Next Permutation

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

`1,2,3` → `1,3,2`
`3,2,1` → `1,2,3`
`1,1,5` → `1,5,1`

Solution 1

My idea is for an array:

1. Start from its last element, traverse backward to find the first one with index i that satisfy num[i-1] < num[i]. So, elements from num[i] to num[n-1] is reversely sorted.
2. To find the next permutation, we have to swap some numbers at different positions, to minimize the increased amount, we have to make the highest changed position as high as possible. Notice that index larger than or equal to i is not possible as num[i,n-1] is reversely sorted. So, we want to increase the number at index i-1, clearly, swap it with the smallest number between num[i,n-1] that is larger than num[i-1]. For example, original number is 121543321, we want to swap the '1' at position 2 with '2' at position 7.
3. The last step is to make the remaining higher position part as small as possible, we just have to reversely sort the num[i,n-1]

The following is my code:

```
public void nextPermutation(int[] num) {
    int n=num.length;
    if(n<2)
        return;
    int index=n-1;
    while(index>0){
        if(num[index-1]<num[index])
            break;
        index--;
    }
    if(index==0){
        reverseSort(num,0,n-1);
        return;
    }
    else{
        int val=num[index-1];
        int j=n-1;
        while(j>=index){
            if(num[j]>val)
                break;
            j--;
        }
        swap(num,j,index-1);
        reverseSort(num,index,n-1);
        return;
    }
}

public void swap(int[] num, int i, int j){
    int temp=0;
    temp=num[i];
    num[i]=num[j];
    num[j]=temp;
}

public void reverseSort(int[] num, int start, int end){
    if(start>end)
        return;
    for(int i=start;i<=(end+start)/2;i++)
        swap(num,i,start+end-i);
}
```

written by yuyibestman original link here

Solution 2

Well, in fact the problem of next permutation has been studied long ago. From the Wikipedia page, in the 14th century, a man named Narayana Pandita gives the following classic and yet quite simple algorithm (with minor modifications in notations to fit the problem statement):

1. Find the largest index `k` such that `nums[k] < nums[k + 1]`. If no such index exists, the permutation is sorted in descending order, just reverse it to ascending order and we are done. For example, the next permutation of `[3, 2, 1]` is `[1, 2, 3]`.
2. Find the largest index `l` greater than `k` such that `nums[k] < nums[l]`.
3. Swap the value of `nums[k]` with that of `nums[l]`.
4. Reverse the sequence from `nums[k + 1]` up to and including the final element `nums[nums.size() - 1]`.

Quite simple, yeah? Now comes the following code, which is barely a translation.

Well, a final note here, the above algorithm is indeed powerful ---**it can handle the cases of duplicates**! If you have tried the problems Permutations and Permutations II, then the following function is also useful. Both of Permutations and Permutations II can be solved easily using this function. Hints: sort `nums` in ascending order, add it to the result of all permutations and then repeatedly generate the next permutation and add it ... until we get back to the original sorted condition. If you want to learn more, please visit this solution and that solution.

```
class Solution {
    void nextPermutation(vector<int>& nums) {
        int k = -1;
        for (int i = nums.size() - 2; i >= 0; i--) {
            if (nums[i] < nums[i + 1]) {
                k = i;
                break;
            }
        }
        if (k == -1) {
            reverse(nums.begin(), nums.end());
            return;
        }
        int l = -1;
        for (int i = nums.size() - 1; i > k; i--) {
            if (nums[i] > nums[k]) {
                l = i;
                break;
            }
        }
        swap(nums[k], nums[l]);
        reverse(nums.begin() + k + 1, nums.end());
    }
};
```

written by jianchao.li.fighter original link here

Solution 3

```
class Solution {
public:
    void nextPermutation(vector<int> &num)
    {
        if (num.empty()) return;

        // in reverse order, find the first number which is in increasing trend ( we call it violated number here)
        int i;
        for (i = num.size()-2; i >= 0; --i)
        {
            if (num[i] < num[i+1]) break;
        }

        // reverse all the numbers after violated number
        reverse(begin(num)+i+1, end(num));
        // if violated number not found, because we have reversed the whole array , then we are done!
        if (i == -1) return;
        // else binary search find the first number larger than the violated numb er
        auto itr = upper_bound(begin(num)+i+1, end(num), num[i]);
        // swap them, done!
        swap(num[i], *itr);
    }
};
```

You might need to think for a while why this would work.

written by AsherBaby original link here