Andrew's Algorithm Solutions

Thursday, July 9, 2015

LeetCode OJ - Jump Game II

Problem:

Please find the problem here.

Solution:

This is an extension of the previous problem, the ideas are similar, but different.

Last time we see a structure that if a position is reachable, then all the previous positions are also reachable, now we extend the concept to numbers of steps.

The key observation here is. If a position is reachable with minimum k steps, then all previous positions are reachable with k steps or less. The structure would look like this.


```
0, 0, 0, 0, 0, 0, 0, 0, 0
 [Min Steps] 0  1  1  1  1  1  1  2  2  2  2
```

If we can fill this array, then we are done, because we can simply read off the minimum number of steps to reach the end.

Just like the last problem, we would want to keep track of horizon. But there are now multiple horizons, with respect to different number of steps, and the horizons changes in time, so let start by tracing them.

```
Step = 0
Horizon = 0
 [Positions] 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
[Jump Steps] 7, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0
 [Min Steps] 0  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
      [Wall] 0  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
```

We do not need to jump at all if we are already at the end (i.e. the array has length 1) - this is trivial. We put a 'wall' at the position 0, meaning if we move beyond position 0, then we have to jump.

As usual, we keep track of the horizon. After reading the first element, we know we can maximally go to position 7, we update the state as follow.

```
Step = 1
Horizon = 7
 [Positions] 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
[Jump Steps] 7, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0
 [Min Steps] 0  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
      [Wall] 0  ?  ?  ?  ?  ?  ?  1  ?  ?  ?  ?
```

My other blogs

- Andrew's App
- Andrew's Complexity
- Andrew's Euclid Solution
- Andrew's Exercise Solutions
- Andrew's Project Euler Solutions
- Andrew's Spanish Lessons
- Andrew's Technical Thoughts

Labels

- adhoc
- all pairs shortest paths
- Approximation Algorithm
- articulation points
- avl tree
- Bellman Ford's
- bipartite
- breadth first search
- bridges
- brute force
- connected components
- counting sort
- depth first search
- Dijkstra's
- disjoint set union find
- divide and conquer
- dynamic programming
- Edmonds Karp's
- Floyd Warshall
- graph
- greedy
- HackerRank
- Haskell 99
- insertion sort
- Kruskal's
- leetcode
- maximum flow
- maximum independent set

We move beyond a wall, so we have to increase our step. We know if we ever move beyond position 7, we have to increase the step. We are now at position 8, what's next?

```
Step = 1
Horizon = 9
 [Positions] 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
[Jump Steps] 7, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0
 [Min Steps] 0  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
      [Wall] 0  ?  ?  ?  ?  ?  ?  1  ?  2  ?  ?
```

Since we know at position 1 we can jump to position 9, and it currently take just one step to get to position 1, so we know (for now) that if we go beyond position 9, then it will take more than two steps, that's why we put a wall there, as usual, we update the horizon, **but not step**, since we have not crossed any wall.

Next we see 9, this **change** things because we know we can reach position B in step 2, in an other words, the wall at position 9 is obsolete, and we will **move** the wall.

```
Step = 1
Horizon = B
 [Positions] 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
[Jump Steps] 7, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0
 [Min Steps] 0  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
      [Wall] 0  ?  ?  ?  ?  ?  ?  1  ?  ?  ?  2
```

So that's it. We can keep iterating until we reach a wall, increase the step, and move on. The key idea is that moving a wall can be done in constant time if we know where the original wall is - this is done in the code using wall_positions.

As an after thought. I might be able to just keep track of two walls (the current wall - which moving beyond it will increase step, and the next wall (which is basically the current horizon).

But since the judge like my code, and I am lazy, maybe next time (or never ... which is the usual case :p)

Code:

```
#include "stdafx.h"

// https://leetcode.com/problems/jump-game-ii/

#include "LEET_JUMP_GAME_II.h"
#include <map>
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

namespace _LEET_JUMP_GAME_II
{
    class Solution {
    public:
        int jump(vector<int>& nums)
        {
            if (nums.size() == 1)
            {
                return 0;
            }

            unsigned int horizon = 0;
```


- memoization
- minimum spanning tree
- Prim's
- quadtree
- segment tree
- single source shortest path
- strongly connected components
- Suffix Tree
- topological sort

About Me

Andrew Au

View my complete profile

Blog Archive

► 2016 (119)

▼ 2015 (218)

► December (5)

► November (8)

► October (5)

► September (34)

► August (29)

▼ July (36)

LeetCode OJ - Search a 2D Matrix II

LeetCode OJ - Rectangle Area

LeetCode OJ - Remove Nth Node From End of List

LeetCode OJ - Summary Ranges

LeetCode OJ - Remove Duplicates from Sorted List

LeetCode OJ - Add and Search Word - Data structure...

LeetCode OJ - Binary Search Tree Iterator

LeetCode OJ - Kth Smallest Element in a BST

LeetCode OJ - Power of Two

LeetCode OJ - Min Stack

LeetCode OJ - Sliding Window Maximum

LeetCode OJ - Number of Digit One

LeetCode OJ - Palindrome Linked List

LeetCode OJ - Product of Array Except Self

LeetCode OJ - Delete Node in a Linked List

LeetCode OJ - 3Sum

LeetCode OJ - Longest Common Prefix

LeetCode OJ - Roman to Integer

LeetCode OJ - Integer to Roman

```
            unsigned int steps = 0;
            vector<unsigned int> walls;
            vector<unsigned int> walls_positions;
            walls.resize(nums.size());
            walls_positions.resize(nums.size());
            for (unsigned int i = 0; i < nums.size(); i++)
            {
                walls[i] = -1;
                walls_positions[i] = -1; // int.max is a good value for not exist
            }
            walls[0] = 0;
            walls_positions[0] = 0;

            for (unsigned int i = 0; i < nums.size(); i++)
            {
                if (i != 0)
                {
                    if (walls[i - 1] != -1)
                    {
                        steps = walls[i - 1] + 1;
                    }
                }
                if (i > horizon)
                {
                    return -1;
                }
                if (i + nums[i] > horizon)
                {
                    unsigned int proposed_wall_position = i + nums[i];
                    if (proposed_wall_position >=  walls.size())
                    {
                        proposed_wall_position = walls.size() - 1;
                    }
                    if (walls_positions[steps + 1] != -1)
                    {
                        walls[walls_positions[steps + 1]] = -1;
                    }
                    walls[proposed_wall_position] = steps + 1;
                    walls_positions[steps + 1] = proposed_wall_position;
                    horizon = proposed_wall_position;
                }
            }

            return steps;
        }
    };
}

using namespace _LEET_JUMP_GAME_II;

int LEET_JUMP_GAME_II()
{
    Solution solution;
    int case1Array[] = { 7, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    int case2Array[] = {9,7,9,4,8,1,6,1,5,6,2,1,7,9,0};
    vector<int> case1 = vector<int>(case1Array, case1Array + _countof(case1Array));
    vector<int> case2 = vector<int>(case2Array, case2Array + _countof(case2Array));
    cout << solution.jump(case1) << endl;
```


LeetCode OJ - Lowest Common Ancestor of a Binary T...

LeetCode OJ - Lowest Common Ancestor of a Binary S...

LeetCode OJ - Container With Most Water

LeetCode OJ - Remove Duplicates from Sorted Array

LeetCode OJ - Jump Game II

LeetCode OJ - Jump Game

LeetCode OJ - Merge Sorted Array

LeetCode OJ - Construct Binary Tree from Inorder a...

LeetCode OJ - Construct Binary Tree from Preorder ...

LeetCode OJ - Regular Expression Matching

LeetCode OJ - Implement Queue using Stacks

LeetCode OJ - Palindrome Number

LeetCode OJ - Binary Tree Right Side View

LeetCode OJ - String to Integer (atoi)

LeetCode OJ - Reverse Integer

LeetCode OJ - ZigZag Conversion

LeetCode OJ - Longest Palindromic Substring

► June (55)

► April (14)

► March (22)

► January (10)

► 2014 (116)

```
    cout << solution.jump(case2) << endl;
    return 0;
}
```

Posted by Andrew Au at 8:22 AM Recommend this on Google

Labels: leetcode

No comments :

Post a Comment

Enter your comment...

Comment as: Select profile...

Publish Preview

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)