**Чернівецький національний університет імені Юрія Федьковича**

**Інститут фізико – технічних та комп’ютерних наук**

Лабораторна робота № 11

на тему

**“ ГЕОМЕТРИЧНІ АЛГОРИТМИ”**

Виконала : студентка

групи № 244(А)

Мельничук А.Г.

Затверджено : доцентом ЧНУ

Угрин Д.І.

Чернівці 2022

**Хід роботи**

**Тема :** геометричні алгоритми.

**Мета роботи :** познайомитися із основними геометричними алгоритмами.

**Завдання до роботи :** Побудувати опуклу оболонку наданих точок алгоритмом Грехема ;

*Код програми :*

#include <iostream>

#include <stack>

#include <stdlib.h>

using namespace std;

// Part of Cosmos by OpenGenus Foundation

// A C++ program to find convex hull of a set of points.

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

int 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; //колінеарний

return (val > 0) ? 1 : 2; // за годинниковою 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;

// Вибираємо самий низ або ліворуч

// 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;

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++;

}

// Якщо модифікований масив точок містить менше 3 точок,

// опукла оболонка неможлива

if (m < 3) return;

// Створіть порожній стек і натисніть перші три точки

// до нього.

stack<Point> S;

S.push(points[0]);

S.push(points[1]);

S.push(points[2]);

for (int i = 3; i < m; i++)

{

// Продовжуйте видаляти вершину, поки кут утворюється

// точки поруч із вершиною, вершина та точки[i] робить

// не лівий поворот

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

S.pop();

S.push(points[i]);

}

// Тепер стек має вихідні точки, вивести вміст стека

while (!S.empty())

{

Point p = S.top();

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

S.pop();

}

}

int main()

{

Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},

{0, 0}, {1, 2}, {3, 1}, {3, 3}};

int n = sizeof(points) / sizeof(points[0]);

convexHull(points, n);

return 0;

}

*Мій опис виконаного завдання :* як і було вказано в завданні , ми скористались алгоритмом Грехема для побудови опуклої оболонки .

Складність завдання :

Часова складність у найгіршому випадку: Θ(N log N)

Середня складність справи: Θ(N log N)

Кращий випадок складності часу: Θ(N log N)

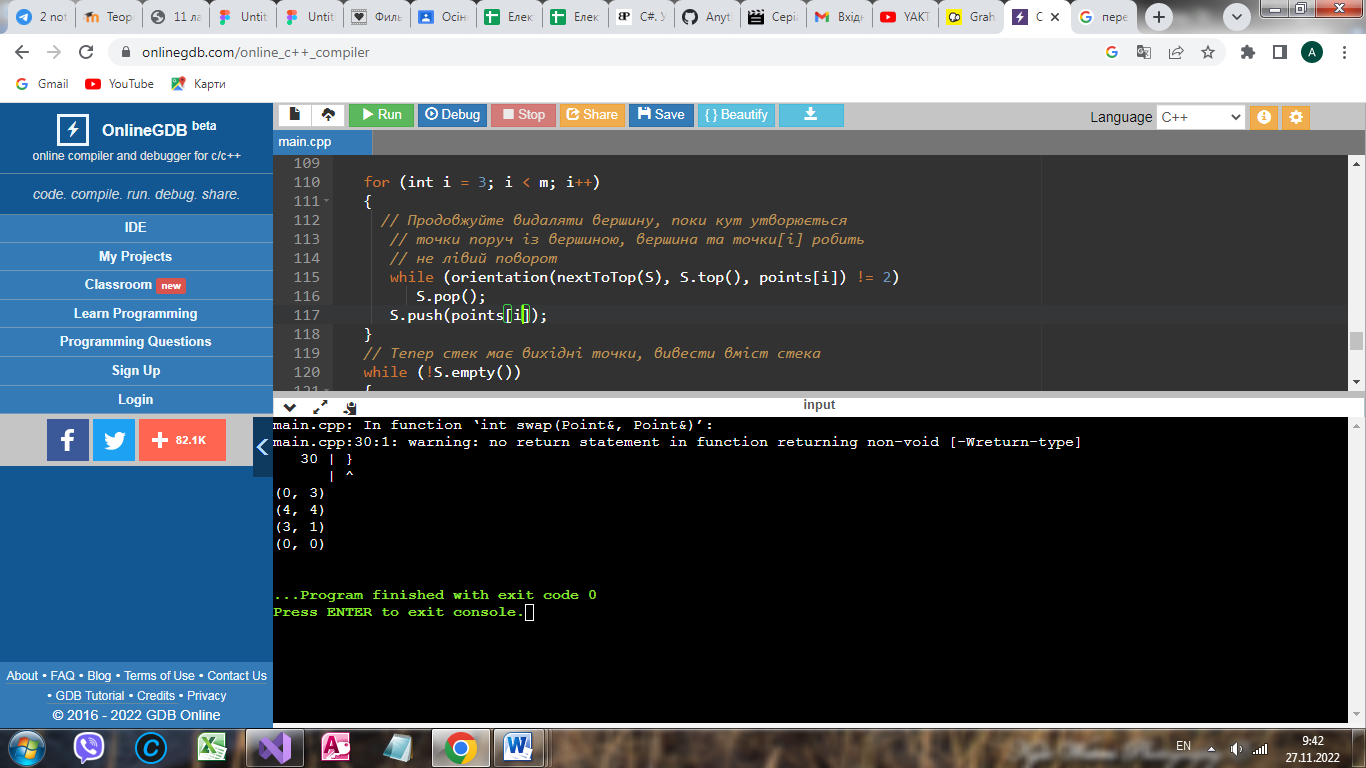
Просторова складність: Θ(N)

Алгоритм займає O(nLogn) часу, якщо ми використовуємо алгоритм сортування O(nLogn).

====================================================================================

Перший крок (пошук найнижчої точки) займає O(n) часу. Другий крок (сортування точок) займає O(nLogn) часу. Третій крок займає O(n) часу. На третьому кроці кожен елемент штовхається та висувається щонайбільше один раз. Таким чином, шостий крок для обробки точок по одній займає O(n) часу, припускаючи, що операції зі стеком займають O(1) часу. Загальна складність: O(n) + O(nLogn) + O(n) + O(n), тобто O(nLogn).

*Скрін виводу консолі :*



*Висновок :* під час виконання даної лабораторної роботи ми навчились використовувати алгоритм Грехема для знаходження опуклої оболонки , також розглянули як може бути побудована ця оболонка, за годинниковою стрілкою чи проти неї.