05solar

백준

[c++] [BOJ] 백준 32032 / 만보기 대행 서비스

lotus05f 2024. 7. 25. 17:46

백준 32032 만보기 대행 서비스 ucpc 예선 문제 c++

[c++] [BOJ] 백준 32032 / 만보기 대행 서비스

문제 바로가기

문제 전문 보기 >>

닫기

문제

요즘 세상에는 일상생활 속 미션을 수행하면서 포인트를 얻고 소정의 상품도 얻어갈 수 있는 앱테크 서비스가 많다. 만보기 미션은 여러 앱테크 서비스에서 널리 사용하고 있는데, 매일 Dm 걷기에 성공하면 약간의 포인트를 얻을 수 있다.

매일 Dm씩 걷는 게 생각보다 번거로운 일이기 때문에, 미션을 직접 수행하기 귀찮아하는 사람들을 위해 한별이는 만보기 미션을 대행해주는 사업을 하는 스타트업 스타트한별을 창업하였다. 스타트한별에서는 우선 스타트한별 사옥을 지나는 동서쪽으로 길게 이어진 도로 위에 1m 간격으로 보관함을 설치하고 정수 번호를 매겼다. 스타트한별 사옥에서 동쪽으로 Am 떨어진 보관함의 번호는 A, 서쪽으로 Am 떨어진 보관함의 번호는 −A, 사옥에 있는 보관함의 번호는 0이다.

당신은 스타트한별 사옥에서 출발하여 모든 고객의 미션을 수행한 다음 회사로 복귀해야 한다. 당신이 업무를 시작하기 전에 이미 모든 고객이 Xi번 보관함에 휴대폰을 놓아두었다. 당신은 Xi번 보관함까지 가서 휴대폰을 직접 집어야 하고, 집은 이후 Dm 이상 움직인 다음 Xi번 보관함에 휴대폰을 반납해야 한다. 당신은 충분히 큰 가방을 갖고 업무를 수행하기 때문에 여러 휴대폰을 동시에 넣어서 움직일 수 있다. 당신의 이동기록은 근무기록으로 반영되기 때문에 반드시 도로 위에서만 움직여야 한다.

모든 고객의 미션을 수행하고 복귀하기 위해 움직여야 할 최소 거리를 구하는 프로그램을 작성하여라.

입력

첫 줄에 고객의 수 N과 미션을 수행하기 위해 걸어야 할 최소 이동 거리 D가 공백을 사이에 두고 주어진다. (1 ≤ N ≤ 1 000 000; 1 ≤ D ≤ 109)

둘째 줄에 각 고객이 휴대폰을 놓은 보관함의 번호를 나타내는 N개의 정수 Xi가 공백을 사이에 두고 주어진다. (−109 ≤ Xi ≤ 109)

휴대폰의 위치가 서로 겹치거나 스타트한별 사옥과 같은 곳에 있을 수 있다.

모든 입력 값은 정수이다.

출력

첫 줄에 고객 N명의 미션을 모두 수행하고 복귀하기 위해 필요한 최소 이동 거리를 출력한다.

정답이 정수가 아닌 경우, 정답보다 작거나 같은 가장 큰 정수를 출력한다.

문제 설명 :

N명이 고객이 존재하는 서비스에서 휴대폰을 수거 한 채로 Dm를 이동하여야 한다.

휴대폰이 들어있는 각 보관함의 위치는 입력의 두번째 줄의 음수과 양수로 들어오며 사옥의 위치는 0이다.

풀이 과정:

1. 입력을 받는다

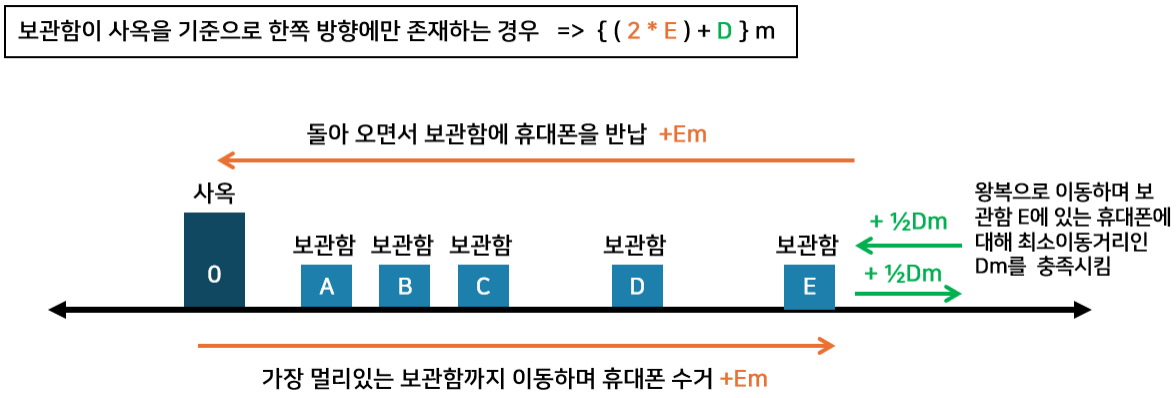
1 - (i) 고객 수와 최대 이동 거리수를 받기 (int)

1 - (ii) 보관함의 위치를 받기 (long long vector로 받았습니다.)

2 최소 이동 거리 계산하기

2 - (i) 보관함이 0 (사옥의 위치) 를 기준으로 한쪽 방향 (양수 혹은 음수) 에만 존재하는 경우

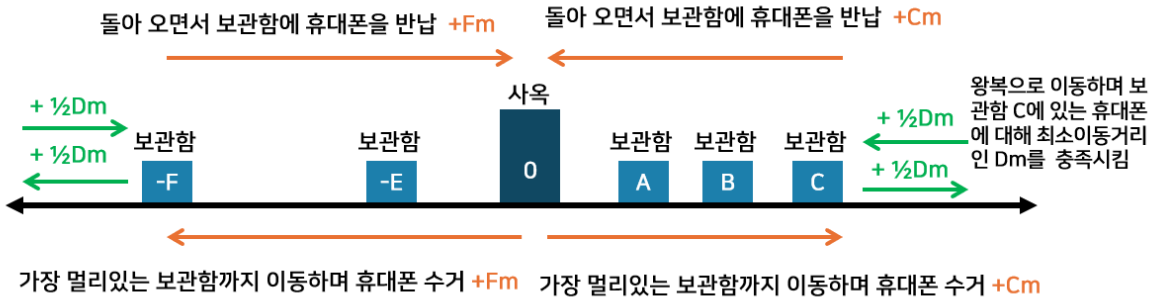
-> 가장 멀리 있는 보관함의 위치 (가장 작은수 혹은 가장 큰 수) 를 왕복하는 거리 + 최소 이동 거리 (D)



2 - (ii) 보관함이 사옥 기준 양쪽에 존재하는 경우 (양수, 음수 둘다 존재)

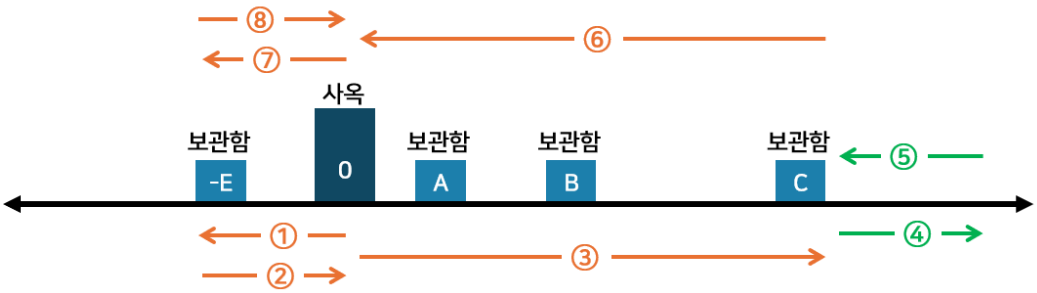
2 - (ii) - ① case1 : 모든 위치를 왕복한 후 최소이동거리를 두번 더하는 경우

- 보관함이 사옥의 양쪽에 존재하는 경우
Case1 : 보관함의 양쪽 끝까지 다녀오고 최소이동거리를 두번 가는 경우 => { (2*C) + (2*F) + (2*D) } m



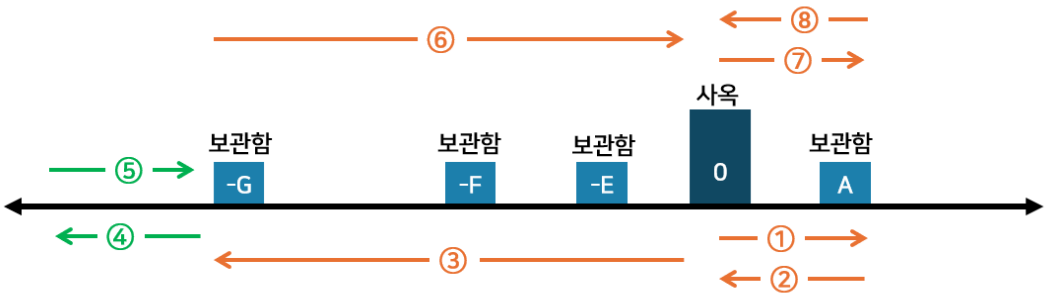
2 - (ii) - ② case2 : 가장 왼쪽 위치를 두번 왕복하고 오른쪽을 왕복하며 최소이동거리를 걷는 경우

- 보관함이 사옥의 양쪽에 존재하는 경우
Case2 : 가장 왼쪽 위치를 두번 왕복하고 오른쪽을 왕복하며 최소이동거리를 걷는 경우 => { (4 * E) + (2 * C) + D } m



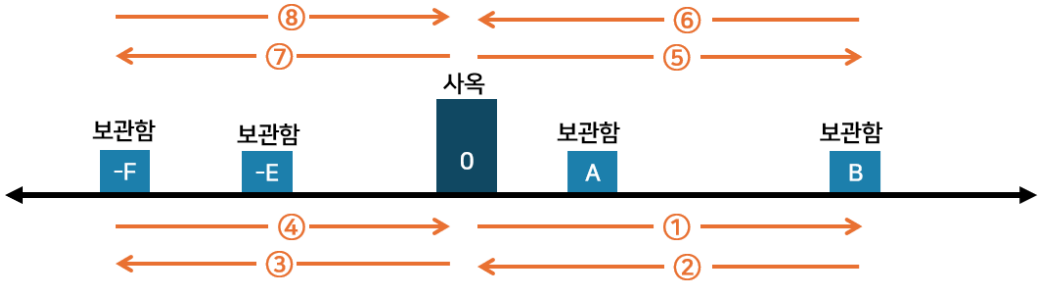
2 - (ii) - ③ case3 : 가장 오른쪽 위치를 두번 왕복하고 왼쪽을 왕복하며 최소이동거리를 걷는 경우

- 보관함이 사옥의 양쪽에 존재하는 경우
Case2 : 가장 오른쪽 위치를 두번 왕복하고 왼쪽을 왕복하며 최소이동거리를 걷는 경우 => { (4 * A) + (2 * G) + D } m



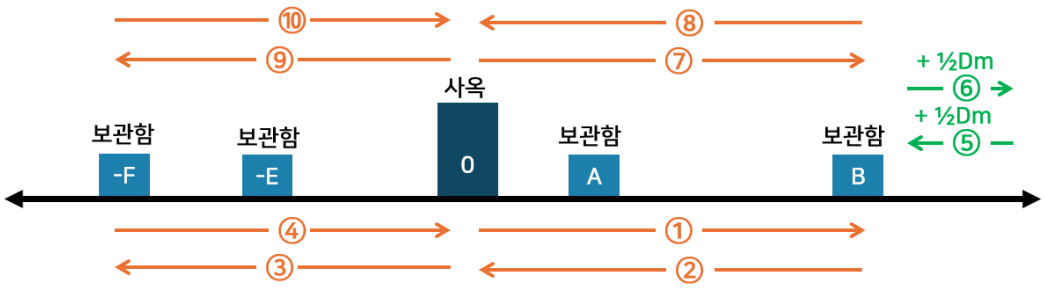
2 - (ii) - ④ case4 - 1 : 가장 먼 두 지점 사이의 거리가 최소 이동거리보다 큰 경우 > > 가장 먼곳까지 두번 왕복

- 보관함이 사옥의 양쪽에 존재하는 경우
Case4 -1 : 가장 양쪽에 위치한 보관함 사이의 거리가 최소 이동 거리 (D)보다 큰 경우
> 가장 먼 곳까지 두번 왕복 => { (4 * F) + (4 * B) } m



2 - (ii) - ⑤ case4 - 2 : 가장 먼 두 지점 사이의 거리가 최소 이동거리보다 작은 경우 >> 가장 먼곳까지 두번 왕복 + 최소 이동거리 한번 이동

- 보관함이 사옥의 양쪽에 존재하는 경우
Case4 -2 : 가장 양쪽에 위치한 보관함 사이의 거리가 최소 이동 거리 (D)보다 작은 경우
> 가장 먼 곳까지 두번 왕복 + 최소 이동거리 한번 이동 => { (4 * F) + (4 * B) + (D) } m



3 . 2번 과정에서 구한 이동 거리 중 최소값을 출력

전체 코드 >>

닫기

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int N, D;
    cin >> N >> D;

    vector<long long> boxes(N);
    for (long long& box : boxes) {
```

```
        cin >> box;
    }
    sort(boxes.begin(), boxes.end());

    long long left_far = boxes.front();
    long long right_far = boxes.back();

    if (left_far * right_far >= 0) {
        long long max_distance = max(abs(left_far), abs(right_far));
        cout << max_distance * 2 + D << endl;
    } else {
        long long case1 = abs(left_far) * 2 + abs(right_far) * 2 + D * 2;
        long long case2 = abs(left_far) * 4 + abs(right_far) * 2 + D;
        long long case3 = abs(left_far) * 2 + abs(right_far) * 4 + D;
        long long distance_between = abs(right_far - left_far);
        long long option4;
        if (distance_between * 2 >= D) {
            option4 = distance_between * 4;
        } else {
            option4 = abs(left_far) * 2 + abs(right_far) * 2 + D;
        }

        cout << min({ case1, case2, case3, option4 }) << endl;
    }

    return 0;
}
```