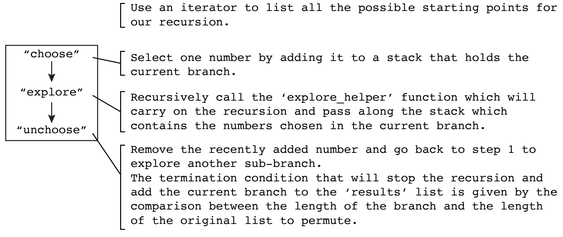
Backtracking Notes

Every backtracking problem can be solved by the following strategy:



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:
    def subsets(self, nums: List[any]) -> List[List[any]]:
        answer = []
        subset = []

        def backtrack(i):
            if i >= len(nums):
                answer.append(subset.copy())
                return

            subset.append(nums[i])
            backtrack(i + 1)

            subset.pop()
            backtrack(i + 1)

        backtrack(0)

        return answer
```

add to the `subset`, try the backtrack by incrementing `i`;
remove from `subset`, try the backtrack by incrementing `i`

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.

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

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

```
class SolutionSubsetII:
    def subsetWithDup(self, nums: List[any]) -> List[List[any]]:
        answer = []
        subset = []
        nums.sort()

        def backtrack(i):
            if i == len(nums):
                answer.append(subset.copy())
                return

            subset.append(nums[i])
            backtrack(i + 1)
            subset.pop()

            while (i + 1 < len(nums) and
                   nums[i] == nums[i + 1]):
                i += 1
            backtrack(i + 1)

        backtrack(0)

        return answer
```

sort the `nums` first

add to the `subset`, try the backtrack by incrementing `i`;
remove from `subset`, ~~try the backtrack by incrementing `i`~~

keep incrementing `i` if the `nums[i]` is a duplicate of the previous.

then try the backtrack by incrementing `i`;

this allows getting all duplicated values. without it, returns a set without duplicates (incorrect).

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

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 to target.
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:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        answer = []
        permutation = []

        def backtrack(i):
            if sum(permutation) == target:
                answer.append(permutation.copy())
                return
            if i >= len(candidates) or sum(permutation) > target:
                return

            permutation.append(candidates[i])
            backtrack(i)

            permutation.pop()
            backtrack(i + 1)

        backtrack(0)

        return answer
```

check if the sum() of `permutation` == target
also check that it doesn't hit edge cases

add to the `permutation`, try the backtrack with `i`;

remove last item from `permutation`, try the backtrack by incrementing `i`

"same number may be chosen from `candidates` **an unlimited number of times**"

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

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]]
```

```
class SolutionCombinationSumII:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        answer = []
        permutation = []
        candidates.sort()

        def backtrack(i):
            if sum(permutation) == target:
                answer.append(permutation.copy())
            if sum(permutation) >= target:
                return

            prev = -1
            for idx in range(i, len(candidates)):
                if candidates[idx] == prev:
                    continue
                permutation.append(candidates[idx])
                backtrack(idx + 1)
                permutation.pop()
                prev = candidates[idx]

        backtrack(0)

        return answer
```

sort the `candidates` first

for loop; `idx` starting from `i` -> end of length of `candidates`

this SKIPS the `idx` if there is a duplicate, runs `continue`

add to the `permutation`, try the backtrack with `i + 1`;

`candidates.sort()` and this code allows skipping duplicate values "Number in `candidates` may only be **used once**"