징검다리

징검다리

문제 설명

출발지점부터 distance만큼 떨어진 곳에 도착지점이 있습니다. 그리고 그사이에는 바위들이 놓여있습니다. 바위 중 몇 개를 제거하려고 합니다.
예를 들어, 도착지점이 25만큼 떨어져 있고, 바위가 [2, 14, 11, 21, 17] 지점에 놓여있을 때 바위 2개를 제거하면 출발지점, 도착지점, 바위 간의 거리가 아래와 같습니다.

제거한 바위의 위치	각 바위 사이의 거리	거리의 최솟값
[21, 17]	[2, 9, 3, 11]	2
[2, 21]	[11, 3, 3, 8]	3
[2, 11]	[14, 3, 4, 4]	3
[11, 21]	[2, 12, 3, 8]	2
[2, 14]	[11, 6, 4, 4]	4

위에서 구한 거리의 최솟값 중에 가장 큰 값은 4입니다.

출발지점부터 도착지점까지의 거리 distance, 바위들이 있는 위치를 담은 배열 rocks, 제거할 바위의 수 n이 매개변수로 주어질 때, 바위를 n개 제거한 뒤 각 지점 사이의 거리의 최솟값 중에 가장 큰 값을 return 하도록 solution 함수를 작성해주세요.

제한사항

- 도착지점까지의 거리 distance는 1 이상 1,000,000,000 이하입니다.
- 바위는 1개 이상 50,000개 이하가 있습니다.
- n 은 1 이상 `바위의 개수` 이하입니다.

입출력 예

distance	rocks	n	return
25	[2, 14, 11, 21, 17]	2	4

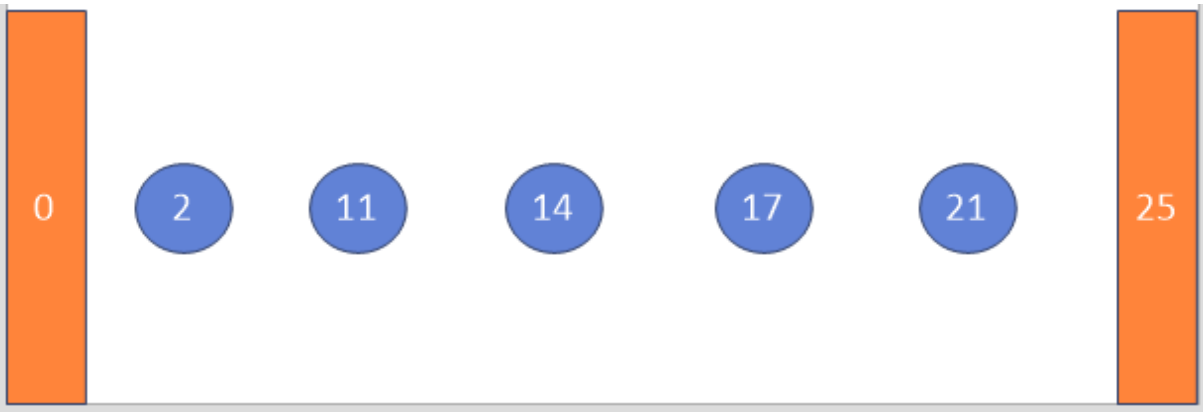
입출력 예 설명

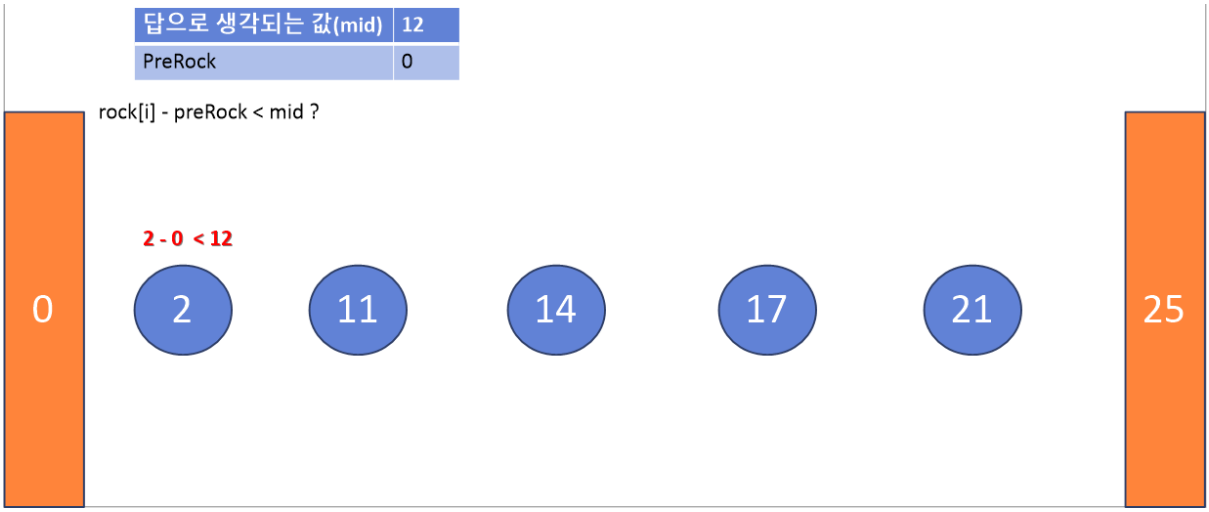
문제에 나온 예와 같습니다.

출처

※ 공지 - 2020년 2월 17일 테스트케이스가 추가되었습니다.

1. 답이 다양하게 나올 수 있는 문제에서 최소, 최대 값을 구하는 문제
2. 결과 값을 가정하고 범위 내에서 가능한지 판별 하는 형태로 값을 좁혀나간다.

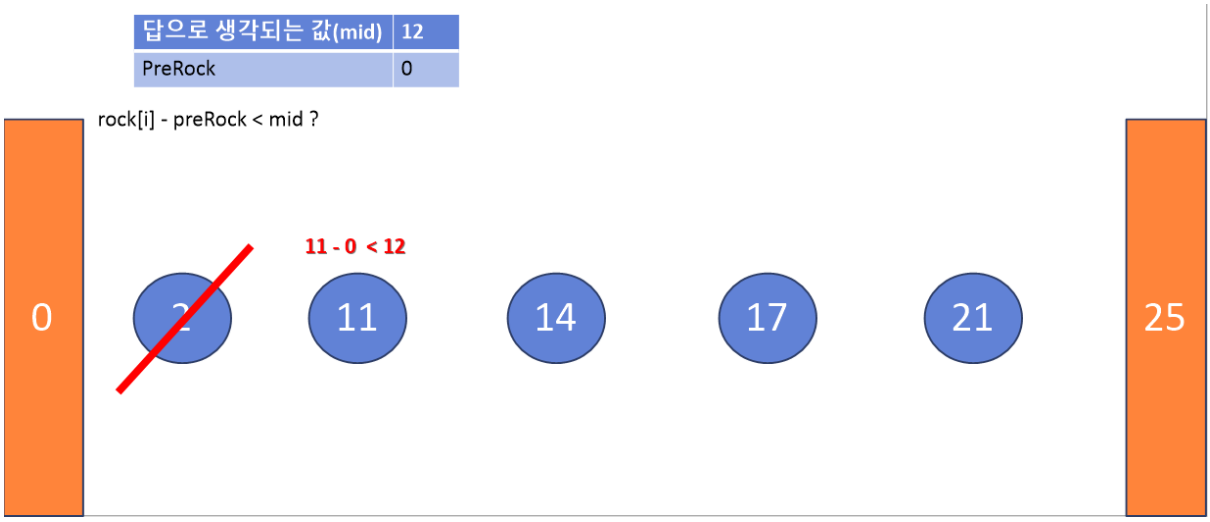




내가 생각하는 최소값은 12 인데
현재 돌 사이의 거리가 2 밖에 안됨

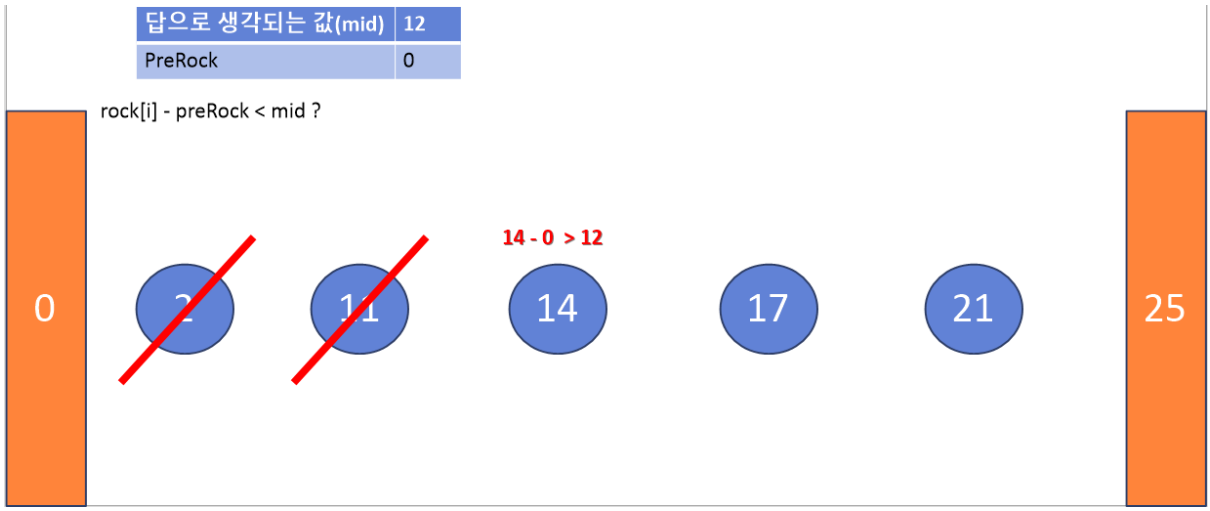
=> 이 돌은 현재 내 예상과 다르므로 이 돌을 제거하여 더 긴 길이를 찾아야함.

돌을 제거하고 count 기록 count = 1



마찬 가지로 예상 값보다 길이가 작으므로 돌을 제거함

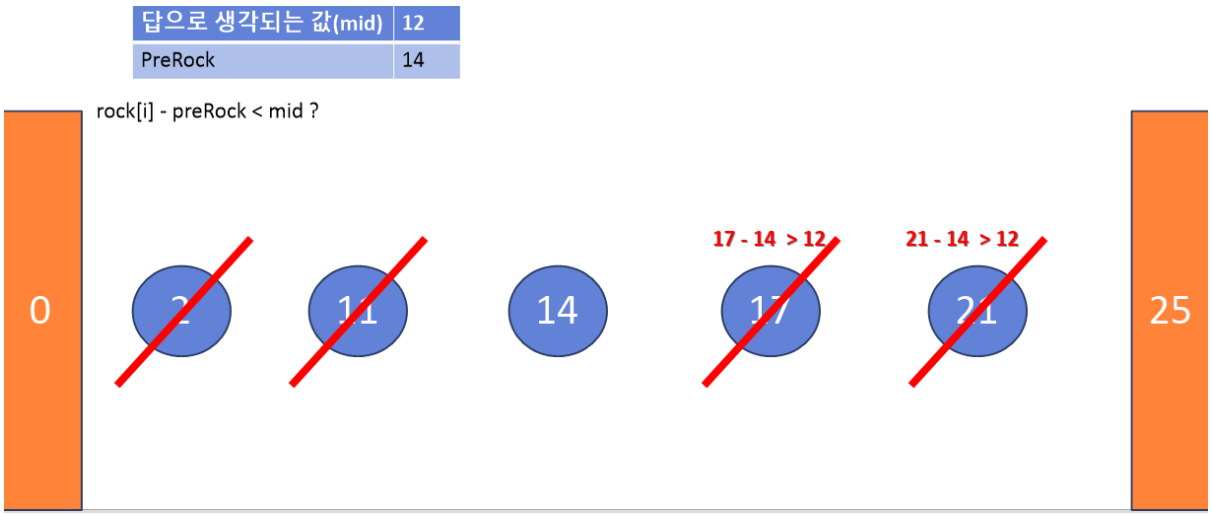
`count = 2`



현재 돌 간 거리가 내 예상보다 커졌으므로 해당 돌은 밟고 지나갈 수 있음

제거 되지 않은 돌부터 다시 위의 과정을 반복해야 하므로

제거 되지 않은 돌을 `preRock`으로 다시 설정함



결과적으로 예상 값을 최소로 갖게 돌을 제거하다보면 총 4개의 돌이 제거가 됨

* 총 지워야 하는 n 보다 더 많은 수의 돌이 지워졌다.

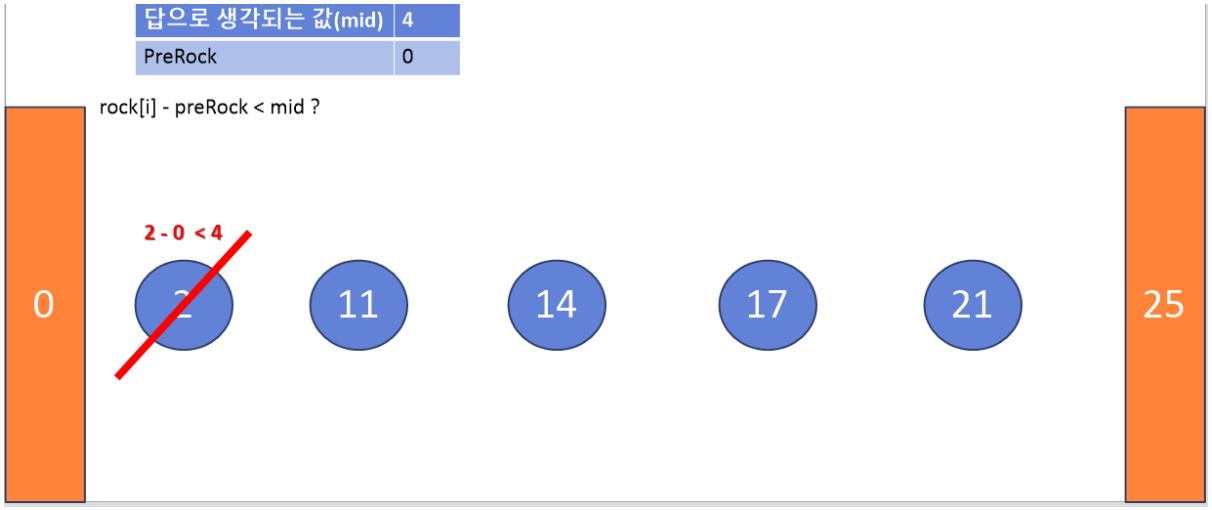
  => 예상 값을 크게 설정했기 때문에 조건을 만족하기 위해 제거된 돌이 많았다.

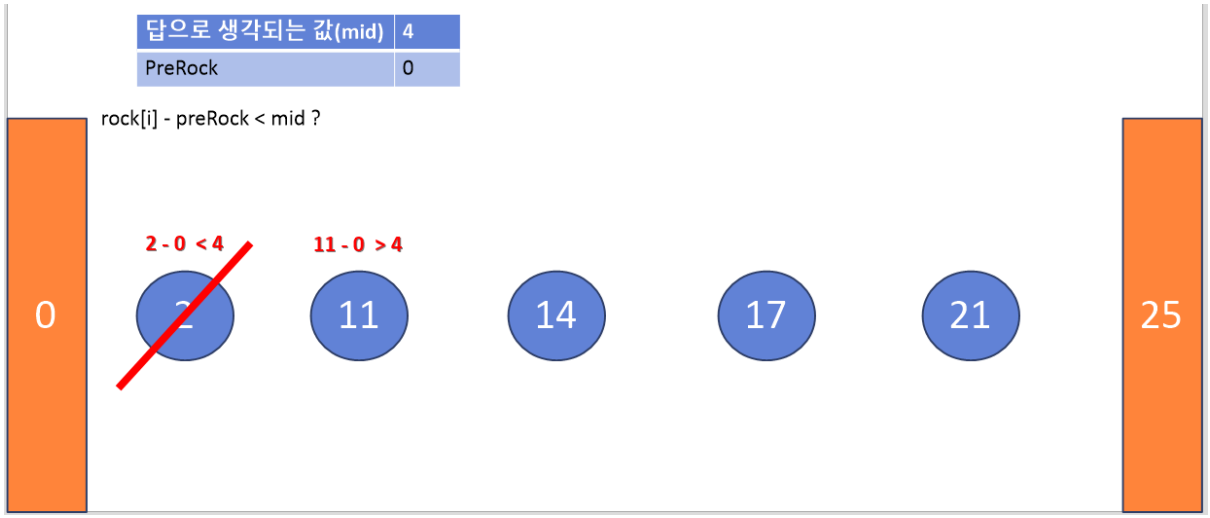
  => 이분 탐색 범위에서 큰 값을 중간값보다 1 작은 값으로 바꾸고 다시 중간값을 구한다.

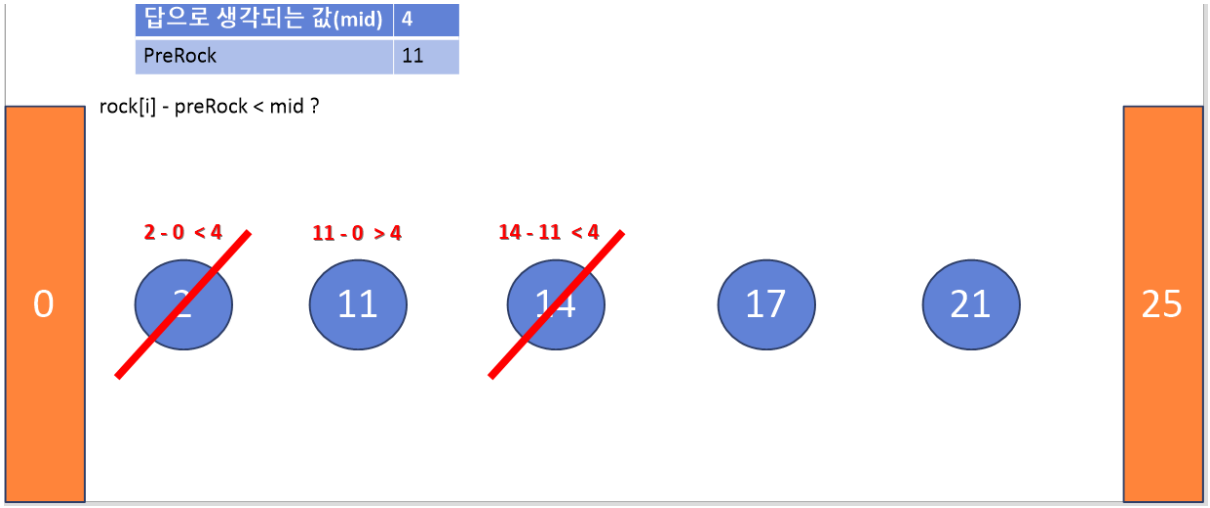
총 지워야 하는 n 보다 더 적은 수의 돌이 지워졌다.

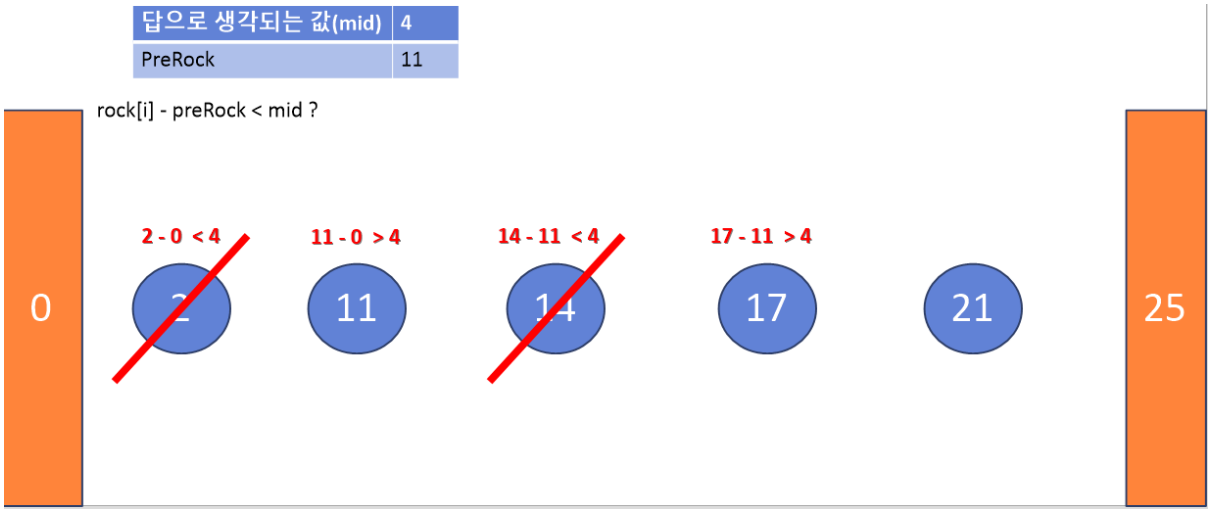
  => 예상 값이 작게 설정했기 때문에 조건을 만족하기 위해 제거된 돌이 적었다.

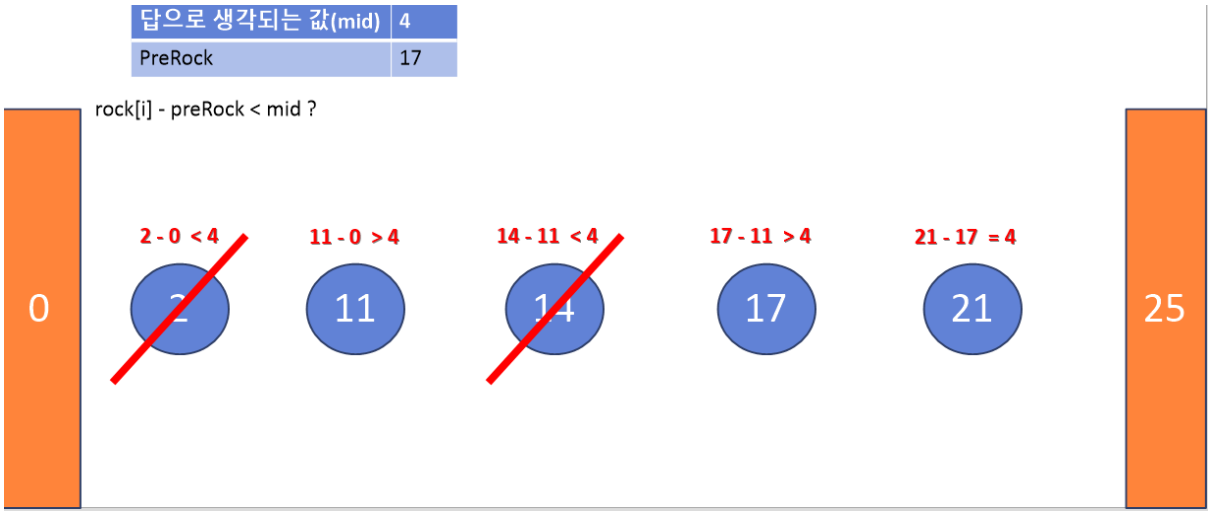
  => 이분 탐색 범위에서 작은 값을 중간값보다 1 큰 값으로 바꾸고 다시 중간값을 구한다.




답으로 생각되는 값(mid) 4
PreRock 0
rock[i] - preRock < mid ?
2 - 0 < 4
11 - 0 > 4
0
2
11
14
17
21
25


답으로 생각되는 값(mid) 4
PreRock 11
rock[i] - preRock < mid ?
2 - 0 < 4
11 - 0 > 4
14 - 11 < 4
0
2
11
14
17
21
25





총 2개만 제거 된 채로 만족 결과 도출

코드

```
import java.util.*;
class Solution {
      public static int solution(int distance, int[] rocks, int n) {
    int answer = 0;

    Arrays.sort(rocks);
```

```
    int left = 0;
    int right = distance;
    int mid = 0;
    while (left <= right) {
      mid = (left + right) / 2;

      int preRock = 0;
      int cnt = 0;

      for (int i = 0; i < rocks.length; i++) {
        if (rocks[i] - preRock < mid) {
          cnt++;
          if (cnt > n) {
            break;
          }
        } else {
          preRock = rocks[i];
        }
      }
      // 너무 많이 제거
      if (cnt > n) {
        right = mid - 1;
      } else {
        if (mid > answer) {
          answer = mid;
        }
        left = mid + 1;
      }
    }
    return answer;
  }
}
```