/\*

Given a set of distinct integers, nums, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example,

If nums = [1,2,3], a solution is:

[

[3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2],

[]

]

way-1 : 递归往子序列中添加

way-2:

具体的思路是这样的：

初始状态: []

第0次，加入S[0]: [], [1]

第1次，加入S[1]: [], [1], [2], [1, 2]

第1次，加入S[2]: [], [1], [2], [1, 2], [3], [1,3], [2,3], [1,2,3]

从上面可以看出，第0次->第1次，把S[1]加入到S[0]的每个subset中，形成新的subset(即[2], [1, 2])这堆新的subset和之前S[0]时候的那堆老的subset(即[], [1])一起构成S[1]的结果(即 [], [1], [2], [1, 2])。

重复这个过程直到把S的每个元素都加入了当前的集合。

way-3:

第三个思考方向，从数学角度来看，假设原集合有n个元素，那么原集合的子集合的个数是2的n次方，记为2 ^ n。对应着从0~2 ^ n - 1这2 ^ n个数。这2 ^ n个数如果用二进制表示，可以发现一共有n位。每位要么取0，要么取1。如果第i位取0，则说明元集合的第i个元素不出现在当前新生成的子集合中，反之，如果第i位取1，则说明元集合的第i个元素出现在当前新生成的子集合中。这个思路可以通过比特位操作来实现。

\*/

class Solution {

public:

void guocheng(int n,int k,int pos,vector<vector<int>> &result,vector<int> m1,vector<int>& nums)

{ //k代表还要选几个 pos代表上一个是选的是多少

if(k==0)

{

result.push\_back(m1);

return;

}

for(int i=pos+1;i<=n-k;i++)

{

m1.push\_back(nums[i]);

guocheng(n,k-1,i,result,m1,nums);

m1.pop\_back();

}

}

vector<vector<int>> subsets(vector<int>& nums)

{

//way-1

/\*

vector<vector<int>> result;

vector<int> m1;

result.push\_back(m1);

for(int i=1;i<=nums.size();i++)//i控制总长度

{

m1.clear();

guocheng(nums.size(),i,-1,result,m1,nums);

}

return result;

\*/

//way-2

/\*

vector<vector<int>> ret;

vector<int> m1;

ret.push\_back(m1);

int n=nums.size();

for(int i=0;i<n;i++)

{

int k=ret.size();

for(int j=0;j<k;j++)

{

ret.push\_back(ret[j]);

ret[ret.size()-1].push\_back(nums[i]);

}

}

return ret;

\*/

//way-3

vector<vector<int>> ret;

int n=nums.size();

for(int i=0;i<(1<<n);i++) // 1<<n 等于2^n

{

vector<int> m1;

for(int j=0;j<n;j++)

{

if((i>>j) & 1) // i的第j位是否是1

m1.push\_back(nums[j]);

}

ret.push\_back(m1);

}

return ret;

}

};