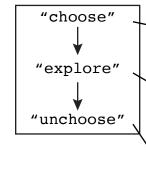
Backtracking Notes

Every backtracking problem can be solved by the following strategy:

```
[ Use an iterator to list all the possible starting points for
our recursion.
```



Select one number by adding it to a stack that holds the L current branch.

Recursively call the 'explore_helper' function which will

contains the numbers chosen in the current branch.

carry on the recursion and pass along the stack which

Remove the recently added number and go back to step 1 to explore another sub-branch. The termination condition that will stop the recursion and add the current branch to the 'results' list is given by the

comparison between the length of the branch and the length

of the original list to permute.

Subset (LC 78)

Given an integer array nums of unique elements, return all possible subsets (the power set).

The solution set **must not contain duplicate subsets.** Return the solution in any order.

```
lst = ["a", "b", "c"]
answer = [
   ['a', 'b', 'c'],
   ['a', 'b'],
   ['a', 'c'],
   ['a'],
   ['b', 'c'],
   ['b'],
   ['c'],
   []]
```

class SolutionSubset:

Subset II (LC 90)

Given an integer array nums that may contain duplicates, return all possible subsets (the power set).

The solution set must not contain duplicate subsets. Return the solution in any order.

```
def subsets(self, nums: List[any]) -> List[List[any]]:
                                                                 def subsetWithDup(self, nums: List[any]) -> List[List[any]]:
    answer = []
                                                                     answer = []
    subset = []
                                                                     subset = []
                                                                                      sort the nums first
                                                                     nums.sort()
    def backtrack(i):
       if i >= len(nums):
                                                                     def backtrack(i):
                                                                         if i == len(nums):
            answer.append(subset.copy())
                                                                             answer.append(subset.copy())
            return
                                                                             return
                                                                                                    add to the subset,
        subset.append(nums[i])
                                add to the subset,
                                                                                                     try the backtrack by incrementing i;
        backtrack(i + 1)
                                                                         subset.append(nums[i])
                                  try the backtrack by incrementing i;
                                                                         backtrack(i + 1)
                                                                                                   remove from subset,
        subset.pop()
                                                                         subset.pop()
                                                                                                     try the backtrack by incrementing i
                                remove from subset,
        backtrack(i + 1)
                                  try the backtrack by incrementing i
                                                                                                        keep incrementing i if the nums[i]
                                                                         while (i + 1 < len(nums)) and
                                                                                 nums[i] == nums[i + 1]):
    backtrack(0)
                                                                                                            is a duplicate of the previous.
                                                                             i += 1
                                                                         backtrack(i + 1)
                                                                                         without it, returns a set without duplicates (incorrect).
    return answer
                                                                     backtrack(0)
                                                                     return answer
```

class SolutionSubsetII:

Combination Sum (LC 39)

Given a collection of candidate numbers (candidates) and a target number (target), return a list of all unique combinations of candidates where the chosen numbers sum

The same number may be chosen from candidates an unlimited number of times.

The solution set **must not** contain duplicate combinations.

```
candidates = [2, 3, 5]
target = 8
answer = [
    [2, 2, 2, 2],
    [2, 3, 3],
    [3, 5]]
```

class SolutionCombinationSum:

['a', 'c'],

['b', 'c'],

[ˈaˈ],

['b'], ['c'],

Combination Sum II (LC 40)

Given a collection of candidate numbers (candidates) and a target number (targett), find all unique combinations in candidates where the candidate numbers sum to target. Number in candidates may only be **used once**.

```
candidates = [10,1,2,7,6,1,5]
target = 8
answer = [
    [1,1,6],
    [1,2,5],
    [1,7],
   [2,6]]
```

```
def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
                                                                                                      def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
    answer = []
                                                                                                          answer = []
                                                                                                          permutation = []
    permutation = []
                                                                                                          candidates.sort() sort the candidates first
    def backtrack(i):
        if sum(permutation) == target:
                                                                                                          def backtrack(i):
                                                      check if the sum() of permutation == target
             answer.append(permutation.copy())
                                                                                                              if sum(permutation) == target:
                                                       also check that it doesn't hit edge cases
                                                                                                                   answer.append(permutation.copy())
        if i >= len(candidates) or sum(permutation) > target:
                                                                                                              if sum(permutation) >= target:
                                                                                                                   return
        permutation.append(candidates[i]) add to the permutation,
                                                                                                               prev = -1
                                                                                                                                                         for loop; idx starting from i -> end of length of candidates
        backtrack(i)
                                                                                                               for idx in range(i, len(candidates)):
                                                                                                                       candidates. sort() and this code allows be used once.

Candidates sort() and this code allows be used once.

Candidates may only be used once.
                                                   try the backtrack with i;
                                                                                                                   if candidates[idx] == prev:
                                                                                                                                                           this SKIPS the idx if there is a duplicate, runs continue
                                                  remove last item from permutation,
        permutation.pop()
                                                   try the backtrack by incrementing i
                                                                                                                    permutation.append(candidates[idx])
        backtrack(i + 1)
                                                                                                                   backtrack(idx + 1)
                                 "same number may be chosen from candidates an unlimited number of times"
                                                                                                                   permutation.pop()
    backtrack(0)
                                                                                                                   prev = candidates[idx]
    return answer
                                                                                                          backtrack(0)
                                                                                                          return answer
```

class SolutionCombinationSumII:

the for-loop inside of the backtrack() function ensures that all indices that are left in candidates is tested, and also allows skipping duplicates.