

컴퓨터알고리즘과실습 실습10

2020112046 이재원

실행 결과

```
----- 10-1. 주어진 점들에 대한 Convex Hull 결정 및 출력 -----
주어진 점으로 결정된 단순 다각형: B - E - F - H - G - A - C - D - B
결정된 Convex Hull: B - E - H - D - B

----- 10-1. 100개의 random한 점에 의한 Convex Hull 결정 및 출력 -----

랜덤한 점들의 좌표:
0(94, 58) / 1(23, 27) / 2(7, 31) / 3(89, 8) / 4(11, 83) / 5(56, 92) / 6(73, 33) / 7(92, 96) / 8(81, 2) / 9(45, 25) / 10(5, 28) / 11(67, 74) / 12(46, 9) / 13(45, 31) / 14(61, 85) / 15(99, 47) / 16(96, 56) / 17(45, 91) / 18(0, 53) / 19(89, 34) / 20(36, 58) / 21(92, 83) / 22(49, 43) / 23(31, 3) / 24(0, 27) / 25(73, 1) / 26(44, 85) / 27(8, 92) / 28(92, 18) / 29(60, 35) / 30(6, 44) / 31(0, 58) / 32(89, 29) / 33(95, 31) / 34(84, 59) / 35(95, 9) / 36(2, 98) / 37(81, 67) / 38(53, 21) / 39(16, 94) / 40(32, 99) / 41(16, 93) / 42(29, 91) / 43(14, 49) / 44(66, 52) / 45(23, 9) / 46(12, 99) / 47(67, 64) / 48(96, 33) / 49(79, 42) / 50(42, 17) / 51(23, 34) / 52(23, 84) / 53(88, 43) / 54(12, 33) / 55(36, 61) / 56(28, 30) / 57(89, 80) / 58(10, 44) / 59(98, 39) / 60(58, 59) / 61(50, 75) / 62(19, 47) / 63(41, 33) / 64(16, 82) / 65(51, 67) / 66(88, 75) / 67(68, 20) / 68(23, 34) / 69(69, 45) / 70(94, 62) / 71(24, 2) / 72(12, 77) / 73(7, 76) / 74(29, 65) / 75(16, 51) / 76(70, 95) / 77(0, 40) / 78(82, 69) / 79(80, 82) / 80(34, 46) / 81(63, 27) / 82(41, 80) / 83(35, 16) / 84(0, 97) / 85(81, 6) / 86(46, 76) / 87(77, 66) / 88(5, 69) / 89(9, 59) / 90(78, 42) / 91(41, 87) / 92(85, 4) / 93(22, 23) / 94(74, 24) / 95(21, 74) / 96(86, 23) / 97(36, 25) / 98(76, 27) / 99(40, 41) /

랜덤한 점으로 결정된 단순 다각형:
25 - 8 - 92 - 3 - 85 - 28 - 33 - 48 - 59 - 96 - 32 - 15 - 19 - 16 - 0 - 53 - 70 - 35 - 21 - 66 - 57 - 7 - 34 - 49 - 78 - 90 - 37 - 98 - 79 - 87 - 9 - 4 - 6 - 76 - 11 - 69 - 47 - 44 - 14 - 5 - 60 - 67 - 61 - 17 - 65 - 26 - 86 - 91 - 29 - 81 - 82 - 40 - 42 - 22 - 52 - 39 - 55 - 41 - 46 - 20 - 74 - 64 - 95 - 27 - 36 - 4 - 84 - 72 - 99 - 80 - 73 - 13 - 38 - 63 - 88 - 89 - 75 - 9 - 62 - 43 - 31 - 18 - 58 - 51 - 68 - 97 - 56 - 30 - 77 - 54 - 1 - 50 - 2 - 93 - 10 - 83 - 24 - 12 - 45 - 23 - 71 - 25

결정된 Convex Hull: 25 - 8 - 92 - 3 - 28 - 59 - 15 - 35 - 7 - 40 - 46 - 36 - 84 - 31 - 18 - 77 - 24 - 71 - 25

----- 10-2. 3차원 평면상에서 랜덤한 N개의 점을 생성하고 최근접 점 쌍 찾기 -----

랜덤한 점들의 좌표:
0(39, 9, 55) / 1(9, 28, 21) / 2(77, 25, 34) / 3(5, 28, 3) / 4(91, 65, 72) / 5(90, 20, 45) / 6(42, 93, 8) / 7(74, 82, 48) / 8(71, 13, 87) / 9(88, 51, 30) / 10(18, 1, 85) / 11(14, 44, 17) / 12(96, 79, 72) / 13(7, 88, 18) / 14(58, 62, 44) / 15(58, 40, 59) / 16(11, 49, 94) / 17(92, 83, 23) / 18(69, 33, 31) / 19(18, 92, 48) / 20(99, 59, 1) / 21(77, 55, 61) / 22(88, 29, 75) / 23(4, 19, 46) / 24(53, 38, 4) / 25(58, 50, 35) / 26(71, 43, 57) / 27(11, 74, 52) / 28(24, 81, 77) / 29(25, 30, 37) / 30(48, 93, 70) / 31(77, 41, 58) / 32(49, 66, 86) / 33(86, 14, 16) / 34(47, 18, 12) / 35(5, 60, 15) / 36(37, 95, 30) / 37(66, 97, 42) / 38(34, 60, 99) / 39(90, 60, 63) / 40(15, 9, 80) / 41(38, 96, 47) / 42(44, 16, 45) / 43(83, 88, 40) / 44(57, 77, 16) / 45(41, 76, 34) / 46(73, 42, 79) / 47(8, 82, 73) / 48(47, 52, 23) / 49(63, 61, 26) / 50(65, 84, 65) / 51(75, 68, 32) / 52(9, 30, 73) / 53(95, 37, 64) / 54(91, 5, 38) / 55(69, 98, 19) / 56(25, 41, 4) / 57(24, 74, 41) / 58(54, 40, 74) / 59(21, 80, 84) / 60(75, 70, 7) / 61(41, 77, 19) / 62(7, 93, 20) / 63(51, 13, 25) / 64(18, 58, 88) / 65(65, 52, 43) / 66(3, 49, 16) / 67(96, 12, 38) / 68(35, 30, 96) / 69(92, 79, 27) / 70(39, 75, 14) / 71(10, 95, 53) / 72(84, 35, 42) / 73(39, 27, 20) / 74(74, 19, 74) / 75(90, 33, 47) / 76(25, 27, 26) / 77(38, 7, 72) / 78(93, 26, 31) / 79(38, 26, 7) / 80(59, 60, 78) / 81(64, 97, 3) / 82(77, 23, 69) / 83(72, 9, 4) / 84(88, 78, 29) / 85(18, 79, 47) / 86(52, 86, 51) / 87(43, 74, 7) / 88(12, 26, 76) / 89(65, 26, 90) / 90(36, 22, 8) / 91(8, 35, 33) / 92(15, 21, 57) / 93(98, 37, 88) / 94(56, 89, 46) / 95(41, 87, 87) / 96(4, 63, 71) / 97(54, 29, 91) / 98(55, 40, 57) / 99(2, 58, 3) /

최근접 점 쌍: 98(55, 40, 57) - 15(58, 40, 59)
최근접 점 쌍 사이의 거리: 3.60555
```

코드에 주석으로 설명을 상세히 작성해 놓았다.

코드

```
#include <math.h>

#include <string.h>

#include <algorithm>

#include <iostream>

#include <map>

#include <random>

#include <vector>
```

```
using namespace std;
```

```
typedef struct point {  
    int x;  
    int y;  
    char c;  
};
```

```
typedef struct point3 {  
    int x;  
    int y;  
    int z;  
    char c;  
};
```

```
typedef struct line {  
    point p1;  
    point p2;  
};
```

```
point polygon[100];
```

```
float ComputeAngle(point p1, point p2) {  
    int dx, dy, ax, ay;  
    float angle;  
    dx = p2.x - p1.x;  
    ax = abs(dx);  
    dy = p2.y - p1.y;
```

```

ay = abs(dy);

angle = (ax + ay == 0) ? 0.0 : (float)dy / (ax + ay);

if (dx < 0)

angle = 2.0 - angle;

else if (dy < 0)

angle = 4.0 + angle;

return angle * 90.0;

}

```

// 1: clockwise(시계방향), -1: counter-clockwise(반시계방향), 0: collinear(일직선상에 있음)

```

int Direction(point A, point B, point C) {

int dxAB = B.x - A.x;

int dyAB = B.y - A.y;

int dxAC = C.x - A.x;

int dyAC = C.y - A.y;


if (dxAB * dyAC < dyAB * dxAC) return 1;

if (dxAB * dyAC > dyAB * dxAC) return -1;

if (dxAB * dyAC == dyAB * dxAC) {

if (dxAB == 0 && dyAB == 0) return 0;

// A가 가운데

if ((dxAB * dxAC < 0) || (dyAB * dyAC < 0)) return -1;

// C가 가운데

else if ((dxAB * dxAB + dyAB * dyAB) < (dxAC * dxAC + dyAC * dyAC))

return 0;

// B가 가운데

else

return 1;

}

}

```

// make Convex Hull with Graham Scan

```
void GrahamScan(vector<point> &polygon) {
```

```
    int i = 0;
```

```
    int n = polygon.size();
```

// parameter polygon is simple polygon

// remove the point that cannot be the convex hull

```
    while (i < n) {
```

```
        while (Direction(polygon[i], polygon[(i + 1) % n], polygon[(i + 2) % n]) == 1) {
```

```
            polygon.erase(polygon.begin() + i + 1);
```

```
            n--;
```

```
            i--;
```

```
        }
```

```
        i++;
```

```
    }
```

```
}
```

// find closest pair of point3s using divide and conquer with sorted pointList, return closest pair of points

// Parameter로는 나뉜 Point의 집합. 초기 Parameter로 정렬된 전체 Point의 집합이 들어옴.

```
pair<point3, point3> ClosestPair(vector<point3> &pointList) {
```

```
    int n = pointList.size();
```

*// Base Case - 점의 집합을 더 이상 나눌 수 없을 때 (3개 이하일 때) 에는 brute force로 최단거리를 구한다.
(한번 더 나뉘 계산하는 것보다 빠름)*

```
    if (n <= 3) {
```

```
        pair<point3, point3> result;
```

```
        float minDist = 1000000000;
```

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

```

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

// 두 점 사이의 거리. sqrt(dx^2 + dy^2 + dz^2) = distance of two points

float dist = sqrt(pow(pointList[i].x - pointList[j].x, 2) + pow(pointList[i].y - pointList[j].y, 2) + pow(pointList[i].z -
pointList[j].z, 2));

if (dist < minDist) {

minDist = dist;

result.first = pointList[i];

result.second = pointList[j];

}

}

}

return result;

}

// Divide - 점의 집합을 x축을 기준으로 반으로 나뉘,

else {

vector<point3> left, right;

for (int i = 0; i < n / 2; i++) {

left.push_back(pointList[i]);

}

for (int i = n / 2; i < n; i++) {

right.push_back(pointList[i]);

}

// Conquer - 왼쪽과 오른쪽으로 나눈 점의 집합들에 대해 ClosestPair 재귀 호출.

pair<point3, point3> leftPair = ClosestPair(left);

pair<point3, point3> rightPair = ClosestPair(right);

// 각각의 집합에서 return된 최근접 점 쌍에 대해 최단거리를 구한 후, 둘 중 더 짧은 거리를 가진 쌍을
선택. (그것이 합쳐진 집합의 최단거리 쌍이 됨)

float leftDist = sqrt(pow(leftPair.first.x - leftPair.second.x, 2) + pow(leftPair.first.y - leftPair.second.y, 2) +
pow(leftPair.first.z - leftPair.second.z, 2));

float rightDist = sqrt(pow(rightPair.first.x - rightPair.second.x, 2) + pow(rightPair.first.y - rightPair.second.y, 2) +
pow(rightPair.first.z - rightPair.second.z, 2));

float minDist = min(leftDist, rightDist);

// leftPair와 rightPair 중 더 짧은 거리를 가진 쌍을 선택.

pair<point3, point3> result;

```

```

if (minDist == leftDist) {

result = leftPair;

} else {

result = rightPair;

}

// Merge - x 축을 기준으로 가운데(중간 영역)에 있는 점들을 찾아서, 그 점들을 기준으로 최단거리를 구한다.
vector<point3> middle;

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

// x 축을 기준으로 가운데에 있는 점들을 찾아서 middle에 넣는다. (minDist 이내에 있는 점들)
if (abs(pointList[i].x - pointList[n / 2].x) < minDist) {

middle.push_back(pointList[i]);

}

}

// middle에 있는 점들을 y 축을 기준으로 정렬한다. by insertion sort
for (int i = 1; i < middle.size(); i++) {

point3 temp = middle[i];

int j = i - 1;

while (j >= 0 && middle[j].y > temp.y) {

middle[j + 1] = middle[j];

j--;

}

middle[j + 1] = temp;

}

// middle에 있는 점들 중 최근접 점 쌍을 찾는다.
pair<point3, point3> middlePair;

float middleDist = 1000000000;

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

for (int j = i + 1; j < middle.size() && (middle[j].y - middle[i].y) < minDist; j++) {

float dist = sqrt(pow(middle[i].x - middle[j].x, 2) + pow(middle[i].y - middle[j].y, 2) + pow(middle[i].z - middle[j].z, 2));

if (dist < middleDist) {

middleDist = dist;

```

```

middlePair.first = middle[i];

middlePair.second = middle[j];

}

}

}

// middle에 있는 점들 중 최근접 점 쌍의 거리가 minDist보다 짧으면, result를 middlePair로 바꾼다.
if (middleDist < minDist) {

result = middlePair;

}

return result;

}

}

```

```

int main() {

// -----
// ----- 10-1. 주어진 점들에 대한 Convex Hull 결정 및 출력 -----
// -----

point poly[8];

//--

poly[0].x = 3;
poly[0].y = 4;
poly[0].c = 'A';

poly[1].x = 1;
poly[1].y = 2;
poly[1].c = 'B';

poly[2].x = 2;
poly[2].y = 5;
poly[2].c = 'C';

poly[3].x = 2;
poly[3].y = 6;
poly[3].c = 'D';

poly[4].x = 9;

```

```
poly[4].y = 3;

poly[4].c = 'E';

poly[5].x = 5;

poly[5].y = 3;

poly[5].c = 'F';

poly[6].x = 6;

poly[6].y = 4;

poly[6].c = 'G';

poly[7].x = 8;

poly[7].y = 4;

poly[7].c = 'H';

//--
```

```
float polyAngle[8];

// 기준점 : y좌표가 가장 작은 점

int polyBasePoint = 0;

for (int i = 1; i < 8; i++) {

    if (poly[i].y < poly[polyBasePoint].y) {

        polyBasePoint = i;

    }

}

// compute Angles of poly - with respect to basePoint

for (int i = 0; i < 8; i++) {

    polyAngle[i] = ComputeAngle(poly[polyBasePoint], poly[i]);

}

// make map of poly and its angles

map<char, float> polyMap;

for (int i = 0; i < 8; i++) {

    polyMap.insert(pair<char, float>(poly[i].c, polyAngle[i]));

}

// sort map by value with Insertion sort

vector<pair<char, float> > polyVec;
```



```

for (auto &p : polyMap) {
    polyVec.push_back(p);
}

for (int i = 1; i < polyVec.size(); i++) {
    for (int j = 0; j < i; j++) {
        if (polyVec[i].second < polyVec[j].second) {
            polyVec.insert(polyVec.begin() + j, polyVec[i]);
            polyVec.erase(polyVec.begin() + i + 1);
            break;
        }
    }
}

// make sorted poly vector include x, y with polyVec
vector<point> sortedPoly;

for (auto &p : polyVec) {
    // A -> 0, B -> 1, C -> 2, D -> 3, E -> 4, F -> 5, G -> 6, H -> 7
    sortedPoly.push_back(poly[p.first - 65]);
}

// print sorted map
cout << "----- 10-1. 주어진 점들에 대한 Convex Hull 결정 및 출력 -----" << endl;

// 결정된 단순 다각형 출력 (print sorted map)
cout << "주어진 점으로 결정된 단순 다각형: ";

for (auto &p : polyVec) {
    cout << p.first << " - ";
}

cout << polyVec[0].first << endl;

// make Convex Hull with Graham Scan
GrahamScan(sortedPoly);

```

```

// print Convex Hull

cout << "결정된 Convex Hull: ";

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

cout << sortedPoly[i].c << " - ";

}

cout << sortedPoly[0].c << endl

<< endl;


// -----
// ----- 10-1. N개의 random 한 점에 의한 Convex Hull 결정 및 출력 -----
// -----


vector<int> xList, yList;

int N = 100;


random_device rd;

for (int i = 0; i < N; i++) {

// x,y의 좌표값은 100 이하

xList.push_back(rd() % 100);

yList.push_back(rd() % 100);

}


// make random polygon

for (int i = 0; i < N; i++) {

polygon[i].x = xList[i];

polygon[i].y = yList[i];

polygon[i].c = i;

}

```

```

// compute angles of polygon

float polygonAngle[N];

// 기준점 : y좌표가 가장 작은 점

int polygonBasePoint = 0;

for (int i = 0; i < N; i++) {

if (polygon[i].y < polygon[polygonBasePoint].y) {

polygonBasePoint = i;

}

}

// compute Angles of poly - with respect to basePoint

for (int i = 0; i < N; i++) {

polygonAngle[i] = ComputeAngle(polygon[polygonBasePoint], polygon[i]);

}

// make map of poly and its angles

map<char, float> polygonMap;

for (int i = 0; i < N; i++) {

polygonMap.insert(pair<char, float>(polygon[i].c, polygonAngle[i]));

}

// sort map by value with Insertion sort

int compareCount = 0;

vector<pair<char, float> > polygonVec;

for (auto &p : polygonMap) {

polygonVec.push_back(p);

}

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

for (int j = 0; j < i; j++) {

compareCount++;

if (polygonVec[i].second < polygonVec[j].second) {

polygonVec.insert(polygonVec.begin() + j, polygonVec[i]);

polygonVec.erase(polygonVec.begin() + i + 1);

break;

}

}

}

```

```
}  
}  
}
```

```
// print sorted map
```

```
cout << "----- 10-1. 100개의 random한 점에 의한 Convex Hull 결정 및 출력 -----" << endl;
```

```
cout << "랜덤한 점들의 좌표: " << endl;
```

```
for (int i = 0; i < N; i++) {
```

```
cout << to_string(polygon[i].c) << "(" << polygon[i].x << ", " << polygon[i].y << ")"
```

```
<< " / ";
```

```
}
```

```
cout << endl
```

```
<< endl;
```

```
// 결정된 단순 다각형 출력 (print sorted map), 수평각 계산 회수, 각의 비교 회수 출력
```

```
cout << "랜덤한 점으로 결정된 단순 다각형: " << endl;
```

```
for (auto &p : polygonVec) {
```

```
cout << to_string(p.first) << " - ";
```

```
}
```

```
cout << to_string(polygonVec[0].first) << endl
```

```
<< endl;
```

```
// make sorted poly vector include x, y with polygonVec
```

```
vector<point> sortedPolygon;
```

```
for (auto &p : polygonVec) {
```

```
sortedPolygon.push_back(polygon[p.first]);
```

```
}
```

```
// make Convex Hull with Graham Scan with sortedPolygon
```

```
GrahamScan(sortedPolygon);
```

```
// print Convex Hull
```

```
cout << "결정된 Convex Hull: ";
```

```
for (int i = 0; i < sortedPolygon.size(); i++) {
```

```
cout << to_string(sortedPolygon[i].c) << " - ";
```

```
}
```

```
cout << to_string(sortedPolygon[0].c) << endl
```

```
<< endl;
```

```
// -----
```

```
// ----- 10-2. 3차원 평면상에서 랜덤한 N개의 점을 생성하고 최근접 점 쌍 찾기 -----
```

```
// -----
```

```
// create 100 random point
```

```
vector<point3> pointList;
```

```
for (int i = 0; i < 100; i++) {
```

```
point3 p;
```

```
// x, y, z의 좌표값은 100 이하
```

```
p.x = rd() % 100;
```

```
p.y = rd() % 100;
```

```
p.z = rd() % 100;
```

```
p.c = i;
```

```
pointList.push_back(p);
```

```
}
```

```
// print random point
```

```
cout << "----- 10-2. 3차원 평면상에서 랜덤한 N개의 점을 생성하고 최근접 점 쌍 찾기 -----  
" << endl;
```

```

cout << "랜덤한 점들의 좌표: " << endl;

for (int i = 0; i < 100; i++) {

cout << to_string(pointList[i].c) << "(" << pointList[i].x << ", " << pointList[i].y << ", " << pointList[i].z << ")"

<< " / ";

}

// sort pointList by x (Insertion sort)

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

for (int j = 0; j < i; j++) {

if (pointList[i].x < pointList[j].x) {

pointList.insert(pointList.begin() + j, pointList[i]);

pointList.erase(pointList.begin() + i + 1);

break;

}

}

}

// find closest pair

pair<point3, point3> closestPair = ClosestPair(pointList);

// print closest pair

cout << endl

<< endl;

cout << "최근접 점 쌍: " << to_string(closestPair.first.c) << "(" << closestPair.first.x << ", " << closestPair.first.y << ", "

<< closestPair.first.z << ")"

<< " - " << to_string(closestPair.second.c) << "(" << closestPair.second.x << ", " << closestPair.second.y << ", " <<

closestPair.second.z << ")" << endl;

cout << "최근접 점 쌍 사이의 거리 : " << sqrt(pow(closestPair.first.x - closestPair.second.x, 2) +

pow(closestPair.first.y - closestPair.second.y, 2) + pow(closestPair.first.z - closestPair.second.z, 2)) << endl;

}

```

