数据结构课程设计报告

Leadial

目录

一、课程设计概述

本次数据结构课程设计共完成 3 道题：一元稀疏多项式计算器的设计与实现、迷宫游戏的设计与实现、内部排序算法的性能分析的程序设计与实现问题。

使用语言：C++

编译环境：Visual Studio 2015

二、课程设计题目一

实验内容

一元稀疏多项式计算器的设计与实现

需求分析

1. 在命令行程序中输入并建立多项式；
2. 能够以一定的格式输出多项式；
3. 能够对输入的多项式进行一定的处理，进行加减运算；
4. 能够代入 x 值计算结果；
5. 操作指引明确，注释清晰，界面美观。

概要设计

ADT Polynomial {

基本操作：

```
    Polynomial CreatePNode(Polynomial, int);
    // 建立一元多项式
    void Insert(Polynomial, Polynomial);
    // 往多项式中插入结点
    void Display(Polynomial);
    // 输出多项式
    int Compare(Polynomial, Polynomial);
    // 控制 Calculate()中的开关
    void Result(Polynomial, float);
    // 代入 x 值计算结果
    Polynomial Calculate(Polynomial, Polynomial, char);
    // 代入数据进行运算
}
```

存储结构

```
typedef struct PNode {
   float coef; // 系数
   int expn;   // 指数
   struct PNode *next;
}PNode, *Polynomial;
```

流程图

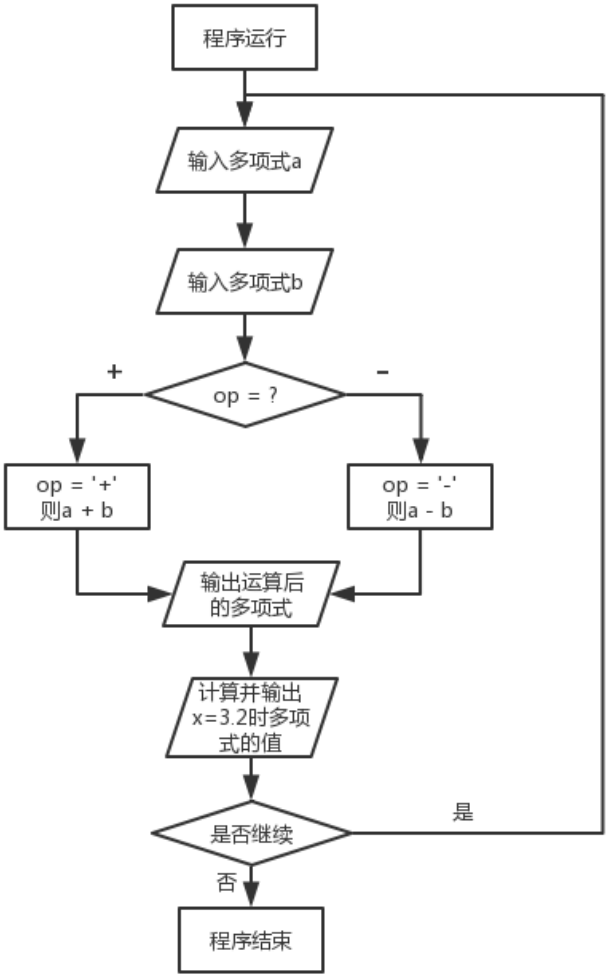


图 1 题目一流程图

详细设计

```
void Insert(Polynomial p, Polynomial q) {
    if (p->coef == 0) delete p; // 系数为 0 则释放结点
    else {
        Polynomial p1, p2; // 用于循环查找插入位置的两个指针
        p1 = q;
        p2 = q->next;
```

```
        while (p2&&p->expn < p2->expn) {
            // 循环，q->next == NULL 或待插入项的指数小于
q->next->expn 时为插入位置
            p1 = p2;
            p2 = p2->next; // 指针后移
        }
        if (p2&&p->expn == p2->expn) {
            // 当 q->next 指数与待插入项指数相同时，进行合并
            p2->coef += p->coef;
            delete p; // 插入完毕后释放结点
            if (!p2->coef) {
                // 系数为 0 则释放结点，删除该项
                p1->next = p2->next;
                delete p2;
            }
        }
        else {
          // 执行此步时，q->next == NULL && p->expn > p2->expn，
为新的项，插入链表
            p->next = p2;
            p1->next = p;
        }
```

```
    }
}
Polynomial Calculate(Polynomial a, Polynomial b , char op) {
    Polynomial qa = a->next;
    Polynomial qb = b->next;
    Polynomial headc, hc, qc;
    hc = new PNode; // 建立头结点
    hc->next = NULL;
    headc = hc;     // 存放头结点
    while (qa || qb) {
        qc = new PNode;
        switch (Compare(qa, qb)) {
        case 1: { // qc 存放 qa 的元素
            qc->coef = qa->coef;
            qc->expn = qa->expn;
            qa = qa->next;
            break;
        }
        case 0: { // 进行系数的相加
            if (op == '+') qc->coef = qa->coef + qb->coef;
            else if (op == '-') qc->coef = qa->coef - qb->coef;
            qc->expn = qa->expn;
```

```
            qa = qa->next;
            qb = qb->next;
            break;
        }
        case -1: { // qc 存放 qb 的元素
            qc->coef = qb->coef;
            qc->expn = qb->expn;
            qb = qb->next;
            break;
        }
        }
        if (qc->coef != 0){ // 插入元素，指针后移
            qc->next = hc->next;
            hc->next = qc;
            hc = qc;
        }
        else delete qc; // 系数为 0 则释放结点
    }
    return headc;
}
void Result(Polynomial p, float x) {
    float res = 0;     // 计算结果
```

```
    Polynomial pp; // pp 作为指针
    pp = p->next;
    if (pp != NULL) do {
        res += pp->coef * pow(x, pp->expn);
        pp = pp->next;
    } while (pp);
    cout << endl << "x = " << x << " 时多项式的值为" << res
<< endl;
}
```

调试分析

本程序主要的思路是用带头节点的单链表数据结构存储多项式，头节点会对程序的执行产生一定的影响，需要多加注意。

问题一：

现象：输出的项数比真实项数要多 1。

原因：没有注意到头结点，在计数时将其算作一项，输出前应该将这一项减去。

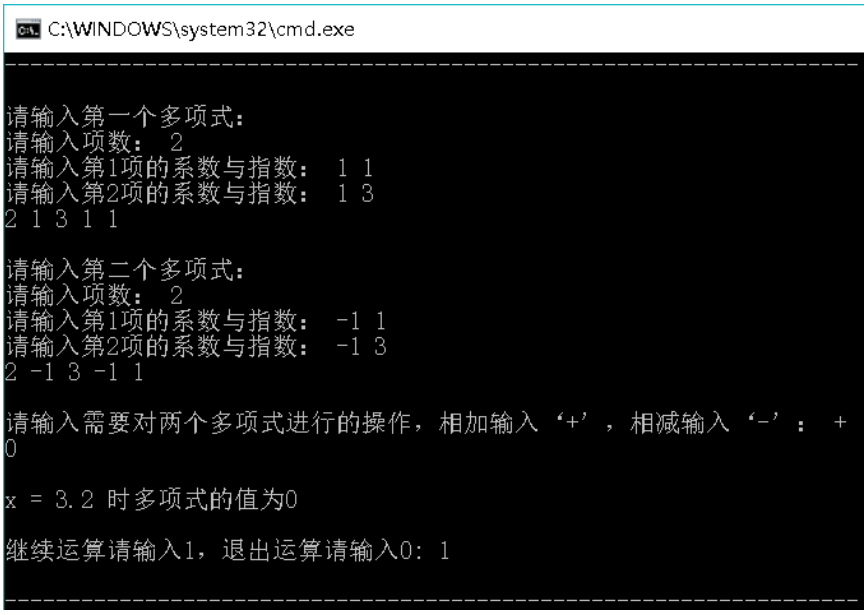
问题二：

现象：在输出项数 n 后，本应该直接输出多项式，但却多出两个较大的负值。且这两个负值没有对多项式的相加（减）造成影响，进行加（减）操作后输出的多项式仍带有两个较大负值。

原因：也是没有注意到头结点，在输出时应该将指针指向头结点的下一个结点。

【运行结果分析】

a) 多项式相减



b) 多项式为 0

图 2 题目一运行结果分析

从运行结果来看，测试了五组数据，其中如图 2 所示的两组数据具有一定的刁难性。图 2 a)中为多项式相减，与正常设计的相加不同，故有可能产生错误；图 2 b)中多项式的相加结果为 0，也有可能会出现引起程序崩溃的错误。这两组数据皆能通过运行，所以认为该

程序运行正确。

三、课程设计题目二

实验内容

迷宫游戏的设计与实现

需求分析

1. 设置一个迷宫，求出一条从入口到出口的通路，或得出没有通路的结论；

2. 实现一个链表作存储结构的栈类型，用非递归的方式求解迷宫；

3. 用递归的算法求得迷宫中所有可能的通路；

4. 以矩阵形式输出迷宫及其通路。

概要设计

ADT LinkStack {

基本操作：

```
int InitStack(LinkStack &);
// 构造一个空栈 S
int StackEmpty(LinkStack);
// 判断栈是否为空
int StackLength(LinkStack);
// 返回栈 S 的长度
```

```
    int Push(LinkStack &, Elem);
    // 元素入栈
    int Pop(LinkStack &, Elem &);
    // 元素出栈
    void GetTop(LinkStack, Elem &);
    // 取栈 S 的栈顶元素
    string ShowDir(int);
    // 将 int 型的方向转换成汉字
    void MazePath(Pos, Pos, int [11][10], int [4][2]);
    // 寻找通路核心算法
}
ADT Recur {
    int Trace(int, int);
    // 递归查找出口
    void Pace(int, int);
    // 以三元组(i, j, d)的形式输出通路
}
```

存储结构

```
typedef struct {
   int x;
   int y; // 记录位置
}Pos;
```

```
typedef struct {
    Pos pos; // 当前位置
    int dir; // 当前方向
}Elem;
typedef struct SNode {
    Elem data;
    SNode *next; // 指向下一个节点
}SNode, *LinkStack;
```

流程图

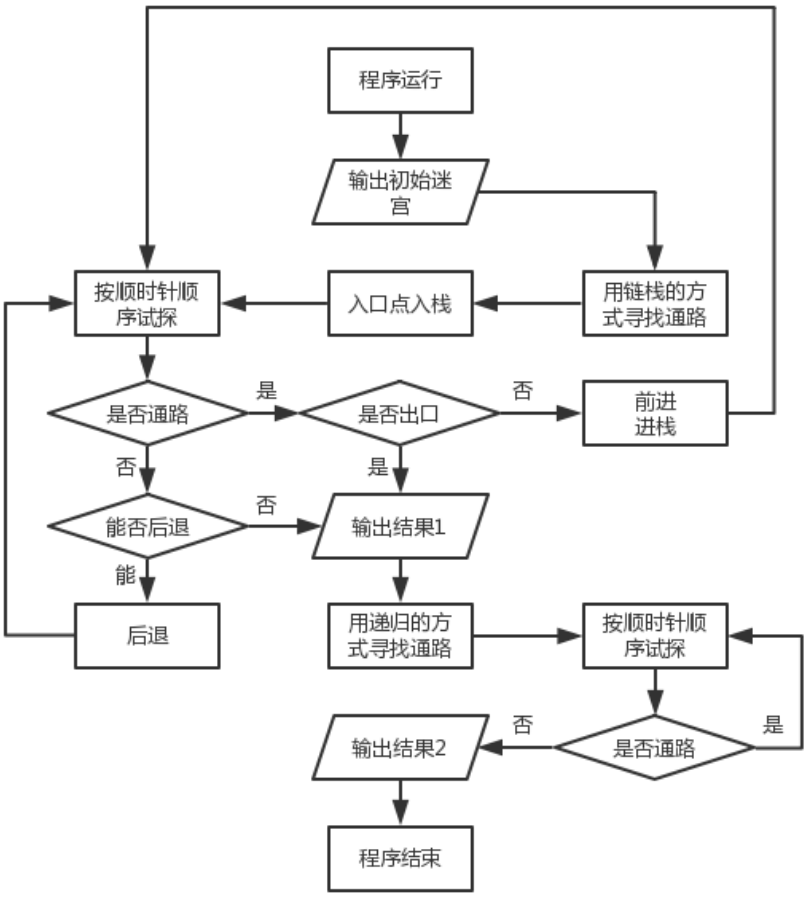


图 3 题目二流程图

详细设计

```
void MazePath(Pos start, Pos end, int maze[11][10], int diradd[4][2]) {
    int i, j, d; // 临时储存元素的 x, y 与方向
    int a, b;    // 临时储存元素的 x, y
    Elem fin;    // 主要用于存放倒数第二个元素
    Elem elem, e;
    LinkStack S1, S2;
    InitStack(S1);
    InitStack(S2); // S2 用于存放倒置的栈
    maze[start.x][start.y] = 2; // 标示入口已经走过
    elem.pos.x = start.x;
    elem.pos.y = start.y;
    elem.dir = -1; // 初始化
    Push(S1, elem);
    while (!StackEmpty(S1)) {
        // 有路可走
        Pop(S1, elem);
        i = elem.pos.x;
        j = elem.pos.y;
        d = elem.dir + 1; // 下一个方向
        while (d < 4) {
```

```
// 以顺时针的顺序试探东南西北
a = i + diradd[d][0];
b = j + diradd[d][1];
fin.dir = d;
if (a == end.x && b == end.y && maze[a][b] == 0) {
   // 到达出口
   Push(S1, fin); // 将倒数第二个元素入栈
   elem.pos.x = a;
   elem.pos.y = b;
   elem.dir = 4; // 将 4 标记为出口
   Push(S1, elem);
   while (!StackEmpty(S1)) {
      // 逆置序列
      Pop(S1, e);
      Push(S2, e);
   }
   while (!StackEmpty(S2)) {
      // 输出迷宫路径
      Pop(S2, e);
      string Showdir;
      Showdir = ShowDir(e.dir);
      cout << "(" << e.pos.x << ", " << e.pos.y <<
```

```
", " << Showdir << ")" << endl;
            }
            return; // 跳出循环
        }
        if (maze[a][b] == 0) {
            // 找到可以前进的非出口点
            maze[a][b] = 2; // 标记走过此点
            elem.pos.x = i;
            elem.pos.y = j;
            elem.dir = d;
            Push(S1, elem); // 当前位置入栈
            fin.pos.x = i = a;
            fin.pos.y = j = b; // 此处可用于存放出口前一个
结点
            d = -1; // 下一点转化为当前点
        }
        d++;
    }
  }
  cout << "没有通路。" << endl;
}int Trace(int x, int y) {
  Maze[x][y] = 2;
```

```
    if ((x == 9 && y == 8)) flag = 1; // 设置出口为(9,8)
    if (flag != 1 && Maze[x - 1][y] == 0) Trace(x - 1, y); //
判断向北是否有路
    if (flag != 1 && Maze[x][y + 1] == 0) Trace(x, y + 1); //
判断向东是否有路
    if (flag != 1 && Maze[x + 1][y] == 0) Trace(x + 1, y); //
判断向南是否有路
    if (flag != 1 && Maze[x][y - 1] == 0) Trace(x, y - 1); //
判断向西是否有路
    if (flag != 1) Maze[x][y] = 0; // 非通路时恢复为0
    return flag;
}
```

调试分析

本程序最复杂的部分是用链栈的数据结构存储迷宫的路径，对栈进行操作的步骤需要做到比较精确，在编写程序前要有十分清晰的逻辑。

问题一：

现象：输出栈中元素时倒数第二个元素有缺失。

原因：运用了在函数各个位置输出所在点的方法，去尝试检验哪一步有遗漏。最终发现在到达出口时，前一个点的数据被覆盖掉了。于是再新设置了一个元素，来储存出口前的一个点的位置，在发现出口时该元素入栈。

问题二：

现象：路径行至(6, 5)处又沿着原路返回，最后给出没有通路的结论。

原因：原来的设想是用 3 表示墙，但由此引发了各种错误。采用了设置断点的方式，让迷宫逐步输出，发现在(6, 5)处变为了障碍。后来把道路非通的标志去除，改为及时将非通的点出栈。有利有弊，此时只能输出路径，若将迷宫整体输出，死路也会被路径填满，实际应用中会使迷宫失去其意义。但可再根据栈设置一个函数，重新描绘路径。

问题三：

现象：递归实现的迷宫运行后一直无反应。

原因：设置断点，逐步输出迷宫，发现出现了死循环，对走过的路径标志有误。后来在合适的位置定义行走过的路径，死循环的问题得到解决。

【运行结果分析】

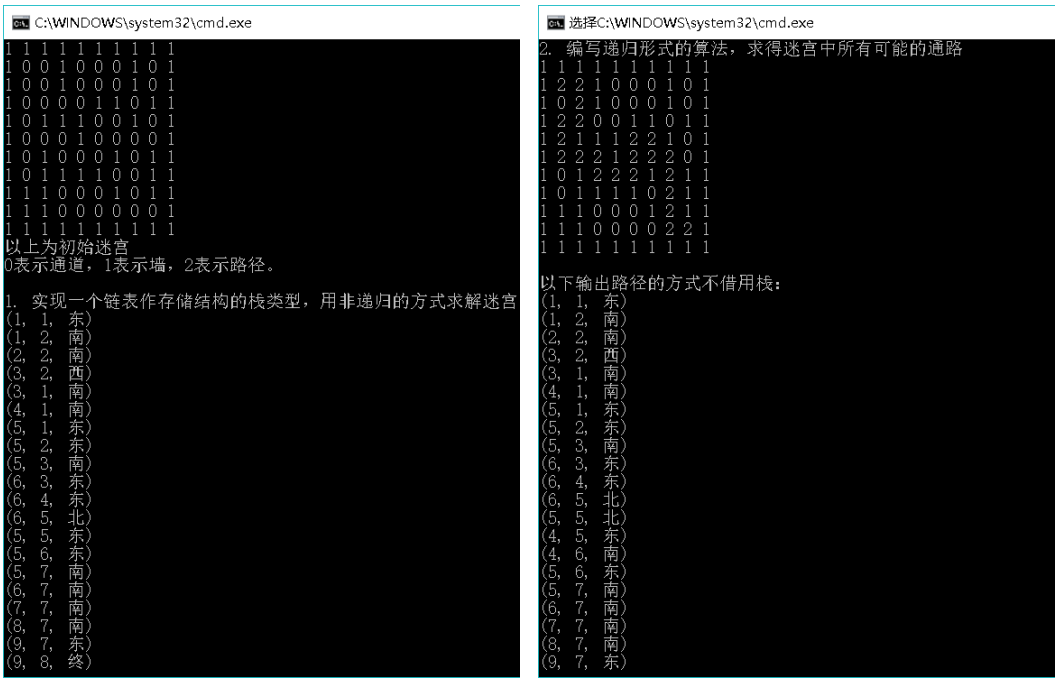


图 4 题目二运行结果分析

链栈与递归的方法都能输出行进至迷宫出口的路径，虽然路径有些许差异，链栈优先向东试探，递归优先向北试探，但都成功找到出口。尝试将迷宫的(6, 5)改成 1，都得出没有通路的结论。故认为该程序运行正确。

四、课程设计题目三

实验内容

内部排序算法的性能分析的程序设计与实现

需求分析

1. 比较起泡排序、插入排序、简单选择排序、归并排序、快速排序、希尔排序、堆排序算法的关键字比较次数和移动次数；

2. 待排序表的表长不小于 100，表中数据随机产生，且至少用 5 组不同数据作比较。

概要设计

```
class SST {
public:
   int Create();
   // 构造一个含 n 个数据元素的静态查找表 ST。
   int Search_Seq(int);
   // 在顺序表 ST 中顺序查找其关键字等于 key 的数据元素。
   int TraverseSST(int(*Visit)(int e));
```

```
    // 遍历表
    int RandomCreate();
    // 核心算法，随机构造一个含 n 个数据元素的静态查找表 ST。
    int reset();
    // 重置 ST
};
ADT Sort {
     void CompareSortTime(SST);
     // 核心算法，比较各算法排序时间
};
```

存储结构

```
typedef struct {
    int elem[MAXSIZE + 1];  // 数据元素存储空间基址
    int length; // 表长度
}SSTable;
```

流程图

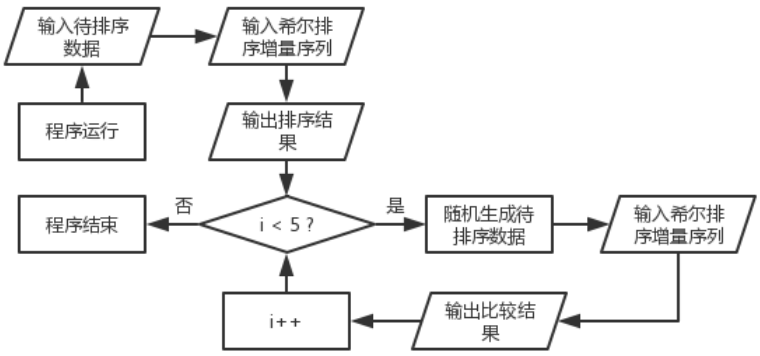


图 5 题目三流程图

详细设计

```
int SST::RandomCreate() {
    // 随机构造一个含 n 个数据元素的静态查找表 ST。
    int n;
    cout << "请输入需随机创建的数据元素的个数 n（n < 1000，建议 1000)：";
    cin >> n;
    for (int i = 1; i <= n; i++) ST.elem[i] = temp.elem[i] = rand();
    ST.length = n;
    temp.length = n;
    return OK;
}
void CompareSortTime(SST ST2) {
    int dlta[100];
    int t = 3;
    cout << "请输入希尔排序增量序列容量：";
    cin >> t;
    cout << "请输入希尔排序增量序列：";
    for (int i = 0; i < t; i++)cin >> dlta[i];
    dlta[t] = '\0';
```

```
    int comt = 0, movt = 0;
    BubbleSort(ST2.ST, comt, movt);
    cout << endl << "起泡排序关键字参加比较次数为：" << comt
<< "，移动次数为：" << movt;
    ST2.reset();
    comt = 0, movt = 0;
    InsertSort(ST2.ST, comt, movt);
    cout << endl << "插入排序关键字参加比较次数为：" << comt
<< "，移动次数为：" << movt;
    ST2.reset();
    comt = 0, movt = 0;
    SelectSort(ST2.ST, comt, movt);
    cout << endl << "简单选择排序关键字参加比较次数为：" <<
comt << "，移动次数为：" << movt;
    ST2.reset();
    comt = 0, movt = 0;
    QuickSort(ST2.ST, comt, movt);
    cout << endl << "快速排序关键字参加比较次数为：" << comt
<< "，移动次数为：" << movt;
    ST2.reset();
    comt = 0, movt = 0;
    ShellSort(ST2.ST, dlta, t, comt, movt);
```

```
    cout << endl << "希尔排序关键字参加比较次数为：" << comt
<< "，移动次数为：" << movt;
    ST2.reset();
    comt = 0, movt = 0;
    HeapSort(ST2.ST, comt, movt);
    cout << endl << "堆排序关键字参加比较次数为：" << comt <<
"，移动次数为：" << movt;
    ST2.reset();
    comt = 0, movt = 0;
    MergeSort(ST2.ST, comt, movt);
    cout << endl << "归并排序关键字参加比较次数为：" << comt
<< "，移动次数为：" << movt << endl;;
}
```

调试分析

本程序用顺序存储结构实现对各内部排序方法进行比较，使用起来较为方便。

问题一：

现象：有时程序能执行成功，有时不可以。重启电脑后的第一次执行都会成功，其后执行六次大概会有一次成功。随机生成数据时只能生成一部分，而且每次生成的数据量不一定。错误提示是“已触发了一个断点”、“stack overflow”。

原因：数据元素采用了int *elem的定义，网上说这是内存越界所致。

数据量小的时候没有异常情况，数据量大时就会出错。根据网上的提示，尝试了连接 Microsoft 符号服务器等方法没有效果，更有修改编译器内核等等远超出个人当前能力的思路。因此后面改为了顺序表的结构，问题解决。

问题二：

现象：计算关键字比较次数和交换次数不准确。

原因：对题目所表达的意思不太理解，所以添加的参与计数的部分可能不足。有一些比较次数，只要序列长度一致便不会发生变化。有些比较次数过少，不明白哪一部分是题目所要求得的数据。只要题意更清晰，需要实现的计数只需要直接添加即可。

【运行结果分析】

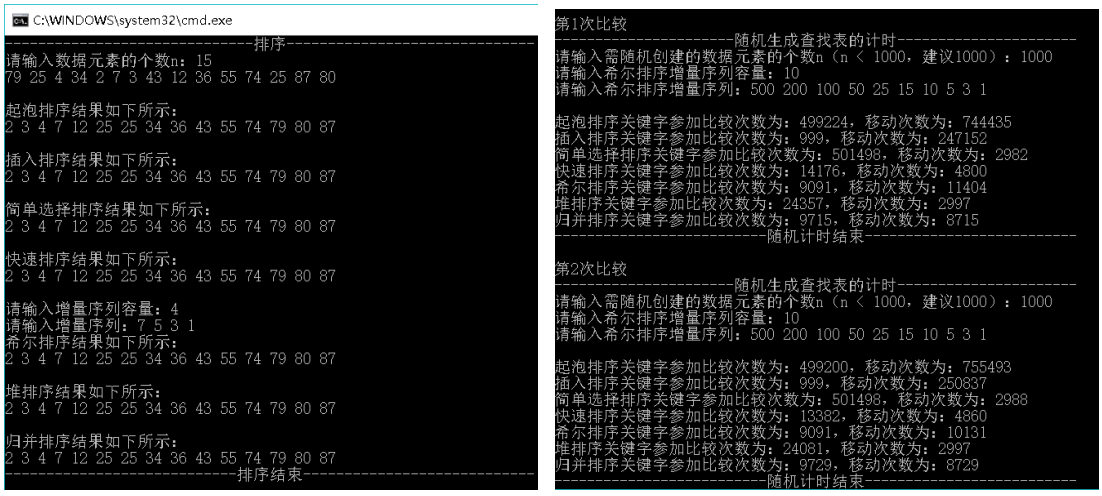


图 6 题目三运行结果分析

图 6 中左图为一组测试数据排序后得出的结果，用以检验进行排序的代码是否有效，结果表明排序代码正常运行，其后的次数比较均基于排序的代码，所以次数的统计有效。且五次排序中比较和移动次数都有一定差异，所以认为该程序运行正确。

五、结束语

在本次课程设计中，我得到了很大的锻炼。

1. 我增进了文献阅读的能力。虽然我没有把所有的问题都做完，可是除了多项式的问题找不到相应的文献外，我针对其它的问题也查阅了许多相关的文献。由于时间不足，无法将剩余的两个问题实现，但思路已经搞明白了。比如哈夫曼树，简单来说，就是每次把权值最低的两个元素合成一个子树，然后重新排序，再把权值最低的两个合成子树，最后就会权值较高的在上面。文献中的论述会比平时阅读到的许多资料都更为严谨，更能将一些细节性的部分讲透，对我们平时的学习有很大的帮助。

2. 我增进了解决实际问题的能力。书本上讲的只是一些比较普遍性的东西，在实际的应用中还会面临许多的问题，需要自己去思考如何实现。在编写代码之前，要明确需要实行的步骤，通过什么样的方法一步步实现目标。如书本上的一些伪码，当拿到一个问题的时候，需要将它拆分成一个个小的问题，逐个击破。解决这些问题，也让我对书本上的知识有了更深刻的理解。同时，除了课程设计所要求解决的问题外，一个模型还可以为解决其它问题提供帮助。例如迷宫问题，可以为实现连连看的小游戏提供参考。哈夫曼树还可以应用在矢量地图的压缩，这个应用便和我们的专业有了相关之处了。学以致用，知识才能发挥它的价值。

3. 我发现了自己的不足。在运用课本的知识时，我仍然不够熟

练，无法快速地想到该应用什么方法，采取怎样的方式，去解决一个问题。在编写代码的时候需要的时间太长，思路还不够清晰，表述有时候过于复杂，不够直观。在解题过程中出现的许多错误，都和不够熟练有很大的关系。这就要求我要多加练习，多进行实际的上机操作，提升自己的解题思路和能力，培养严谨、清晰的思维。

感谢老师这一学期的辛勤教导，我必将更努力地学习，将学到的知识真正地运用到实处。

参考文献

[1] 胡佳,赵福生. 广度优先搜索在迷宫问题中的应用[J]. 江西教育学院学报, 2013,34(03):27-29.

[2] 何昭青. 运用 Huffman 编码进行数据压缩的新算法[J]. 科学技术与工程, 2008(16):4531-4535.

[3] 江燕,周军,罗冬梅,尼亚孜买买提,李莉. 内部排序算法的分析与比较[J]. 电脑编程技巧与维护, 2014(21):23-24.

[4] 李晓飞. Huffman 编解码及其快速算法研究[J]. 现代电子技术, 2009,32(21):102-104+108.

[5]廖国勇,王广超.用遗传算法解迷宫问题[J].华东交通大学学报,2006(02):138-140.

[6] 刘兴科,陈轲,于晓光. Huffman 编码在矢量地图压缩中的应用[J]. 测绘科学技术学报, 2014,31(01):89-92.

[7] 涂海丽. 求迷宫中从入口到出口的路径的算法及实现[J]. 中国科技信息, 2008(23):54+56.

[8] 谭浩强. C++面向对象程序设计（第 2 版）[M]. 北京：清华大学出版社, 2014.

[9] 陶圣哲. 几种典型内部排序算法性能分析[J]. 电脑知识与技术, 2016,12(26):26-29.

[10] 文国知. 基于 C 语言的自适应 Huffman 编码算法分析及实现研究[J]. 武汉工业学院学报, 2011,30(02):53-57+62.

[11] 遇娜,简广宁. 回溯法求解迷宫问题[J]. 天津职业院校联合学报, 2011,13(08):46-49.

[12] 严蔚敏, 吴伟民. 数据结构[M]. 北京：清华大学出版社, 2007.

[13] 朱素英. 迷宫问题的图论解法探讨[J]. 湖南人文科技学院学报, 2006(03):73-75.

[14] 左羽. 数据结构与算法课程的入门教学范例——迷宫问题[J]. 科技广场, 2009(11):53-56.

[15] Brian W. Kernighan, Dennis M. Ritchie. C 程序设计语言[M]. 北京：机械工业出版社, 2004.