

**Ahsania Mission University of Science & Technology**

**Lab Report**

**Lab No:** 07

**Course Code:** CSE2202

**Course Title:** Computer Algorithm Sessional.

**Submitted By:**

Afrin Nahrin Nipu

ID: 1012320005101025

1st Batch, 2nd Year, 2nd Semester

Department of Computer science and Engineering,

Ahsania Mission University of Science & Technology

**Submitted To:**

Md. Fahim Faisal

Lecturer,

Department of Computer science and Engineering,

Ahsania Mission University of Science & Technology

**Task No.:** 01

**Problem Statement:** Convex Hull using Brute Force Algorithm.

**Source Code:**

#include <iostream>

#include <vector>

using namespace std;

struct Point

{

int x, y;

};

int direction(Point a, Point b, Point c)

{

return (b.x - a.x)\*(c.y - a.y) - (b.y - a.y)\*(c.x - a.x);

}

void convexHull(vector<Point>& points)

{

int n = points.size();

cout << "Convex Hull Edges:\n";

for (int i = 0; i < n; i++)

{

for (int j = i + 1; j < n; j++)

{

int pos = 0, neg = 0;

for (int k = 0; k < n; k++)

{

if (k == i || k == j)

{

continue;

}

int d = direction(points[i], points[j], points[k]);

if (d > 0)

{

pos++;

}

else if (d < 0)

{

neg++;

}

}

if (pos == 0 || neg == 0)

{

cout << "(" << points[i].x << "," << points[i].y << ") - ("<< points[j].x << "," << points[j].y << ")\n";

}

}

}

}

int main()

{

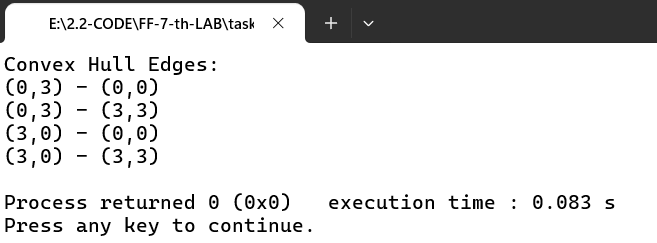
vector<Point> points = {{0, 3}, {2, 2}, {1, 1}, {2, 1}, {3, 0}, {0, 0}, {3, 3}};

convexHull(points);

return 0;

}

**Output:**

****

**Task No.:** 02

**Problem Statement:** Convex Hull using Graham's Scan Algorithm.

**Source Code:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <stack>

using namespace std;

struct Point

{

int x, y;

};

Point p0;

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;

return (val > 0) ? 1 : 2;

}

int distSq(Point p1, Point p2)

{

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

}

bool compare(Point p1, Point p2)

{

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

if (o == 0)

return distSq(p0, p1) < distSq(p0, p2);

return (o == 2);

}

void grahamScan(vector<Point>& points)

{

int n = points.size();

int ymin = points[0].y, minIndex = 0;

for (int i = 1; i < n; i++)

{

if ((points[i].y < ymin) || (points[i].y == ymin && points[i].x < points[minIndex].x))

{

ymin = points[i].y;

minIndex = i;

}

}

swap(points[0], points[minIndex]);

p0 = points[0];

sort(points.begin() + 1, points.end(), compare);

stack<Point> hull;

hull.push(points[0]);

hull.push(points[1]);

hull.push(points[2]);

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

{

while (hull.size() > 1)

{

Point top = hull.top();

hull.pop();

Point nextToTop = hull.top();

if (orientation(nextToTop, top, points[i]) != 2)

continue;

else

{

hull.push(top);

break;

}

}

hull.push(points[i]);

}

cout << "Convex Hull Points (Graham's Scan):\n";

while (!hull.empty())

{

Point p = hull.top();

cout << "(" << p.x << ", " << p.y << ")\n";

hull.pop();

}

}

int main()

{

vector<Point> points = {{0, 3}, {1, 1}, {2, 2}, {4, 4}, {0, 0}, {1, 2}, {3, 1}, {3, 3}};

grahamScan(points);

return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**Task No.:** 03

**Problem Statement:** Convex Hull using QuickHull Algorithm

**Source Code:**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

struct Point

{

int x, y;

};

// Function to find distance of point C from line AB

int distance(Point A, Point B, Point C)

{

return abs((C.y - A.y) \* B.x - (C.x - A.x) \* B.y + C.x \* A.y - C.y \* A.x);

}

// Determine the side of point P with respect to line AB

int findSide(Point A, Point B, Point P)

{

int val = (P.y - A.y) \* (B.x - A.x) - (B.y - A.y) \* (P.x - A.x);

if (val > 0) return 1;

if (val < 0) return -1;

return 0;

}

// Recursive QuickHull function

void quickHull(vector<Point>& points, Point A, Point B, int side, vector<Point>& hull)

{

int index = -1;

int max\_dist = 0;

for (int i = 0; i < points.size(); i++)

{

int temp = distance(A, B, points[i]);

if (findSide(A, B, points[i]) == side && temp > max\_dist)

{

index = i;

max\_dist = temp;

}

}

if (index == -1)

{

hull.push\_back(A);

hull.push\_back(B);

return;

}

quickHull(points, points[index], A, -findSide(points[index], A, B), hull);

quickHull(points, points[index], B, -findSide(points[index], B, A), hull);

}

// Driver function

void findConvexHull(vector<Point>& points)

{

if (points.size() < 3)

{

cout << "Convex hull not possible\\n";

return;

}

int min\_x = 0, max\_x = 0;

for (int i = 1; i < points.size(); i++)

{

if (points[i].x < points[min\_x].x) min\_x = i;

if (points[i].x > points[max\_x].x) max\_x = i;

}

vector<Point> hull;

quickHull(points, points[min\_x], points[max\_x], 1, hull);

quickHull(points, points[min\_x], points[max\_x], -1, hull);

cout << "Convex Hull Points (QuickHull):"<<"\n";

for (auto& p : hull)

{

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

}

}

int main()

{

vector<Point> points = {{0, 3}, {1, 1}, {2, 2}, {4, 4}, {0, 0}, {1, 2}, {3, 1}, {3, 3}};

findConvexHull(points);

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated