**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 | 常用数据结构及其应用 | | | 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 | | 2025.3.18 | |

# 实验目的

本次实验旨在通过实践深入理解常用数据结构与算法的原理及其应用，提升问题解决能力和编程实现技巧。实验内容涵盖优先队列、哈希表、动态数组、基数排序等核心数据结构，以及双指针、哈希冲突解决、元素查找与排序等经典算法。此外，实验结合LeetCode编程题目，强化对实际问题的分析与解决能力，并通过理论问答环节加深对数据结构与算法的理解。

# 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

编写一个程序，利用优先队例求出一个无序整数序列中的第k小的元素。数据的测试集长度自定，利用随机数生成，种子填充为个人学号（以李明为例，如202211672200），写出程序并完成数据测试。粘贴测试结果。程序请以文本方式粘贴在实验报告中。

1. import heapq

2. import random

3.

4.

5. def find\_kth\_smallest(arr, k):

6. if k <= 0 or k > len(arr):

7. return None

8. heap = []

9. for num in arr:

10. if len(heap) < k:

11. heapq.heappush(heap, -num)

12. else:

13. if num < -heap[0]:

14. heapq.heappop(heap)

15. heapq.heappush(heap, -num)

16. return -heap[0]

17.

18.

19. seed = 202211672411

20. random.seed(seed)

21.

22. n = 10

23. arr = [random.randint(1, 100) for \_ in range(n)]

24. print("生成的数组:", arr)

25.

26. for k in range(1, 6):

27. sorted\_arr = sorted(arr)

28. expected = sorted\_arr[k - 1] if k <= len(arr) else None

29. result = find\_kth\_smallest(arr, k)

30. print(f"\n排序后的数组: {sorted\_arr}")

31. print(f"第{k}小的元素: {result}")

上述代码使用heapq模块实现查找第k小的元素，实现优先队列的方式是最大堆。最大堆的结构可以理解为一棵平衡二叉树，根节点（堆顶）的元素一定是最大的，heapd模块的heappush可以自动插入并平衡。但由于heapq模块仅支持最小堆，因此在输入元素的时候要取负数，在返回的时候再取一次负数。

find\_kth\_smallest函数的输入参数是未排序的数组arr，和最k小的参数k，当堆元素数量小于k时，直接入堆；否则，新来的元素与堆顶比较，如果更小就进行更换。

其中，程序输出结果如图1所示。

文本

AI 生成的内容可能不正确。

**图1 实验任务1输出结果**

## 3.2 实验任务2

编写一个实验程序，设计一种好的数据结构，尽可能高效地实现元素的插入、删除、按值查找和按序号查找（假设所有元素的值均不相同）。

测试数据集请采用随机生成。数量自定，如生成1000个各不相同的数据等。可以生随机整数，也可以生成随机字符串。随机生成整数后，请向数据集中添加个人的学号（以李明为例，如202211672200）；如果生成随机字符串，请添加个人姓名的拼音全拼（以李明为例，如 LiMing）。

建议采用动态数组，容器（如map）等实现。

1. import random

2.

3.

4. class EfficientDataStructure:

5. def \_\_init\_\_(self):

6. self.array = [] # 动态数组存储元素

7. # 动态数组在初始化时不需要指定固定大小，而是在插入元素时自动扩展容量。Python中的列表就是一种典型的动态数组

8. self.value\_to\_index = {} # 哈希表：值 → 数组索引

9.

10. def insert(self, value):

11. """插入元素"""

12. if value in self.value\_to\_index: # 如果要插入的元素在字典里面了，为了不重复，不插入

13. return False

14. self.array.append(value) # 没有重复就插入动态数组

15. self.value\_to\_index[value] = len(self.array) - 1 # 字典的键是元素，值是在动态数组中的索引

16. return True

17.

18. def delete(self, value):

19. """删除元素"""

20. if value not in self.value\_to\_index: # 如果字典的键中不存在这个元素，即动态数组中没有这个元素

21. return False # 那就删除不了

22. index = self.value\_to\_index[value] # 否则先从字典中找个要删除元素的索引

23. last\_val = self.array[-1] # 获取数组末尾元素

24.

25. # 交换待删除元素与末尾元素

26. self.array[index], self.array[-1] = self.array[-1], self.array[index]

27. self.value\_to\_index[last\_val] = index

28.

29. # 交换后删除的时间复杂度是O(1)，不然后面的元素都要向前面移动

30. self.array.pop()

31. del self.value\_to\_index[value] # 因为待删除的元素已经在最后，且pop出去了，所以不需要更新字典

32. return True

33.

34. def get\_by\_value(self, value):

35. """按值查找是否存在"""

36. return value in self.value\_to\_index

37.

38. def get\_by\_index(self, index):

39. """按序号查找元素"""

40. if index < 0 or index >= len(self.array):

41. return None

42. return self.array[index] # 直接通过字典进行查找

43.

44. def size(self):

45. return len(self.array)

46.

47.

48. def generate\_test\_data(data\_type='int', size=1000, seed=202211672411):

49. """生成测试数据集"""

50. random.seed(seed)

51. data = set()

52.

53. if data\_type == 'int':

54. # 生成唯一随机整数

55. while len(data) < size - 1:

56. data.add(random.randint(0, 10 \*\* 12))

57. # 添加学号

58. student\_id = 202211672411

59. data.add(student\_id)

60. else:

61. while len(data) < size - 1:

62. s = ''.join(random.choices('abcdefghijklmnopqrstuvwxyz', k=10))

63. data.add(s)

64. # 添加姓名拼音

65. data.add("LiChuanTao")

66.

67. return list(data)

68.

69.

70.

71. if \_\_name\_\_ == "\_\_main\_\_":

72. test\_data = generate\_test\_data(data\_type='int', size=1000) # 生成1000个各不相同的整数

73.

74. # 初始化数据结构

75. ds = EfficientDataStructure()

76. for num in test\_data:

77. ds.insert(num)

78.

79. # 按值查找

80. student\_id = 202211672411

81. print(f"黎川滔的学号 {student\_id} 是否在数组中？", ds.get\_by\_value(student\_id))

82.

83. # 按序号查找

84. print("数组中索引为 0 的元素：", ds.get\_by\_index(0))

85.

86. # 删除元素

87. ds.delete(student\_id)

88. print("删除黎川滔学号后数组的大小", ds.size())

89. print(f"黎川滔的学号 {student\_id} 在删除后是否还存在？", ds.get\_by\_value(student\_id))

在上述代码中，为了设计一个高效的数据结构实现元素的插入、删除和查找，可以结合动态数组和哈希表。在Python中，列表是一个动态数组，不需要事先指定数组的大小；字典是一个哈希结构，字典的键用来存储元素，对应的值用来存储元素在数组的索引。这样的数据结构在插入、删除和查找中，时间复杂度都非常可观。

其中，任务2输出的结果如图2所示。

文本

AI 生成的内容可能不正确。

**图2 实验任务2输出结果**

## 3.3 实验任务3

请参考visualgo中的Hashtable，网址 <https://visualgo.net/zh/hashtable>；及Hello算法中的第6章哈希表 <https://www.hello-algo.com/chapter_hashing/> 尝试理解Linear Probing、Quadratic Probing、Double Hashing、Separate Chaining等哈希算法。

要求了解Hash的实现与基本应用。

请运用哈希结构，求解LeetCode1两数之和（用哈希表实现）；说明该求解方法的优点，存在哪些不足。

1. class Solution:

2. def twoSum(self, nums: List[int], target: int) -> List[int]:

3. dict = {} # 字典

4. for j, x in enumerate(nums): # 从左边开始遍历，键是元素，值是索引

5. if target - x in dict: # 如果目标值减去元素在字典中出现过

6. return [dict[target - x], j] # 那就返回字典对应键的值和当前j

7. dict[x] = j # 如果没有就在字典中赋值存储

哈希表的查找和插入操作的时间复杂度均为O(1)，遍历数组一次，时间复杂度为O(n)。然而，尽管哈希表的时间复杂度可观，但遇到大规模数据时，空间复杂度较高。上述代码在LeetCode的运行结构如图3所示。

|  |  |
| --- | --- |
| 图形用户界面, 文本, 应用程序  AI 生成的内容可能不正确。 | 图形用户界面, 文本, 应用程序, 电子邮件  AI 生成的内容可能不正确。 |
| (a) | (b) |

**图3 哈希表解决两数之和运行结果**

当发生哈希冲突时，线性探测会从冲突的位置开始，​依次检查下一个位置，直到找到一个空闲的位置来存储数据。线性探测的代码如下所示。

1. class HashTable:

2. def \_\_init\_\_(self, size):

3. self.size = size # 哈希表的大小

4. self.table = [None] \* size # 初始化哈希表，所有位置为空

5.

6. def hash\_function(self, key):

7. """哈希函数：计算键的哈希值"""

8. return key % self.size

9.

10. def insert(self, key, value):

11. """插入键值对到哈希表中"""

12. index = self.hash\_function(key) # 计算初始索引

13. while self.table[index] is not None: # 如果当前位置被占用

14. if self.table[index][0] == key: # 如果键已存在，更新值

15. self.table[index] = (key, value)

16. return

17. index = (index + 1) % self.size # 线性探测：检查下一个位置

18. self.table[index] = (key, value) # 找到空闲位置，插入键值对

19.

20. def search(self, key):

21. """查找键对应的值"""

22. index = self.hash\_function(key) # 计算初始索引

23. while self.table[index] is not None: # 如果当前位置被占用

24. if self.table[index][0] == key: # 如果找到键

25. return self.table[index][1] # 返回对应的值

26. index = (index + 1) % self.size # 线性探测：检查下一个位置

27. return None # 未找到键

28.

29. def delete(self, key):

30. """删除键值对"""

31. index = self.hash\_function(key) # 计算初始索引

32. while self.table[index] is not None: # 如果当前位置被占用

33. if self.table[index][0] == key: # 如果找到键

34. self.table[index] = None # 删除键值对

35. self.rehash(index) # 重新哈希后续的键值对

36. return

37. index = (index + 1) % self.size # 线性探测：检查下一个位置

38.

39. def rehash(self, start\_index):

40. """重新哈希从 start\_index 开始的所有键值对"""

41. index = (start\_index + 1) % self.size # 从下一个位置开始

42. while self.table[index] is not None: # 如果当前位置被占用

43. key, value = self.table[index] # 获取键值对

44. self.table[index] = None # 清空当前位置

45. self.insert(key, value) # 重新插入键值对

46. index = (index + 1) % self.size # 检查下一个位置

47.

48. def \_\_str\_\_(self):

49. """打印哈希表内容"""

50. return "\n".join(f"Index {i}: {self.table[i]}" for i in range(self.size))

51.

54. if \_\_name\_\_ == "\_\_main\_\_":

55. ht = HashTable(10) # 创建大小为 10 的哈希表

56.

57. # 插入键值对

58. ht.insert(10, "A")

59. ht.insert(20, "B")

60. ht.insert(30, "C")

61. ht.insert(15, "D") # 哈希冲突，使用线性探测

62. print("哈希表内容：")

63. print(ht)

64.

65. # 查找键

66. print("\n查找键 20 的值：", ht.search(20))

67. print("查找键 15 的值：", ht.search(15))

68.

69. # 删除键

70. ht.delete(20)

71. print("\n删除键 20 后的哈希表内容：")

72. print(ht)

## 3.4 实验任务4

请参考Hello算法，11.10基数排序 <https://www.hello-algo.com/chapter_sorting/radix_sort/> 实现基数排序。数据请自动生成，数据格式以我校学号为例，如202211672200；其中前4位为入学年份；中间四位为学院与专业；后四位为入学年份（两位为班级序号，如21为1班，22为2班，23为3班等）及班级内部学号；请生成50000名学生的学号，并完成排序。理解基数排序思想。

基数排序是一种非比较型的排序算法，它通过将数据按位分配到不同的“桶”中，然后按位依次排序，最终得到有序序列。基数排序的核心思想是：​从最低位到最高位，依次对每一位进行排序。相比快速排序这种低时间复杂度的算法，基数排序适用于待排序数据位数较大的情况，将自然数拆成每一位进行排序。代码如下所示。

1. import random

2.

3.

4. def generate\_student\_ids(num\_students=50000):

5. """生成指定数量的学生学号"""

6. student\_ids = []

7. for \_ in range(num\_students):

8. year = random.randint(2020, 2023) # 入学年份：2020-2023

9. dept = random.randint(1000, 9999) # 学院与专业：1000-9999

10. class\_num = random.randint(1, 3) # 班级序号：1-3

11. student\_num = random.randint(1, 99) # 班级内部学号：1-99

12. # 格式化学号：入学年份（4位） + 学院与专业（4位） + 班级序号（2位） + 班级内部学号（2位）

13. student\_id = f"{year}{dept:04d}{class\_num:02d}{student\_num:02d}"

14. student\_ids.append(student\_id)

15. return student\_ids

16.

17.

18. def radix\_sort(arr):

19. """基数排序"""

20. max\_length = len(max(arr, key=len)) # 获取最长学号的长度

21. for i in range(max\_length - 1, -1, -1): # 从最低位到最高位依次排序

22. buckets = [[] for \_ in range(10)] # 初始化10个桶（0-9）

23. for num in arr:

24. # 获取当前位的数字，如果不足则补0

25. digit = int(num[i]) if i < len(num) else 0

26. buckets[digit].append(num) # 将数字放入对应的桶中

27. arr = [num for bucket in buckets for num in bucket] # 将桶中的数字按顺序合并

28. return arr

29.

32. if \_\_name\_\_ == "\_\_main\_\_":

33. # 生成50000个学生学号

34. student\_ids = generate\_student\_ids(50000)

35. print("前10个未排序的学号：", student\_ids[:10])

36.

37. # 基数排序

38. sorted\_student\_ids = radix\_sort(student\_ids)

39. print("\n前10个排序后的学号：", sorted\_student\_ids[:10])

40.

41. # 检查是否有序

42. is\_sorted = all(sorted\_student\_ids[i] <= sorted\_student\_ids[i + 1] for i in range(len(sorted\_student\_ids) - 1))

43. print("\n学号是否有序：", is\_sorted)

## 3.5 实验任务5

LeetCode 27 移除元素

<https://leetcode.cn/problems/remove-element/description/>

1. class Solution:

2. def removeElement(self, nums: List[int], val: int) -> int:

3. slow = -1

4. for fast in range(0, len(nums)):

5. if nums[fast] != val:

6. slow += 1

7. nums[slow] = nums[fast]

8. return slow + 1

如果不调用库函数去做，就用双指针的思路。以下是库函数的代码，用nums[:]而不是nums，是因为需要原地操作。

1. class Solution:

2. def removeElement(self, nums: List[int], val: int) -> int:

3. nums[:] = [x for x in nums if x != val]

4. return len(nums)

LeetCode 206 反转链表

<https://leetcode.cn/problems/reverse-linked-list/description/>

1. class Solution:

2. def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:

3. if head == None:

4. return

5. cur, pre = head, None

6. while cur:

7. temp = cur.next

8. cur.next = pre

9. pre = cur

10. cur = temp

11. return pre

反转链表和删除链表指定元素不同的点是，前者需要双指针，但是后者只需要一个指针去遍历即可。在这道题中，由于需要寻找下一个节点，所以需要一个临时变量temp去过渡。

LeetCode 26 删除有序数组中的重复项

<https://leetcode.cn/problems/remove-duplicates-from-sorted-array/description/>

1. class Solution:

2. def removeDuplicates(self, nums: List[int]) -> int:

3. slow = 0

4. for fast in range(1, len(nums)):

5. if nums[slow] != nums[fast]:

6. slow += 1

7. nums[slow] = nums[fast]

8. return slow + 1

这道题用双指针完成非常简洁，之所以fast从1开始遍历，是因为要比较重复性，所以是slow的后一个开始的。先slow+1再赋值，这是因为考虑的是相邻元素。

LeetCode 1700 无法吃午餐的学生数量

<https://leetcode.cn/problems/number-of-students-unable-to-eat-lunch/description/>

1. class Solution:

2. def countStudents(self, students: List[int], sandwiches: List[int]) -> int:

3. from collections import deque

4. students = deque(students)

5. sandwiches = deque(sandwiches)

6. while students and sandwiches:

7. if students[0] == sandwiches[0]:

8. students.popleft()

9. sandwiches.popleft()

10. else:

11. temp = students.popleft()

12. students.append(temp)

13. if sandwiches and sandwiches[0] not in students:

14. break

15. return len(students)

这道题只需用来队列的弹出插入操作，和一点点模拟的思想就可以了。为了使用deque模块的popleft和append，需要先把两个列表进行队列的转换。需要注意的是，需要在循环里面检查sandwiches是不是没有学生想要的三明治了，没有的话就退出循环，如果不是就一直循环。

LeetCode 1480 一维数组的动态和

<https://leetcode.cn/problems/running-sum-of-1d-array/description/>

1. class Solution:

2. def runningSum(self, nums: List[int]) -> List[int]:

3. slow = 0

4. results = [nums[0]] \* len(nums)

5. for fast in range(1,len(nums)):

6. results[fast] = results[slow] + nums[fast]

7. slow += 1

8. return results

很直观的双指针题目，用来练习很不错。

## 3.6 实验任务6

双指针类型：

1. 同向双指针：两个指针都是从同一个方向开始移动的，用来解决删除元素和滑动窗口等问题；
2. 双向双指针：两个指针分别从两端向中间移动，用来解决有序数组的查找和回文判断等问题；
3. 快慢指针：两个指针从同一端开始，但移动速度不同，用来解决链表节点相关的问题。

哈希表的理解：

对于Python语言，哈希表就是字典的体现，通过键值对的映射关系可以高效存储数据和快速解决问题。比如查找问题，哈希表的查找直接查找键（元素）对应的值（索引）就可以了，时间复杂度是O(1)。除此之外，对于Python而言，有现成的结构使用，不需要另外编写代码实现数据结构。

# 实验总结

我在这次实验的收获非常大，我学习并通过实践掌握了插入排序和基数排序，排序的思路从自然数扩展到位的排序，这对于高位数的排序非常有效。除此之外，我通过几道老师布置的题目，掌握了双指针这一思想，在链表、数组、队列等问题类型中进一步巩固。

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