中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	计科 2 班	专业（方向）	计算机科学与技术
学号	2336126	姓名	李漾

一、 实验题目

Python 程序设计基础 I，II

二、 实验内容

（一）二分查找

1. 算法原理

二分查找适用于有序数组。每次将搜索区间划分为两部分，然后确定目标值可能存在的区间，直到找到目标值或者确定目标值不存在为止。

采用递归思想来实现：

Step1：(基本要素定义)

初始化左右指针，分别指向数组的起始位置和末尾位置。

Step2：(递归环节)

进入循环，每次计算中间位置的索引，并比较该中间值与目标值：

用 if 进行判断

(1)如果中间值等于目标值，则返回中间索引；

(2)如果中间值小于目标值，则将左指针移动到中间索引的右侧；

(3)如果中间值大于目标值，则将右指针移动到中间索引的左侧。

重复 Step2 直到找到目标值或者确定目标值不存在。

2. 伪代码

```
BinarySearch(nums, target):
    left = 0
    right = length(nums) - 1
    while left <= right:
        mid = (left + right) / 2
        if nums[mid] == target:
            return mid
        else if nums[mid] < target:
            left = mid + 1
        else:
```

```
            right = mid - 1
    return -1
```

3. 关键代码展示（带注释）

```python
def BinarySearch(nums, target):
    """
    :param nums: list[int]
    :param target: int
    :return: int
    """
    left= 0
    right=len(nums) - 1  # 初始化左右指针
    while left <= right:
        mid = (left + right) // 2  # 计算中间索引(注意是整除)
        if nums[mid] == target:  # 如果中间值等于目标值，则返回中间索引
            return mid
        elif nums[mid] < target:  # 如果中间值小于目标值，则将左指针移动到中间索引的右侧
            left = mid + 1
        else:  # 如果中间值大于目标值，则将右指针移动到中间索引的左侧
            right = mid - 1
    return -1  # 如果未找到目标值，则返回 -1
```

（二）矩阵加法,乘法

1. 算法原理

矩阵加法：

给定两个相同大小的矩阵 A 和 B，将它们对应位置的元素相加，得到一个新的矩阵。

矩阵乘法：

给定两个矩阵 A 和 B，其维度分别为(n x m)和(m x k)，将 A 的每一行与 B 的每一列进行乘积运算，然后将结果相加，得到新的矩阵。

2. 伪代码

矩阵加法：

```
MatrixAdd(A, B):
    n = A.rows
    result =  创建一个所有元素为 0 的  n x n  矩阵
    for i = 0 to n-1:
        for j = 0 to n-1:
            result[i][j] = A[i][j] + B[i][j]
    return    result
```

矩阵乘法

```
MatrixMul(A, B):
    n = A.rows          result =  创建一个所有元素为 0 的  n x n  矩阵
    for i = 0 to n-1:
        for j = 0 to n-1:
            for k = 0 to n-1:
                result[i][j] += A[i][k] * B[k][j]
    return    result
```

3. 关键代码展示（带注释）

```
def MatrixAdd(A, B):
    """
    :param A: list[list[int]]
    :param B: list[list[int]]
    :return: list[list[int]]
    """
    n = len(A)
    result = [[0] * n for _ in range(n)]  # 创建一个与 A、B 大小相同的零矩阵
    for i in range(n):
        for j in range(n):
            result[i][j] = A[i][j] + B[i][j]  # 对应位置元素相加
    return result

def MatrixMul(A, B):
    """
    :param A: list[list[int]]
    :param B: list[list[int]]
    :return: list[list[int]]
    """
    n = len(A)
    result = [[0] * n for _ in range(n)]   # 创建一个与 A、B 大小相同的零矩阵
    for i in range(n):
        for j in range(n):
            for k in range(n):
                result[i][j] += A[i][k] * B[k][j]  # 矩阵乘法的公式，逐个元素相乘再相加
    return result
```

4. 创新点&优化（如果有）

优化：可以考虑使用 NumPy 库来实现这些操作，因为 NumPy 内置了高效的矩阵运算功能，可以大大提高运行效率。

具体代码在 code2_v2

```
import numpy as np

def MatrixAdd(A, B):
    """
    :param A: list[list[int]]
    :param B: list[list[int]]
    :return: list[list[int]]
    """
    return np.array(A) + np.array(B)

def MatrixMul(A, B):
    """
    :param A: list[list[int]]
    :param B: list[list[int]]
    :return: list[list[int]]
    """
    return np.dot(A, B)
```

（三）字典遍历

1. 算法原理

即原字典的键变为新字典的值，原字典的值变为新字典的键。

2. 伪代码

ReverseKeyValue(dict1):
    新字典 = 空字典
    对于原字典中的每一个键值对 (k, v):
        将新字典中的键值对设为 (v, k)
    返回新字典

3. 关键代码展示（带注释）

```
def ReverseKeyValue(dict1):
    """
    :param dict1: dict
    :return: dict
    """
    r_dict = {}  # 创建一个空字典用于存放颠倒后的键值对
    for k, v in dict1.items():  # 遍历原字典的键值对
        r_dict[v] = k  # 将原字典的值作为新字典的键，原字典的键作为新字典的值
    return r_dict
```

（四）管理 Student 数据

1. 算法原理

初始化方法 (__init__):
打开文件并逐行读取文件内容。
将每行数据按空格分割成学生信息列表。
将每个学生信息列表添加到 self.data 列表中。
添加数据方法 (AddData):
创建包含新学生信息的列表。
将新学生信息列表添加到 self.data 列表中。
排序方法 (SortData):
根据输入的属性名称，使用 sort() 方法对 self.data 列表中的学生信息进行排序。
导出数据方法 (ExportFile):
遍历 self.data 列表中的学生信息。
将每个学生信息列表转换为字符串，并写入到文件中。

2. 伪代码

2. 伪代码
初始化方法 (__init__):
函数 __init__(file_name):
    打开文件 file_name
    对于文件中的每一行:
        将行按空格分割成学生信息列表 student_info
        将 student_info 添加到 self.data 列表中
添加数据方法 (AddData):
函数 AddData(name, stu_num, gender, age):
    创建包含新学生信息的列表 student

将 student 添加到 self.data 列表中
排序方法 (SortData):
函数 SortData(a):
如果 a 为 'name':
使用学生信息的第一个元素进行排序
如果 a 为 'stu_num':
使用学生信息的第二个元素进行排序
如果 a 为 'gender':
使用学生信息的第三个元素进行排序
如果 a 为 'age':
使用学生信息的第四个元素进行排序
导出数据方法 (ExportFile):
函数 ExportFile(filename):
打开文件 filename，准备写入数据
对于 self.data 列表中的每个学生信息 student_info:
将学生信息列表转换为字符串，使用空格分隔，并写入文件
关闭文件

3. 关键代码展示（带注释）

```
class StuData:
    def __init__(self, file_name):
        """
        构造函数，从文件中读取学生数据并存储到 self.data 中
        :param file_name: str, 学生数据文件名
        """
        self.data = []
        with open(file_name, 'r') as file:
            for line in file:
                student = line.strip().split()  # 将每一行的数据按空格分割成列表
                self.data.append(student)  # 将学生信息列表添加到 self.data 中

    def AddData(self, name, stu_num, gender, age):
        """
        添加新的学生数据到 self.data 中
        :param name: str, 学生姓名
        :param stu_num: str, 学号
        :param gender: str, 性别
        :param age: int, 年龄
        """
        student = [name, stu_num, gender, age]  # 创建学生信息列表
        self.data.append(student)  # 将学生信息列表添加到 self.data 中

    def SortData(self, a):
        """
        根据指定属性对学生数据进行排序
        :param a: str, 排序属性，可以是 'name', 'stu_num', 'gender' 或 'age'
        """
        if a == 'name':
            self.data.sort(key=lambda x: x[0])  # 根据姓名排序
        elif a == 'stu_num':
            self.data.sort(key=lambda x: x[1])  # 根据学号排序
        elif a == 'gender':
            self.data.sort(key=lambda x: x[2])  # 根据性别排序
        elif a == 'age':
            self.data.sort(key=lambda x: x[3])  # 根据年龄排序
```

```
#使用了 lambda 函数作为 key 参数，以实现按照指定属性排序的功能。
#lambda 函数根据输入的 x 值（即学生数据的一个列表），返回对应属性的值，然后 sort() 方法根据这个值进行排序。
    def ExportFile(self, filename):
        """
        将学生数据导出到文件
        :param filename: str, 导出文件名
        """
        with open(filename, 'w') as file:
            for student_info in self.data:
                file.write(' '.join(map(str, student_info)) + '\n')  # 将学生信息列表转换为字符串并写入文件中
```

4. 创新点&优化（如果有）

（1）使用上下文管理器 with 来自动关闭文件：

在 __init__() 和 ExportFile() 方法中，使用了上下文管理器 with 来打开和关闭文件。这种方式可以确保文件在使用完毕后自动关闭，而不需要手动调用 file.close() 方法，这样可以避免资源泄漏和文件被错误地锁定的情况。这是一种良好的编程实践，有助于代码的健壮性和可维护性。

（2）使用 map 函数将学生信息列表转换为字符串：

在 ExportFile() 方法中，通过 map() 函数将学生信息列表中的每个元素转换为字符串，然后使用 ''.join() 方法将这些字符串连接起来，以便将学生信息写入文件。这种方法比手动迭代学生信息列表并使用字符串拼接更简洁和高效。

（3）使用 key 参数进行灵活排序：

在 SortData() 方法中，使用了 key 参数来指定排序的规则。通过将 lambda 函数作为 key 参数传递给 sort() 方法，可以根据不同的属性进行灵活的排序。这样可以使排序逻辑更加清晰和灵活，避免了编写多个不同的排序方法。

三、 实验结果及分析

1. 实验结果展示示例

（一）二分查找

```
# 测试样例
nums = [2,3,7,11,35,56,236,293,778]
target = 35
print("Index of target:", BinarySearch(nums, target))
```

```
Index of target: 4
```

（二）矩阵加法，乘法

```
print("Matrix Add:")
result_addition = MatrixAdd(A, B)
for row in result_addition:
    print(row)

print("Matrix Mul:")
result_multiplication = MatrixMul(A, B)
for row in result_multiplication:
    print(row)
```

```
Matrix Add:
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
Matrix Mul:
[[ 30  36  42]
 [ 66  81  96]
 [102 126 150]]
```

（三）字典遍历

```
{'001': 'Alice', '002': 'Bob'}
```

（四）管理 Student 数据

```
Original Data:
['Aaron', '243', 'M', '18']
['Eric', '249', 'M', '19']
['Alex', '812', 'M', '19']
['Leo', '092', 'M', '17']
['Sam', '230', 'F', '18']
['Ruth', '942', 'M', '19']
['Beryl', '091', 'F', '20']
['Cynthia', '920', 'F', '19']

Data after adding new student:
['Aaron', '243', 'M', '18']
['Eric', '249', 'M', '19']
['Alex', '812', 'M', '19']
['Leo', '092', 'M', '17']
['Sam', '230', 'F', '18']
['Ruth', '942', 'M', '19']
['Beryl', '091', 'F', '20']
['Cynthia', '920', 'F', '19']
['Bob', '003', 'M', 20]

Data after sorting by student number:
['Bob', '003', 'M', 20]
['Beryl', '091', 'F', '20']
['Leo', '092', 'M', '17']
['Sam', '230', 'F', '18']
['Aaron', '243', 'M', '18']
['Eric', '249', 'M', '19']
['Alex', '812', 'M', '19']
['Cynthia', '920', 'F', '19']
['Ruth', '942', 'M', '19']
```

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

1. 二分查找

评测指标： 时间复杂度

分析： 二分查找算法的时间复杂度为 O(log n)，其中 n 为数组中元素的个数。因此对于较大规模的有序数组，二分查找算法比暴力查找更加高效。

2. 矩阵加法和乘法

评测指标： 时间复杂度

分析： 矩阵加法的时间复杂度为 O(n^2)，矩阵乘法的时间复杂度为 O(n^3)，其中 n 表示矩阵的大小。因此，随着矩阵规模的增加，矩阵乘法的运行时间会呈现出更快的增长趋势。对于较大规模的矩阵，优化的算法和数据结构（如 分治算法：将两个矩阵分解为更小的子矩阵，然后通过一系列的加法和减法操作来组合这些子矩阵以得到最终的乘积。）可以进一步降低矩阵乘法的时间复杂度。

3. 字典键值对颠倒

评测指标： 时间复杂度

分析： 对于给定的字典，将键值对颠倒的算法时间复杂度为 O(n)，其中 n 表示字典中键值对的数量。对于较大规模的字典，这个算法在时间上表现良好，但在空间上需要额外的存储空间来存放颠倒后的键值对。

4.管理 Student 数据

评测指标：功能完整性

分析：验证代码能够成功读取文件、添加学生数据、按照指定属性排序数据，并将排序后的数据正确导出到文件。每个方法的功能都需要进行测试，确保代码的完整性。

评测指标：可扩展性

分析：在现有代码基础上添加更多的功能，例如删除学生数据、更新学生信息等操作，评估代码的灵活性和可维护性。通过添加新功能并检查代码是否能够轻松扩展，评估其可扩展性。

四、 思考题

1. 编写一个猜数字游戏的程序： a）用 random 模块中的 randint()函数生成一个在 1 到 100 之间的随机整数作为答案；b）程序循环读取用户输入的猜测数字，并向用户提示猜测偏大/偏小，直到猜 中； c）用户猜中后，输出一共猜测了多少次

```
import random

def guess_number():
    # 生成随机答案
    answer = random.randint(1, 100)
    attempts = 0

    while True:
        # 循环读取用户输入的猜测数字
        guess = int(input("请输入一个猜测的数字（1到100之间）："))
        attempts += 1

        # 判断猜测的数字是否等于答案
        if guess == answer:
            print("恭喜你猜对了！答案是", answer)
            break
        elif guess < answer:
            print("猜的数字偏小，请继续尝试。")
        else:
            print("猜的数字偏大，请继续尝试。")

    print("你一共猜测了", attempts, "次。")

if __name__ == "__main__":
    guess_number()
```

2. 编程求出 1 到 100 的和（你能用一行代码完成吗？）

```
result = sum(range(1, 101))
```

3. 餐馆、就餐人数与冰淇淋小店： a）创建一个名为 Restaurant 的类，其方法__init__()设置两个属性：restaurant_name 和 cuisine_type。创建一个名为 describe_restaurant()的方法和一个名为 open_restaurant()的方法，其中前者打印前述两项信息，而后者打印一条消息，指出餐馆正在营业。 b）添加一

个名为 number_served 的属性，并将其默认值设置为 0。根据这个类创建一个名为 restaurant 的实例；打印有多少人在这家餐馆就餐过，然后修改这个值并再次打印它。添加 一个名为 set_number_served()的方法，它让你能够设置就餐人数；调用这个方法并向它传 递一个值，然后再次打印这个值。添加一个名为 increment_number_served()的方法，它让你 能够将就餐人数递增；调用这个方法并向它传递一个这样的值:你认为这家餐馆每天可 能接待的就餐人数。c) 冰淇淋小店是一种特殊的餐馆。编写一个名为 IceCreamStand 的类，让它继承 Restaurant 类。 添加一个名为 flavors 的属性，用于存储一个由各种口味的冰淇淋组成的列表。编写一个显 示这些冰淇淋的方法。创建一个 IceCreamStand 实例，并调用这个方法

```
class Restaurant:
    def __init__(self, restaurant_name, cuisine_type):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        print("Restaurant Name:", self.restaurant_name)
        print("Cuisine Type:", self.cuisine_type)

    def open_restaurant(self):
        print("The restaurant", self.restaurant_name, "is now open.")

    def set_number_served(self, number):
        self.number_served = number

    def increment_number_served(self, increment):
        self.number_served += increment


class IceCreamStand(Restaurant):
    def __init__(self, restaurant_name, cuisine_type, flavors):
        super().__init__(restaurant_name, cuisine_type)
        self.flavors = flavors

    def display_flavors(self):
        print("Available Ice Cream Flavors:")
        for flavor in self.flavors:
            print("-", flavor)
```

五、 参考资料

《数据结构与算法》