**Bài tập ngày 25.10.2024**

Họ và tên: Nguyễn Hoàng Tuấn Anh

MSSV: 102220219

Graham:

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

#include <stack>

using namespace std;

struct Point {

    int x, y;

};

int crossProduct(Point a, Point b, Point c) {

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

}

int distanceSquared(Point a, Point b) {

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

}

bool compare(Point a, Point b, Point origin) {

    int cp = crossProduct(origin, a, b);

    if (cp == 0) {

        return distanceSquared(origin, a) < distanceSquared(origin, b);

    }

    return cp > 0;

}

vector<Point> grahamScan(vector<Point>& points) {

    Point origin = \*min\_element(points.begin(), points.end(), [](Point a, Point b) {

        return (a.y < b.y) || (a.y == b.y && a.x < b.x);

    });

    sort(points.begin(), points.end(), [&](Point a, Point b) {

        return compare(a, b, origin);

    });

    stack<Point> hull;

    hull.push(points[0]);

    hull.push(points[1]);

    for (int i = 2; i < points.size(); ++i) {

        while (hull.size() >= 2) {

            Point top = hull.top(); hull.pop();

            Point nextToTop = hull.top();

            if (crossProduct(nextToTop, top, points[i]) > 0) {

                hull.push(top);

                break;

            }

        }

        hull.push(points[i]);

    }

    vector<Point> result;

    while (!hull.empty()) {

        result.push\_back(hull.top());

        hull.pop();

    }

    return result;

}

double polygonArea(const vector<Point>& hull) {

    int n = hull.size();

    double area = 0.0;

    for (int i = 0; i < n; ++i) {

        int j = (i + 1) % n;

        area += hull[i].x \* hull[j].y - hull[i].y \* hull[j].x;

    }

    return abs(area) / 2.0;

}

int main() {

    int N = 15;

    vector<Point> points(N);

    srand(time(0));

    for (int i = 0; i < N; ++i) {

        points[i].x = rand() % 116;

        points[i].y = rand() % 116;

        cout << "Point " << i + 1 << ": (" << points[i].x << ", " << points[i].y << ")\n";

    }

    vector<Point> hull = grahamScan(points);

    cout << "\nConvex Hull:\n";

    for (const auto& p : hull) {

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

    }

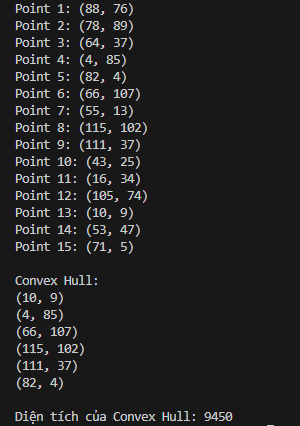
    double area = polygonArea(hull);

    cout << "\nDiện tích của Convex Hull: " << area << endl;

    return 0;

}

Kết quả:



DivideAndConquer:

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

using namespace std;

struct Point {

    int x, y;

};

int crossProduct(Point a, Point b, Point c) {

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

}

void addPointToHull(vector<Point>& hull, Point p) {

    while (hull.size() >= 2 && crossProduct(hull[hull.size() - 2], hull.back(), p) <= 0) {

        hull.pop\_back();

    }

    hull.push\_back(p);

}

vector<Point> divideAndConquer(vector<Point>& points) {

    sort(points.begin(), points.end(), [](Point a, Point b) {

        return (a.x < b.x) || (a.x == b.x && a.y < b.y);

    });

    vector<Point> lowerHull, upperHull;

    for (const auto& p : points) {

        addPointToHull(lowerHull, p);

    }

    for (int i = points.size() - 1; i >= 0; --i) {

        addPointToHull(upperHull, points[i]);

    }

    lowerHull.pop\_back();

    upperHull.pop\_back();

    lowerHull.insert(lowerHull.end(), upperHull.begin(), upperHull.end());

    return lowerHull;

}

double polygonArea(const vector<Point>& hull) {

    int n = hull.size();

    double area = 0.0;

    for (int i = 0; i < n; ++i) {

        int j = (i + 1) % n;

        area += hull[i].x \* hull[j].y - hull[i].y \* hull[j].x;

    }

    return abs(area) / 2.0;

}

int main() {

    int N = 15;

    vector<Point> points(N);

    srand(time(0));

    for (int i = 0; i < N; ++i) {

        points[i].x = rand() % 116;

        points[i].y = rand() % 116;

        cout << "Point " << i + 1 << ": (" << points[i].x << ", " << points[i].y << ")\n";

    }

    vector<Point> hull = divideAndConquer(points);

    cout << "\nConvex Hull:\n";

    for (const auto& p : hull) {

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

    }

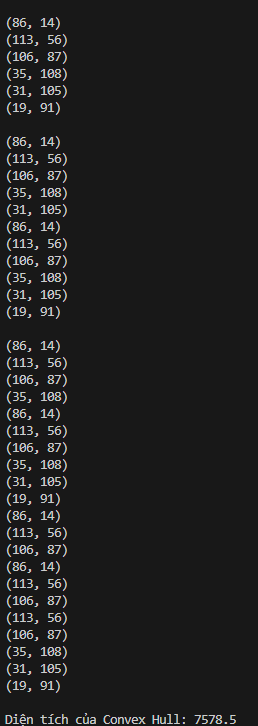
    double area = polygonArea(hull);

    cout << "\nDiện tích của Convex Hull: " << area << endl;

    return 0;

}

Kết quả:



Monotone Chain:

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

using namespace std;

struct Point {

    int x, y;

};

// Hàm tính tích chéo giữa ba điểm

int crossProduct(Point a, Point b, Point c) {

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

}

// Thuật toán Monotone Chain để tìm Convex Hull

vector<Point> monotoneChain(vector<Point>& points) {

    sort(points.begin(), points.end(), [](Point a, Point b) {

        return (a.x < b.x) || (a.x == b.x && a.y < b.y);

    });

    vector<Point> lowerHull, upperHull;

    // Tạo Lower Hull

    for (const auto& p : points) {

        while (lowerHull.size() >= 2 &&

               crossProduct(lowerHull[lowerHull.size() - 2], lowerHull.back(), p) <= 0) {

            lowerHull.pop\_back();

        }

        lowerHull.push\_back(p);

    }

    // Tạo Upper Hull

    for (int i = points.size() - 1; i >= 0; --i) {

        while (upperHull.size() >= 2 &&

               crossProduct(upperHull[upperHull.size() - 2], upperHull.back(), points[i]) <= 0) {

            upperHull.pop\_back();

        }

        upperHull.push\_back(points[i]);

    }

    lowerHull.pop\_back();  // Bỏ điểm trùng

    upperHull.pop\_back();  // Bỏ điểm trùng

    // Kết hợp hai Hull thành một

    lowerHull.insert(lowerHull.end(), upperHull.begin(), upperHull.end());

    return lowerHull;

}

// Hàm tính diện tích đa giác bằng công thức Shoelace

double polygonArea(const vector<Point>& hull) {

    int n = hull.size();

    double area = 0.0;

    for (int i = 0; i < n; ++i) {

        int j = (i + 1) % n;  // Điểm tiếp theo (vòng lại nếu cần)

        area += hull[i].x \* hull[j].y - hull[i].y \* hull[j].x;

    }

    return abs(area) / 2.0;

}

// Hàm main

int main() {

    int N = 15;  // Số điểm ngẫu nhiên

    vector<Point> points(N);

    // Sinh các điểm ngẫu nhiên

    srand(time(0));

    for (int i = 0; i < N; ++i) {

        points[i].x = rand() % 116;

        points[i].y = rand() % 116;

        cout << "Point " << i + 1 << ": (" << points[i].x << ", " << points[i].y << ")\n";

    }

    // Tìm Convex Hull bằng Monotone Chain

    vector<Point> hull = monotoneChain(points);

    // In các điểm trên Hull

    cout << "\nConvex Hull:\n";

    for (const auto& p : hull) {

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

    }

    // Tính diện tích của Convex Hull

    double area = polygonArea(hull);

    cout << "\nDiện tích của Convex Hull: " << area << endl;

    return 0;

}

Kết quả:

