**广东海洋大学学生实验报告书（学生用表）**

**GDOU-B-11-112**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 分治法设计 | | | 课程名称 | | 算法设计与分析 | | | | 课程号 | | 32260035 |
| 学院(系) | 数学与计算机 | | 专业 | 物联网工程 | | | | 班级 | | 1224 | | |
| 学生姓名 | 黎川滔 | 学号 | 202211672411 | | 实验地点 | | 明德楼A1501 | | 实验日期 | | 2025.4.1 | |

# 实验目的

本次实验旨在掌握分治法的基本思想及其在算法设计中的应用。通过具体问题（如假币查找、中位数求解、芯片测试等）实践分治法的实现，理解其递归模式和效率优势。与此同时，比较不同分治策略（如二分法与三分法）的异同，分析其时间复杂度及适用场景。

# 实验环境

处理器：AMD Ryzen 5 6600H with Radeon Graphics。

LeetCode。

# 实验内容

## 3.1 实验任务1

给定若干个正整数a0,a1、...、an-1，从中选出若干个数，使它们的和恰好为k，要求找出选择元素个数最少的解。数值填充请采用随机数生成方式，种子填充为个人学号（202211672411）；为保证数据的可用性，可以考虑只生成有限范围内的数据，如1至100。请根据具体需要调整。

(1)阐述该问题属于回溯法中的哪类问题；

(2)写出解决该问题的算法框架，表述算法可以用流程图或伪代码等；

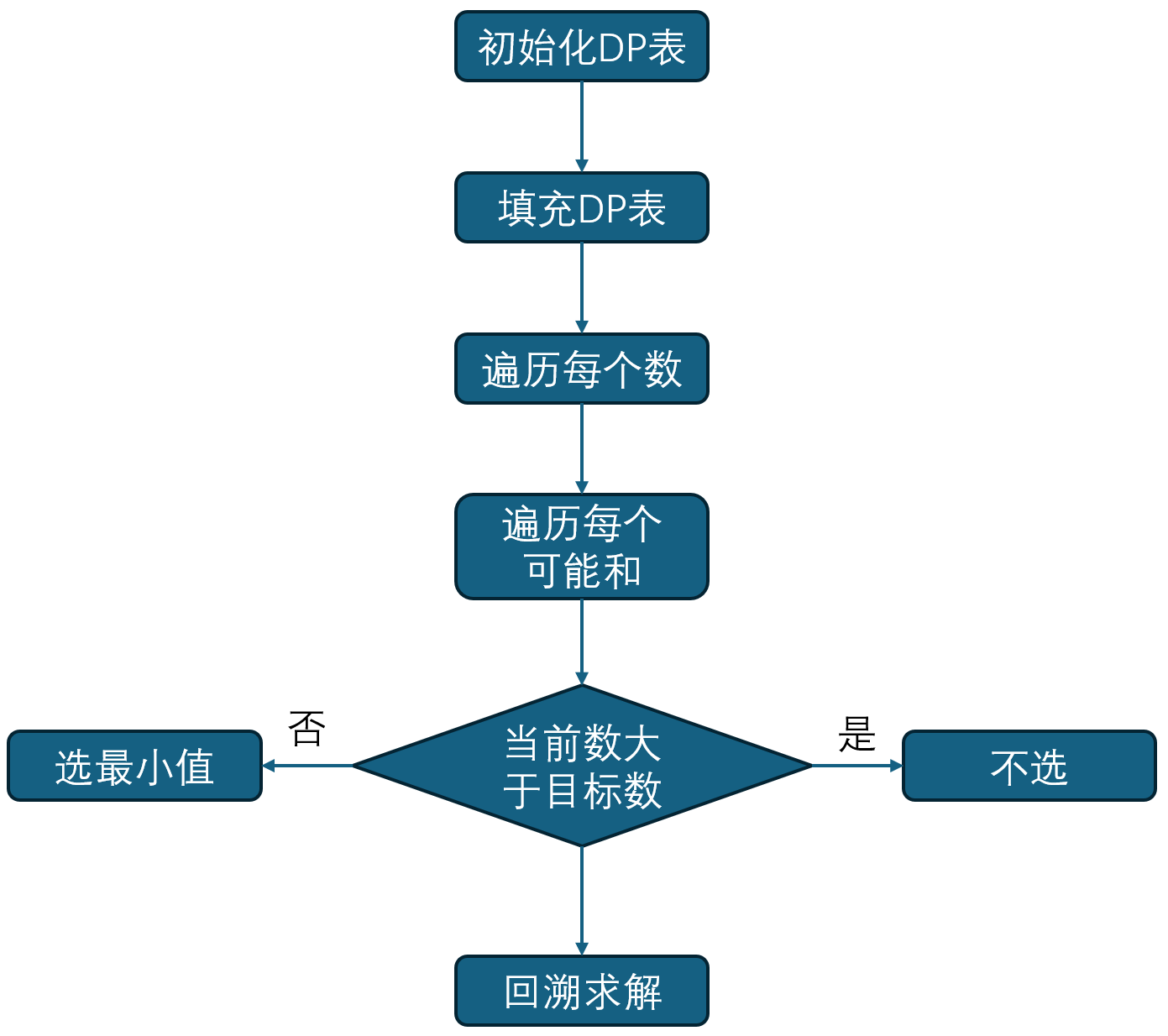
(3)提供程序的源代码，完整代码请以文本方式粘贴在实验报告中，并做适当说明，以便于评阅人评阅；

(4)利用程序展示测试数据，展示测试结果。

(5)分析其时间复杂度。

(1)该问题属于回溯法中的带约束的子集和问题；

(2)流程图：



(3)(4)见源码和测试结果：

import random

def find\_min\_subset(nums, target):

# 动态规划: dp[i][j]表示前i个元素中达到和j所需的最小元素数

dp = [[float('inf')] \* (target + 1) for \_ in range(len(nums) + 1)] # 初始化

dp[0][0] = 0 # 和为0需要0个元素

for i in range(1, len(nums) + 1): # i 从 1 开始遍历

for j in range(target + 1): # j 从 0 到 target

if nums[i - 1] > j: # 遍历每个数，因为是从 1 开始，所以要减去 1

dp[i][j] = dp[i - 1][j] # 如果当前数大于 j，则不能选这个数，因为选了就会超过 j

else:

dp[i][j] = min(dp[i - 1][j], dp[i - 1][j - nums[i - 1]] + 1)

# dp[i-1][j]：不选 nums[i-1]，直接用前 i-1 个数凑 j。

# dp[i-1][j - nums[i-1]] + 1：选 nums[i-1]，那么剩下的和是 j - nums[i-1]，用前 i-1 个数凑这个和，再加 1（因为选了 nums[i-1]）。

# 回溯找出选择的元素

if dp[len(nums)][target] == float('inf'):

return None, None

res = []

i, j = len(nums), target

while i > 0 and j > 0:

if dp[i][j] != dp[i - 1][j]:

res.append(nums[i - 1])

j -= nums[i - 1]

i -= 1

return dp[len(nums)][target], res[::-1]

random.seed(202211672411)

n = 20 # 数组长度

nums = [random.randint(1, 100) for \_ in range(n)] # 随机数组

target = random.randint(1, sum(nums)) # 随机目标和

print("随机数组:", nums)

print("随机目标和:", target)

count, result = find\_min\_subset(nums, target)

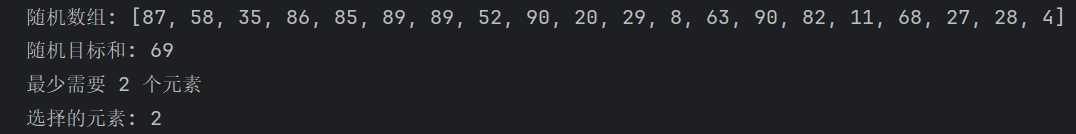
if count is not None:

print(f"最少需要 {count} 个元素")

print("选择的元素:", count)

else:

print("没有找到满足条件的子集")



(5)时间复杂度分析：DP表的填充有两层循环，外层是循环n次，内层是循环target次，回溯是最多遍历n次，故总时间复杂度为。

## 3.2 实验任务2

设计求解有重复元素的排列问题的算法。设有n个元素a[]={a0,a1,...，an-1}，其中可能含有重复的元素，求这些元素的所有不同排列。例如a[ ]={1, 1, 2}，输出结果是(1,1,2)、(1,2,1)、(2,1,1)。数值填充方式，取个人学号最后两位，并顺序递增若干数据；如个人学号为202211672411，取最后两位并适当递增可以是(0,0,1,2,2,3)或(0,0,1,1,2)等。

(1)阐述该问题属于回溯法中的哪类问题；

(2)写出解决该问题的算法框架，表述算法可以用流程图或伪代码等；

(3)提供程序的源代码，完整代码请以文本方式粘贴在实验报告中，并做适当说明，以便于评阅人评阅；

(4)分析其时间复杂度。

(5)利用程序展示测试数据，展示测试结果。

(1)这属于回溯法的排列问题；

(2)生成所有的排列组合 --> 集合去重 --> 输出结果；

(3)(5)见源代码和输出结果：

from itertools import permutations

def unique\_permutations(elements):

# 生成所有可能的排列

all\_perms = permutations(elements)

# 使用set去重

unique\_perms = set(all\_perms)

# 转换为列表并排序

sorted\_perms = sorted(unique\_perms)

return sorted\_perms

nums = [0, 0, 1, 1, 2]

result = unique\_permutations(nums)

for perm in result:

print(perm)

电脑键盘

AI 生成的内容可能不正确。

(6)生成排列有，去重有，排序有，故时间复杂度为。

## 3.3 实验任务3

class Solution:

def subsets(self, nums: List[int]) -> List[List[int]]:

res = [] # 用来存储子集

def dfs(index, path):

res.append(path[:]) # 记录当前路径形成的子集

# 遍历当前索引以及之后的元素

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

path.append(nums[i])

dfs(i+1,path)

path.pop()

dfs(0,[])

return res

这道题要求在给定的一个元素互不相同的整数数组 nums中，找到该数组所有可能的子集，我的代码使用dfs进行回溯。具体而言，对于每个元素，可以选择将其包含在当前子集中，也可以选择不包含。所谓回溯，即通过弹出最后加入的一个元素，回到前一步，再次遍历其他取值。

## 3.4 实验任务4

class Solution:

def combine(self, n: int, k: int) -> List[List[int]]:

return list(combinations([x for x in range(1, n + 1)], k))

首先生成一个从 1 到 n 的整数列表，然后使用 itertools.combinations 函数从生成的列表中选取所有长度为 k 的组合。最后，转换成列表返回结果。

## 3.5 实验任务5

class Solution:

def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:

from itertools import combinations

from typing import List

nums.sort() # 先排序，便于后续去重

res = set() # 使用集合来存储子集以实现去重

# 生成所有可能长度的子集

for i in range(len(nums) + 1):

# 生成长度为i的所有组合

for combo in combinations(nums, i):

res.add(combo) # 将组合作为元组添加到集合中

# 将集合中的元组转换回列表

return [list(subset) for subset in res]

选择第90题完成，此题要求返回一个整数数组 nums 的所有可能的子集，且解集中不能包含重复的子集。子集的顺序可以任意排列。首先，使用 itertools 模块中的 combinations 函数生成所有可能的子集，接着使用集合进行去重，最后转换成列表返回。

# 实验总结

我在这个实验中了解了回溯法的代码算法思想：动态规划和深度优先遍历，并通过老师布置的题目进行了练习巩固。尽管题目难度较大，但还是结合上网搜索资料逐渐理解题目的意思。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 成绩 |  | 指导教师 |  | 日期 |  |