Cut Off Trees for Golf Event

You are asked to cut off trees in a forest for a golf event. The forest is represented as a non-negative 2D map, in this map:

1. `0` represents the `obstacle` can't be reached.
2. `1` represents the `ground` can be walked through.
3. `The place with number bigger than 1` represents a `tree` can be walked through, and this positive number represents the tree's height.

You are asked to cut off **all** the trees in this forest in the order of tree's height - always cut off the tree with lowest height first. And after cutting, the original place has the tree will become a grass (value 1).

You will start from the point (0, 0) and you should output the minimum steps **you need to walk** to cut off all the trees. If you can't cut off all the trees, output -1 in that situation.

You are guaranteed that no two `trees` have the same height and there is at least one tree needs to be cut off.

Example 1:

```
Input:
[
 [1,2,3],
 [0,0,4],
 [7,6,5]
]
Output: 6
```

Example 2:

```
Input:
[
 [1,2,3],
 [0,0,0],
 [7,6,5]
]
Output: -1
```

Example 3:

```
Input:
[
 [2,3,4],
 [0,0,5],
 [8,7,6]
]
Output: 6
Explanation: You started from the point (0,0) and you can cut off the tree in (0,0) d
irectly without walking.
```

Hint: size of the given matrix will not exceed 50x50.

Solution 1

I thought it is a straightforward BFS search, so I write it like the following.
Actually, I have the same question with `Number of Island` problem:
https://discuss.leetcode.com/topic/88586/why-python-bfs-get-time-exceed-error

```
import collections
class Solution(object):
    def cutOffTree(self, G):
        """
        :type forest: List[List[int]]
        :rtype: int
        """
        if not G or not G[0]: return -1
        m, n = len(G), len(G[0])
        trees = []
        for i in xrange(m):
            for j in xrange(n):
                if G[i][j] > 1:
                    trees.append((G[i][j], i, j))
        trees = sorted(trees)
        count = 0
        cx, cy = 0, 0
        for h, x, y in trees:
            step = self.BFS(G, cx, cy, x, y)
            if step == -1:
                return -1
            else:
                count += step
                G[x][y] = 1
                cx, cy = x, y
        return count

    def BFS(self, G, cx, cy, tx, ty):
        m, n = len(G), len(G[0])
        visited = [[False for j in xrange(n)] for i in xrange(m)]
        Q = collections.deque()
        step = -1
        Q.append((cx, cy))
        while len(Q) > 0:
            size = len(Q)
            step += 1
            for i in xrange(size):
                x, y = Q.popleft()
                visited[x][y] = True
                if x == tx and y == ty:
                    return step
                for nx, ny in [(x + 1, y), (x - 1, y), (x, y-1), (x, y + 1)]:
                    if nx < 0 or nx >= m or ny < 0 or ny >= n or G[nx][ny] == 0 or
visited[nx][ny]:
                        continue
                    Q.append((nx, ny))
        return -1
```

written by Andrinux original link here

Solution 2

```java
class Solution {
    static int[][] dir = {{0,1}, {0, -1}, {1, 0}, {-1, 0}};

    public int cutOffTree(List<List<Integer>> forest) {
        if (forest == null || forest.size() == 0) return 0;
        int m = forest.size(), n = forest.get(0).size();

        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> a[2] - b[2]);

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (forest.get(i).get(j) > 1) {
                    pq.add(new int[] {i, j, forest.get(i).get(j)});
                }
            }
        }

        int[] start = new int[2];
        int sum = 0;
        while (!pq.isEmpty()) {
            int[] tree = pq.poll();
            int step = minStep(forest, start, tree, m, n);

            if (step < 0) return -1;
            sum += step;

            start[0] = tree[0];
            start[1] = tree[1];
        }

        return sum;
    }

    private int minStep(List<List<Integer>> forest, int[] start, int[] tree, int m,
int n) {
        int step = 0;
        boolean[][] visited = new boolean[m][n];
        Queue<int[]> queue = new LinkedList<>();
        queue.add(start);
        visited[start[0]][start[1]] = true;

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                int[] curr = queue.poll();
                if (curr[0] == tree[0] && curr[1] == tree[1]) return step;

                for (int[] d : dir) {
                    int nr = curr[0] + d[0];
                    int nc = curr[1] + d[1];
                    if (nr < 0 || nr >= m || nc < 0 || nc >= n
                        || forest.get(nr).get(nc) == 0 || visited[nr][nc]) continue
;
                    queue.add(new int[] {nr, nc});
                    visited[nr][nc] = true;
                    visited[nr][nc] = true;
                }
            }
            step++;
        }

        return -1;
    }
}
```

written by shawngao original link here

Solution 3

My idea to solve this problem is by two steps:

1. Save each `{tree height, tree position}` into an array `heights`, and then sort this array based on its `tree height`. (You can also do this by using `TreeMap` or `PriorityQueue`);
2. Accumulate the shortest steps between each two adajacent points in `heights[]`.

Java version:

```
class Solution {
    int[][] dir = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

    public int cutOffTree(List<List<Integer>> forest) {
        if (forest == null || forest.size() == 0) return -1;
        int m = forest.size(), n = forest.get(0).size(), res = 0;
        //first step: sort the tree position based on its height
        List<int[]> heights = new ArrayList<>();
        for(int i = 0; i<m; i++){
            for(int j = 0; j<n; j++){
                if(forest.get(i).get(j) > 1)heights.add( new int[]{forest.get(i).ge
t(j), i, j} );
            }
        }
        Collections.sort(heights, new Comparator<int[]>() {
            public int compare(int[] arr1, int[] arr2) {
                return Integer.compare(arr1[0], arr2[0]);
            }
        });
        //second step: accumulate the shortest steps between each two adajacent poin
ts in heights[].
        int start_x = 0, start_y = 0;
        for(int i = 0; i<heights.size(); i++){
            int cnt_steps = BFS(forest, m, n, start_x, start_y, heights.get(i)[1], h
eights.get(i)[2]);
            if(cnt_steps == -1)return -1;
            res += cnt_steps;
            start_x = heights.get(i)[1];
            start_y = heights.get(i)[2];
        }
        return res;
    }

    public int BFS(List<List<Integer>> forest, int m, int n, int start_x, int start_
y, int des_x, int des_y){
        if(start_x == des_x && start_y == des_y)return 0;
        int steps = 0;
        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{start_x, start_y});
        int[][] visited = new int[m][n];
        visited[start_x][start_y] = 1;
        while(!q.isEmpty()){
            int qsize = q.size();
            steps++;
            while(qsize-- >0 ){
                int[] cur = q.poll();
                int cur_x = cur[0], cur_y = cur[1];
                for(int k = 0; k<4; k++){
                    int x = cur_x + dir[k][0], y = cur_y + dir[k][1];
                    if(x>=0 && x<m && y>=0 && y<n && forest.get(x).get(y) > 0 && vis
ited[x][y] == 0){
                        if(x == des_x && y == des_y)return steps;
                        visited[x][y] = 1;
                        q.add(new int[]{x,y});
                    }
                }
            }
        }
        return -1;
    }
}
```

C++ version:

```cpp
class Solution {
public:
    vector<vector<int>> dir = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    int cutOffTree(vector<vector<int>>& forest) {
        int m = forest.size(), n , res = 0;
        if(m == 0)return -1;
        n = forest[0].size();
        //first step: sort the tree position based on its height
        vector<vector<int>> heights;
        for(int i = 0; i<m; i++){
            for(int j = 0; j<n; j++){
                if(forest[i][j] > 1)heights.push_back({forest[i][j], i, j});
            }
        }
        sort(heights.begin(), heights.end());
        //second step: accumulate the shortest steps between each two adajacent poin
ts in heights[].
        int start_x = 0, start_y = 0;
        for(int i = 0; i<heights.size(); i++){
            int cnt_steps = BFS(forest, m, n, start_x, start_y, heights[i][1], heigh
ts[i][2]);
            if(cnt_steps == -1)return -1;
            res += cnt_steps;
            start_x = heights[i][1], start_y = heights[i][2];
        }
        return res;
    }

    int BFS(vector<vector<int>>& forest, int m, int n, int start_x, int start_y, int
des_x, int des_y){
        if(start_x == des_x && start_y == des_y)return 0;
        int steps = 0;
        queue<pair<int, int>> q;
        q.push({start_x, start_y});
        vector<vector<int>> visited(m, vector<int>(n, 0));
        visited[start_x][start_y] = 1;
        while(!q.empty()){
            int qsize = q.size();
            steps++;
            while(qsize-- >0 ){
                int cur_x = q.front().first, cur_y = q.front().second;
                q.pop();
                for(int k = 0; k<4; k++){
                    int x = cur_x + dir[k][0], y = cur_y + dir[k][1];
                    if(x>=0 && x<m && y>=0 && y<n && forest[x][y] > 0 && visited[x][
y] == 0){
                        if(x == des_x && y == des_y)return steps;
                        visited[x][y] = 1;
                        q.push({x,y});
                    }
                }
            }
        }
        return -1;
    }
};
```

written by Vincent Cai original link here