POJ 1177 Picture (线段树+离散化+扫描线) 详解

POJ 1177 (线段树+离散化+扫描线)，题目链接为http://poj.org/problem?id=1177

在做本题之前，必须先了解什么是线段树和离散化，请看前一篇博文线段树(segment tree)，里面对线段树和离散化的说明相对比较清楚了。

对于这题，我们的思路步骤如下(代码和下面的文字解释结合着看)：

1.对于输入的N个矩形，有2*N条纵向边，我们把这些边叫做扫描线

2.建立一个struct ScanLine，保留这些扫描线

```
struct ScanLine
{
  int x;//横坐标
  int y1;//扫描线的下端点
  int y2;//扫描线的上端点
  int flag;//若该扫描线属于矩形的左边的竖边，如AB，则叫做入边，值为1，若属于矩形的右边的竖边，如CD，则叫做出边，值为0
};
```

3.建立数组struct ScanLine scan[LEN]；保存输入值，同时用y[LEN]保存所有的纵向坐标

4.对scan数组进行排序，即所有竖边从左往右排序；对y排序并去除重复值，然后离散化，建立线段树。（PS：线段树的node[i].left和node[i].right保存的都是离散化的值，y[node[i].left]和y[node[i].right]保存的就是实际值，这个在代码中很容易理解）

线段树节点struct Node

```
struct Node
{
  int left;
  int right;
  int count;//被覆盖次数
  int line;//所包含的区间数量，如三条[1,2]，[2,3]，[4,5]线段被覆盖，则line=2，因为 [1,2]，[2,3]是连续的。这个是用来辅助计算横边的，如图，在AB和EG之间的横边AK和BL，它们是边界，
  line=1，|AB|+|EG|=2*line*|AB|
```

```
    int lbd;//左端点是否被覆盖，用来辅助对line的计算
    int rbd;//右端点是否被覆盖，用来辅助对line的计算
    int m;//测度，即覆盖的区间长度，如[2,8]就为6
};
```

好的，上面建立了大的框架，然后就开始扫描了。

1.将排序后的scan数组依次输入，执行插入线段insert函数(为入边)或者remove函数(为出边)，同时更新m和line

2.没扫描一次，就要计算一次周长perimeter，这里我们以图中的例子来讲解过程：

首先是AB，它被插入线段树，perimeter = perimeter + |AB|；

然后是EG，它被插入线段树，此时线段树的root节点的测度为|EG|的值，但由于之前之前加过|AB|，因而应该减去|AB|，其实就是减去|KL|，然后再加上line*2*|AK|，这里的line的值是未插入EG时线段树的根节点的line值。

具体代码如下：

```
#include <stdio.h>
#include <algorithm>
#define LEN 10000
using namespace std;

struct Node
{
    int left;
    int right;
    int count;//被覆盖次数
    int line;//所包含的区间数量
    int lbd;//左端点是否被覆盖
    int rbd;//右端点是否被覆盖
    int m;//测度
};

struct ScanLine
{
    int x;
```

```
  int y1;
  int y2;
  int flag;
};

struct Node node[LEN*4];
struct ScanLine scan[LEN];
int y[LEN];

void build(int l, int r, int i)
{
  node[i].left = l;
  node[i].right = r;
  node[i].count = 0;
  node[i].m = 0;
  node[i].line = 0;
  if (r - l > 1)
  {
    int middle = (l + r)/2;
    build(l, middle, 2*i + 1);
    build(middle, r, 2*i + 2);
  }
}

//更新测度m
void update_m(int i)
{
  if (node[i].count > 0)
    node[i].m = y[node[i].right] - y[node[i].left];
  else if (node[i].right - node[i].left == 1)
    node[i].m = 0;
  else
  {
    node[i].m = node[2*i + 1].m + node[2*i + 2].m;
  }
}

//更新line
void update_line(int i)
{
  if (node[i].count > 0)
  {
    node[i].lbd = 1;
    node[i].rbd = 1;
    node[i].line = 1;
```

```
    }
    else if (node[i].right - node[i].left == 1)
    {
      node[i].lbd = 0;
      node[i].rbd = 0;
      node[i].line = 0;
    }
    else
    {
      node[i].lbd = node[2*i + 1].lbd;
      node[i].rbd = node[2*i + 2].rbd;
      node[i].line = node[2*i + 1].line + node[2*i + 2].line - node[2*i + 1].rbd*node[2*i + 2].lbd;
    }
}

void insert(int l, int r, int i)
{
  //在这里要取离散化之前的原值进行比较
  if (y[node[i].left] >= l && y[node[i].right] <= r)
    (node[i].count)++;
  else if (node[i].right - node[i].left == 1)
    return;
  else
  {
    int middle = (node[i].left + node[i].right)/2;
    if (r <= y[middle])
      insert(l, r, 2*i + 1);
    else if (l >= y[middle])
      insert(l, r, 2*i + 2);
    else
    {
      insert(l, y[middle], 2*i + 1);
      insert(y[middle], r, 2*i + 2);
    }
  }
  update_m(i);
  update_line(i);
}

void remove(int l, int r, int i)
{
  ////在这里要取离散化之前的原值进行比较
  if (y[node[i].left] >= l && y[node[i].right] <= r)
    (node[i].count)--;
```

```
    else if (node[i].right - node[i].left == 1)
      return;
    else
    {
      int middle = (node[i].left + node[i].right)/2;
      if (r <= y[middle])
        remove(l, r, 2*i + 1);
      else if (l >= y[middle])
        remove(l, r, 2*i + 2);
      else
      {
        remove(l, y[middle], 2*i + 1);
        remove(y[middle], r, 2*i + 2);
      }
    }
    update_m(i);
    update_line(i);
}


bool cmp(struct ScanLine line1, struct ScanLine line2)
{
    if (line1.x == line2.x)
      return line1.flag > line2.flag;
    return (line1.x < line2.x);
}

int main()
{
    int n;
    scanf("%d", &n);
    int x1, y1, x2, y2;
    int i = 0;
    while (n--)
    {
      scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
      scan[i].x = x1;
      scan[i].y1 = y1;
      scan[i].y2 = y2;
      scan[i].flag = 1;
      y[i++] = y1;
      scan[i].x = x2;
      scan[i].y1 = y1;
      scan[i].y2 = y2;
```

```
        scan[i].flag = 0;
        y[i++] = y2;
      }
      sort(y, y + i);
      sort(scan, scan + i, cmp);
      //y数组中不重复的个数
      int unique_count = unique(y, y + i) - y;
      //离散化，建立线段树
      build(0, unique_count - 1, 0);

      int perimeter = 0;
      int now_m = 0;
      int now_line = 0;

      for (int j = 0; j < i; j++)
      {
        if (scan[j].flag)
          insert(scan[j].y1, scan[j].y2, 0);
        else
          remove(scan[j].y1, scan[j].y2, 0);
        if (j >= 1)
          perimeter += 2*now_line*(scan[j].x - scan[j-1].x);
        perimeter += abs(node[0].m - now_m);
        now_m = node[0].m;
        now_line = node[0].line;
      }

      printf("%d\n", perimeter);
      return 0;
    }
```