2021 여름학기 동국대학교 SW역량강화캠프

15일차. 이분탐색 2

● 이분탐색(이진탐색)

▶ 탐색 범위를 절반씩 줄여나가며 진행하는 탐색

▶ 답이 존재하는 범위 l ~ r을 구간의 길이가 1이 될 때까지 줄인다.

▶ 탐색의 시간복잡도 O(lgN)

● 공유기 설치 (4342)

문제

집 N개가 수직선 위에 있다. 각각의 집의 좌표는 $x_1, \cdots, x_N$이고, 집의 좌표는 모두 다르다.

월리는 집에 공유기 C개를 설치하려고 한다. 최대한 많은 곳에서 와이파이를 사용하려고 하기 때문에, 한 집에는 공유기를 하나만 설치할 수 있고, 가장 인접한 두 공유기 사이의 거리를 가능한 크게 하여 설치하려고 한다.

C개의 공유기를 N개의 집에 설치할 때, 가장 인접한 두 공유기 사이의 거리의 최댓값을 구해보자.

입력

첫째 줄에 집의 개수 $N(2 \leq N \leq 200,000)$과 공유기의 개수 $C(2 \leq C \leq N)$이 하나 이상의 빈 칸을 사이에 두고 주어진다.
둘째 줄에 N개의 집의 좌표를 나타내는 $x_i(0 \leq x_i \leq 1,000,000,000)$가 주어진다.

▶ Parametric Search

▶ 공유기 C개를 설치할 때, 가장 인접한 두 공유기 사이의 거리의 최댓값

▶ 최소 거리 x로 공유기들을 설치할 때 공유기 C개를 모두 설치할 수 있는가?

→ x가 정답 ans 이하이면 YES, x가 ans 초과이면 NO

▶ 이분탐색 가능

▶ 그리디

▶ 최소 거리 x로 공유기들을 설치할 때 공유기 C개를 모두 설치할 수 있는가?

▶ 작은 좌표에서부터 보면서 이전 공유기로부터 x이상 떨어졌으면 설치한다.

핵심 코드

```
Collections.sort(arr);
int l=1, r=arr.get(N-1)-arr.get(0);
while(l<r) {
    int mid = (l+r+1)/2;
    if(chk(mid)) l=mid; // mid의 거리를 두고 설치 가능하다면 mid 이상에서 검사. [l,r] -> [mid, r]
    else r=mid-1; // mid의 거리를 두고 설치 불가능하면 더 짧은 거리에서 시도. [l,r] -> [l, mid-1]
}
System.out.println(l);
```

```
public static boolean chk(int x) {
    int now = arr.get(0), cnt = 1;
    for(int i=1; i<N; i++) {
        if(arr.get(i) >= now + x) {
            now = arr.get(i);
            cnt++;
        }
    }
    if(cnt >= C) {
        return true;
    }
    else return false;
}
```

● 휴게소 설치(4341)

문제

윌리는 고속도로를 가지고 있다. 그리고, 윌리는 현재 고속도로에 휴게소를 N개 가지고 있는데, 휴게소의 위치는 고속도로의 시작으로부터 얼만큼 떨어져 있는지로 주어진다. 그런데 윌리는 지금 휴게소를 M개 더 세우려고 한다.

이미 휴게소가 있는 곳에 휴게소를 또 세울 수 없고, 고속도로의 끝에도 휴게소를 세울 수 없으며, 휴게소는 정수 위치에만 세울 수 있다.

윌리는 이 고속도로를 이용할 때, 모든 휴게소를 방문하는데, 휴게소가 없는 구간의 길이의 최댓값을 최소로 하려고 한다. (반드시 M개를 모두 지어야 한다.)

예를 들어, 고속도로의 길이가 1000이고, 현재 휴게소가 {200, 701, 800}에 있고, 휴게소를 1개 더 세우려고 한다고 해보자.

일단, 지금 이 고속도로를 타고 달릴 때, 휴게소가 없는 구간의 최댓값은 200~701구간인 501이다. 하지만, 새로운 휴게소를 451구간에 짓게 되면, 최대가 251이 되어서 최소가 된다.

▶ Parametric Search

▶ 휴게소 M개를 추가로 설치할 때, 휴게소 사이의 거리의 최댓값

▶ 최대 거리 x로 휴게소들을 설치할 때 M개의 휴게소로 충분한가?

→ x가 정답 ans 미만이면 NO, x가 ans 이상이면 YES

▶ 이분탐색 가능

문제 해설

▶ 그리디

▶ 최대 거리 x로 휴게소들을 설치할 때 M개의 휴게소로 충분한가?

▶ 휴게소가 없는 구간의 길이가 tmp라면,
거리 x 이하로 구간을 쪼개기 위해 추가로 필요한 휴게소의 개수 = (tmp-1) / x

모든 구간들에 대해 필요한 휴게소의 개수 합이 M 이하라면 true

핵심 코드

```
int l = 1, r = L;
while (l < r) {
    int mid = (l + r) / 2;
    if (chk(mid))
        r = mid; // mid의 거리를 두고 설치 가능하다면 mid 이하에서 검사. [l,r] -> [l, mid]
    else
        l = mid + 1; // mid의 거리를 두고 설치 불가능하면 더 짧은 거리에서 시도. [l,r] -> [mid+1, r]
}
System.out.println(l);
```

```
public static boolean chk(int x) {
    int now = 0, cnt = 0;
    for (int i = 0; i < N; i++) {
        int tmp = arr.get(i) - now;
        cnt += (tmp - 1) / x;
        now = arr.get(i);
    }
    cnt += (L - now - 1) / x;
    if (cnt <= M) {
        return true;
    } else
        return false;
}
```

● 동전 금고(4703)

문제

윌리는 용돈이 부족해져서 삼촌을 찾아갔다. 윌리의 삼촌은 동전의 산으로 가득 찬 큰 금고를 가지고 있는데, 그 중 몇몇 동전은 매우 특별한 가치가 있는 동전이다.

하지만 최근 삼촌은 금고에 있던 코인을 옮기는 중, 우연히 아끼던 특별한 동전이 금고로 옮겨져 극도의 스트레스를 받고 있다. 다행히 그 코인을 찾았지만, 아쉽게도 금고 입구와 완전히 반대이고 금고 안에 동전이 산더미처럼 쌓여 있기 때문에 실제로 동전에 닿는 것은 쉬운 일이 아니다.

삼촌은 윌리가 동전의 산에서 특별한 동전을 회수해 을 수 있다면, 용돈을 줄 생각이 있다. 윌리는 동전을 회수하기 위한 장비로 사다리를 가져오기로 결정했지만 사다리의 길이를 얼마짜리로 준비해야 할 지 정하지 못했다. 더 긴 사다리는 더 높은 동전 절벽을 오를 수 있다는 것을 의미하지만 더 많은 비용이 든다. 따라서 윌리는 특별한 동전에 도달할 수 있는 가장 짧은 사다리를 사고 싶어한다.

금고는 왼쪽 위 모서리에 입구가 있고 특별한 동전은 오른쪽 아래 모서리에 있는 다양한 높이의 동전 더미의 직사각형 그리드로 표시될 수 있다. 윌리는 현재 서 있는 동전 더미에서 상하좌우로 인접한 동전 더미로 이동할 수 있다. 윌리는 점프하거나 날 수 없기 때문에 성공적으로 높이 x가 높은 동전더미로 이동하려면 길이 x의 사다리가 필요하다. 하지만 내려가는 것은 높이 제한 없이 중력에 모든 것을 맡길 수 있다.

윌리가 사야 할 사다리의 최단 길이를 출력해보자.

▶ Parametric Search

▶ 사야 할 사다리의 최단 길이

▶ 사다리 길이 x로 (1,1)에서 (M,N)까지 이동 가능한가?

→ x가 정답 ans 미만이면 NO, x가 ans 이상이면 YES

▶ 이분탐색 가능

▶ BFS

▶ 사다리 길이 x로 (1,1)에서 (M,N)을 방문할 수 있는지는 BFS로 탐색 가능

핵심 코드

```
int l=0, r=1000000000;
while(l<r) {
    int mid=(l+r)/2;
    if(bfs(mid)) r=mid;
    else l=mid+1;
}
System.out.println(l);
```

핵심 코드

```
public static boolean bfs(int t) {
    Queue<Integer[]> queue = new LinkedList<>();
    visit = new boolean[M+1][N+1];
    queue.offer(new Integer[] {1, 1}); visit[1][1] = true;

    while(!queue.isEmpty()) {
        Integer[] tmp = queue.poll();
        int x = tmp[0], y = tmp[1];
        for(int i=0; i<4; i++) {
            int newx = x + dx[i], newy = y + dy[i];
            if(newx < 1 || newx > M || newy < 1 || newy > N) continue;
            if(arr[newx][newy] <= arr[x][y] + t && visit[newx][newy] == false) {
                visit[newx][newy] = true;
                queue.offer(new Integer[]{newx, newy});
            }
        }
    }
    return visit[M][N];
}
```