# HW2 보고서

로빛 20기 인턴 2025407006 모시온

## 목차

## 1. hw2 프로젝트 개요

본 프로젝트는 사용자에게서 전체 좌표의 개수와 좌표의 최솟값과 최댓값의 입력을 통해 좌표의 범위를 제한하여 해당 범위 내에서 무작위의 좌표를 생성한 후 두 점 사이의 거리가 가장 가까운 점들의 묶음과 가장 먼 점들의 묶음을 도출하는 코드입니다.
해당 코드는 좌표와 무작위 요소, 동적할당을 활용해야하는 조건이 요구됩니다.

## 2. 알고리즘 설계

2-1. .hpp 설명

```
#include <iostream>
#include <cstdlib>
#include <ctime>
```

hw2.hpp 안에서는 위와 같은 라이브러리를 호출하였습니다.
첫번째부터 순서대로, 기본 입력 및 출력을 위한 모듈, 구조체의 동적할당 및 무작위의 난수 생성을 위한 모듈, 마지막으로 시간을 인식하는 모듈을 통해 각 시간을 시드로서 구분지어 해당 코드에서는 같은 점의 중복을 방지할 수 있도록 사용하는 모듈들이 호출되었습니다.

```
namespace HW2{

//structure 구조체 활용
struct coor {
    int x, y;
};

class Coordinate {
private:
    coor* p;
    int p_number;

public:
    Coordinate(int n, int min, int max);
    ~Coordinate();

    double distance(coor& p1, coor& p2);
    void mmdistance();
    void print();
};
```

이후 네임스페이스를 HW2 로 지정하여, 직접적으로 해당 보고서 및 활동에 연관된 외부인이 해당 코드의 용도를 직관적으로 파악할 수 있도록 하였습니다.

구조체(structure)활용 조건을 달성하기 위하여 coordinate 의 coor 부분을 인용하여 각각 x 좌표, y 좌표를 통해 점의 위치를 멤버 변수로서 사용하였습니다.

이후 Coordinate Class 의 private 영역에, 점들의 좌표를 저장할 coor 배열 포인터 p 와, 점의 개수를 나타내는 p_number 를 선언하였습니다.
Public 영역에서는 Class 의 멤버 함수를 사용하여 생성자 및 소멸자와 함께, 무작위로 생성되는 각각의 점들의 모든 거리를 계산하고 비교하여 최대거리 떨어진 점들의 집합 및 최소거리 떨어진 점들의 집합을 구하도록 하였습니다.

```
//생성자
Coordinate::Coordinate(int n, int min, int max) {
    p_number = n;
    p = new coor[p_number];
    srand((unsigned int)time(NULL));
    for (int i = 0; i < p_number; i++) {
        //좌표는 정수형
        p[i].x = min + rand() % (max - min + 1);
        p[i].y = min + rand() % (max - min + 1);
    }
}

//소멸자
Coordinate::~Coordinate() {
    delete[] p;
}

//거리 계산
double Coordinate::distance(coor& p1, coor& p2) {
    double dx = p1.x - p2.x;
    double dy = p1.y - p2.y;
    return newtonSqrt(dx * dx + dy * dy);
}
```

생성자와 소멸자입니다.
생성자에서는 Class Coordinate 의 멤버 변수들을 초기화하도록 하며 해당 코드에서 가장 중요한 무작위 난수를 통한 랜덤 점을 사용자가 지정한 점의 전체 개수 n(매개변수), p_number 만큼 for 문을 반복, 포인터 배열로 선언한 p 의 x 와 y 의 각각의 값을 정수형태로 무작위로 받을 수 있게 하였습니다.
이때 중요한 점으로, 사용자가 최소값을 지정하기 때문에 min 을 더하며 시작, 그리고 해당 값에 최대값 또한 지정되기 때문에 최대값 – 최소값 +1(최댓값을 포함하기 위함)을 통해 rand 의 범위를 지정하였습니다.

소멸자에서는 포인터 배열로 선언한 변수 p 를 해제하도록 하였습니다.
이후 각각의 점의 거리를 구하는 방법은 과제의 공지된 내용과 동일하게 dx^2 와 dy^2 를 더한 후 제곱근 연산을 적용하는 방식을 적용하였습니다.
이때 해당 dx^2 + dy^2 의 값을 함수 newtonSqrt 로 반환하며, 해당 함수는 별도의 sqrt 함수없이 직접적으로 제곱근을 계산하는 함수로서 거리를 최종적으로 도출합니다.

```
//뉴튼-랩슨법으로 sqrt
double newtonSqrt(double n) {
    if (n < 0) return -1;
    double t = n;
    for (int i = 0; i < 16; i++) {
        if (t < 1.0) break;
        t = (t * t + n) / (2.0 * t);
    }
    return t;
}
```

직접적으로 제곱근 결과를 도출하는 방식은 뉴튼랩슨법의 원리를 참고하였습니다.
제곱근 값을 구하고자하는 값을 n 으로서 전달받으며 이와 근사값 또는 해당 값을 t 를 통해 저장하며 근사값을 계산하기 위해 다음과 같은 점화식을 활용하여

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

현재 정의된 연산 과정 t = t^2 + n / 2t 를 이에 적용하면

$$t_{next} = \frac{t^2 + n}{2t}$$

의 형태를 이루며 이를 현재 16 번 반복을 통해 근사치를 구하도록 하였습니다.
이를 통해 각각의 점들의 모든 거리를 구하며 이후 다음 함수를 통해 모든 랜덤 좌표를 출력합니다.

```
//모든 점 출력
void Coordinate::print() {
    for (int i = 0; i < p_number; i++) {
        std::cout << "Point ";
        std::cout << i+1;
        std::cout << ": nX=";
        std::cout << p[i].x;
        std::cout << " , nY=";
        std::cout << p[i].y << std::endl;
    }
}
```

이는 p_number 만큼 반복하는 for 문에서 포인터 배열 p 의 x, y 값을 순서대로 출력하는 원리로 구동됩니다.

마지막으로 현재 도출된 모든 좌표 서로간의 거리를 구하여 그 값을 각각 아래의

```cpp
//최소, 최대 거리 계산
void Coordinate::mmdistance() {
    if (p_number < 2) {
        std::cout << "Lack Point Number" << std::endl;
        return;
    }

    double min_distance = distance(p[0], p[1]);
    double max_distance = min_distance;
    coor min_coor1 = p[0], min_coor2 = p[1];
    coor max_coor1 = p[0], max_coor2 = p[1];

    for (int i = 0; i < p_number; i++) {
        for (int j = i + 1; j < p_number; j++) {
            double d = distance(p[i], p[j]);
            if (d < min_distance) {
                min_distance = d;
                min_coor1 = p[i];
                min_coor2 = p[j];
            }
            if (d > max_distance) {
                max_distance = d;
                max_coor1 = p[i];
                max_coor2 = p[j];
            }
        }
    }
```

min_distance 와 max_distance 에 저장한 후, 최대거리와 최소거리에 해당하는 좌표를 각각 배열 p 를 통해 min_coorn, max_coorn 에 저장합니다. 전제 점의 각각의 거리를 구하며 해당 거리의 차이를 min_distance 또는 max_distance 와 비교하여 업데이트하는 방식으로 정렬하며, 아래의 코드를 통해 전체 결과를 출력합니다.

```cpp
//결과 출력
std::cout << "-------- Result --------" << std::endl;
std::cout << "MinDist=";
std::cout << min_distance << std::endl;
std::cout << "Pair of Min Coor.(x,y): P1(";
std::cout << min_coor1.x;
std::cout << ",";
std::cout << min_coor1.y;
std::cout << ") & P2(";
std::cout << min_coor2.x;
std::cout << ",";
std::cout << min_coor2.y;
std::cout << ")" << std::endl;

std::cout << "MaxDist=";
std::cout << max_distance;
std::cout << std::endl;
std::cout << "Pair of Max Coor.(x,y): P1(";
std::cout << max_coor1.x;
std::cout << ",";
std::cout << max_coor1.y;
std::cout << ") & P2(";
std::cout << max_coor2.x;
std::cout << ",";
std::cout << max_coor2.y;
std::cout << ")" << std::endl;
```

2-2. .cpp 설명

. `#include "hw2.hpp"`

앞서 설명한 것과 같이 헤더파일에 iostream 를 호출하였으므로 추가적인 호출없이도 기본적인 입출력을 사용할 수 있기에 include 는 하나만 호출하였습니다.

`using namespace HW2;`

헤더파일에서 사용하는 namespace 를 불러왔습니다.

```cpp
int main() {
    double min, max;
    double n;

    std::cout << "******** HW2 Point Distance Computation ********" << std::endl;
    std::cout << std::endl;
    std::cout << std::endl;

    std::cout <<"Please define the number of points: ";
    std::cin >> n;

    if(n <= 2 || !n || std::cin.fail()){
        std::cout << "종료합니다" << std::endl;
        std::cin.clear();
        std::cin.ignore(5, '\n');
        return -1;
    }
```

사용자의 점의 개수를 입력받을 n 과 최소값 min, 최대값 max 를 실수형으로 선언하며 이때 n 의 입력이 올바르지 않은 경우를 대비하여 조건문을 통해 다른 자료형(문자)을 입력하거나 최소 2 개의 점이 필요한 상황에서 1 개 또는 그 이하의 점 개수를 입력할 경우 다음과 같이 예외처리될 수 있도록 하였습니다.

```
sion@sion-Laptop:~/intern_ws/hw2/build$ ./test
******** HW2 Point Distance Computation ********


Please define the number of points: k
종료합니다
sion@sion-Laptop:~/intern_ws/hw2/build$ ./test
******** HW2 Point Distance Computation ********


Please define the number of points: -1
종료합니다
sion@sion-Laptop:~/intern_ws/hw2/build$ ./test
******** HW2 Point Distance Computation ********


Please define the number of points: 1
종료합니다
```

이는 max 와 min 에도 동일하게 적용하여 올바르지 못한 값의 입력에 대해 대비하였습니다.

```cpp
std::cout << "Please define minimum of coor. value: ";
std::cin >> min;
if(min < 0 || !min || std::cin.fail()){
    std::cout << "종료합니다" << std::endl;
    std::cin.clear();
    std::cin.ignore(5, '\n');
    return -1;
}

std::cout << "Please define maximum of coor. value: ";
std::cin >> max;

if(max < min || !max || std::cin.fail()){
    std::cout << "종료합니다" << std::endl;
    std::cin.clear();
    std::cin.ignore(5, '\n');
    return -1;
}
```

마지막으로 올바르게 입력된 값만이 있을 때,

```
std::cout << "Generate Random points" << std::endl;
Coordinate compute((int)n, (int)min, (int)max);
compute.print();

compute.mmdistance();

std::cout << "******** Completed ********" << std::endl;
return 0;
```

좌표 연산을 compute 의 멤버 변수에 각각의 값을 정수형
(과제 공지 예시에서는 정수형만이 입력된 것을 참고하여 정수만이 입력되도록 하였습니다.)
으로 변환한 후 초기화하여 결과를 도출할 수 있도록 하였습니다.

## 3. 실행 결과