Algorithm Design and Analysis

Homework 1

Haochen Huang
5140309485

October 8, 2016

1

1.1 Question

Given an array of non-negative integers, you are initially positioned at the first index of the array.
Each element in the array represents your maximum jump length at that position.
Determine if you are able to reach the last index.
For example:
A = [2,3,1,1,4], return true.
A = [3,2,1,0,4], return false.
Input:
int A[]: the input array.
int N: length of A.
Output:
return true or false.

1.2 Answer

At every position, record the farthest position it can reach in a jump. Check every position by index and once the record includes the last index, return true.
Time complexity O(n)

1.3 Code

C++ :

```
bool Jump(int A[],int N) {
    int range = 0;
    for (int i=0; i < N && i <= range; ++i) {
        range = max(range, i + A[i]);
        if (range>=N) return true;
    }
    return false;
}
```

2

2.1 Question

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

For example:

Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

Note:

You can assume that you can always reach the last index.

Input:

int A[]: the input array.

int N: length of A.

Output:

return minimum number of jumps.

2.2 Answer

Use the greedy algorithm. Record three parameters.

num is the number of jumps.

curReach means the maximum index after **num** jumps.

curMax means the maximum index one can reach from index 0 to i.

When i is larger than **curReach**, we do not care the actual route of the jump, but we are sure that we need one more jump to cover the current index i.

If the **curReach** reaches the index N-1, the **num** is already the minimum number of jumps.

Time complexity O(n)

2.3 Code

C++ :

```
int Jump2(int A[], int N) {
        int num=0,curReach=0,curMax=0;
        for (int i = 0; i < N; ++i) {
            if (i > curReach) {
                curReach = curMax;
                ++num;
                if (curReach >= N-1) break;
            }
            curMax = max(curMax, i+A[i]);
        }
        return num;
}
```

3

3.1 Question

There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

a. Each child must have at least one candy.

b. Children with a higher rating get more candies than their neighbors.(assume no equal rating neighbors)

What is the minimum candies you must give?

Input:

int A[]: the input array of rating values.

int N: length of A,(number of children).

Output:

return minimum number of candies you must give.

3.2 Answer

This problem can be done with the greedy method. **num[n]** is the number of candy for each child. If one child's rate is greater than his left neighbors rate, then the best choice is giving this child **num[i]=num [i -1]+1** candies. If not, we just give the child one candy temporarily.

Then we need to satisfy the requirement that if the child's rate is greater than his right neighbor's rate, he should get more candies. We only need to reverse the array and use the greedy method again. We also need to make sure that the num[i] should be greater than the initial number of candies in the first operation. So **num[i] = max(num [i] , num [i+1]+1);**This operation will not influence the situation that children with a higher rating get more candies than their left neighbors.

Time complexity O(n)

3.3 Code

C++ :

```
int Pro3(int A[], int N) {
    int * num = new int[N];
    int ans =N;
    for(int i=1;i<N;++i){
        if(A[i] > A[i-1]){
            num[i] = num[i-1]+1;
        }
        else
            num[i] = 1;
    }
    for(int i = N-2;i>=0;i--){
        if(A[i] > A[i+1]){
                num[i] = max(num[i], num[i+1]+1);
```

```
14          }
15      }
16      for(int i=0;i<N;i++){
17          ans += num[i];
18      }
19      return ans;
20  }
```