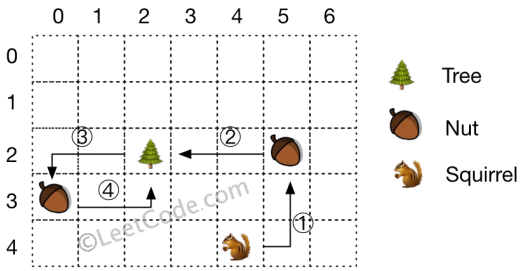
Squirrel Simulation

There's a tree, a squirrel, and several nuts. Positions are represented by the cells in a 2D grid. Your goal is to find the **minimal** distance for the squirrel to collect all the nuts and put them under the tree one by one. The squirrel can only take at most **one nut** at one time and can move in four directions - up, down, left and right, to the adjacent cell. The distance is represented by the number of moves.

Example 1:

```
Input:
Height : 5
Width : 7
Tree position : [2,2]
Squirrel : [4,4]
Nuts : [[3,0], [2,5]]
Output: 12
Explanation:
```



Note:

1. All given positions won't overlap.
2. The squirrel can take at most one nut at one time.
3. The given positions of nuts have no order.
4. Height and width are positive integers. 3
5. The given positions contain at least one nut, only one tree and one squirrel.

Solution 1

Proof: Let the minimum distance from each nut to the tree be `a_1, ..., a_n` and let the minimum distance from each nut to the initial squirrel position be `b_1, ..., b_n`. Note that the minimum distance between two positions in the matrix is determined by their Manhattan distance.

Then, if the squirrel were to start at the tree, then the minimum total distance required to collect all the nuts is `2a_1 + ... + 2a_n`. However, since the squirrel starts elsewhere, we just need to substitute one of the `2a_i` terms with `a_i + b_i`. Or equivalently, we replace one of the `a_i` terms in the sum with `b_i`. To minimize the total sum value at the end, we choose `i` that maximizes `a_i - b_i`.

```
public int minDistance(int height, int width, int[] tree, int[] squirrel, int[][] nu
ts) {
    int sum = 0, maxDiff = Integer.MIN_VALUE;
    for (int[] nut : nuts) {
        int dist = Math.abs(tree[0] - nut[0]) + Math.abs(tree[1] - nut[1]);
        sum += 2*dist;
        maxDiff = Math.max(maxDiff, dist - Math.abs(squirrel[0] - nut[0]) - Math.ab
s(squirrel[1] - nut[1]));
    }
    return sum - maxDiff;
}
```

written by compton_scatter original link here

Solution 2

Time complexity O(n), n = number of nuts.

```
public class Solution {
    public int minDistance(int height, int width, int[] tree, int[] squirrel, int[][
] nuts) {
        int n = nuts.length;
        int[] nutToTree = new int[n];
        int[] nutToSquirrel = new int[n];

        int sum = 0;
        for (int i = 0; i < n; i++) {
            nutToTree[i] = Math.abs(nuts[i][0] - tree[0]) + Math.abs(nuts[i][1] - t
ree[1]);
            sum += nutToTree[i] * 2;
            nutToSquirrel[i] = Math.abs(nuts[i][0] - squirrel[0]) + Math.abs(nuts[i
][1] - squirrel[1]);
        }

        int min = Integer.MAX_VALUE;
        for (int i = 0; i < n; i++) {
            int dist = sum + nutToSquirrel[i] - nutToTree[i];
            min = Math.min(min, dist);
        }

        return min;
    }
}
```

written by shawngao original link here

Solution 3

```
class Solution {
public:
    int minDistance(int height, int width, vector<int>& tree, vector<int>& squirrel,
vector<vector<int>>& nuts) {
        int delta = INT32_MIN;
        int total = 0;
        for (const vector<int>& p : nuts) {
            total += distance(p, tree) * 2;
            delta = max(delta, distance(tree, p) - distance(p, squirrel));
        }
        return total - delta;
    }

    int distance(const vector<int>& p1, const vector<int>& p2) {
        return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1]);
    }
};
```

written by oxFFFFFFFF original link here