**广州大学学生实验报告**

**开课学院及实验室：**计算机学院/电子信息楼418A **2017年11月29日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学 院** | 计算机 | **年级/专业/班** | 软件171 | **姓名** | 谢金宏 | **学号** | 1706300001 |
| **实验课程名称** | 程序设计基础 | | | | | **成绩** |  |
| **实验项目名称** | **实验6程序设计基础应用实验** | | | | | **指导老师** | 张为 |

(\*\*\*（1）报告只能为文字和图片；（2）实验项目名称不能有任何改动；（3）必须填写正确的姓名和学号；否则会影响最后成绩的统计，后果自负\*\*\*)

**一、实验目的及要求**

1．培养学生将实际问题转化为数学模型再用C语言编程解决该问题的综合应用方法；

2．训练学生灵活掌握本课程所学的有关章节知识；

3．掌握基本的数据结构知识和更多的计算方法；

4． 掌握模块化程序设计的方法。

**二、实验设备与平台**

1. 实验设备：计算机；

2. 平台：Windows 10操作系统，GCC 7.2.0。

**三、实验内容及步骤**

**说明：三个实验题可任选其中两个完成。**

我选择完成实验1（折半查找法）与实验3（汉诺塔动画）。

**实验6\_1：**用折半查找法完成数据的查找工作。

（1）题目分析

折半查找法是一种效率较高的查找方法。

折半查找法的前提是：数据已按照一定的规律（升序或降序）排列好。

折半查找法的基本思路是：先检索居中的一个数据，看它是否为所需要的数据，如果不是，则判断要找的数据是在居中数的哪一边，下次就在这个范围内找，如下图：

ο ο \* ο ο ο ο ο ο ο ο ο ο ο ο

| | |

a c b

图 6\_1 折半查找法的前提

图 6\_1 中有15个数设按升序排列好。如果想找的是第3个数，由于它小于居中的那个数，因此必然在a – c 之间，这样就把b – c 区间排除在下次查找的范围之外了。第2次就在a – c 范围内按上法查找，依次类推，每次将查找的范围缩小一半。

假如有一组数为：8，13，21，28，35，41，52，63，71，76，81，95，101，150，164。要找21这个数。为了更直观，我们将这15个数画成如图 6\_2 所示的倒立的树形。“树”根在上面，“树枝”和“树叶”在下面，每一个数据称为一个“结点”。树跟结点是15个数中居中的那个数（63），它有两个“子结点”，左边的比它小，右边的比它大，以下逐级都是按这个规律（有关“树”的知识在“数据结构”教材中有详细的介绍）。可以按以下步骤查找：

63

28 95

13 41 76 150

8 21 35 52 71 81 101 164

图 6\_2 查找子结点

1）先将要找的数21与数列中的“树跟结点”比较，由于21 <63，因此，21显然在63左面的范围内，而且是1 – 7个数之间的一个数（由于21<63，所以已知道不可能是第8个数63）。

2）再找第1 – 7个数中居中位置的一个数（第4个数，即28），将21与28相比，由于21<28，因此，所找的数必然在1 – 3个数的范围内。

3）再找1 – 3个数中居中位置的一个数13（即第2个数），即将21与13相比，由于21>13,因此，21必然在13的右面。

4）由于21下面不在有分支，因此下一次就直接找到21。

以上一共查找了四次（即树的深度或树的层次是4），就找到了所需的数。

解题的具体方法如下：设有三个变量top、mid和bot分别指向数列的开头、中间和末尾，如果用a作为数组名，则top、mid、bot的值分别是a[0]、a[n/2]、a[n]的下标，即0、n/2、n。见图 6\_3 。

8 13 21 28 35 41 52 63 71 76 81 95 101 150 164

① top mid bot

② top mid bot

③ top mid bot

④ top mid bot

图 6\_3 子结点查找步骤

1)开始时令 top=0，bot=14，mid=7，见图 6\_3 中①。由于x<a[mid]，故确定x应在a[0] – a[6]范围内。

2）令新的bot=mid-1=6，算出mid=3,见图 6\_3 中②。

3）由于x<a[mid]，令新的bot=mid-1=2，即x应在a[0] – a[2]之中，算出mid=1，见图 6\_3 中③。

4）由于x>a[mid]，故令新的top=mid+1，x应在a[top] – a[bot]之间，今top=bot=mid，判断x=a[mid]，找到了21这个数，见图 6\_3中④。

如果要找的数不是数列中的数，例如，x=23,当第4次做判断时，x>a[mid]，按照以上规律，令top=mid+1，即top=3，出现top>bot的情况，也就是top跳到bot右面去了，而x理应在top与bot位置之间，由于top>bot，所以x不可能出现。

（2）流程图（N – S图）

见图 6\_4所示。

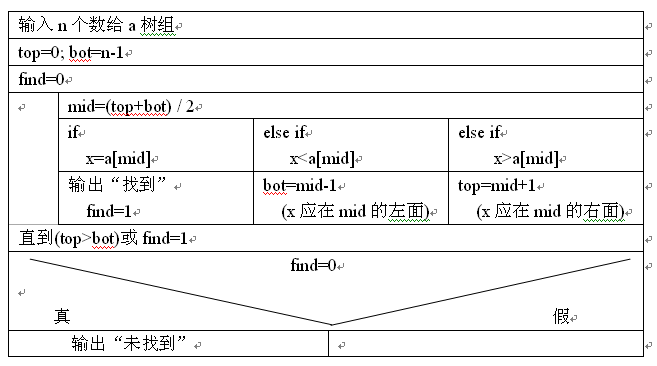


图 6\_4 流程图（N – S图）

(3) 源程序

//sy6\_1.cpp

#include <iostream>

void binary\_search(int a[], int n, int x)

{

    int mid, top, bot, find;

    top = 0;

    bot = n-1;

    find = 0;

    do

    {

        mid = (top + bot) / 2;

        if(x == a[mid])

        {

            std::cout << "find: " << x << ", it is " << mid + 1 << "th.";

            find = 1;

        }

        else if(x < a[mid]) bot = mid - 1;

        else if(x > a[mid])

            top = mid+1;

    }

    while((top<=bot) && (find==0));

    if(find == 0) std::cout << x << " is not found.";

}

int main()

{

    int a[100], x, i;

    int n = 15;

    std::cout << "input the numbers:";

    for(i = 0; i < n; i++)

        std::cin >> a[i];

    std::cout << "input x to look for:";

    std::cin >> x;

    binary\_search(a, n, x);

}

（4）运行程序

1）仔细阅读源程序，然后将你认为正确的答案写在下面。

源程序很好地实现了二分搜索。

此外，原始程序中由部分拼写和语法错误已被改正。

2）将源程序输入到计算机，进行编辑、编译、连接和运行，然后将运行结果记录在下面。

**第一次执行：**

input the numbers:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

input x to look for:16

16 is not found.

第二次执行：

**第一次执行：**

input the numbers:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

input x to look for:3

find: 3, it is 3th.

第三次执行：

**第一次执行：**

input the numbers:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

input x to look for:14

find: 14, it is 14th.

（5）结果分析

1）比较分析：将人工读题的答案、上机运行结果以及参考答案进行比较，在结论处填写“正确”或“不正确”。如果不正确则重新修改运行，直到正确为止。并记录出错原因的修改方法。

结论：正确

（6）重点提问

1）请注意，在程序中的 do…while 循环中，while 后面的表达式的写法和N-S 图中的循环表示方法的关系。在N-S 图中“直到top>bot或find=1”，意味着top>bot 或find=1两者中有一个成立就终止循环，在do…while循环中应该表示为：“当top<bot和find=0”时继续执行循环，两者中有一个不满足就不再执行循环。请特别注意“或”和“与”的用法。分析程序中使用循环条件(top <= bot)&&(find = = 0)的原因是什么？

如果没有找到元素(find == 0)并且还存在区间可以搜索(top <= bot)，就继续循环。

2）该题是在15个数中查找，如果使程序能完成在任意n个数中查找，应如何修改程序？

// 只需要对main()函数稍加改动

int main()

{

int a[100], x, i, n;

std::cout << "how many elements there are in total? ";

std::cin >> n;

std::cout << "input the numbers:";

for(i = 0; i < n; i++)

std::cin >> a[i];

std::cout << "input x to look for:";

std::cin >> x;

binary\_search(a, n, x);

}

3）在程序中使用下标访问数组，其实还可以用指针变量访问数组完成编程，只要将函数binare\_search() 稍作修改即可，主函数不变。首先将top、mid、bot改为指针类型，即 \*top、\*mid、\*bot，然后top的初值为a，bot 的初值为a+(n-1)，mid的值为：mid=top+(bot-top)/2。修改后的binare\_search 函数如下：

// 拼写错误已被修正

void binary\_search(int a[], int n, int x)

{

    int \*mid, \*top, \*bot, find;

    top = a;

    bot = a + (n - 1);

    find = 0;

    do

    {

//! mid = (top + bot) / 2;

        mid = top + (bot - top) / 2;

        if(x == \*mid)

        {

            std::cout << "find: " << x << ", it is " << mid - a + 1 << "th.";

            find = 1;

        }

        else if(x < \*mid) bot = mid - 1;

        else if(x > \*mid) top = mid + 1;

    }

    while ((top <= bot) && (find == 0));

    if (find==0) std::cout << x << " is not found.";

}

请上机调试，看运行效果是否一样？并认真体会两种方法的区别是什么？

运行效果相同。

两种方法思想相同，只是具体实现不同。注意到在寻找中间元素时，不能直接使用指针来相加。

**实验6\_2：** 动态分配内存方法

实验6\_2．利用动态分配内存方法，实现高精度整数相加。

未选做此实验。

**实验6\_3：**汉诺塔问题

图6\_5是汉诺塔的初始状态，要将若干个从小到大的园盘从A柱移动到C柱，见图6\_6 汉诺塔完成状态。

**A**

**B**

**C**

图6\_5 汉诺塔初始状态

要求每次只能移动一片，且不能将大片压在小片上，移动时可以借助第三根柱暂时存放。

**A**

**B**

**C**

图6\_6 汉诺塔完成状态

要求编写程序模拟该过程。

/\* 汉诺塔“动画” \*/

#include <iostream>

#include <cstdio>

#include <vector>

using namespace std;

void show();

void move(int, int, int, int);

struct Tower

{

vector<int> disks;

char name;

Tower(char name)

{

this->name = name;

}

operator >> (Tower &oth) // 不能写成(Tower oth)

{

int tmp = disks.back();

oth.disks.push\_back(tmp);

disks.pop\_back();

cout << name << " --> " << oth.name << endl;

show();

}

} towers[] = {Tower('A'), Tower('B'), Tower('C')};

void show()

{

for (int i = 0; i < 3; i++)

{

cout << towers[i].name << ": ";

vector<int>::iterator it = towers[i].disks.begin();

while (it != towers[i].disks.end())

{

cout << \*it << " ";

it++;

}

cout << endl;

}

cout << endl;

}

void move(int num, int src, int ast, int dst)

{

if (num == 1) towers[src] >> towers[dst];

else

{

move(num - 1, src, dst, ast);

move(1, src, ast, dst);

move(num - 1, ast, src, dst);

}

}

int main()

{

int n;

printf("How many disks you want me to display?\n");

cin >> n;

// 建塔

for (int i = n; i > 0; i--)

{

towers[0].disks.push\_back(i);

}

show();

move(n, 0, 1, 2);

system("pause");

}