Poisonous Plants

locked

by vatsalchanana

Editorial by bertho_coder

We will discuss 3 different appproaches to solve this problem.

Let's first construct an $O(N^2)$ solution

Suppose that the number of days after which no plants die is equal to x. Let's simulate the process of dying for x days. On the i-th day, we can loop through the plants which are not dead yet and check whether it should die today or not. In the worst case, x can be equal to $N-1$ where N is the number of plants. So the complexity of this solution is $O(N^2)$ which isn't fast enough to pass under the given time limit.

An O(NlogN) solution

Suppose we're simulating the process for the i-th day where $i > 1$. We don't need to loop through all the plants to see whether it should die today or not. We only need to check those plants whose neighboring plants got updated due to the death of some plant on the $(i-1)$-th day.

We maintain 3 sets:

- $alive$ - which stores the indices of the plants which are still alive
- $goingToDie$ - which stores the indices of the plants which are going to die on the current day
- $nextToDie$ - which stores the indices of plants which are going to die on the next day

Initially, all the plants are in the set $alive$. We loop through all the plants and check whether it should die on the first day or not. If it should, then we insert it to the set $goingToDie$. Then we do the following while this set is not empty:

1. We first remove the indices which are in the set $goingToDie$ from the set $alive$.
2. Then for every index in the set $goingToDie$, suppose it is p, we binary search on the set $alive$ to find out the smallest index greater than p and the largest index smaller than p. The plants of these two indices are going to be neighbors on the next day. We check their heights and decide whether the plant on the right side should die on the next day or not. If it should, then we insert it to the set $nextToDie$.
3. We do this assignment: $goingToDie = nextToDie$, increment the number of days and go to step 1.

Time Complexity: As we insert any index into any of the sets for at most once, the time complexity of this solution is $O(NlogN)$.

Solution

C++

```
#include<bits/stdc++.h>
using namespace std;
#define MAX     100000

set<int> alive, goingToDie, nextToDie;
int height[MAX+5];
```

Statistics

Difficulty: Hard

Time Complexity:	$O(n)$ Required

Knowledge: Stack, Set, Binary Search

Publish Date: Aug 02 2015

```
const int inf = 1000000000;

int main()
{
    int i, j, k, v, n, day = 0;

    scanf("%d", &n);
    for(i = 1; i <= n; i++) scanf("%d", &height[i]);

    height[0] = inf;
    for(i = 1; i <= n; i++)
    {
        alive.insert(i);
        if(height[i] > height[i-1]) goingToDie.insert(i);
    }

    while(goingToDie.size())
    {
        day++;

        for(auto x : goingToDie)
            alive.erase(x);

        nextToDie.clear();
        for(auto x : goingToDie)
        {
            auto itr = alive.lower_bound(x);
            if(itr == alive.begin() || itr == alive.end()) continue;

            auto previous = itr;
            previous--;
            if(height[*itr] > height[*previous]) nextToDie.insert(*itr);
        }

        goingToDie = nextToDie;
    }

    printf("%d\n", day);
    return 0;
}
```

Let's solve it in O(n)

Each plant must contain less or equal amount of pesticide than the plant to it's left in order to survive. Maintain a stack where every element on the stack contains the following information about a plant:

- the amount of pesticide contained in the plant
- the number of days that have passed.

Initially, push $\{P[0], 0\}$ onto the stack, where $P[i]$ is the amount of pesticide contained in the i-th plant.

Then iterate over the plants from left to right. Suppose that the i-th plant is now being considered. If the stack isn't empty, pop the top element from the stack and compare it with $P[i]$.

- If $P[i]$ is greater than this value, then the i-th plant will die on the first day. So add $\{P[i], 1\}$ to the stack.
- Otherwise, pop as long as the value in the top of the stack is greater than or equal to $P[i]$. Update the answer with the second integer of the pair pushed onto the stack. If the stack becomes empty then add $\{P[i], 0\}$ to the stack otherwise add $P[i], cur\_date + 1$.

Solution

C++

```
#include <vector>
#include <list>
#include <map>
#include <set>
#include <queue>
#include <deque>
#include <stack>
#include <bitset>
#include <algorithm>
#include <functional>
```

```
#include <numeric>
#include <limits>
#include <tuple>
#include <utility>
#include <sstream>
#include <iostream>
#include <iomanip>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <cassert>
using namespace std;


#define MAX 100005

typedef struct ele
{
    int v1,v2;
}ele;



int  main()
{
    int n;
    cin>>n;
    assert(n>=1&&n<=100000);
    vector<int>a(n);


    for(int i=0;i<n;i++)
    {
        cin>>a[i];
        assert(a[i]>=0 && a[i]<=1000000000);
        a[i]=-a[i];
    }


    stack<ele>s;
    int maxa=0;

    for(int  i=0;i<n;i++)
    {
        if(s.empty())
        {
            s.push({a[i],0});
        }
        else
        {
            ele temp=s.top();

            if(a[i]<temp.v1)
            {
                int sc=1;
                maxa=max(maxa,sc);
                s.push({a[i],sc});
            }
            else
            {

                ele v=s.top();
                int pr=v.v2;
                while(!s.empty()&&v.v1<=a[i])
                {
                    s.pop();
                    if(s.empty())
                        break;
                    pr=max(pr,v.v2);
                    v=s.top();
                }

                if(s.empty())
                {
                    s.push({a[i],0});
                }

                else
                {
                    s.push({a[i],pr+1});
                    maxa=max(maxa,pr+1);
                }
            }
        }
    }
    cout<<maxa<<endl;
}
```