2), flag);
54      if (tree[right].l<y2)
55          update(right, max(tree[right].l, y1), y2, flag);
56      pushup(rt);
57  }
58
59  int main() {
60      int cnt=0, n;
61      double x1, y1, x2, y2, ans;
62      while (scanf("%d",&n) && n) {
63          ans=0;
64          for (int i=0; i<n; i++) {
65              scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
66              p[i]=node(x1,y1,y2,1);
67              p[i+n]=node(x2,y1,y2,-1);
68              s[i+1]=y1;
69              s[i+n+1]=y2;
70          }
71          sort(s+1, s+1+2*n); //离散化
72          sort(p, p+2*n, cmp);
73          build(1, 1, 2*n);
74          update(1, p[0].y1, p[0].y2, p[0].flag);
75          for (int i=1; i<2*n; i++) {
76              ans+=(p[i].x-p[i-1].x)*tree[1].sum;
77              update(1, p[i].y1, p[i].y2, p[i].flag);
78          }
79          printf("Test case #%d\n",++cnt);
80          printf("Total explored area: %.2lf\n\n",ans);
81      }
82      return 0;
83  }
```

最小球覆盖

模拟退火，玄学算法。

```
// delta 接近 1 但是小于 1，越接近 1 时间越久
// T (初始温度) 可调大
#define eps 1e-8
#define T 100
#define delta 0.98
#define INF 1e30

int n;
struct point3D
{
    double x, y, z;
}p[105];

double dist(point3D A, point3D 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() {
    point3D s=p[0];
    double t=T;
    double ans=INF;
    while(t>eps) {
        int k=0;
        for (int i=0; i<n; i++)
            if (dist(s, p[i]) > dist(s, p[k]))
                k=i;
        double d=dist(s, p[k]);
        ans=min(ans,d);
        s.x+=(p[k].x-s.x)/d*t;
        s.y+=(p[k].y-s.y)/d*t;
        s.z+=(p[k].z-s.z)/d*t;
        t*=delta;
    }
    return ans;
}
```

数据结构

并查集

```
// 初始化
void init() {
    for (int i=0; i<=n; i++)
        fa[i]=i;
}

// 查询
int find(int x) {
    if (fa[x]==x) return x;
    else return fa[x]=find(fa[x]);
}

// 合并
void combine(int x, int y) {
    x=find(x), y=find(y);
    if (x==y) return;
    fa[x]=y;
}

// 启发式合并
int size[MAXN];
void combine(int x, int y) {
    x=find(x), y=find(y);
    if (x==y) return;
    if (size[x]>size[y])
        swap(x,y);
    fa[x]=y;
    size[y]+=size[x];
}
```

树状数组

单点修改，区间查询

```
#define MAXN 1000005
ll BIT[MAXN],A[MAXN];

inline int lowbit(int x) { return x&-x; }

void update(int p, ll x) {
    while(p<=n) BIT[p]+=x,p+=lowbit(p);
}

// A[1]+A[2]...A[p] 的和
ll query(int p) {
    ll ret=0;
    while(p) ret+=BIT[p],p-=lowbit(p);
```

```
    return ret;
}

// 读入
for (int i=1; i<=n; i++) {
    scanf("%lld",&A[i]);
    update(i,A[i]);
}

// 查询 [l,r]
query(r)-query(l-1)
```

区间修改，单点查询

```
#define MAXN 1000005
ll BIT[MAXN];
ll A[MAXN],D[MAXN];

inline int lowbit(int x) { return x&-x; }

void update(int p, ll x) {
    while(p<=n) BIT[p]+=x,p+=lowbit(p);
}

// A[i]=D[1]+D[2]...+D[i]
ll query(int p) {
    ll ret=0;
    while(p) ret+=BIT[p],p-=lowbit(p);
    return ret;
}

// 更新 [l,r],+x
update(l,x),update(r+1,-x);

// 查询点 A[p]
query(p);
```

区间修改，区间查询

```
#define MAXN 1000005
ll BIT[MAXN],BIT1[MAXN];
ll A[MAXN],D[MAXN];

inline int lowbit(int x) { return x&-x; }

void update(int p, ll x) {
    for (int i=p; i<=n; i+=lowbit(i))
        BIT[i]+=x,BIT1[i]+=x*p;
}

ll query(int p) {
```

```
    ll ret=0;
    for (int i=p; i; i-=lowbit(i))
        ret+=BIT[i]*(p+1)-BIT1[i];
    return ret;
}

// 更新 [l,r],+x
update(l,x),update(r+1,-x);

// 查询 [l,r]
query(r)-query(l-1)
```

线段树

单点修改

```
#define left rt<<1
#define right rt<<1|1
#define MAXN 100005

struct SegTree {
    int l, r;
    ll sum;
}t[MAXN<<2];

void pushup(int rt) {
    t[rt].sum=t[left].sum+t[right].sum;
}

void build(int l, int r, int rt) {
    t[rt].sum=0;
    t[rt].l=l; t[rt].r=r;
    if (l==r) {
        scanf("%lld",&t[rt].sum);
        return;
    }
    int m=(l+r)>>1;
    build(l, m, left);
    build(m+1, r, right);
    pushup(rt);
}

void update(int p, ll c, int rt) {
    if (t[rt].l==t[rt].r) {
        t[rt].sum+=c;
        return;
    }
    int m=(t[rt].l+t[rt].r)>>1;
    if (p<=m) update(p, c, left);
    else update(p, c, right);
    pushup(rt);
```

```
}

ll query(int L, int R, int rt) {
    if (L<=t[rt].l && t[rt].r<=R) {
        return t[rt].sum;
    }
    int m=(t[rt].l+t[rt].r)>>1;
    ll ret=0;
    if (L<=m) ret+=query(L, R, left);
    if (m<R) ret+=query(L, R, right);
    return ret;
}
```

区间修改

```
#define left rt<<1
#define right rt<<1|1
#define MAXN 100005

struct SegTree {
    int l, r;
    ll sum, lazy;
}t[MAXN<<2];

void pushup(int rt) {
    t[rt].sum=t[left].sum+t[right].sum;
}

void pushdown(int rt) {
    int m=t[rt].r-t[rt].l+1;
    if (t[rt].lazy) {
        t[left].sum+=t[rt].lazy*(m-(m>>1));
        t[right].sum+=t[rt].lazy*(m>>1);
        t[left].lazy+=t[rt].lazy;
        t[right].lazy+=t[rt].lazy;
        t[rt].lazy=0;
    }
}

void build(int l, int r, int rt) {
    t[rt].lazy=t[rt].sum=0;
    t[rt].l=l; t[rt].r=r;
    if (l==r) {
        scanf("%lld",&t[rt].sum);
        return;
    }
    int m=(l+r)>>1;
    build(l, m, left);
    build(m+1, r, right);
    pushup(rt);
}
```

```
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  }
```

区间合并

```
1   #define left rt<<1
2   #define right rt<<1|1
3   #define MAXN 100005
4
5   struct segTree {
6       int l, r, len;
7       int ls, rs, ms;
8   }t[MAXN<<2];
9
10  void pushup(int rt) {
11      t[rt].ls=t[left].ls;
12      t[rt].rs=t[right].rs;
13      if (t[left].ls==t[left].len) {
14          t[rt].ls+=t[right].ls;
15      }
16      if (t[right].rs==t[right].len) {
17          t[rt].rs+=t[left].rs;
18      }
19      t[rt].ms=max(t[rt].ls, t[rt].rs);
20      t[rt].ms=max(t[rt].ms, t[left].rs+t[right].ls);
21  }
22
23  void build(int l,int r,int rt) {
```

```
    t[rt].l=l; t[rt].r=r; t[rt].len=r-l+1;
    t[rt].ls=t[rt].rs=t[rt].ms=1;
    if (l==r) {
        return;
    }
    int m=(l+r)>>1;
    build(l, m, left);
    build(m+1, r, right);
    pushup(rt);
}

void update(int p,int c,int rt) {
    if (t[rt].l==t[rt].r) {
        t[rt].ls=t[rt].rs=t[rt].ms=c;
        return;
    }
    int m=(t[rt].l+t[rt].r)>>1;
    if (p<=m) update(p, c, left);
    else update(p, c, right);
    pushup(rt);
}

int query(int rt,int p) {
    if (t[rt].l==t[rt].r) {
        return t[rt].ms;
    }
    int ret=0;
    int m=(t[rt].l+t[rt].r)>>1;
    if (p<=m) {
        if (p>=t[left].r-t[left].rs+1) {
            return ret=t[left].rs+t[right].ls;
        } else {
            return query(left, p);
        }
    } else {
        if (t[right].ls+t[right].l-1>=p) {
            return ret=t[right].ls+t[left].rs;
        } else {
            return query(right, p);
        }
    }
}
```

图论

基础

```cpp
/*************** 链式前向星 ***************/
struct Edge {
    int to, next;
    //ll w;
    //bool cut;
}edge[MAXM*2];
int head[MAXN], tot;
// 别忘记初始化!
inline void init() {
    tot=0;
    memset(head,-1,sizeof(head));
}
// 加边, 无向图记得加反向边
inline void addedge(int u, int v) {
    edge[tot].to=v;
    //edge[tot].w=w;
    edge[tot].next=head[u];
    head[u]=tot++;
}
// 遍历
for (int i=head[u]; i!=-1; i=edge[i].next)


/*************** 去重边 ***************/
// 先抄一遍链式前向星

struct Edge1 {
    int u, v;
    ll w;
}e[MAXM];

bool cmp(Edge1 e1, Edge1 e2) {
    if (e1.u==e2.u && e1.v==e2.v)
        return e1.w<e2.w;
    if (e1.u==e2.u)
        return e1.v<e2.v;
    return e1.u<e2.u;
}

int main() {
    int u, v; ll w;
    for (int i=1; i<=M; i++) {
        scanf("%d%d%lld", &u, &v, &w);
        e[i].u=u; e[i].v=v; e[i].w=w;
    }
    sort(e+1, e+1+M, cmp);
    for (int i=1; i<=M; i++) {
```

```
        if (i!=1 && e[i].u==e[i-1].u && e[i].v==e[i-1].v)
            continue;
        u=e[i].u; v=e[i].v; w=e[i].w;
        addedge(u, v, w);
        addedge(v,u,w);
    }
    return 0;
}
```

分层图

写法 1:

```
#define pii pair<ll, int>
const ll inf=0x3f3f3f3f;
const int MAXN=4500005;
using namespace std;

struct Edge {
    int to,next;
    ll w;
} edge[MAXN];
int head[MAXN],tot;
ll dis[MAXN],ans;
int vis[MAXN];
int n,m,s,t,k;

void init() {
    tot=0;
    memset(head,-1,sizeof(head));
    memset(dis,inf,sizeof(dis));
    memset(vis,0,sizeof(vis));
}

void addedge(int u,int v,ll w) {
    edge[tot].to=v;
    edge[tot].w=w;
    edge[tot].next=head[u];
    head[u]=tot++;
}

void dijkstra() {
    priority_queue<pii,vector<pii>,greater<pii> > q;
    dis[s]=0; q.push({dis[s],s});
    while(!q.empty()) {
        int now=q.top().second;
        q.pop();
        if (vis[now]) continue;
        vis[now]=1;
        for (int i=head[now];i!=-1;i=edge[i].next) {
            int v=edge[i].to;
            if(!vis[v]&&dis[v]>dis[now]+edge[i].w) {
```

```
                dis[v]=dis[now]+edge[i].w;
                q.push({dis[v],v});
            }
        }
    }
}

int main() {
    int T,u,v; ll w;
    scanf("%d",&T);
    while(T--) {
        init();
        cin >> n >> m >> k;
        // 起点为 1, 终点为 n
        s=1; t=n;
        for (int i=1;i<=m;i++) {
            scanf("%d%d%lld",&u,&v,&w);
            for (int j=0;j<=k;j++) {
                addedge(u+j*n,v+j*n,w);
                //addedge(v+j*n,u+j*n,w);
                if (j!=k) {
                    addedge(u+j*n,v+(j+1)*n,0);
                    //addedge(v+j*n,u+(j+1)*n,0);
                }
            }
        }
        ans=1e18;
        dijkstra();
        for (int i=0;i<=k;i++)
            ans=min(ans,dis[t+i*n]);
        cout << ans << endl;
    }
    return 0;
}
```

写法 2:

```
#define pii pair<ll, int>
const ll inf=0x3f3f3f3f;
const int MAXN=300005;
using namespace std;

struct node {
    int x,y;
    ll w;
    node(){}
    node(int _x,int _y,ll _w) {
        x=_x; y=_y; w=_w;
    }
    friend bool operator < (const node a,const node b) {
        return a.w > b.w;
    }
};
```

```
17  struct Edge {
18      int to,next;
19      ll w;
20  } edge[MAXN*4];
21  int head[MAXN*4],tot;
22  ll dis[MAXN][15],ans;
23  int vis[MAXN][15];
24  int n,m,s,t,k;
25
26  void init() {
27      tot=0;
28      memset(head,-1,sizeof(head));
29      memset(dis,inf,sizeof(dis));
30      memset(vis,0,sizeof(vis));
31  }
32
33  void addedge(int u,int v,ll w) {
34      edge[tot].to=v;
35      edge[tot].w=w;
36      edge[tot].next=head[u];
37      head[u]=tot++;
38  }
39
40  void dijkstra() {
41      priority_queue<node> q;
42      dis[s][0]=0;
43      q.push(node(s,0,0));
44      while(!q.empty()) {
45          int x=q.top().x,y=q.top().y;
46          q.pop();
47          if (vis[x][y]) continue;
48          vis[x][y]=1;
49          for (int i=head[x];i!=-1;i=edge[i].next) {
50              int to=edge[i].to;
51              if (dis[x][y]+edge[i].w<dis[to][y]) {
52                  dis[to][y]=dis[x][y]+edge[i].w;
53                  q.push(node(to,y,dis[to][y]));
54              }
55              if (y+1<=k && dis[x][y]<dis[to][y+1]) {
56                  dis[to][y+1]=dis[x][y];
57                  q.push(node(to,y+1,dis[to][y+1]));
58              }
59          }
60      }
61  }
62
63  int main() {
64      int T,u,v; ll w;
65      scanf("%d",&T);
66      while(T--) {
67          init();
68          cin >> n >> m >> k;
```

```
        s=1; t=n;
        for (int i=1;i<=m;i++) {
            scanf("%d%d%lld",&u,&v,&w);
            addedge(u,v,w);
        }
        ans=1e18;
        dijkstra();
        for (int i=0;i<=k;i++)
            ans=min(ans,dis[n][i]);
        cout << ans << endl;
    }
    return 0;
}
```

A*(K 短路)

```
#define MAXN 1005
#define MAXM 200005
#define inf 0x3f3f3f3f
//#define inf 1e9
using namespace std;

inline const int read() {
    register int x=0,f=1;
    register char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9'){x=(x<<3)+(x<<1)+ch-'0';ch=getchar();}
    return x*f;
}
struct Edge{
    int to,w,next;
}e1[MAXM],e2[MAXM];
bool vis[MAXN];
struct node {
    int id; // 当前结点编号
    int f; // 经过当前结点的最短路
    int g; // s-> 当前结点的最短路
    node (int id=0,int f=0,int g=0):id(id),f(f),g(g){}
    bool operator < (const node &a) const {
        if (f==a.f) return g>a.g;
        return f>a.f;
    }
};
int tot,n,m,k,head1[MAXN],head2[MAXN];
int dis[MAXN];
void addedge(int u,int v,int w) {
    e1[tot].to=v;
    e1[tot].w=w;
    e1[tot].next=head1[u];
    head1[u]=tot++;
    e2[tot].to=u;
```

```
    e2[tot].w=w;
    e2[tot].next=head2[v];
    head2[v]=tot++;
}
void init() {
    tot=0;
    memset(head1,-1,sizeof(head1));
    memset(head2,-1,sizeof(head2));
    memset(vis,0,sizeof(vis));
}
void spfa(int s) {
    queue<int> Q;
    memset(dis,inf,sizeof(dis));
    //for (int i=1;i<=n;i++)
    //    dis[i]=1e9;
    dis[s]=0; vis[s]=1;
    Q.push(s);
    while(!Q.empty()) {
        int Now=Q.front(); Q.pop();
        vis[Now]=0;
        for (int i=head2[Now];i!=-1;i=e2[i].next) {
            int v=e2[i].to,w=e2[i].w;
            if (dis[v]>dis[Now]+w) {
                dis[v]=dis[Now]+w;
                if (!vis[v]) {
                    vis[v]=1;
                    Q.push(v);
                }
            }
        }
    }
}
void Astar(int s,int t) {
    if (s==t) k++;
    priority_queue<node> Q;
    Q.push(node(s,0,0));
    int cnt=0;
    while(!Q.empty()) {
        node h=Q.top(); Q.pop();
        if (h.id==t) {
            if (++cnt==k) {
                printf("%d\n",h.f);
                return;
            }
        }
        for (int i=head1[h.id];i!=-1;i=e1[i].next) {
            Q.push(node(e1[i].to,h.g+e1[i].w+dis[e1[i].to],h.g+e1[i].w));
        }
    }
    puts("-1");
}

```

```
88  int main() {
89      int u,v,w,s,t;
90      init();
91      n=read(); m=read();
92      for (int i=1;i<=m;i++) {
93          u=read(); v=read(); w=read();
94          addedge(u,v,w);
95      }
96      s=read(); t=read(); k=read();
97      spfa(t);
98      Astar(s,t);
99      return 0;
100 }
```