

《数据结构》

实验报告

姓名：

学号：

成绩：

电子与计算机工程学院

School of Electronic & Computer Engineering

2023年12月

【实验名称】查找算法的实现

【实验目的】

一、掌握顺序查找算法的实现。

二、掌握折半查找算法的实现。

三、掌握哈希表、哈希函数与哈希冲突的概念，掌握哈希表的构造方法和哈希表查找算法的实现。

【实验内容】

1. 实验要求：

1．仔细阅读查找实验代码，补充下面4个函数的实现过程。补充完成main（）函数的定义， 给出测试结果。

（1）创建查找表

void CreateTable(SSTable &ST) { }

（2）输出查找表元素

void Traverse(SSTable ST){ }

（3）在顺序表ST中顺序查找等于key的数据元素。若找到，则返回该元素在表中的位置，否则返回0。

int Search\_Seq(SSTable ST,ElemType key){ }

（4）在有序表ST中折半查找等于key的数据元素。若找到，则返回该元素在表中的位置，否则返回0。

int Search\_Bin(SSTable ST,ElemType key) { }

2. 阅读并理解哈希表实验代码，给出详尽的注释，调试程序，给出运行结果，并进行分析。

3．二分查找的拓展（五选一）

（1）搜索二维矩阵 II（LeetCode240）

编写一个高效的算法来搜索 m x n 矩阵 matrix 中的一个目标值 target 。该矩阵具有以下特性：

• 每行的元素从左到右升序排列。

• 每列的元素从上到下升序排列。

示例 1：



输入：matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 5

输出：true

示例 2：



输入：matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 20

输出：false

提示：

• m == matrix.length

• n == matrix[i].length

• 1 <= n, m <= 300

• -109 <= matrix[i][j] <= 109

• 每行的所有元素从左到右升序排列

• 每列的所有元素从上到下升序排列

• -109 <= target <= 109

（2）搜索插入位置（LeetCode35）

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。请必须使用时间复杂度为 O(log n) 的算法。

示例 1:

输入: nums = [1,3,5,6], target = 5

输出: 2

示例 2:

输入: nums = [1,3,5,6], target = 2

输出: 1

示例 3:

输入: nums = [1,3,5,6], target = 7

输出: 4

提示:

1 <= nums.length <= 104

-104 <= nums[i] <= 104

nums 为 无重复元素 的 升序 排列数组

-104 <= target <= 104

（3）在排序数组中查找元素的第一个和最后一个位置（LeetCode34）

给你一个按照非递减顺序排列的整数数组 nums，和一个目标值 target。请你找出给定目标值在数组中的开始位置和结束位置。如果数组中不存在目标值 target，返回 [-1, -1]。

你必须设计并实现时间复杂度为 O(log n) 的算法解决此问题。

示例 1：

输入：nums = [5,7,7,8,8,10], target = 8

输出：[3,4]

示例 2：

输入：nums = [5,7,7,8,8,10], target = 6

输出：[-1,-1]

示例 3：

输入：nums = [], target = 0

输出：[-1,-1]

提示：

0 <= nums.length <= 105

-109 <= nums[i] <= 109

nums 是一个非递减数组

-109 <= target <= 109

（4）搜索旋转排序数组（LeetCode33）

整数数组 nums 按升序排列，数组中的值互不相同 。

在传递给函数之前，nums 在预先未知的某个下标 k（0 <= k < nums.length）上进行了 旋转，使数组变为 [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]（下标 从 0 开始 计数）。例如， [0,1,2,4,5,6,7] 在下标 3 处经旋转后可能变为 [4,5,6,7,0,1,2] 。

给你旋转后的数组 nums 和一个整数 target ，如果 nums 中存在这个目标值 target ，则返回它的下标，否则返回 -1 。你必须设计一个时间复杂度为 O(log n) 的算法解决此问题。

示例 1：

输入：nums = [4,5,6,7,0,1,2], target = 0

输出：4

示例 2：

输入：nums = [4,5,6,7,0,1,2], target = 3

输出：-1

示例 3：

输入：nums = [1], target = 0

输出：-1

提示：

1 <= nums.length <= 5000

-104 <= nums[i] <= 104

nums 中的每个值都 独一无二

题目数据保证 nums 在预先未知的某个下标上进行了旋转

-104 <= target <= 104

（5）搜索旋转排序数组II（LeetCode81）

在（4）基础上修改条件，数组数据允许有重复值

4.选作题目

利用平衡二叉树实现一个动态查找表，实现动态查找表的三种基本功能：查找、插入和删除。

1. 程序清单

1.

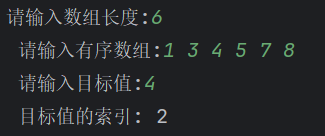
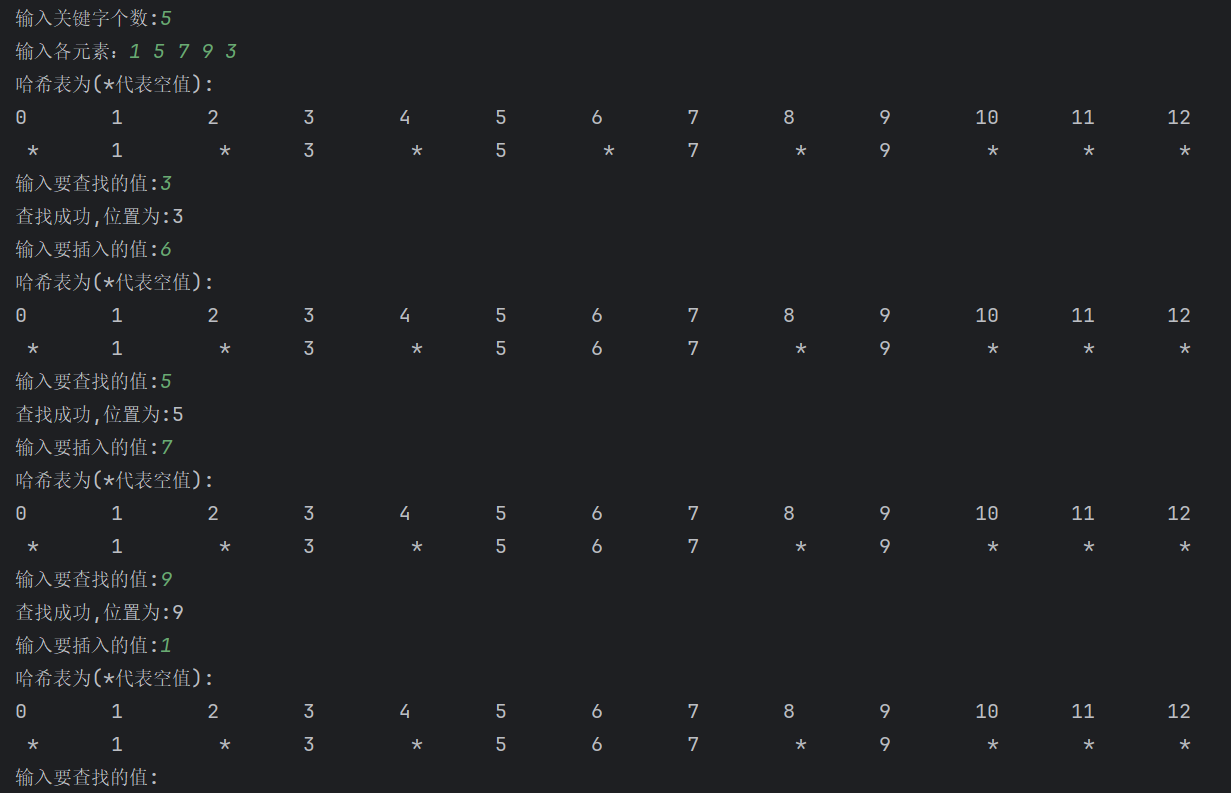
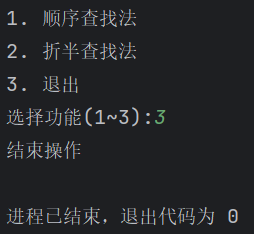
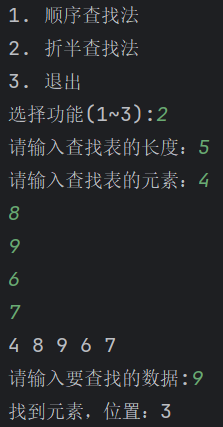
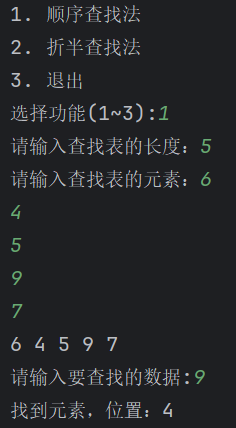
1. #include<iostream>
2. #include <stdlib.h>
3. using namespace std;
4. *//顺序查找表的定义*
5. typedef int ElemType;
6. typedef struct{
7. ElemType \*elem;   *//数据元素存储空间基址，0号单元留空*
8. int length;       *//表长度*
9. }SSTable;
10. *//创建查找表*
11. void CreateTable(SSTable &ST) {
12. cout << "请输入查找表的长度：";
13. cin >> ST.length;
14. ST.elem = new ElemType[ST.length + 1]; *// 创建存储空间*
15. cout << "请输入查找表的元素：";
16. for (int i = 1; i <= ST.length; i++) {
17. cin >> ST.elem[i];
18. }
19. }
20. *//输出查找表元素*
21. void Traverse(SSTable ST)
22. {
23. for (int i = 1; i <= ST.length; i++) {
24. cout << ST.elem[i] << " ";
25. }
26. cout << endl;
27. }
28. *//在顺序表ST中顺序查找等于key的数据元素。*
29. *//若找到，则返回该元素在表中的位置，否则返回0*
30. int Search\_Seq(SSTable ST, ElemType key){
31. for (int i = 1; i <= ST.length; i++) {
32. if (ST.elem[i] == key) {
33. return i;
34. }
35. }
36. return 0;
37. }
38. *//在有序表ST中折半查找等于key的数据元素。*
39. *//若找到，则返回该元素在表中的位置，否则返回0*
40. int Search\_Bin(SSTable ST, ElemType key) {
41. int low = 1, high = ST.length, mid;
42. while (low <= high) {
43. mid = (low + high) / 2;
44. if (ST.elem[mid] == key) {
45. return mid;
46. }
47. else if (ST.elem[mid] < key) {
48. low = mid + 1;
49. }
50. else {
51. high = mid - 1;
52. }
53. }
54. return 0;
55. }
56. int main(void)
57. {
58. SSTable ST;
59. int c = 0;
60. ElemType key;
61. while (c != 3)
62. {
63. cout << endl << "1. 顺序查找法";
64. cout << endl << "2. 折半查找法";
65. cout << endl << "3. 退出";
66. cout << endl << "选择功能(1~3):";
67. cin >> c;
68. switch (c)
69. {
70. case 1:
71. CreateTable(ST);
72. Traverse(ST);
73. cout << "请输入要查找的数据:";
74. cin >> key;
75. if (int pos = Search\_Seq(ST, key)) {
76. cout << "找到元素，位置：" << pos << endl;
77. }
78. else {
79. cout << "未找到元素" << endl;
80. }
81. break;
82. case 2:
83. CreateTable(ST);
84. Traverse(ST);
85. cout << "请输入要查找的数据:";
86. cin >> key;
87. if (int pos = Search\_Bin(ST, key)) {
88. cout << "找到元素，位置：" << pos << endl;
89. }
90. else {
91. cout << "未找到元素" << endl;
92. }
93. break;
94. case 3:
95. cout<<"结束操作"<<endl;
96. break;
97. }
98. }
99. delete [] ST.elem; *// 释放存储空间*
100. }

2.

1. #include <iostream>
2. #include <stdlib.h>
3. #define HASHSIZE 13  *// 定义哈希表长*
4. #define M 13  *// 定义除留余数法中的质因数*
5. #define SUCCESS 1  *// 查找成功*
6. #define UNSUCCESS 0  *// 查找失败*
7. #define DUPLICATE -1  *// 表中已有与e有相同关键字的元素*
8. using namespace std;
9. typedef int Keytype;
10. typedef struct {   *// 数据元素定义*
11. Keytype key;
12. int other;
13. } Elemtype;
14. typedef struct {   *// 哈希表定义*
15. Elemtype r[HASHSIZE];  *// 哈希表使用一个Elemtype类型的数组来保存数据*
16. int length;  *// 哈希表中元素的个数*
17. } Hashtable;
18. int p;  *// 哈希地址*
19. int Hash(int k)    *// 计算哈希地址的函数*
20. {
21. p = k % M;
22. return p;
23. }
24. int SearchHash(Hashtable H, Keytype K) *// 查找函数*
25. {
26. p = Hash(K);  *// 根据关键字计算哈希地址*
27. while (H.r[p].key != NULL && K != H.r[p].key)  *// 若哈希表该位置的关键字不为空且不等于要查找的关键字*
28. p = (p + 1) % M;  *// 线性探测再散列，查找下一个位置*
29. if (K == H.r[p].key)  *// 如果找到了键值等于K的元素*
30. return SUCCESS;  *// 查找成功*
31. else
32. return UNSUCCESS;  *// 查找失败*
33. }
34. int InsertHash(Hashtable& H, Elemtype e) *// 插入函数*
35. {
36. if (SearchHash(H, e.key) == SUCCESS)  *// 首先先在哈希表中搜索是否已存在与e.key相同的元素*
37. return DUPLICATE;  *// 如果已存在，返回重复标志*
38. H.r[p].key = e.key;  *// 若不存在相同的元素，则在哈希地址为p的位置插入新元素*
39. ++H.length;  *// 哈希表长度加1*
40. }
41. void CreateHashTable(Hashtable& H, int n) *// 创建哈希表*
42. {
43. Elemtype e;
44. int flag;
45. cout << "输入各元素：";
46. while (n--)
47. {
48. cin >> e.key;  *// 输入关键字*
49. flag = InsertHash(H, e);  *// 调用插入函数进行插入操作*
50. if (flag == DUPLICATE)  *// 如果返回的标志为重复标志*
51. {
52. cout << "有重复值,重新输入\n";
53. n++;
54. }
55. }
56. }
57. void Show(Hashtable H, int n) *// 输出哈希表*
58. {
59. cout << "哈希表为(\*代表空值):\n";
60. for (int i = 0; i < HASHSIZE; i++)
61. cout << i << '\t';  *// 输出哈希表的列号*
62. cout << "\n";
63. for (int i = 0; i < HASHSIZE; i++)
64. {
65. if (H.r[i].key != NULL)
66. cout << H.r[i].key << '\t';  *// 输出哈希表中的关键字值*
67. else
68. cout << " \* " << '\t';  *// 输出空值标记*
69. }
70. cout << "\n";
71. }
72. int main()
73. {
74. int n;
75. Keytype e;
76. Hashtable H;
77. Elemtype x;
78. for (int i = 0; i < HASHSIZE; i++)
79. H.r[i].key = NULL;  *// 初始化哈希表的关键字为空*
80. cout << "输入关键字个数:";
81. cin >> n;
82. CreateHashTable(H, n);  *// 创建哈希表*
83. Show(H, n);  *// 显示哈希表*
84. while (1)
85. {
86. cout << "输入要查找的值:";
87. cin >> e;
88. if (SearchHash(H, e))
89. cout << "查找成功,位置为:" << p << "\n";  *// 查找成功*
90. else
91. cout << "查找失败\n";  *// 查找失败*
92. cout << "输入要插入的值:";
93. cin >> x.key;
94. InsertHash(H, x);
95. Show(H, n);
96. }
97. }

3.

1. #include <iostream>
2. #include <vector>
3. using namespace std;
4. int searchInsert(vector<int>& nums, int target) {
5. int left = 0, right = nums.size() - 1;
6. int mid;
7. while (left <= right) {  *// 进行二分查找*
8. mid = left + (right - left) / 2;
9. if (nums[mid] == target) {  *// 目标值在数组中*
10. return mid;
11. } else if (nums[mid] < target) {
12. left = mid + 1;
13. } else {
14. right = mid - 1;
15. }
16. }
17. *// 目标值不在数组中，返回应该插入的位置*
18. return left;
19. }
20. int main() {
21. vector<int> nums;
22. int numCount, target;
23. cout << "请输入数组长度: ";
24. cin >> numCount;
25. nums.reserve(numCount);
26. cout << "请输入有序数组: ";
27. for (int i = 0; i < numCount; i++) {
28. int num;
29. cin >> num;
30. nums.push\_back(num);
31. }
32. cout << "请输入目标值: ";
33. cin >> target;
34. int index = searchInsert(nums, target);
35. cout << "目标值的索引: " << index << endl;
36. return 0;
37. }
38. 结果截图



【实验体会】