Candies

Editorial by kevinsogo

This problem can be solved with a greedy technique.

We denote by C_i the number of candies given to the i^{th} child.

Let's classify the children into four kinds, depending on the comparisons of their ratings with their neighbors.

- If $\text{rating}_{i-1} \geq \text{rating}_i \leq \text{rating}_{i+1}$, we say Child i is a **valley**.
- If $\text{rating}_{i-1} < \text{rating}_i \leq \text{rating}_{i+1}$, we say Child i is a **rise**.
- If $\text{rating}_{i-1} \geq \text{rating}_i > \text{rating}_{i+1}$, we say Child i is a **fall**.
- If $\text{rating}_{i-1} < \text{rating}_i > \text{rating}_{i+1}$, we say Child i is a **peak**.

For the leftmost and rightmost child, assume they each have a neighbor with an infinite rating, so they can also be classified according to this scheme. (In particular, the leftmost child cannot be a rise or a peak, and the rightmost child cannot be a fall or a peak.)

Given this classification, we can intuitively distribute the candies this way:

- For each valley child, give him/her 1 candy.
- For each rise child, give him/her $C_{i-1} + 1$ candies.
- For each fall child, give him/her $C_{i+1} + 1$ candies.
- For each peak child, give him/her $\max(C_{i-1}, C_{i+1}) + 1$ candies.

Observe that this gives a valid distribution of candies. But does it give the minimum total number of candies?

Amazingly, it does! But only as long as you distribute the candies in the right order. Here's one good order:

- Distribute the candies to the valleys.
- Distribute the candies to the rises from left to right.
- Distribute the candies to the falls from right to left.
- Distribute the candies to the peaks.

Note that the order in which we distribute candies to rises and falls is pretty important!

Here's a Python implementation:

```
INF = 10**9 # a number larger than all ratings
n = input()
a = [input() for i in xrange(n)]

# add sentinels
a = [INF] + a + [INF]

candies = [0]*(n+1)
# populate 'valleys'
for i in xrange(1,n+1):
    if a[i-1] >= a[i] <= a[i+1]:
        candies[i] = 1

# populate 'rises'
for i in xrange(1,n+1):
    if a[i-1] < a[i] <= a[i+1]:
        candies[i] = candies[i-1] + 1

# populate 'falls'
for i in xrange(n,0,-1):
    if a[i-1] >= a[i] > a[i+1]:
        candies[i] = candies[i+1] + 1

# populate 'peaks'
for i in xrange(1,n+1):
    if a[i-1] < a[i] > a[i+1]:
        candies[i] = max(candies[i-1], candies[i+1]) + 1

# print the total number of candies
print sum(candies)
```

Of course, we need to prove why this assignment works. We will prove two things:

- It gives a valid distribution of candies.
- It gives a the distribution with the minimum sum.

Is it valid?

The first thing to notice is that: All children are assigned a positive number of candies. We just have to show that for every two neighbors with different ratings, the one with the higher rating is given more candies.

Consider two neighboring children i and $i+1$ with different ratings. There are two cases:

Case 1: $\textbf{rating}_i < \textbf{rating}_{i+1}$.

In this case, the only possible cases are:

- Child i is a valley and Child $i+1$ is a peak.
- Child i is a valley and Child $i+1$ is a rise.
- Child i is a rise and Child $i+1$ is a peak.
- Child i is a rise and Child $i+1$ is a rise.

In the first three cases, we can see that Child i is given candies earlier than Child $i+1$ (due to the order we're giving out candies), and indeed, we see that $C_i < C_{i+1}$. But actually, the same holds as well in the last case, since we're giving candies to rises from left to right.

Case 2: $\textbf{rating}_i > \textbf{rating}_{i+1}$.

This is similar. In this case, the only possible cases are:

- Child i is a peak and Child $i+1$ is a valley.
- Child i is a fall and Child $i+1$ is a valley.
- Child i is a peak and Child $i+1$ is a fall.
- Child i is a fall and Child $i+1$ is a fall.

In the first three cases, we can see that Child $i+1$ is given candies earlier than Child i, and indeed, we see that $C_i > C_{i+1}$. But actually, the same holds as well in the last case, since we're giving candies to falls from right to left.

This shows that the candy distribution is valid for every pair of neighbors, hence the overall distribution is valid as well!

Is it the minimum?

Proving that this gives the minimum total candies is a little easier. We just have to notice that these inequalities hold:

- For each valley child i, $C_i \geq 1$.
- For each rise child i, $C_i \geq C_{i-1} + 1$.
- For each fall child i, $C_i \geq C_{i+1} + 1$.
- For each peak child i, $C_i \geq \textbf{max}(C_{i-1}, C_{i+1}) + 1$.

This easily follows from the constraints of the problem. Since the algorithm above satisfies each inequality in the tightest possible way, this means the overall number of candies is minimized.

The time complexity is $O(N)$, which is optimal.

Note that there are many other solutions as well, some requiring more advanced techniques, and some requiring even fewer passes through the array. This editorial just describes one.

OK Set by Omar Khaled

Problem Setter's code:

```
#include <bits/stdc++.h>

using namespace std;

int rate [100000 + 10];
long long values [100000 + 10];   // use long long

int main()
{
    int n;
    long long ans = 0, tmp = 0;

    cin >> n;
```

```
    for(int i = 0; i < n; i++) cin >> rate[i];

    values[0] = 1;
                                        // case 1
    for(int i = 1; i < n; i++){         // scan input array from left to right

        if(rate[i] > rate[i - 1]){

            values[i] = values[i - 1] + 1;  // case 1, a

        } else values[i] = 1;               // case 1,b
    }

    ans = values[n - 1];                // case 2

    for(int i = n - 2; i >= 0; i--){    // scan input array from right to left

        if(rate[i] > rate[i + 1]){

            tmp = values[i + 1] + 1;        // case 2, a

        } else tmp = 1;                     // case 2, b

        ans += max(tmp, values[i]);   // maximum of the two values for child (i)
        values[i] = tmp;
    }

    cout << ans;

    return 0;
}
```

Feedback

Was this editorial helpful?

Yes | No