《数据结构》课程实践报告

院、系	计算机学院	年级专业	21 计算机科学与技术	姓名	赵鹏	学号	2127405037
实验布置日期	2022.10.25		提交日期	2022.11.5		成绩	

课程实践实验六：查找算法的实现及性能测试与比较

一、问题描述及要求

在顺序线性表中存放 n 个整数，n 的值由用户输入确定，线性表可以是有序表或无序表。

比较各查找算法在不同情况下的时间性能。

各查找算法的实测时间性能包括两个指标:算法执行的绝对时间和关键字的平均比较次数。

各查找算法要求评测查找成功与不成功的两种情形。

为了能比较出各种查找算法执行的绝对时间，需要对表中的数据进行较大量的查找，设为 m 次，m 的值也由用户输入确定。当输入 m 为 1000000 时，则对线性表作 1000000 次查找。

(1) 比较在有序表和无序表中进行顺序查找时，查找成功和查找失败时的算法执行的绝对时间和关键字的平均比较次数。

(2) 比较在同一有序表中进行顺序查找和二分查找时的时间性能。

(3) 比较在同一有序表中进行非递归二分查找和递归二分查找的时间性能。

二、概要设计

1.问题分析

本次实验的主要内容是实现顺序查找、递归二分查找和非递归二分查找三种算法，并测试其平均绝对查找时间和平均比较次数来比较其性能。

2.系统功能

本次实验为了实现测试三种查找算法的性能实现了以下的功能:

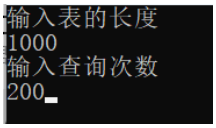
1．根据用户输入的表长生成有序表和无序表，用于测试顺序查找在有序表和无序表中的查找性能。

2．实现顺序查找、二分查找、非递归二分查找三种查找算法，并计算比较次数和绝对运行时间，

3．设计主函数，测试三种查找算法在查找成功和查找失败是的绝对运行时间和平均比较次数。

3.运行界面设计

本次程序的运行界面较为简单，与用户进行的交互较少。仅需要输入表长和查询次数即可。如下图所示：



4.总体设计思路

为了实现查找功能，需要对顺序表类添加三个 public 的查找函数。对为了实现递归二分查找，需要额外添加 private 类型的递归函数。

为了实现成功查找和失败查找的情况，在生成表中选择只生成一定 0,2,4,6,8…的偶数，用偶数模拟查找成功，用奇数模拟查找失败，对于给定的查找次数 m，将其均分到 n 个元素中即可，如 n=15,m=3。成功查找时查找 0,10,20 三个数，失败查找时查找 1,11,21 三个数。以此实现平均。

为实现程序所需要的功能，在实现过程中设计了以下类及方法：

1. **SearchList**

类方法	功能
SearchList(int size,bool unordered)	生成指定长度的有序/无序 顺序表
sequenceSearch(int target,Info &info)	顺序查找，并记录查找信息
binarySearchRecursive(int target,Info &info)	递归二分查找并记录查找信息(主调方法)
binarySearchNonRecursive(int target, Info& info)	非递归二分查找并记录查找信息
binaryRecursive(int low, int high, int target, Info& info)	递归二分查找函数(递归函数)

2. **Timer**

类方法	功能
Timer()	计时器类的构造函数，记录生成时间
Runtime()	返回当前时间和生成时间之差(s)

5.程序结构设计

为使得程序的结构清晰，在实现过程中设计了以下文件：

文件名	功能
SearchList.h	声明支持三种查找算法的顺序表类
SearchList.cpp	实现顺序表类
Timer.hpp	声明并实现计时器类
main.cpp	测试各种查找在不同情况下的性能

其中，SearchList.h 额外声明了 Info 类型，用于存储查找的绝对运行时间和比较次数。在查找过程中通过引用传入 Info 类型变量，统计查找时的比较次数。

Timer 类用于在测试时统计绝对运行时间并将其传递给 Info 类型的变量。

三、详细设计

本次实验的重点即为顺序表的生成和三种查找的的实现。下面依次叙述三种查找的实现。

1. 顺序表的生成

为了实现数据的存储，利用数组存储所有元素。元素的值由用户输入的 n 决定，对于有序表，则生成 0,2,4,6,8…的偶数，这样做便可以利用查找奇数实现查找失败的情况。对于无序表，只需在生成有序表后随机交换元素即可。为了便于操作，在构造函数中除了表长以外增加了一个 bool 类型的参数 unordered，用于表示是否生成无序表。

代码实现如下：

```
SearchList::SearchList(int _size, bool unordered)
{
    size = _size;
    for (int i = 0; i < size; i++)
        datas[i] = 2 * i;
    if (unordered)
    {
        std::srand(std::time(0));
        for (int i = 0; i < size; i++)
        {
            int p = std::rand() % size;
            std::swap(datas[p], datas[i]);
        }
    }
    }
```

2. 顺序查找

顺序查找即从前往后依次比较表中的每个元素，同时记录比较信息。遇到与目标值相等的元素直接返回索引即可。若遍历完都无法找到结果则返回无法找到。

代码实现如下：

```
int SearchList::sequenceSearch(int target, Info& info)
{
    Timer timer;
    for (int i = 0; i < size; ++i)
    {
        if (info.compare++,  target == datas[i])
        {
            info.time = timer.runtime();
            return i;
```

```
        }
    }
    info.time = timer.runtime();
    return NOT_FOUND;
}
```

3．递归二分查找

二分查找可以在有序表中以 log 的时间复杂度查找元素。递归二分查找首先取左右端点为 low=0 和 high=size-1。每次选取 low 和 high 的中间位置 mid=(low+high)>>1;若 datas[mid]=target 则直接返回。若 datas[mid]<target 则递归查找右边的部分。若 datas[mid]>target 则递归查找左边部分。需要注意的是，在类中递归二分查找需要设计两个函数:主调函数和递归函数。

代码实现如下:

```
int SearchList::binarySearchRecursive(int target, Info& info)
{
    Timer timer;
    int res = binaryRecursive(0, size - 1, target, info);
    info.time = timer.runtime();
    return res;
}
int SearchList::binaryRecursive(int low, int high, int target, Info& info)
{
    if (high < low)return NOT_FOUND;
    int mid = low + high >> 1;

    if (info.compare++,target < datas[mid])
    {
        return binaryRecursive(low, mid - 1, target, info);
    }
    else if (info.compare++, target > datas[mid])
    {
        return binaryRecursive(mid + 1, high, target, info);
    }
    return mid;
}
```

4.非递归二分查找

非递归二分查找的思路和递归二分查找类似。只是把递归改为 while 循环。把递归改变 low 和 high 的值改为在循环中修改 low 和 high 的值。

代码实现如下:

```
int SearchList::binarySearchNonRecursive(int target, Info& info)
{
    Timer timer;
    int low = 0, high = size - 1, mid;
    while (low <= high)
    {
        mid = low + high >> 1;
        if (info.compare++, target < datas[mid])
            high = mid - 1;
        else if (info.compare++, target > datas[mid])
            low = mid + 1;
        else
        {
            info.time = timer.runtime();
            return mid;
        }
    }
    info.time = timer.runtime();
    return NOT_FOUND;
}
```

5.测试设计

由于顺序表在生成时生成的为 0,2,4,6,8…的偶数，所以可以利用查询偶数代表测试成功。查询奇数测试查询失败。对于给定的 m，将 m 次查找均分到长度为 n 的数组中即可测试平均查找情况。

四、实验结果

对于序列长度为 50000，查询次数为 50000 的测试结果如下：

有序表成功顺序查找情况下:
平均查找时间：0.000036s
平均比较次数：26584

有序表失败顺序查找的情况下
平均查找时间：0.000068s
平均比较次数：49999

无序表成功顺序查找情况下:
平均查找时间：0.000049s
平均比较次数：36317

无序表失败顺序查找的情况下
平均查找时间：0.000068s
平均比较次数：49999

有序表成功递归二分查找情况下：
平均查找时间：0.0000001200s
平均比较次数：18

有序表的失败递归二分查找情况下：
平均查找时间：0.0000001600s
平均比较次数：19

有序表成功非递归二分查找情况下：
平均查找时间：0.0000000600s
平均比较次数：18

有序表失败非递归二分查找情况下：
平均查找时间：0.0000000600s
平均比较次数：18

五、实验分析与探讨

对于有序表和无序表的顺序查找。有序表的绝对运行时间和绝对比较次数差距不大。但在上述测试中略优于无序表。这与无序表的生成方法有着很大的关系，在数据量较小的情况下误差较大。

对于有序表二分查找的$0(\log_2 n)$的时间复杂度和$\log_2 n+1$的比较次数均由于顺序查找。因为二分查找每次将查找范围缩小了一半，极大的减少了比较次数。

对于有序表的递归二分查找和非递归二分查找，通过绝对运行时间可以看出非递归二分查找有着一定的优势。这是因为递归调用本身需要使用系统栈，每次分配函数内存以及栈都需要一定的时间。

六、小结

通过本次实验，我对查找算法有了更加深刻的理解。了解到了各种查找算法的实现方法以及优缺点。学会了在不同情况下选择不同的查找算法以获得最优的查找性能。