class Solution:

"""

@param nums: A set of numbers

@return: A list of lists

"""

def subsets(self, nums):

# write your code here

if nums is None:

return []

result = []

nums.sort()

self.dfs(nums, **0**, [], result)

return result

def dfs(self, nums, pos, subset, res):

res.append([] + subset)

for i in range(pos, len(nums)):

subset.append(nums[i])

self.dfs(nums, i + **1**, subset, res)

subset.pop()

def dfs(self, nums, pos, subset, res):

res.append([] + subset)

for i in range(pos, len(nums)):

if (i != pos and nums[i] == nums[i - **1**]):

continue

subset.append(nums[i])

self.dfs(nums, i + **1**, subset, res)

subset.pop()

class Solution:

def permute(self, nums):

# write your code here

if nums is None:

return []

result, visited = [], [False] \* len(nums)

self.dfs(nums, visited, [], result)

return result

def dfs(self, nums, visited, permutation, res):

if len(nums) == len(permutation):

res.append([] + permutation)

for i in range(**0**, len(nums)):

if visited[i]:

continue

permutation.append(nums[i])

visited[i] = True

self.dfs(nums, visited, permutation, res)

visited[i] = False

permutation.pop()

def dfs(self, nums, visited, permutation, res):

if len(permutation) == len(nums):

res.append([] + permutation)

for i in range(**0**, len(nums)):

if visited[i]:

continue

if (i > **0** and nums[i] == nums[i - **1**] and visited[i - **1**] == False):

continue

permutation.append(nums[i])

visited[i] = True

self.dfs(nums, visited, permutation, res)

visited[i] = False

permutation.pop()

class Solution:

"""

@param n: Given the range of numbers

@param k: Given the numbers of combinations

@return: All the combinations of k numbers out of 1..n

"""

def combine(self, n, k):

# write your code here

result = []

self.dfs(n, k, **1**, [], result)

return result

def dfs(self, n, k, pos, combination, res):

if len(combination) == k:

res.append([] + combination)

for i in range(pos, n + **1**): //the range is pos to n+**1**

combination.append(i)

self.dfs(n, k, i + **1**, combination, res)

combination.pop()

Given two integers *n* and *k*, return all possible combinations of *k* numbers out of 1 ... *n*. Given n = 4 and k = 2, a solution is:

[[2,4], [3,4], [2,3], [1,2], [1,3], [1,4]]

class Solution:

"""

@param num: Given the candidate numbers

@param target: Given the target number

@return: All the combinations that sum to target

"""

def combinationSum2(self, num, target):

# write your code here

num.sort()

result, temp, visited = [], [], [False] \* len(num)

self.dfs(num, target, **0**, **0**, temp, visited, result)

return result

def dfs(self, nums, target, pos, current, temp, visited, res):

if current == target:

res.append(temp[:])

for i in range(pos, len(nums)):

if current + nums[i] <= target:

if (i > **0** and nums[i] == nums[i - **1**] and visited[i - **1**] == False):

continue

temp.append(nums[i])

visited[i] = True

self.dfs(nums, target, i + **1**, current + nums[i], temp, visited, res)

visited[i] = False

temp.pop()

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.Each number in C may only be used once in the combination. Given candidate set [10,1,6,7,2,1,5] and target 8,A solution set is:

[[1,7], [1,2,5], [2,6], [1,1,6]]