Чернівецький національний університет ім. Юрія Федьковича

Факультет математики та інформатики

Кафедра математичного моделювання

Теорія алгоритмів

Виконав: Апатьєв Нікіта Дмитрович

Чернівці 2022

**Варіант № 8**

**Завдання лабораторної роботи "Побудова опуклої оболонки".**

У старовинному саду росли *N* (1≤*N*≤1000) рідкісних плодових дерев. Цим садом опікувався один літній досвідчений садівник і йому були відомі координати всіх дерев у саді (*xi*,*yi*), де *і*=1,2,3,…, *N*. Він акуратно записував урожайність кожного дерева, доглядав і поливав їх. Допоможіть цій поважній людині обчислити довжину огорожі саду, яка пройде по самих крайніх деревах.

#include <iostream>

#include <stack>

#include <stdlib.h>

#include <vector>

#include <cmath>

using namespace std;

struct Point

{

    int x, y;

};

// Глобальна точка, необхідна для сортування точок із посиланням

// до першої точки Використовується у функції порівняння qsort()

Point p0;

// Допоміжна функція для пошуку вгорі в стеку

Point nextToTop(stack<Point>& S)

{

    Point p = S.top();

    S.pop();

    Point res = S.top();

    S.push(p);

    return res;

}

// Допоміжна функція для обміну місцями двох точок

void swap(Point& p1, Point& p2)

{

    Point temp = p1;

    p1 = p2;

    p2 = temp;

}

// Допоміжна функція для повернення квадрата відстані

// між p1 і p2

int distSq(Point p1, Point p2)

{

    return (p1.x - p2.x) \* (p1.x - p2.x) +

        (p1.y - p2.y) \* (p1.y - p2.y);

}

// Знайти орієнтацію впорядкованого триплету (p, q, r).

// Функція повертає наступні значення

// 0 --> p, q і r колінеарні

// 1 --> За годинниковою стрілкою

// 2 --> Проти годинникової стрілки

int orientation(Point p, Point q, Point r)

{

    int val = (q.y - p.y) \* (r.x - q.x) -

        (q.x - p.x) \* (r.y - q.y);

    if (val == 0) return 0;  // колінеарна

    return (val > 0) ? 1 : 2; // годинникова або проти годинникової стрілки

}

// Функція, яка використовується функцією бібліотеки qsort() для сортування масиву

// точки відносно першої точки

int compare(const void\* vp1, const void\* vp2)

{

    Point\* p1 = (Point\*)vp1;

    Point\* p2 = (Point\*)vp2;

    // Знайти орієнтацію

    int o = orientation(p0, \*p1, \*p2);

    if (o == 0)

        return (distSq(p0, \*p2) >= distSq(p0, \*p1)) ? -1 : 1;

    return (o == 2) ? -1 : 1;

}

// Друкує опуклу оболонку набору з n точок.

void convexHull(Point points[], int n)

{

    // Знайти найнижчу точку

    int ymin = points[0].y, min = 0;

    for (int i = 1; i < n; i++)

    {

        int y = points[i].y;

        // Виберіть крайню нижню частину або виберіть ліву

        // найбільше очок у разі нічиєї

        if ((y < ymin) || (ymin == y &&

            points[i].x < points[min].x))

            ymin = points[i].y, min = i;

    }

    // Розмістіть крайню нижню точку на першому місці

    swap(points[0], points[min]);

    // Сортування n-1 балів відносно першого пункту.

    // Точка p1 стоїть перед p2 у відсортованому виведенні, якщо p2

    // має більший полярний кут (проти годинникової стрілки

    // напрямок), ніж p1

    p0 = points[0];

    qsort(&points[1], n - 1, sizeof(Point), compare);

    // Якщо дві або більше точок утворюють однаковий кут із p0,

    // Видалення всіх, крім того, що знаходиться найдальше від p0

    // Запам'ятайте, що у вищевказаному сортуванні нашим критерієм було

    // зберегти найдальшу точку в кінці, коли більше ніж

    // одна точка має однаковий кут.

    int m = 1; // Ініціалізація розміру зміненого масиву

    for (int i = 1; i < n; i++)

    {

        // Продовжуйте видаляти i, поки кути i та i+1 однакові

        // відносно p0

        while (i < n - 1 && orientation(p0, points[i],

            points[i + 1]) == 0)

            i++;

        points[m] = points[i];

        m++;  // Оновити розмір зміненого масиву

    }

    // Якщо модифікований масив точок містить менше 3 точок,

    // опукла оболонка неможлива

    if (m < 3) return;

    // Створіть порожній стек і введіть перші три точки

    // до нього.

    stack<Point> S;

    S.push(points[0]);

    S.push(points[1]);

    S.push(points[2]);

    // Обробити решту n-3 точок

    for (int i = 3; i < m; i++)

    {

        // Продовжуйте видаляти вершину, поки кут утворюється

        // точки next-to-top, top, та points[i] роблять

        //не лівий поворот

        while (S.size() > 1 && orientation(nextToTop(S), S.top(), points[i]) != 2)

            S.pop();

        S.push(points[i]);

    }

    vector <Point> path;

    double suma = 0;

    // Тепер стек має вихідні точки, вивести вміст стека

    cout << "Точки, що утворюють опуклу оболонку: " << endl;

    while (!S.empty())

    {

        Point p = S.top();

        cout << "(" << p.x << ", " << p.y << ")" << endl;

        path.push\_back(p);

        S.pop();

    }

    // Знаходження довжини паркана.

    for (int i = 0; i < path.size() - 1; i++) {

        double p = sqrt(pow(path[i + 1].x - path[i].x, 2) + pow(path[i + 1].y - path[i].y, 2));

        suma += p;

    }

    suma += sqrt(pow(path[0].x - path[path.size() - 1].x, 2) + pow(path[0].y - path[path.size() - 1].y, 2));

    cout << "Довжина паркану: " << suma << endl;

}

int main()

{

    setlocale(LC\_ALL, "UKR");

    cout << "Введiть кiлькiсть точок: ";

    int n; cin >> n;

    cout << "Введiть точки." << endl;

    Point \*points =  new Point[n];

    for (int i = 0; i < n; i++) {

        int x, y; cin >> x >> y;

        points[i].x = x;

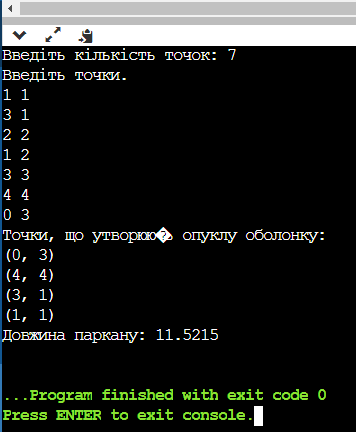
        points[i].y = y;

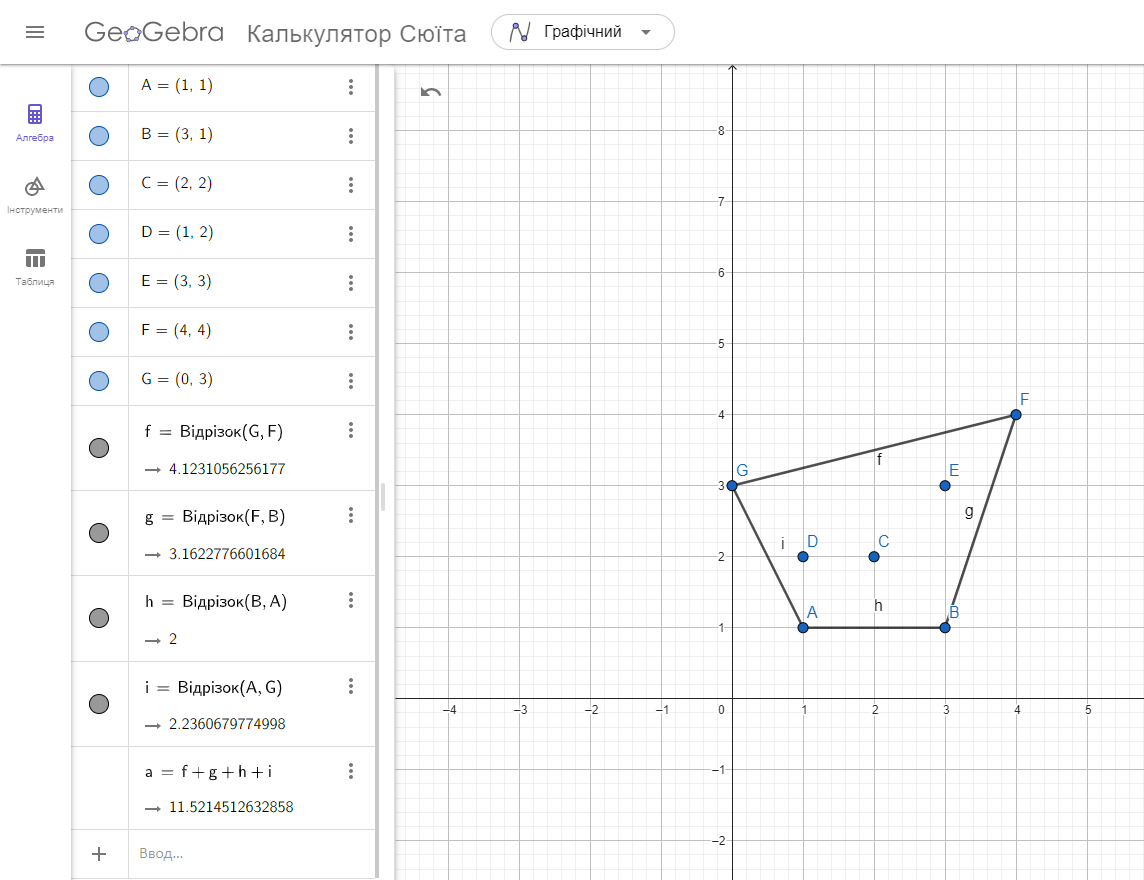
    }

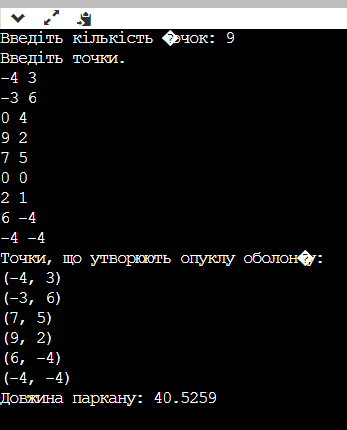
    convexHull(points, n);

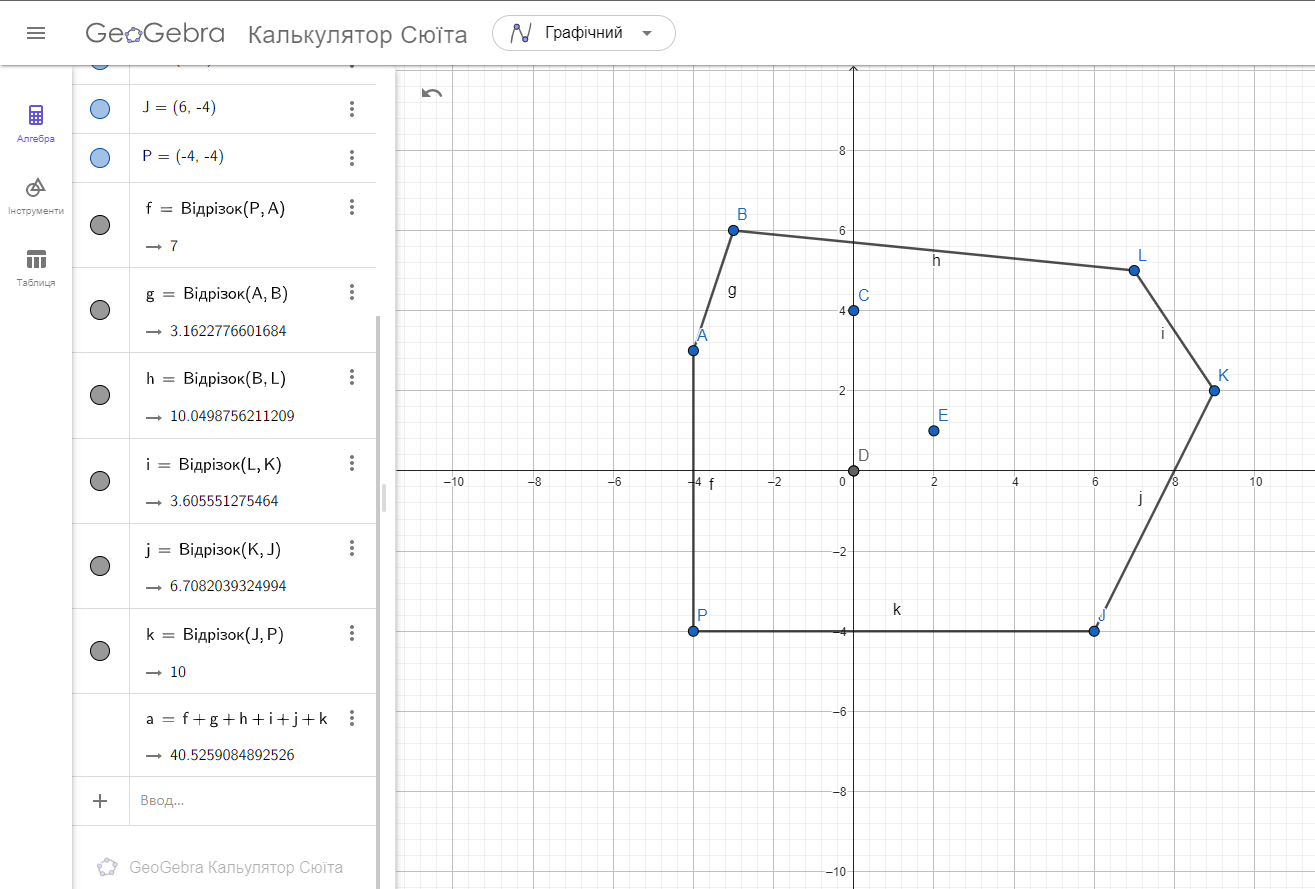
    return 0;

}

****

****

****

****

