Homework2

CSI3108-1 Algorithms

27167033 남다예

1. 알고리즘

클래스

CollinearPts	메인 함수가 포함되어 있는 클래스이다.

Coordinate	점들의 좌표들을 저장하는 클래스. x, y좌표를 저장하고, count 안에는 몇 개의 collinear라인이 지나는지를 저장한다.
	
	`public static class Coordinate {`
	`int x = 0;`
	`int y = 0;`
	`int count = 0;`
	`}`

Line	직선들의 정보를 저장하는 클래스. 클래스 내의 함수로 두 점의 좌표를 넣으면 기울기와 x절편을 계산하여 저장한다.
	
	`public class Line {`
	`float slope;`
	`float inter_x;`
	
	`Line(int a_x, int a_y, int b_x, int b_y){`
	`this.slope = calculateSlope(a_x, a_y, b_x, b_y);`
	`this.inter_x = calculateItsX(a_x, a_y, b_x, b_y);`
	`}`
	
	`static float calculateSlope(int a_x, int a_y, int b_x, int b_y);`
	`static float calculateItsX(int a_x, int a_y, int b_x, int b_y);`
	`}`

자료구조

pts_array

```
ArrayList<Coordinate> pts_array = new ArrayList<Coordinate>();
```

좌표들을 lexicographical order에 따라 정렬하기 위해서 ArrayList안에 있는 comparator를 override하여 사용하였다. 먼저 두 점들의 x좌표들을 비교한 후, 같다면 y좌표들을 비교하여 정렬하였다.

```
public static Comparator<Coordinate> lexAscOrder = new Comparator<Coordinate>() {
    @Override
    public int compare(Coordinate o1, Coordinate o2) {
    if (o1.x == o2.x)
        return o1.y > o2.y ? 1 : (o1.y == o2.y ? 0 : -1);
    else
        return o1.x > o2.x ? 1 : (o1.x == o2.x ? 0 : -1);
    }
};
```

line_map

```
HashMap<Line, TreeSet<Integer>> line_map = new HashMap<Line, TreeSet<Integer>>();
```

Line

```
public class Line {
        float slope;
        float inter_x;
}
```

TreeSet 〈Integer〉 Point들의 index를 저장

```
public int hashCode(){
    int hashcode = 17;
    hashcode = 31 * hashcode + Float.floatToIntBits(slope);
    hashcode = 31 * hashcode + Float.floatToIntBits(inter_x);
    return hashcode;
}

public boolean equals(Object obj){
    if (obj instanceof Line) {
        Line pp = (Line) obj;
        return ((Float.compare(pp.slope, this.slope)==0) && (Float.compare(pp.inter_x,
        this.inter_x)==0));
    }
    else {
        return false;
    }
}
```

key를 Line이라는 클래스로 사용하였기 때문에 자바에 기본적으로 구현되어있는 hashcode와 equals를 사용할 수 없어 hashcode()와 equals()를 Line 클래스 안에 override하였다. 이 때 중복을 막고 효율적으로 해쉬맵을 사용하기 위해 17과 31등의 소수를 사용하였고, equals와 hashCode 안에서 같은 element들만을 사용하였다.

두 점들로 만들어질 수 있는 모든 직선들의 정보를 저장하기 위해 해쉬맵을 사용하였다. class Line 안에 slope와 inter_x(x절편)들을 저장하고 이 것을 키로 사용하였다. 따라서 같은 slope와 inter_x를 가진 두 점이 있다면 O(1)로 바로 확인할 수 있고, TreeSet에 추가하여 그 라인 위에 있는 점들의 index 또한 O(1)로 알 수 있도록 하였다. 이 때 TreeSet을 사용한 이유는 점들의 index들을 중복되지 않게 저장하기 위함이다.

MAIN ALGORITHM

1. 주어진 인풋을 차례로 읽으면서 좌표 정보들을 Coordinate 클래스 안에 넣어 pts_array 안에 add시킨다.
2. 좌표의 입력이 끝나면 pts_array의 comparator를 사용하여 lexicographical order로 정렬한다.
3. 모든 점들 사이의 기울기와 x절편을 계산하여 hashmap에 Line을 키로 put 한다. 이 때 같은 key를 가진 value가 이미 있다면 새로 put하지 않고 value에 점의 정보만 add한다
4. hash map의 구성이 끝나면 value가 점 3개 이상 포함하고 있는 것들만 찾아 pts_array 안에서 그 점들의 Coordinate.count에 1씩 더해준다. 이 과정에서 각 점마다 몇 개의 라인을 지나는지 알 수 있다.
5. pts_array 안에 있는 점 중 2개 이상의 라인을 지나는 점을(Coordinate.count가 2 이상인 점) 찾아 출력한다.

Coordinate	
(1,1), (1,2)	
Hash Map	
Key	Value
slope	x절편	points
∞	1	0, 1

Coordinate	
(1,1), (1,3)	
Hash Map	
Key	Value
slope	x절편	points
∞	1	0, 1, 2

Coordinate	
(1,1), (1,4)	
Hash Map	
Key	Value
slope	x절편	points
∞	1	0, 1, 2, 3

Coordinate	
(1,1), (2,2)	
Hash Map	
Key	Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4

Coordinate	
(1,1), (3,3)	
Hash Map	
Key	Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5

Coordinate	
(1,2), (1,3)	
Hash Map	
Key	Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5

Coordinate	
(1,2), (1,4)	
Hash Map	
Key	Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5

Coordinate	
(1,2), (2,2)	
Hash Map	
Key	Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5
0	2	1, 4

Coordinate		
(1,2), (3,3)		
Hash Map		
Key		Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5
0	2	1, 4
0.5	1.5	1, 5

Coordinate		
(1,3), (1,4)		
Hash Map		
Key		Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5
0	2	1, 4
0.5	1.5	1, 5

Coordinate		
(1,3), (2,2)		
Hash Map		
Key		Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5
0	2	1, 4
0.5	1.5	1, 5
-1	4	2, 4

Coordinate		
(1,3), (3,3)		
Hash Map		
Key		Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5
0	2	1, 4
0.5	1.5	1, 5
-1	4	2, 4
0	3	2, 5

Coordinate		
(1,4), (2,2)		
Hash Map		
Key		Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5
0	2	1, 4
0.5	1.5	1, 5
-1	4	2, 4
0	3	2, 5
-2	6	3, 4

Coordinate		
(1,4), (3,3)		
Hash Map		
Key		Value
slope	x절편	points
∞	1	0, 1, 2, 3
1	0	0, 4, 5
0	2	1, 4
0.5	1.5	1, 5
-1	4	2, 4
0	3	2, 5
-2	6	3, 4
-0.5	4.5	3, 5

Coordinate		
(2,2), (3,3)		
Hash Map		
Key		Value
slope	x절편	points
∞	**1**	**0, 1, 2, 3**
1	**0**	**0, 4, 5**
0	2	1, 4
0.5	1.5	1, 5
-1	4	2, 4
0	3	2, 5
-2	6	3, 4
-0.5	4.5	3, 5

해쉬 맵에 정보를 모두 put 한 결과, (slope, x절편) 이 (∞,1), (1,0) 인 경우에만 3개 이상의 점을 지난다는 것을 알 수 있고, 이 선들이 collinear 라인이라는 것 또한 알 수 있다. 이 때 index가 0 인 점만 한 개 이상의 collinear 라인 위에 존재하므로 (1,1)인 점 하나만 결과값으로 출력하게 된다.

2. 예시 결과 및 실행시간

1. runtimeTest1.txt

➜ 0

2. runtimeTest2.txt

➜ 0

3. runtimeTest3.txt

➜ 0

Test Case	1	2	3	4	5	6	7	8	9	10	평균
1	0	0	0	0	0	0	1	1	1	1	0.4
2	35	39	42	39	38	36	45	45	54	43	41.6
3	37	34	41	37	47	29	46	64	36	51	42.2

실행시간을 분석 해 본 결과, 테스트케이스 1의 경우가 평균 0.4 ms, 2의 경우 41.6ms, 3의 경우가 42.2ms이 걸리는 것을 확인할 수 있었다. 가장 짧은 테스트 케이스의 실행시간이 가장 낮았고, test2와 test3의 경우 input의 길이가 길다보니 이 과정에서 오래 걸린 것으로 추정할 수 있다. 또한 점들의 갯수가 많아지만 그만큼 직선들의 기울기와 x 절편을 계산하고 저장하는 과정에서 시간이 지체될 수 있기 때문에 실행 시간은 논리적으로 합당해 보인다.