Candy

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:

- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

Solution 1

```
 int candy(vector<int> &ratings)
{
    int size=ratings.size();
    if(size<=1)
        return size;
    vector<int> num(size,1);
    for (int i = 1; i < size; i++)
    {
        if(ratings[i]>ratings[i-1])
            num[i]=num[i-1]+1;
    }
    for (int i= size-1; i>0 ; i--)
    {
        if(ratings[i-1]>ratings[i])
            num[i-1]=max(num[i]+1,num[i-1]);
    }
    int result=0;
    for (int i = 0; i < size; i++)
    {
        result+=num[i];
       // cout<<num[i]<<" ";
    }
    return result;
}
```

written by 1145520074 original link here

Solution 2

Hi guys!

This solution picks each element from the input array only once. First, we give a candy to the first child. Then for each child we have three cases:

1. His/her rating is equal to the previous one -> give 1 candy.
2. His/her rating is greater than the previous one -> give him (previous + 1) candies.
3. His/her rating is less than the previous one -> don't know what to do yet, let's just count the number of such consequent cases.

When we enter 1 or 2 condition we can check our count from 3. If it's not zero then we know that we were descending before and we have everything to update our total candies amount: number of children in descending sequence of raitings - coundDown, number of candies given at peak - prev (we don't update prev when descending). Total number of candies for "descending" children can be found through arithmetic progression formula (1+2+...+countDown). Plus we need to update our peak child if his number of candies is less then or equal to countDown.

Here's a pretty concise code below.

```
public class Solution {
    public int candy(int[] ratings) {
        if (ratings == null || ratings.length == 0) return 0;
        int total = 1, prev = 1, countDown = 0;
        for (int i = 1; i < ratings.length; i++) {
            if (ratings[i] >= ratings[i-1]) {
                if (countDown > 0) {
                    total += countDown*(countDown+1)/2; // arithmetic progression
                    if (countDown >= prev) total += countDown - prev + 1;
                    countDown = 0;
                    prev = 1;
                }
                prev = ratings[i] == ratings[i-1] ? 1 : prev+1;
                total += prev;
            } else countDown++;
        }
        if (countDown > 0) { // if we were descending at the end
            total += countDown*(countDown+1)/2;
            if (countDown >= prev) total += countDown - prev + 1;
        }
        return total;
    }
}
```

Have a nice coding!

written by shpolsky original link here

Solution 3

```
public int candy(int[] ratings) {
    int candies[] = new int[ratings.length];
    Arrays.fill(candies, 1);// Give each child 1 candy

    for (int i = 1; i < candies.length; i++){// Scan from left to right, to make
sure right higher rated child gets 1 more candy than left lower rated child
        if (ratings[i] > ratings[i - 1]) candies[i] = (candies[i - 1] + 1);
    }

    for (int i = candies.length - 2; i >= 0; i--) {// Scan from right to left, to
make sure left higher rated child gets 1 more candy than right lower rated child
        if (ratings[i] > ratings[i + 1]) candies[i] = Math.max(candies[i], (candi
es[i + 1] + 1));
    }

    int sum = 0;
    for (int candy : candies)
        sum += candy;
    return sum;
}
```

written by Pixel_ original link here