```
/*
给定一个可包含重复数字的序列，返回所有不重复的全排列。

示例:

输入: [1,1,2]
输出:
[
  [1,1,2],
  [1,2,1],
  [2,1,1]
]

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/permutations-ii
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 与046 全排列思想类似,都是一个数字一个数字的填,直到填充缓存的长度等于数组的长度就保存一遍.
- 考虑到与与046 全排列不同的是,这个数组里面是有重复元素的,难点在于如何去除重复的排列.
- 第一种思路是先全排列再利用set去重,虽然很慢,但是可以学习到set要怎么用.
- 第二种思路是先对原始数组进行排序,最简单的情况的就是2个重复，遇到重复两个的第一个正常explore，遇到重复两个的第二个就直接unchoose或者不进入choose流程，这样就能筛选出来了.
- 方法二致谢: happygirlzt

方法一:C++_不筛选回溯+set去重.

```
class Solution
{

    private:

        void helper(    set<vector<int>>&       svi     ,
                        vector<int>&            cur     ,
                        vector<int>&            nums    ,
                        vector<int>&            visited
                    )
        {
            if(cur.size() == nums.size())
            {
                svi.insert(cur);
                return;
            }


            for(int i = 0 ; i < nums.size() ; i++)
            {
```

```
                if(visited[i])
                {
                    continue;
                }
                else
                {
                    cur.push_back(nums[i]);             // choose
                    visited[i]  =   1;
                    helper(svi,cur,nums,visited);       // explore
                    visited[i]  =   0;                  // unchoose
                    cur.pop_back();
                }
            }
        }

    public:

        vector<vector<int>> permuteUnique(vector<int>& nums)
        {
            vector<vector<int>> vvi;
            set<vector<int>>    svi;
            vector<int>         cur;
            vector<int>         visited(nums.size(),0);

            if(nums.size() < 1)
            {
                return vvi;
            }
            helper(svi, cur, nums , visited);

            set<vector<int>> ::iterator iter = svi.begin();
            while(iter != svi.end())
            {
                vvi.push_back(*iter);
                iter++;
            }

            return vvi;
        }
};

/*
执行结果:
通过
显示详情
执行用时 :208 ms, 在所有 cpp 提交中击败了11.33% 的用户
内存消耗 :10.7 MB, 在所有 cpp 提交中击败了53.01%的用户
*/

```

方法二:C++_排序+筛选回溯

```
class Solution
{
    private:
        void helper(    vector<vector<int>>&    vvi,
```

```
                        vector<int>&            cur,
                        vector<int>&            nums,
                        vector<int>&            visited
                    )
        {
            if(cur.size() == nums.size())
            {
                vvi.push_back(cur);
                return;
            }

            for(int i = 0 ; i < nums.size() ; i++)
            {
                if(visited[i])
                {
                    continue;
                }

                /* 跳过与上一个重复的,最基本的重复情况就是重复两个*/
                if(i > 0 && nums[i] == nums[i-1] && !visited[i-1]) //
                {
                    continue;
                }

                cur.push_back(nums[i]);         /* choose*/
                visited[i] = 1;
                helper(vvi,cur,nums,visited);   /* explore*/
                visited[i] = 0;                 /* unchoose*/
                cur.pop_back();

            }

        }

    public:
        vector<vector<int>> permuteUnique(vector<int>& nums)
        {
            vector<vector<int>> vvi                     ;
            vector<int>         cur                     ;
            vector<int>         visited(nums.size(),0)  ;
            if(nums.size() < 1)
            {
                return vvi;
            }
            /*先排序一下*/
            sort(nums.begin(),nums.end());
            helper(vvi,cur,nums,visited);
            return vvi;

        }
};

/*
执行结果:
通过
显示详情
执行用时 :28 ms, 在所有 cpp 提交中击败了93.70% 的用户
内存消耗 :9.8 MB, 在所有 cpp 提交中击败了93.27%的用户
```

```
63 */
```

AlimyBreak
2019.11.23