1) 선분 간 가까운 거리 출력하기

Closet pair 알고리즘의 동작방식을 이해한다.
Closet pair 알고리즘의 시간복잡도를 계산한다.
Closet pair 문제를 Divide and Conquer 알고리즘을 이용하여 해결한다.

● 사용 언어

Python

● 사용 라이브러리

sys (Command Line input을 위해)
math (두 점 사이의 거리를 계산하기 위해)

● 사용 알고리즘

Divide and conquer

■ Closest Pair [Big O(n lg n)]

Brute Force [이 알고리즘에선 Big O (n^2)]

● 아이디어

Closest pair 알고리즘 내에서, 두 선분의 최단거리를 구한다.
(겹치는 선분을 제거하기 때문에 가능한 일)
Brute Force를 통해 모든 선분 사이의 최단거리를 구한다.
(Closet Pair 알고리즘의 결과가 '선분'의 최단거리임을 장담할 수 없음)

● 순서

1) 겹치는 선분들을 모두 제거한다.
2) Closest Pair 알고리즘을 사용하여 두 점 사이 최단거리를 구한다.
두 점 사이의 최단 거리가 두 선분 사이의 최단 거리임을 확정지을 수 없음
-> 모든 두 선분 사이의 거리를 구하고, 두 점 사이의 최단거리를 구해 minimum을 반환
3) 두 점이 한 선분내에 있지 않다면 두 선분 사이의 거리를 구한다.

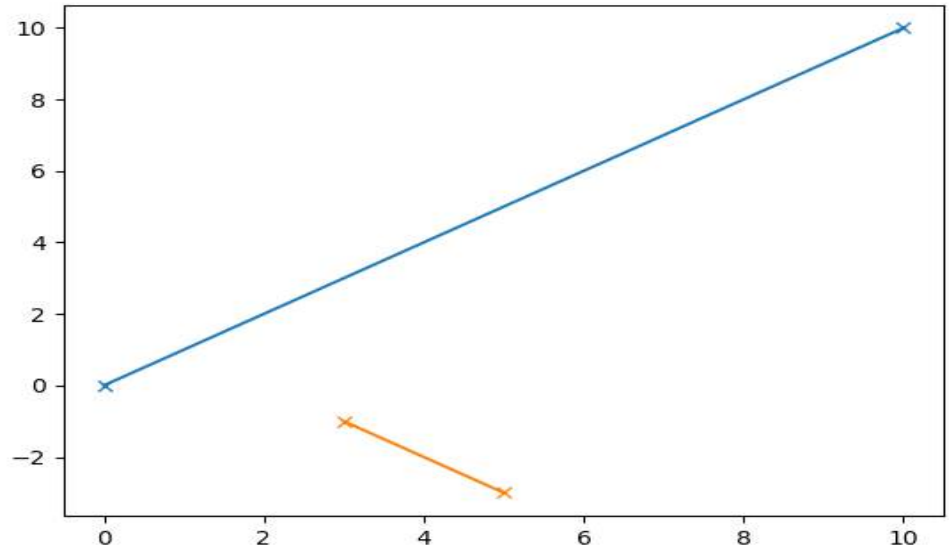
■ 두 점 사이의 최단 거리가 두 선분 사이의 최단 거리가 아님을 증명

[(0, 0) , (10, 10)] -> y = x

[(3, -1) , (5, -3)] -> y = -x + 2

두 직선의 교점 (1, 1) 과 (3, -1) 의 거리는 2.8 이나,

두 점 사이의 최단거리 (0, 0) (3, -1) 은 3.1 이다.




```
def main() :
    '''
    if len(sys.argv) != 2 :
        print("Usage : PA01-1.py [INPUT FILE]")
        return
    '''

    inputFilename = "PA01-1_input.txt"#sys.argv[1]

    global Segments
    line_segments = list()
    points = list()
    del_array = list()
    Distance = list()

    # 파일을 읽어 배열에 저장
    with open(inputFilename, "r") as f :
        tmpList = f.readlines()
        for b in tmpList :
            tmpSegment = list(map(int, b.rstrip().split()))
            if tmpSegment[0] == -1 :
                break

            if len(tmpSegment) != 1 : # 처음 개수, -1 제외
                point1 = Point(tmpSegment[0], tmpSegment[1])
                point2 = Point(tmpSegment[2], tmpSegment[3])
                line_segments.append(LineSegment(point1, point2))

                # 교차점 있는 선분 확인.
                # 교차점이 있는 선분 확인
                # O(n^2)  -> 파일 for list,  순회 for   n*n
                for i in range(0, len(line_segments)-1) :
                    if segments_intersect( line_segments[i], line_segments[-1] ) :
                        del_array.extend([line_segments[i],line_segments[len(line_segments)-1]])

    # 교차점이 있는 선분 제거
    del_array = set (del_array)
    line_segments = list(set(line_segments) - del_array)

    # 교차점을 모두 제거한 이후 선분이 1개 이하일 때 거리 0을 출력후 리턴.
    if len(line_segments) <= 1 :
        print("0")
        print("Line Segments Less than 2")
        return

    # Closest point 알고리즘을 위해 선분 점들을 배열에 저장
    for i in range(len(line_segments)) :
        l = line_segments[i]
        p1 = (l.p1.x, l.p1.y)
        p2 = (l.p2.x, l.p2.y)
        points.append(p1)
        points.append(p2)
        Segments.append([p1, p2])
        # 모든 선분의 거리를 배열에 추가
        for j in range(i+1, len(line_segments)) :
            Distance.append( segments_distance(line_segments[i], line_segments[j]) )

    print(Segments)

    # Closest pair 거리 추가
    Distance.append(divideMin(points))
    print(divideMin(points))

    print(min(Distance))

main()
```

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019,
1)] on win32
Type "help", "copyright", "credits" or "licens
>>>
 RESTART: C:₩Users₩lejae₩Desktop₩알고리즘₩MyBC
고리즘₩Assignment #1₩PA01-1.py
[[(1, 1), (9, 9)], [(4, 0), (6, -3)]]
3.1622776601683795
2.8284271247461903
>>>
 RESTART: C:₩Users₩lejae₩Desktop₩알고리즘₩MyBC
고리즘₩Assignment #1₩PA01-1.py
[[(4, 0), (6, -2)], [(0, 0), (10, 10)]]
4.0
2.8284271247461903
>>>
 RESTART: C:₩Users₩lejae₩Desktop₩알고리즘₩MyBC
고리즘₩Assignment #1₩PA01-1.py
[[(4, 0), (7, -1)], [(0, 0), (10, 10)]]
4.0
2.8284271247461903
>>>
 RESTART: C:₩Users₩lejae₩Desktop₩알고리즘₩MyBC
고리즘₩Assignment #1₩PA01-1.py
[[(0, 0), (10, 10)], [(3, -1), (5, -3)]]
3.1622776601683795
2.8284271247461903
>>>
 RESTART: C:₩Users₩lejae₩Desktop₩알고리즘₩MyBC
고리즘₩Assignment #1₩PA01-1.py
[[(3, -1), (5, -3)], [(0, 0), (10, 10)]]
3.1622776601683795
2.8284271247461903
>>>
```

■ 알고리즘 분석

closest pair

```
from math import sqrt, pow

INF = sys.maxsize
# 거리를 반환하는 함수
def distance(a, b):
    global Segments
    # 두 점이 같은 선분 내일 때 INF 반환
    for i in Segments :
        if a in i :
            if b in i :
                return INF
    return sqrt(pow(a[0] - b[0],2) + pow(a[1] - b[1],2))

# 최단 거리를 반환
def bruteMin(points, current=INF):
    if len(points) < 2: return current
    else:
        head = points[0]
        del points[0]
        newMin = min([distance(head, x) for x in points])
        newCurrent = min([newMin, current])
        return bruteMin(points, newCurrent)

# closet pair T(n) = 2T(2/n) + O(n)
def divideMin(points):
    half = len(sorted(points))//2
    minimum = min([bruteMin(points[:half]), bruteMin(points[half:])])
    nearLines = filter(lambda x: x[0] > half - minimum and x[0] < half + minimum, points)
    nearLine = list()
    for l in nearLines :
        nearLine.append(l)
    return min([bruteMin(nearLine), minimum])
```

Closest Pair 알고리즘이다.

T(n) = 2T(2/n) + O(n) 이므로 O(n log n) 이다.

■ 두 직선 사이의 거리

두 직선이 교차하는지의 확인은 CCW를 이용하여 판정하였다.

두 직선 사이의 거리는 점과 선분 사이의 거리를 이용하였는데

선분이 두 개이므로 네 점에서 거리를 판별한다.

알고리즘은 파일 내에 있다.

재귀 함수가 아니고, 단순 계산이므로 시간 복잡도는 O(1) 이다.

■ 교차 판별

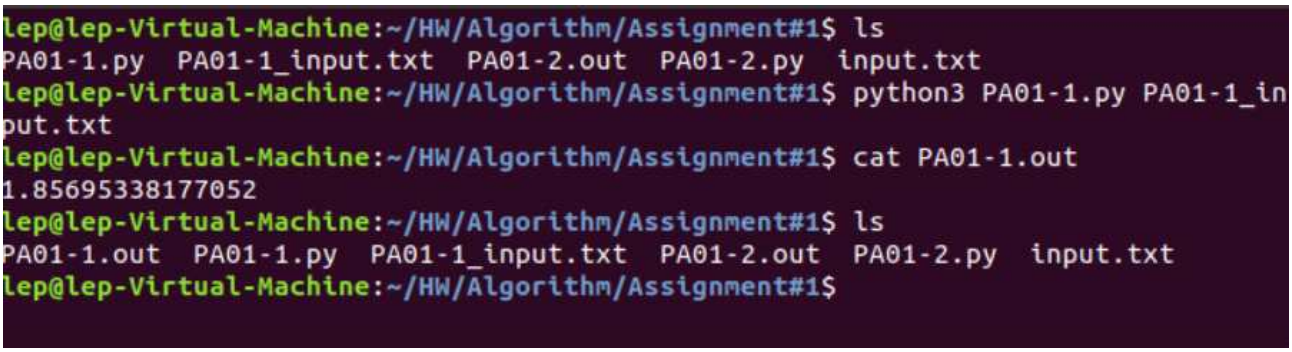
파일을 읽으면서 이전의 모든 선분을 확인하므로 이중 for 문이고 시간 복잡도는 O(n^2) 이다.

```
# Main

Segments = list()

def main() :

    if len(sys.argv) != 2 :
        print("Usage : PA01-1.py [INPUT FILE]")
        return


    inputFilename = sys.argv[1]

    global Segments
    line_segments = list()
    points = list()
    del_array = list()
    Distance = list()

    # 파일을 읽어 배열에 저장
    with open(inputFilename, "r") as f :
        tmpList = f.readlines()
        for b in tmpList :
            tmpSegment = list(map(int, b.rstrip().split()))
            if tmpSegment[0] == -1 :
                break
```

```
            if len(tmpSegment) != 1 : # 처음 개수, -1 제외
                point1 = Point(tmpSegment[0], tmpSegment[1])
                point2 = Point(tmpSegment[2], tmpSegment[3])
                line_segments.append(LineSegment(point1, point2))

                # 교차점 있는 선분 확인.
                # 교차점이 있는 선분 확인
                # O(n^2)  -> 파일 for list,  순회 for   n*n
                for i in range(0, len(line_segments)-1) :
                    if segments_intersect( line_segments[i], line_segments[-1] ) :
                        del_array.extend([line_segments[i],line_segments[len(line_segments)-1]])

    # 교차점이 있는 선분 제거
    del_array = set (del_array)
    line_segments = list(set(line_segments) - del_array)

    # 교차점을 모두 제거한 이후 선분이 1개 이하일 때 거리 0을 출력후 리턴.
    if len(line_segments) <= 1 :
        print("0")
        print("Line Segments Less than 2")
        return

    # Closest point 알고리즘을 위해 선분 점들을 배열에 저장
    for i in range(len(line_segments)) :
        l = line_segments[i]
        p1 = (l.p1.x, l.p1.y)
        p2 = (l.p2.x, l.p2.y)
        points.append(p1)
        points.append(p2)
        Segments.append([p1, p2])
        # 모든 선분의 거리를 배열에 추가
        for j in range(i+1, len(line_segments)) :
            Distance.append( segments_distance(line_segments[i], line_segments[j]) )

    # Closest pair 거리 추가
    Distance.append(divideMin(points))
    points.sort()
    print(min(Distance))

main()
```

모든 시간 복잡도를 고려한 총 시간 복잡도는 O(n^2) 이다.

결과 스크린샷


lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$ ls
PA01-1.py PA01-1_input.txt PA01-2.out PA01-2.py input.txt
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$ python3 PA01-1.py PA01-1_in
out.txt
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$ cat PA01-1.out
1.85695338177052
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$ ls
PA01-1.out PA01-1.py PA01-1_input.txt PA01-2.out PA01-2.py input.txt
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$

2) 아파트 스카이라인 그리기

Skyline 알고리즘의 동작방식을 이해한다.
Skyline 알고리즘의 시간복잡도를 계산한다.
Skyline 문제를 Divide and Conquer 알고리즘을 이용하여 해결한다.

● 사용 언어

Python

● 사용 라이브러리

sys (Command Line input을 위해)

● 사용 알고리즘

Divide and conquer

■ Merge Sort [Big O(n lg n)]

● 아이디어

Merge Sort 알고리즘 내에서 변화 Point를 찾아 추가한다.

skyline 알고리즘을 살펴보자. (merge_sort를 사용하였다.)

```
def skyline(buildingList) :

    # empty
    if not buildingList :
        return list()

    if len(buildingList) == 1 :
        l,h,r = buildingList[0]
        return [[l, h], [r , 0]]

    count = len(buildingList)

    mid = count // 2
    left = skyline(buildingList[:mid])
    right = skyline(buildingList[mid:])

    return merge_sort(left, right)
```

파라미터엔 [x1,h,x2] 형태의 배열들이 저장되어있고 이를 나누어
모든 배열을 [[x1, h], [x2,h]] 로 쪼개어 Merge한다. (Merge Sort)

```
def merge_sort(left, right) :
```

```
    i,j = 0,0
    leftMaxHeight,rightMaxHeight = 0,0
    result = list()
    # point = 오른쪽 꼭짓점이 추가되는 튜플의 기준
    while i < len(left) and j < len(right) :
        if left[i][0] < right[j][0]:
            leftMaxHeight = left[i][1]
            point = left[i][0]
            i += 1
        elif right[j][0] < left[i][0]:
            rightMaxHeight = right[j][1]
            point = right[j][0]
            j += 1
        else:
            leftMaxHeight = left[i][1]
            rightMaxHeight = right[j][1]
            point = right[j][0]
            i += 1
            j += 1

        if not result or result[-1][1] != max(leftMaxHeight, rightMaxHeight) :
            result.append([point, max(leftMaxHeight, rightMaxHeight)])

    # 사용되지 않은 인자 Merge
    result.extend(right[j:])
    result.extend(left[i:])

    return result
```

Merge Sort 내부에서는 result에 결과 리스트를 저장하여 돌려주는데 약간의 수정을 하였다.
point (추가될 꼭짓점의 x좌표)와 좌우의 최대 높이를 result에 추가한다.
이 때 result함수가 비어있거나, 높이가 변화하는 경우에만 result에 추가해야하므로 관련
내용을 추가하였다.

또한, 사용하지 않은 리스트 내 인자도 합쳐줘야하므로 extend 해 주었다.

시간복잡도는 $T(n) = T(n/2) + O(n)$
$O(n \lg n)$ 이므로 Merge Sort와 같은 시간복잡도가 되겠다.

결과 스크린샷

```
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$ python3 PA01-2.py
Usage : PA01-2.py [INPUT FILE]
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$ python3 PA01-2.py input.txt
Success!!
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$ ls
PA01-2.out  PA01-2.py  input.txt
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$ cat PA01-2.out
1, 11
3, 13
9, 0
12, 7
16, 3
19, 18
22, 3
25, 0
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#1$
```