**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 | 贪心法设计 | | | 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 | | 2025.4.8 | |

# 实验目的

本次实验

# 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：求解会议安排问题

有一组会议A和一组会议室B，A[i]表示第i个会议的参加人数，B[j]表示第j个会议室最多可以容纳的人数。当且仅当A[i]<=B[j]时第j个会议室可以用于举办第i个会议。给定数组A和数组B，试问最多可以同时举办多少个会议。例如，A[]=(1，2，3)，B[]=(3，2，4)，结果为3；若A[]=(3，4，3，1)，B[]=(1，2，2，6)，结果为2。

(1) 会议A数组与B数组分别采用个人学号（202211672411）与学号除以2生成数据；可以根据需要限制数值范围，如限制数组的数值1-100等；生成数据的数值如果不合适，可以自行调整，请在报告中写明具体的调整情况；

(2) 写出解决该问题的贪心准则；

(3) 分析其时间复杂度；

(4) 提供程序的源代码，代码请以文本方式粘贴在实验报告中；

(5) 展示运行效果。可以采用多组测试数据，展示并说明输入、输出结果。

(1) (2) 会议数组 A，其中 A[i] 表示第 i 个会议的参加人数；会议室数组 B，其中 B[j] 表示第 j 个会议室最多可以容纳的人数。只有当 A[i] <= B[j] 时，会议室 j 可以用于举办会议 i。对于此题可以采用贪心思想，首先将数组 A 和 B 进行升序，这样可以优先用最小的会议室满足最小的会议需求，避免浪费较大的会议室。接着，初始化两个指针 i 和 j 分别指向 A 和 B 的起始位置，进行 A 和 B 的遍历，如果 A[i] <= B[j]，则会议室 j 可以举办会议 i，计数加一，并移动 i 和 j；否则移动 j 寻找更大的会议室。最终，计数即为最多可以同时举办的会议数量。

(3) 主要取决于 max\_meetings 的排序部分，故时间复杂度为：。

(4) (5) 代码和结果如下：

def max\_meetings(A, B):

A\_sorted = sorted(A)

B\_sorted = sorted(B)

i = j = count = 0

while i < len(A\_sorted) and j < len(B\_sorted):

if A\_sorted[i] <= B\_sorted[j]:

count += 1

i += 1

j += 1

else:

j += 1

return count

student\_id = "202211672411"

A = [int(digit) for digit in student\_id if digit != '0']

B\_digits = [int(digit) for digit in str(int(student\_id) // 2) if digit != '0']

B = B\_digits[:len(A)] if len(B\_digits) > len(A) else B\_digits + [1] \* (len(A) - len(B\_digits)) # 补齐长度

print("A:", A)

print("B:", B)

max\_meetings\_count = max\_meetings(A, B)

print("最多可以同时举办的会议数量:", max\_meetings\_count)

电脑萤幕

AI 生成的内容可能不正确。

## 3.2 实验任务2：多机调度问题

设n（3<=n<=30）个独立作业，由5台机器M1、M2、M3、M4、M5加工处理。各作业所需的处理时间分别为N[i]。请写出解题思路，算法分析及程序代码。

(1) n个独作业，请根据要求利用学号填充随机生成，N[i]也采用学号（202211672411）随机生成；生成数据不适当的，可采用适当增减数值处理，写清处理的情况即可；

(2) 写明不同情况解决该问题的方法分析及时间复杂度分析；

(3) 提供程序的源代码，代码请以文本方式粘贴在实验报告中；

(4) 展示原始数据及处理方案情况。

(1) 多机调度问题是一个 NP-Hard 问题，首先将将作业按处理时间从大到小排序，每次将当前最长的作业分配给当前负载最小的机器。具体而言，代码使用使用最大堆维护机器负载，从而实现每次将最长作业分配给当前负载最小的机器。

(2) 排序作业的时间复杂度为O(n log n)，分配作业到机器的时间复杂度为O(n log 5)，故总时间复杂度为O(n log n)。

(3) (4) 见代码和输出结果：

import heapq

def generate\_jobs(n, student\_id="202211672411"):

digits = [int(d) for d in student\_id if d != '0']

jobs = digits \* (n // len(digits)) + digits[:n % len(digits)]

return jobs

def schedule\_jobs(jobs, m=5):

jobs\_sorted = sorted(jobs, reverse=True)

machines = [0] \* m

heapq.heapify(machines)

for job in jobs\_sorted:

earliest\_machine = heapq.heappop(machines)

heapq.heappush(machines, earliest\_machine + job)

makespan = max(machines)

return makespan, machines

n = 15

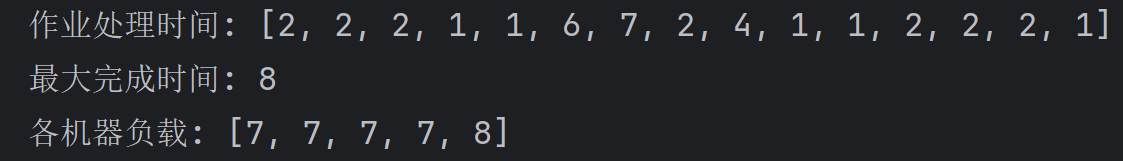
jobs = generate\_jobs(n)

print("作业处理时间:", jobs)

makespan, machine\_loads = schedule\_jobs(jobs)

print("最大完成时间:", makespan)

print("各机器负载:", machine\_loads)



## 3.3 实验任务3：Huffman编码树

假设有n个字符，也可以用a、b、c...等指代；各个字符的频数F；为简化题目，直接以整数代替。如字符a出现的频数是45，b现现的频数是55等。

class Node:

def \_\_init\_\_(self, char=None, freq=0, left=None, right=None):

self.char = char # 字符（仅叶子节点有）

self.freq = freq # 频数

self.left = left # 左子节点

self.right = right # 右子节点

def \_\_lt\_\_(self, other):

return self.freq < other.freq # 用于优先队列比较

def generate\_codes(root, current\_code="", code\_dict=None):

if code\_dict is None:

code\_dict = {}

if root.char is not None: # 叶子节点

code\_dict[root.char] = current\_code

else: # 内部节点

generate\_codes(root.left, current\_code + "0", code\_dict)

generate\_codes(root.right, current\_code + "1", code\_dict)

return code\_dict

def build\_huffman\_tree(freq\_dict):

# 创建优先队列

heap = []

for char, freq in freq\_dict.items():

heapq.heappush(heap, Node(char=char, freq=freq))

# 合并节点直到只剩一个根节点

while len(heap) > 1:

left = heapq.heappop(heap)

right = heapq.heappop(heap)

merged = Node(freq=left.freq + right.freq, left=left, right=right)

heapq.heappush(heap, merged)

return heapq.heappop(heap)

freq = {

'a': 45,

'b': 55,

'c': 30,

'd': 20,

'e': 15

}

huffman\_tree = build\_huffman\_tree(freq)

huffman\_codes = generate\_codes(huffman\_tree)

print("字符 | 频数 | 哈夫曼编码")

print("----------------------")

for char, code in sorted(huffman\_codes.items(), key=lambda x: len(x[1])):

print(f"{char:^4} | {freq[char]:^4} | {code}")

日历

AI 生成的内容可能不正确。

首先定义一个节点类，每个节点包含字符(char)、频数(freq)和左右子节点指针。接着，初始化优先队列。据特容颜，将每个字符及其频数创建为叶子节点，使用heapq.heappush将节点插入最小堆，堆顶总是频数最小的节点。然后合并节点循环​​，当堆中节点数大于1时继续合并，每次取出频数最小的两个节点(left和right)。最后创建新节点，其频数为两个子节点频数之和，将新节点重新插入堆中。当堆中只剩一个节点时，这就是哈夫曼树的根节点。

## 3.4 实验任务4：LeetCode121—买卖股票的最佳时机

class Solution:

def maxProfit(self, prices: List[int]) -> int:

from typing import List

if not prices:

return 0

min\_price = float('inf')

max\_profit = 0

for price in prices:

min\_price = min(min\_price, price)

max\_profit = max(max\_profit, price - min\_price)

return max\_profit

通过一次遍历数组来解决，记录当前最小价格和最大利润。用 min\_price 初始化为无穷大，用于跟踪迄今为止看到的最低价格，用 max\_profit 初始化为0，用于记录最大利润。接着遍历价格数组​​，使用内置的 min() 函数更新最低价格；使用内置的 max() 函数计算当前价格与最低价格的差，并更新最大利润。最终，返回计算得到的最大利润。

## 3.4 实验任务5：LeetCode122—买卖股票的最佳时机II

class Solution:

def maxProfit(self, prices: List[int]) -> int:

from typing import List

return sum(max(prices[i+1] - prices[i], 0) for i in range(len(prices)-1))

只要后一天的价格比前一天高，就进行买卖操作，将所有正收益累加就是最大利润。具体而言，使用生成器表达式遍历所有相邻日期的价格差。

# 实验总结

我在这个实验中了解了贪心法的代码算法思想，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 |  |