

数据结构

实验报告（七）

折半查找

学号：3020205015

姓名：石云天

班级：智能机器平台 2 班

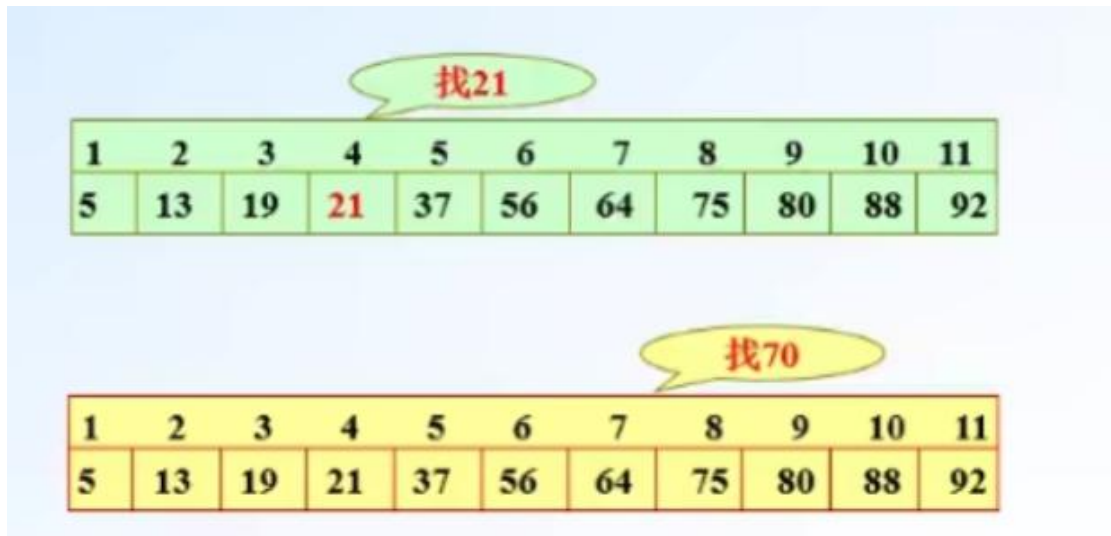
日期：2022.12.17

目 录

一、实验内容描述	3
二、实验步骤	3
三、程序设计	4
(一) 抽象数据类型 ADT	4
(二) 算法简述	4
(三) 程序代码	5
四、调试分析	5
(一) 调试过程和主要错误	5
(二) 时间复杂度	5
五、程序测试	6
六、实验总结	6
附录 1: 程序源代码: BinarySearch	7

一、实验内容描述

本次实验主题是折半查找算法，实现后在图 1 的样例上进行测试：



1	2	3	4	5	6	7	8	9	10	11
5	13	19	21	37	56	64	75	80	88	92

1	2	3	4	5	6	7	8	9	10	11
5	13	19	21	37	56	64	75	80	88	92

图 1 实验要求的测试样例

二、实验步骤

- (1) 根据上课所讲，回顾折半查找的基本概念，认识到折半查找算法实质上是插入排序的一种改进。选取顺序存储结构作为数据结构，利用数组进行存储。
- (2) 仔细阅读实验要求，考虑折半查找算法的核心思想与实现方法。
- (3) 利用 CodeBlocks 编译器，配置环境，基于 C++ 语言将算法用程序实现。
- (4) 编译运行程序，使用样例进行程序测试，观察所编程序是否实现要求的功能。
- (5) 考察算法的时间复杂度和空间复杂度，评价算法的优劣，进一步优化程序。
- (6) 撰写实验报告，进行实验总结与反思。

三、程序设计

(一) 抽象数据类型 ADT

查找表的抽象数据类型:

ADT SSTable{

数据对象: $D = \{a_i | a_i \in \text{int}, i = 1, 2, 3, \dots\}$

数据关系: $R = \{ \langle a_i, a_{i+1} \rangle | a_i, a_{i+1} \in D, i = 1, 2, 3, \dots \}$

基本操作:

SSTable(int size = DEFAULT_SIZE)

操作结果: 根据数据量构造查找表, 默认数据量为 **DEFAULT_SIZE(10)**

SSTable(KeyType *data, int size)

操作结果: 从数组中构造查找表, 数组大小为 **size**

~SSTable()

操作结果: 销毁查找表, 释放存储空间

int GetSize()

操作结果: 返回查找表中的数据量

void PrintTable()

操作结果: 打印查找表内的数据

KeyType &operator[] (int index)

操作结果: 重构下标运算符, 可通过下标运算符直接访问查找表中的数据

}

(二) 算法简述

在进行折半查找前, 首先要对数组中的元素进行排序。因此将数组中的元素放到查找表中。查找表会调用 `sort` 函数对元素进行排序。排序完成后即可进行折半查找操作。

折半查找的具体实现方式是, 首先定义最小下标和最大下标, 表示当前查找的范围。随后计算最小下标和最大下标的平均值(向下取整), 比较以该平均值为下标的数据与需要查找的数据的大小。如果相同则返回该平均值。如果需要查找的数据较小, 那么将新的最大下标设为平均值减 1, 表示新的查找范围在左半部分。如果需要查找的数据较大, 那么将想不到最小下标设为平均值加 1, 表示新的查找范围在右半部分。循环上述操作, 直至最小下标大于最大下标, 如果还未查找到, 则说明查找表中没有需要查找的数据, 返回-1, 具体流程可参考下图 2:

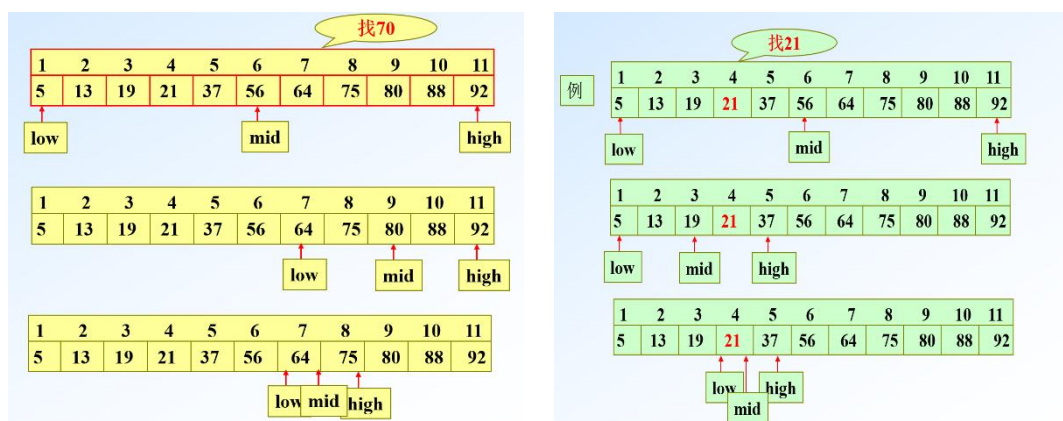


图 2 折半查找算法示意图

(三) 程序代码

为保证实验报告的清晰和可读性，将源代码以附录形式附于文末。

四、调试分析

(一) 调试过程和主要错误

在程序编写完成后，使用 CodeBlocks 编译并运行，基本没有出现问题。最初由于空指针的问题没有注意，编译器报错 Segmentation Fault，仔细检查代码后，经过适当修改，程序顺利运行。

(二) 时间复杂度

与传统查找算法相比，折半查找降低了时间复杂度，假设有序表长度为 $n = 2h - 1$ （则 $h = \log_2(n + 1)$ ），则用于描述折半查找的判定树是深度为 h 的满二叉树。树中有 1 个层次为 1 的节点，2 个层次为 2 的点， $2h - 1$ 个层次为 h 的点。

五、程序测试

在完成全部程序编写后，输入测试样例进行测试，所得结果均满足要求。
BinarySearch 算法测试结果见图 3。



图 3 BinarySearch 算法测试结果

六、实验总结

通过这次试验，我发现我对查找这一部分的理解不够深入全面，需要不断巩固学习，加深理解。同时。在编程过程中需要完成某些特定目标时，我不能很快的想出其对应的操作，需要课下不断练习以熟能生巧，还可以多查阅一些资料以开阔自己的思路。在本次实验中，我编写并实现了折半查找算法，在此过程中不断调试，寻找问题，并不断简化代码，提升函数执行速度。除此之外，在本次实验过程中，编写、调试程序花费了很长时间：首先是折半查找要求必须在顺序存储的有序表中才能适用，其他是不可以的。其次对于空指针的理解更加深刻，我最初以为 Segmentation Fault 类型报错是因为数组越界，经过不断查阅资料发现是因为调用了空指针，空指针不会指向任何实体，因此在程序编写过程中需要格外注意各指针变量指向的变化，在 delete 操作完成后，最好在后面加一行将指针置为 NULL 的代码，这可以有效避免调用空指针的错误。最后是关于结果部分不正确的情况，这一问题往往是比较棘手的，此时需要在边界条件上入手，寻找没有关注到的情况，让思考更加全面周到，有助于顺利解决问题。在本次实验后，我还需要精益求精，不断改进程序，优化函数性能，实现预期目标与功能所需。

附录 1：程序源代码： BinarySearch

```
#include<bits/stdc++.h>

using namespace std;

int BinarySearch(int arr[],int length,int key)
{
    int low = 0;//定义初始最小
    int high = length - 1;//定义初始最大
    int mid;//定义中间值
    while(low <= high)
    {
        mid = (low + high) / 2;//找中间值
        if(key == arr[mid])//判断 min 与 key 是否相等
            return mid;
        else if(key > arr[mid])//如果 key>mid 新区间为[mid+1,high]
            low = mid + 1;
        else//如果 key<mid 新区间为[liw,mid-1]
            high = mid - 1;
    }
    return -1;//如果数组中无目标值 key,则返回-1;
}

int main()
{
    cout << "测试一 " << endl;
    int arr[11] = {5, 13, 19, 21, 37, 56, 64, 75, 80, 88, 92};
    cout << BinarySearch(arr,(sizeof(arr) / sizeof(arr[0])),21);
    cout << endl;
    cout << "测试二 " << endl;
    cout << BinarySearch(arr,(sizeof(arr) / sizeof(arr[0])),70);
}
```