Звіт по лабораторній роботі №4

З предмету Комп`ютерна графіка

Студента групи ІПС-31

Олійника Микита

**Тема**: регіональний пошук методом дерева регіонів

**Алгоритм**:

Препроцесінг:

1. Відсортуємо даний набір точок за Х координатою
2. Побудуємо дерево відрізків на даному наборі точок:
   1. У кожній вершині дерева, якій відповідає інтервал [l; r) зберігаємо масив точок з набору з індиксами [l; r), відсортованими за Y координатою
   2. У листових вершинах дерева зберігаємо масив з одного елемента, у не листових отримуємо масив процедурою злиття двох масивів нащадків (за аналогією з сортуванням злиттям)

Обробка запиту:

1. Знайдемо бінарним пошуком по відсортованому набору індекси що відповідають межам запиту за віссю X
2. Зробимо спуск по дереву відрізків за знайденими індексами:
   1. Для кожної тупікової вершини знайдемо бінарним пошуком індекси що відповідають межам запиту за віссю Y
   2. Додами точки між знайденими індексами до відповіді

**Код**:

#include <iostream>  
#include <fstream>  
#include <vector>  
#include <deque>  
#include <algorithm>  
  
using namespace std;  
  
const int INF = 1e9;  
  
struct Point  
{  
 int x, y;  
 Point(int a = 0, int b = 0) : x(a), y(b) {}  
 bool operator<(const Point &p) const  
 {  
 return x < p.x; // для початкового сортування  
 }  
 friend ostream &operator<<(ostream &os, const Point &p)  
 {  
 return os << p.x << ' ' << p.y;  
 }  
 friend istream &operator>>(istream &is, Point &p)  
 {  
 return is >> p.x >> p.y;  
 }  
};  
  
class SegmentTree  
{  
 vector < vector < Point > > data;  
 vector < Point > points;  
 void build(int v, int l, int r)  
 {  
 data[v] = vector < Point >(r - l);  
 int lp = 0, rp = 0;  
 if (l + 1 == r)  
 data[v][0] = points[l]; // у листових вершинах масив з одного елемента  
 else  
 {  
 int m = (l + r) >> 1;  
 build(v << 1, l, m);  
 build(v << 1 | 1, m, r);  
  
 for (int i = 0; i < r - l; ++i) // у не листових вершинах масив отримується злиттям масивів нащадків  
 {  
 if (lp != m - l && (rp == r - m || data[v << 1][lp].y < data[v << 1 | 1][rp].y))  
 {  
 data[v][i] = data[v << 1][lp];  
 ++lp;  
 }  
 else  
 {  
 data[v][i] = data[v << 1 | 1][rp];  
 ++rp;  
 }  
 }  
 }  
 }  
 int binary\_search(const vector < Point > &v, int value, bool coordinate) // знаходить бінарним пошуком перший індекс у даному масиві що більше або дорівнює даному значенню  
 {  
 int l = -1, r = v.size();  
 while (l < r - 1)  
 {  
 int m = (l + r) >> 1;  
 int m\_value = coordinate ? v[m].y : v[m].x;  
 if (m\_value < value)  
 l = m;  
 else  
 r = m;  
 }  
 return l + 1;  
 }  
 void query(int v, int l, int r, int xl, int xr, int yl, int yr, vector < Point > &result)  
 {  
 if (l == xl && xr == r)  
 { // у тупіковій вершині знаходимо індекси відповідні межам запиту за віссю Y  
 int ql = binary\_search(data[v], yl, 1);  
 int qr = binary\_search(data[v], yr, 1);   
 for (int i = ql; i < qr; ++i) // додаємо знайдені точки до відповіді  
 result.push\_back(data[v][i]);  
 }  
 else  
 {  
 int m = (l + r) >> 1;  
 if (xr <= m)  
 query(v << 1, l, m, xl, xr, yl, yr, result);  
 else if (xl >= m)  
 query(v << 1 | 1, m, r, xl, xr, yl, yr, result);  
 else  
 {  
 query(v << 1, l, m, xl, m, yl, yr, result);  
 query(v << 1 | 1, m, r, m, xr, yl, yr, result);  
 }  
 }  
 }  
public:  
 explicit SegmentTree(vector < Point > const &in)  
 {  
 points = in;  
 sort(points.begin(), points.end()); // сортуємо даний список точок  
 data.resize(in.size() << 2);  
 build(1, 0, points.size()); // будуємо дерево відрізків  
 }  
 vector < Point > query(Point lower\_left, Point upper\_right)  
 {  
 vector < Point > result;  
 int xl = binary\_search(points, lower\_left.x, 0); // знаходимо індекси відповідні межам запиту за віссю Х  
 int xr = binary\_search(points, upper\_right.x+1, 0);  
 query(1, 0, points.size(), xl, xr, lower\_left.y, upper\_right.y+1, result);   
 return result;  
 }  
};  
  
int main()  
{  
 ifstream fin("../input.txt");  
  
 int n;  
 fin >> n;  
  
 vector < Point > in;  
  
 for (int i = 0; i < n; ++i)  
 {  
 int x, y;  
 fin >> x >> y;  
 in.emplace\_back(x, y);  
 }  
 SegmentTree st = SegmentTree(in);  
 while (true)  
 {  
 Point lower\_left, upper\_right;  
 cout << "Enter four coordinates:" << endl;  
 cin >> lower\_left >> upper\_right;  
 vector < Point > response = st.query(lower\_left, upper\_right);  
 cout << "Number of points found: " << response.size() << endl;  
 for (Point p: response)  
 cout << p << endl;  
 cout << endl;  
 }  
 return 0;  
}