77. Combinations

Solved

Medium | Topics | Companies

Given two integers `n` and `k`, return *all possible combinations of* `k` *numbers chosen from the range* `[1, n]`.

You may return the answer in **any order**.

Example 1:

```
Input: n = 4, k = 2
Output: [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
Explanation: There are 4 choose 2 = 6 total combinations.
Note that combinations are unordered, i.e., [1,2] and [2,1] are considered to be the same combination.
```

Example 2:

```
Input: n = 1, k = 1
Output: [[1]]
Explanation: There is 1 choose 1 = 1 total combination.
```

Constraints:

- `1 <= n <= 20`
- `1 <= k <= n`

Classic example of Backtracking approach

naive approach

```cpp
class Solution {
public:
    void find(vector<int> &nums,int i,int k,vector<int> &currPer,vector<vector<int>> &ans){
        if(i == nums.size()){
            if(currPer.size() == k)
                ans.push_back(currPer);
            return;
        }

        currPer.push_back(nums[i]);
        find(nums,i+1,k,currPer,ans);
        currPer.pop_back();
        find(nums,i+1,k,currPer,ans);
    }

    vector<vector<int>> combine(int n, int k) {
        vector<int> nums(n);

        for(int i=0;i<n;i++)
            nums[i] = i+1;
```

```
        vector<vector<int>> ans;
        vector<int> currPer;

        find(nums,0,k,currPer,ans);

        return ans;
    }
};
```

optimization 1: we don't need to create an explicit nums vector.

```
class Solution {
public:
    void find(int i,int n,int k,vector<int> &currPer,vector<vector<int>> &ans){
        if(i == n){
            if(currPer.size() == k)
                ans.push_back(currPer);
            return;
        }

        currPer.push_back(i+1);
        find(i+1,n,k,currPer,ans);
        currPer.pop_back();
        find(i+1,n,k,currPer,ans);
    }

    vector<vector<int>> combine(int n, int k) {

        vector<vector<int>> ans;
        vector<int> currPer;

        find(0,n,k,currPer,ans);

        return ans;
    }
};
```

optimization 2: we can eliminate some recursive calls that are not going to get us to the required answer.

```
class Solution {
public:
    void find(int i,int n,int k,vector<int> &currPer,vector<vector<int>> &ans){
        if(i == n || (currPer.size()+n-i) < k){
            if(currPer.size() == k)
                ans.push_back(currPer);
```

```
            return;
        }

        currPer.push_back(i+1);
        find(i+1,n,k,currPer,ans);
        currPer.pop_back();
        find(i+1,n,k,currPer,ans);
    }

    vector<vector<int>> combine(int n, int k) {

        vector<vector<int>> ans;
        vector<int> currPer;

        find(0,n,k,currPer,ans);

        return ans;
    }
};
```

↳ i.e. if we can't create the k sized vector with the current elements of vector and by considering ALL the remaining elements, then there is no point of going to that branches. So we can avoid exploring them.