Звіт по лабораторній роботі №8

З предмету Комп`ютерна графіка

Студента групи ІПС-31

Олійника Микита

**Тема**: апроксимація опуклої оболонки

**Алгоритм**:

1. Знаходимо найвищу лівішу та найнижчу правішу точки даного набору
2. Починаючи з найвищої лівішої точки, будуємо верхню половину опуклої оболонки
   1. Розбиваємо простір між двома знайденими точками на К комірок
   2. Для кожної комірці знаходимо найвищу точку
   3. Для отриманих найвищих точок будуємо опуклу оболонку алгоритмом Єндрю за лінійний час
3. Перевертаємо всі точки обох осях та починаючи з найнижчої правішої точки будуємо аналогічно нижню половину опуклої оболонки

**Код**:

#include <iostream>  
#include <fstream>  
#include <vector>  
#include <deque>  
  
using namespace std;  
  
const int INF = 1e9;  
  
struct Point  
{  
 int x, y;  
 Point(int a, int b) : x(a), y(b) {}  
 Point operator\*(int mult) const  
 {  
 return Point(x\*mult, y\*mult);  
 }  
 void operator\*=(int mult)  
 {  
 x \*= mult;  
 y \*= mult;  
 }  
 Point operator-()   
 {  
 return (\*this)\*(-1);  
 }  
  
 bool operator<(const Point &p) const  
 {  
 return x < p.x || (x == p.x && y > p.y); // для пошуку найвищої лівішої точки  
 }  
 friend ostream &operator<<(ostream &os, Point const &p)  
 {  
 return os << p.x << ' ' << p.y;  
 }  
};  
  
int area(const Point &a, const Point &b, const Point &c)  
{  
 return (b.x - a.x)\*(c.y - a.y) - (b.y - a.y)\*(c.x - a.x); // орієнтована площа трикутника  
}  
  
bool left(const Point &a, const Point &b, const Point &c)  
{  
 return area(a, b, c) > 0; // предикат «проти годинникової стрілки»  
}  
  
deque < int > andrew(int start, const vector < int > &bucket\_top\_i, const vector < Point > &in)  
{ // алгоритм Ендрю для побудови опуклої оболонки заданого набору точок, відсортованого за Х координатою  
 deque < int > result;  
 result.push\_front(start);  
  
 for (int i = 1; i < bucket\_top\_i.size(); ++i)  
 if (i != -1)  
 {  
 int next = bucket\_top\_i[i];  
 while (result.size() >= 2)  
 {  
 int front = result.front();  
 int second\_front = \*(result.begin() + 1);  
 if (left(in[second\_front], in[front], in[next]))  
 result.pop\_front();  
 else  
 break;  
 }  
 result.push\_front(next);  
 }  
 return result;  
}  
  
int main()  
{  
 ifstream fin("../input.txt");  
  
 int n, k;  
 fin >> n >> k;  
  
 int upper\_left\_i = 0, lower\_right\_i = 0;  
 vector < Point > in;  
  
 for (int i = 0; i < n; ++i)  
 {  
 int x, y;  
 fin >> x >> y;  
 Point p(x, y);  
 in.emplace\_back(p);  
 if (p < in[upper\_left\_i]) // шукаємо найвищу лівішу точки та найнижчу правішу  
 upper\_left\_i = i;  
 if (in[lower\_right\_i] < p)  
 lower\_right\_i = i;  
 }  
  
 vector < int > start = {upper\_left\_i, lower\_right\_i};  
 vector < vector < int > > bucket\_top\_i(2, vector < int >(k + 1, -1));  
 vector < int > multiplier = {1, -1};  
 vector < deque < int > > hull(2, deque < int >());  
  
 for (int part = 0; part <= 1; ++part) // ітеруємось по половинам опуклої оболонки  
 {  
 for (int i = 0; i < in.size(); ++i)  
 in[i] \*= multiplier[part]; // перевертаємо площину по обох осях у випадку нижньої половини оболонки  
 Point start\_p = in[start[part]];  
 Point end\_p = in[start[1 - part]];  
 for (int i = 0; i < in.size(); ++i)  
 if (i != start[part])  
 {  
 Point p = in[i];  
 int bucket = (p.x - start\_p.x)\*k/(end\_p.x - start\_p.x); // шукаємо комірку до якої відноситься точка  
 if (bucket\_top\_i[part][bucket] == -1 || p.y > in[bucket\_top\_i[part][bucket]].y)  
 bucket\_top\_i[part][bucket] = i; // оновлюємо найвищу точку у комірці  
 }  
 hull[part] = andrew(start[part], bucket\_top\_i[part], in); // будуємо опуклу оболонку для найвищих точок  
 }  
 for (int part = 0; part <= 1; ++part)  
 for (auto it = hull[part].rbegin(); it != hull[part].rend(); ++it)  
 {  
 int i = \*it;  
 if (i != hull[1 - part].front()) // з`єднуємо дві половини опуклої оболонки  
 cout << -in[i] << endl;  
 }  
 return 0;  
}