

알고리즘

- HW03 : Divide & Conquer -

제 출 일	2017년 10월 12일
분 반	02반
담당교수	공은배
학 과	컴퓨터공학과
학 번	201302423
이 름	신종욱

ㄱ. Loop Invariant

1. 해결 방법

Loop Invariant는 이진탐색을 LoopInvariant에 맞게 구현하면된다.

정리하자면

Precondition : A[1..n]은 정렬된 배열이다. (구현은 0번째 배열부터임으로 실제로는 A[0..n-1]로 구현했다)

Postcondition : 해당하는값이 A[1..n]에 있다면 해당 위치를 출력(return)해야한다.

Loop-invariant : 해당하는 값이 A[1..n]에 있다면 키는 A[p..q]에 있다고 할 수 있다

여기서 p와 q의 초기조건 즉 간단히 만족하는 것을 찾아야하는데 p=0 q=n-1로 하면 항상 옳다

Loop-invariant가 거짓이 될 때는 p가 q보다 클 경우 즉 $q < p$ 이다 역인 $q \geq p$ 를 while문의 조건에 넣어주면된다.

그 후 이진탐색을 구현 하였다. 만약 값이 없을경우를 대비해서 마지막에 if문도 하나 추가하였다.

하지만 애초에 Preconditon에 어긋남으로 이런일은 없어야한다.

```
int p = 0;
int q = count-1; //p와 q의 초기조건
int mid = 0;
while(q >= p) { //p가 q보다 클때 해당되지않기때문에 not상태인 반대를 조건으로 넣었다
    mid = (p+q)/2;
    if(key < data[mid]) {
        q = mid-1;
    }
    else if(key > data[mid]) {
        p = mid+1;
    }
    //이진탐색으로 검색할곳을 좁여가면서 하는 과정이다.
    else if(key == data[mid]) {
        System.out.println("찾으시는 값은 "+mid+"인덱스에 있습니다.");
        break;
    }
    if(p == count || mid == 0) {
        break; //만약 없을경우를 생각해서 예외 처리를 하기위해 만들어 졌다.
    }
}
```

2. 실행 결과

BinarySearch [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (2017. 10. 9. 오후 8:59:28)

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

90

찾으시는 값은 89인덱스에 있습니다.

검색완료

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

1

찾으시는 값은 0인덱스에 있습니다.

검색완료

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

33

찾으시는 값은 32인덱스에 있습니다.

검색완료

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

10000

찾으시는 값은 9999인덱스에 있습니다.

검색완료

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

0

없는 숫자입니다. 숫자를 다시 입력하세요

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

-456

없는 숫자입니다. 숫자를 다시 입력하세요

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

10001

없는 숫자입니다. 숫자를 다시 입력하세요

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

234234

없는 숫자입니다. 숫자를 다시 입력하세요

프로그램을 종료하려면 -1을, 숫자를 탐색하려면 수를 입력해주세요

└.Closest Pair

1. 해결 방법

Point클래스는 x값 y값을 함께 가진채로 정렬을 해야하기 때문에 List로 선언하였다 List로 선언하면 Collection.sort기능으로 정렬하기 편한것도 또 하나의 이유였다.

일단 3개이하일 때 최소값을 찾도록 한다음 그 외는 양쪽을 다시 Clopair 메소드를 사용하여 재귀적으로 구현하였다.

여기서 나온 left right의 ClosestPair를 alpha로 선언후 중심축을 기준으로 Window를 만들어줬다.

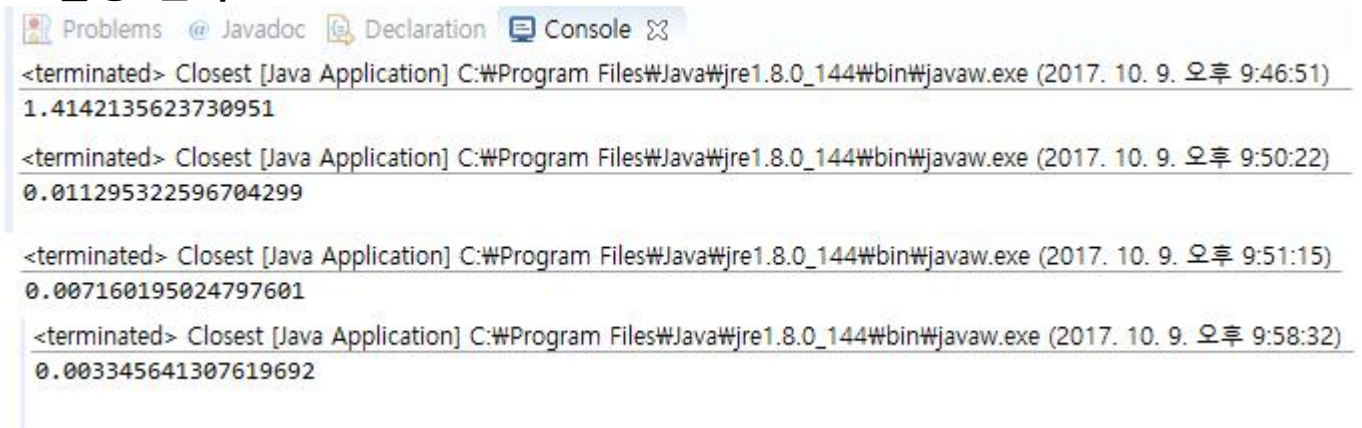
이제 window에서의 최소거리를 구하여야하는데 일단 y축으로 정렬해야한다

여기서 문제점은 x축으로 정렬로 구현하여서 y축으로도 정렬해야해서 두 개의 point 클래스로 선언하여서 2번째 point는 1번째 point 클래스를 상속 받지만 ComparableTo 메소드를 Override 하여서 y값을 비교하도록 하였다.

그리고 새로선언해서 y축정렬 하기 때문에 저장공간의 낭비는 있을지몰라도 본래의 x축의 정렬된 data값도 건드리지않았다.

window에서 최소값을 구할땐 옆 인덱스와 y축값의 차가 alpha이상이면 break를 해서 다음 인덱스부터 비교하기로 하여 실행시간을 줄일 수 있다.

2. 실행 결과



```
<terminated> Closest [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (2017. 10. 9. 오후 9:46:51)
1.4142135623730951

<terminated> Closest [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (2017. 10. 9. 오후 9:50:22)
0.011295322596704299

<terminated> Closest [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (2017. 10. 9. 오후 9:51:15)
0.007160195024797601

<terminated> Closest [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (2017. 10. 9. 오후 9:58:32)
0.003345641307619692
```

3. 느낀 점

Loop Invariant는 이진탐색을 기반으로 생각해서 구현해서 쉽게했다.

Closest Pair는 생각보다 오래걸렸는데 일단 알고리즘 자체가 상당히 복잡했기 때문에 코딩이 길어졌다.

그래도 이번과제를 하면서 확실히 이해하게 되었다.