喜刷刷

Monday, November 17, 2014

[LeetCode] Permutations I, II

Permutations I

Given a collection of numbers, return all possible permutations.

For example,

`[1,2,3]` have the following permutations:

`[1,2,3]`, `[1,3,2]`, `[2,1,3]`, `[2,3,1]`, `[3,1,2]`, and `[3,2,1]`.

Permutations II

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,

`[1,1,2]` have the following unique permutations:

`[1,1,2]`, `[1,2,1]`, and `[2,1,1]`.

思路：Permutations I

方法1：插入法

与subset I的方法2很相近。以题中例子说明：
当只有1时候：[1]
当加入2以后：[2, 1], [1, 2]
当加入3以后：[3, 2, 1], [2, 3, 1], [2, 1, 3], [3, 1, 2], [1, 3, 2], [1, 2, 3]
前3个permutation分别对应将3插入[2, 1]的0, 1, 2的位置。同理后3个为插入[1, 2]的。因此可以用逐个插入数字来构造所有permutations。

```
class Solution {
public:
    vector<vector<int> > permute(vector<int> &num) {
        vector<vector<int>> allPer;
        if(num.empty()) return allPer;
        allPer.push_back(vector<int>(1,num[0]));

        for(int i=1; i<num.size(); i++) {
            int n = allPer.size();
            for(int j=0; j<n; j++) {
                for(int k=0; k<allPer[j].size(); k++) {
                    vector<int> per = allPer[j];
                    per.insert(per.begin()+k, num[i]);
                    allPer.push_back(per);
                }
                allPer[j].push_back(num[i]);
            }
        }
        return allPer;
    }
};
```

方法2：backtracking法

和combination/subset不同，数字不同的排列顺序算作不同的permutation。所以我们需要用一个辅助数组来记录当前递归层时，哪些数字已经在上层的递
了。

```
class Solution {
public:
    vector<vector<int> > permute(vector<int> &num) {
        vector<vector<int>> allPer;
        if(num.empty()) return allPer;
        vector<bool> used(num.size(), false);
        vector<int> per;
        findPermutations(num, used, per, allPer);
        return allPer;
    }

    void findPermutations(vector<int> &num, vector<bool> &used, vector<int> &per, vector<vector<int>> &allPe
        if(per.size()==num.size()) {
            allPer.push_back(per);
            return;
        }

        for(int i=0; i<num.size(); i++) {
            if(used[i]) continue;
            used[i] = true;
            per.push_back(num[i]);
            findPermutations(num, used, per, allPer);
            used[i] = false;
            per.pop_back();
        }
    }
};
```

思路：Permutations I

与I的区别在于有重复元素，所以在解集中要去重复。思路和combination II, subset II的去重复基本一致。通过排序 + 每层递归跳过重复数字。注意这里的
的是一直到当前递归层，还未被使用的数字中的重复。

```
class Solution {
public:
    vector<vector<int> > permuteUnique(vector<int> &num) {
        vector<vector<int>> allPer;
        if(num.empty()) return allPer;
        sort(num.begin(),num.end());
        vector<int> per;
        vector<bool> used(num.size(),false);
        findPerUniq(num, used, per, allPer);
        return allPer;
    }

    void findPerUniq(vector<int> &num, vector<bool> &used, vector<int> &per, vector<vector<int>> &allPer) {
        if(per.size()==num.size()) {
            allPer.push_back(per);
            return;
        }

        for(int i=0; i<num.size(); i++) {
            if(used[i]) continue;
            if(i>0 && num[i]==num[i-1] && !used[i-1]) continue;
            used[i] = true;
            per.push_back(num[i]);
            findPerUniq(num, used, per, allPer);
            per.pop_back();
            used[i] = false;
        }
    }
};
```

Posted by Yanbing Shi at 11:36 PM

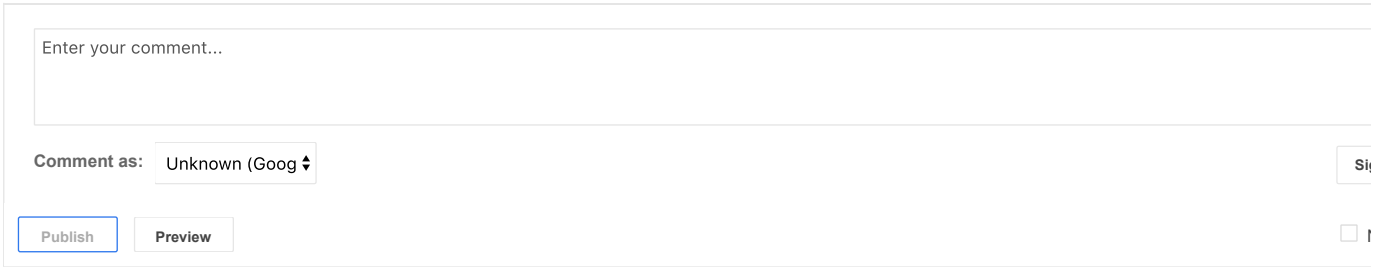
Labels: algorithm, backtracking, Leetcode, recursive

1 comment:

马友实 May 7, 2015 at 4:52 AM

good work

Reply



Newer Post Home

Subscribe to: Post Comments (Atom)

Simple template. Powered by Blogger.