Missing Ranges

Given a sorted integer array where the range of elements are [*lower*, *upper*] inclusive, return its missing ranges.

For example, given `[0, 1, 3, 50, 75]`, *lower* = 0 and *upper* = 99, return `["2", "4->49", "51->74", "76->99"]`.

Solution 1

```
public class Solution {
    public List<String> findMissingRanges(int[] A, int lower, int upper) {
        List<String> result = new ArrayList<String>();
        int pre = lower - 1;
        for(int i = 0 ; i <= A.length  ; i++){
            int after = i == A.length ? upper + 1 : A[i];
            if(pre + 2 == after){
                result.add(String.valueOf(pre + 1));
            }else if(pre + 2 < after){
                result.add(String.valueOf(pre + 1) + "->" + String.valueOf(after
- 1));
            }
            pre = after;
        }
        return result;
    }
}
```

written by jccg1000021953 original link here

Solution 2

```
public List<String> findMissingRanges(int[] a, int lo, int hi) {
  List<String> res = new ArrayList<String>();

  // the next number we need to find
  int next = lo;

  for (int i = 0; i < a.length; i++) {
    // not within the range yet
    if (a[i] < next) continue;

    // continue to find the next one
    if (a[i] == next) {
      next++;
      continue;
    }

    // get the missing range string format
    res.add(getRange(next, a[i] - 1));

    // now we need to find the next number
    next = a[i] + 1;
  }

  // do a final check
  if (next <= hi) res.add(getRange(next, hi));

  return res;
}

String getRange(int n1, int n2) {
  return (n1 == n2) ? String.valueOf(n1) : String.format("%d->%d", n1, n2);
}
```

written by jeantimex original link here

Solution 3

I noticed that OJ currently does not have test cases which involves extreme integer values, i.e. INT*MIN/INT*MAX. For instance, the following code:

```
vector<string> res;
char buf[50];
void addMissingRange(int left, int right, bool inc_left = false, bool inc_right =
false)
{
    if (right < left) return; // The range does not exist
    else if (right == left) sprintf(buf, "%d", left); // The range has only one e
lement
    else sprintf(buf, "%d->%d", left, right); // A two element range

    res.push_back(buf);
}

vector<string> findMissingRanges(int A[], int n, int lower, int upper) {
    int last = lower-1;
    for (int i = 0; i < n; ++i)
    {
        addMissingRange(last+1, A[i]-1);
        last = A[i];
    }
    addMissingRange(last+1, upper); // Add the last range.
    return res;
}
```

would pass OJ, but as a matter of fact, it fails on inputs like this:

```
A = [INT_MAX]; lower = 0, upper = INT_MAX;
```

The expected output should be: ["0->2147483646"],

but the actual output produced by the code above is: ["0->2147483646", "-2147483648->2147483647"]

It is because 'last+1' in the second last row overflows to INT_MIN, thus creating a giant range between 'last' and 'upper'.

So my questions are:

1. Do you guys think that we should add those corner cases to OJ?
2. If I would like to make sure my code works for ALL possible inputs, is there any elegant trick that I can use to avoid the overflow problem?

Thanks.

written by stellari original link here