题目：

Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]


Output: true


Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: [1, 2, 3, 5]


Output: false


Explanation: The array cannot be partitioned into equal sum subsets.
```

思路：

两个特殊情况：1.如果能够选的最大数字大于等于 desiredTotal，第一个人又不傻肯定选最大的值～那样他就赢啦～即：

if(maxn >= desiredTotal) return true;

2.如果所有能够选的数字的总和都小于 desiredTotal，再怎么选两个人都不可能赢，所以肯定输～：

总和就是首项加末项的和乘以项数除以 2～：if((1 + maxn) * maxn / 2 < desiredTotal) return false;

然后建立一个 canWin 函数，需要用 visited 标记某个数字是否被选过～因为可选的数字最大不超过 20，则可以用一个整型数组标记，因为整型有 32 位～如果 1 被选过就把第 1 位（不是第 0 位哦～当然也可以从 0 开始啦～）标记为 1，如果 2 被选过就把第 2 位标记为 1～这样保证所有数字不重复～

所以传入两个值，一个是想要达到（或者说超过，也就是大于等于啦）的目标数字 target，另一个是 visited 数字标记哪些数字被选过了～

用 map 标记当前情况在 map 表中是否存在，存在的话结果保存在 map 里面～如果我们发现这个 visited 也就是这个数字选择的情况已经被保存过了，就直接返回在 map 里面保存的结果～

遍历 i 从 1 到 maxn（maxn 是可以选择的最大值，即 maxChoosableInteger），表示考虑选择 i 的情况，用 mask = 1 << i，如果 mask 和 visited 进行与运算，如果等于 0 说明当前的 visited 没有被访问过，就可以考虑这个 i 的情况，如果要选的这个 i 大于 target，不傻的这个人就会选择 i 因为肯定能赢啦～还有种情况自己能赢，就是对方输了，即：canWin(target - i, mask | visited) == false，

（mask | visited 表示把 i 那位也标记为 1～）这个时候把 visited 情况存起来并且 return true，表示赢了～如果所有数字都遍历完还是没有 return true，那就最后 return false～return false 之前不要忘记把当前状态存储起来～也就是 m[visited] = false; ～

注意：调了两个小时的 bug 后来才发现！(mask & visited) == 0 一开始忘记加括号了，写成了 mask & visited == 0，但是&运算符比==优先级低，所以==会先运算。

1.时间：O（ ）; 空间：O（ ）

```
class Solution {
public:
    bool canIWin(int maxChoosableInteger, int desiredTotal) {
        if (maxChoosableInteger > desiredTotal) return true;
        if ((1 + maxChoosableInteger) * maxChoosableInteger / 2 < desiredTotal) return false;
        return dfs(maxChoosableInteger, desiredTotal, 0);
    }
private:
    bool dfs(const int maxmaxChoosableInteger, int target, int visited){
        if (hashTable.find(visited) != hashTable.end()) return hashTable[visited];
        for (int i = 0; i < maxmaxChoosableInteger; ++i){
            int mask = 1 << i;
            if ((mask & visited) == 0 &&
                (target <= (i + 1) || !dfs(maxmaxChoosableInteger, target - (i + 1), mask | visited))){
                    hashTable[visited] = true;
                    return true;
            }
        }
        hashTable[visited] = false;
```

```
            return false;
        }
    private:
        std::unordered_map<int, bool> hashTable;
};
```