南京信息工程大学

《数据结构》课程设计

题　　目　数据结构应用

姓名___朱宸扬_____

学号___202183760012__

专业___信安_____

二O二一　年 12　月 24　日

数据结构应用

朱宸扬

计算机与软件学院，信安，21 奇安信

摘　要：本文涵盖了三个重要的计算机科学和信息技术领域的主题，包括后缀表达式求值、哈夫曼编码系统和图书管理系统。首先，介绍了后缀表达式求值的基本概念和原理，详细说明了如何通过栈数据结构实现后缀表达式的有效求值过程。后缀表达式求值在编译器和计算器等应用中具有广泛的应用，因其简洁的表示形式和高效的计算性能而备受关注。其次，探讨了哈夫曼编码系统，这是一种用于数据压缩的有效方法。通过构建哈夫曼树，该编码系统能够为不同字符分配不同的编码，确保最短编码用于频率较高的字符，从而实现数据压缩。哈夫曼编码在通信和存储领域得到广泛应用，为数据传输提供了高效的解决方案。最后，介绍了图书管理系统，该系统在图书馆和其他文献管理环境中起到关键作用。通过数据库管理、借还管理、搜索功能等模块，图书管理系统能够高效地管理大量图书信息，提供方便快捷的检索和借阅服务。这对于图书馆和其他知识管理机构来说是不可或缺的工具，有助于提升信息组织和检索的效率。

关键词：表达式求值，哈夫曼，图书管理

第一章　链表的应用（或栈和队列的应用）

中缀表达式转换为后缀表达式
后缀表达式求值

中缀变后缀，要利用到字符栈
遇到数字直接进表达式，遇到符号，进栈或者进表达式
栈为空，符号就进栈
栈不为空，就进行比较，优先级高的进表达式，低的在栈里
比如乘号遇到加号，乘号就进栈。

```
string infixToPostfix(const string& infix) {
   stack<char> operators;
   string postfix;
   for (char ch : infix) {
       if (isalnum(ch)) {
           postfix += ch;  // 操作数直接添加到后缀表达式中
       } else if (ch == '(') {
           operators.push(ch);
       } else if (ch == ')') {
           while (!operators.empty() && operators.top() != '(') {
               postfix += operators.top();
               operators.pop();
           }
           operators.pop();  // 弹出左括号
       } else {  // 运算符
           while    (!operators.empty()    &&    precedence(ch)    <=
precedence(operators.top())) {
               postfix += operators.top();
               operators.pop();
           }
           operators.push(ch);
       }
   }
```

```
    while (!operators.empty()) {
        postfix += operators.top();
        operators.pop();
    }
    return postfix;
}
```

后缀计算，要利用数字栈
数字直接进栈，遇到符号，就让栈中最上面两个数字用这个符号运算，结果再进栈，最后会只剩一个数，就是结果

```
double evaluatePostfix(const string& postfix) {
    stack<double> operands;

    for (char ch : postfix) {
        if (isalnum(ch)) {
            operands.push(ch - '0');  // 将字符转换为数字
        } else {
            double operand2 = operands.top();
            operands.pop();
            double operand1 = operands.top();
            operands.pop();

            switch (ch) {
                case '+':
                    operands.push(operand1 + operand2);
                    break;
                case '-':
                    operands.push(operand1 - operand2);
                    break;
                case '*':
                    operands.push(operand1 * operand2);
```

```
                    break;
                case '/':
                    operands.push(operand1 / operand2);
                    break;

            }
        }
    }

    return operands.top();
}
```

第二章　树结构的应用（或图结构应用）

哈夫曼文本编码系统

1 数据结构:

代码定义了一个 Node 结构，表示 Huffman 树中的一个节点，存储字符数据、频率以及左右子节点的指针。

Compare 结构被用作优先队列的比较器，确保具有较低频率的节点具有较高的优先级。

2Huffman 树构建:

buildHuffmanTree 函数接受字符频率的映射，并使用优先队列构建 Huffman 树。它重复组合两个最低频率的节点，直到只剩一个根节点为止。
Huffman 编码生成:

generateHuffmanCodes 函数递归地遍历 Huffman 树，为每个字符生成二进制编码。

```
Node* buildHuffmanTree(const unordered_map<char, int>& charFrequency) {
    priority_queue<Node*, vector<Node*>, Compare> minHeap;

    for (const auto& pair : charFrequency) {
        minHeap.push(new Node(pair.first, pair.second));
    }

    while (minHeap.size() > 1) {
        Node* left = minHeap.top();
        minHeap.pop();
        Node* right = minHeap.top();
        minHeap.pop();

        Node* internalNode = new Node('$', left->frequency +
right->frequency);
        internalNode->left = left;
        internalNode->right = right;
        minHeap.push(internalNode);
    }

    return minHeap.top();
}
```

3 编码和解码:

huffmanEncode 使用生成的 Huffman 编码对给定文本进行编码。
huffmanDecode 使用 Huffman 树解码编码文本。
保存 Huffman 树到文件:
saveHuffmanTreeToFile 函数将 Huffman 树中每个节点的字符数据和频率写入文件。

```
// 哈夫曼编码
string huffmanEncode(const string& text, const unordered_map<char,
string>& huffmanCodes) {
```

```
    string encodedText;
    for (char ch : text) {
        encodedText += huffmanCodes.at(ch);
    }
    return encodedText;
}

// 哈夫曼译码
string huffmanDecode(const string& encodedText, const Node* root) {
    string decodedText;
    const Node* current = root;

    for (char bit : encodedText) {
        if (bit == '0') {
            current = current->left;
        } else if (bit == '1') {
            current = current->right;
        }

        if (current->data != '$') {
            decodedText += current->data;
            current = root;
        }
    }

    return decodedText;
}
```

4 用户输入和输出:

主函数提示用户输入文本。
然后计算文本中每个字符的频率。
构建 Huffman 树，生成 Huffman 编码，并打印每个字符的编码。
使用 Huffman 编码对输入文本进行编码并打印结果。
将 Huffman 树保存到文件（hfmTree.txt），将编码文本保存到另一个文件

（CodeFile.txt）。

5 打印 Huffman 树的树状图。

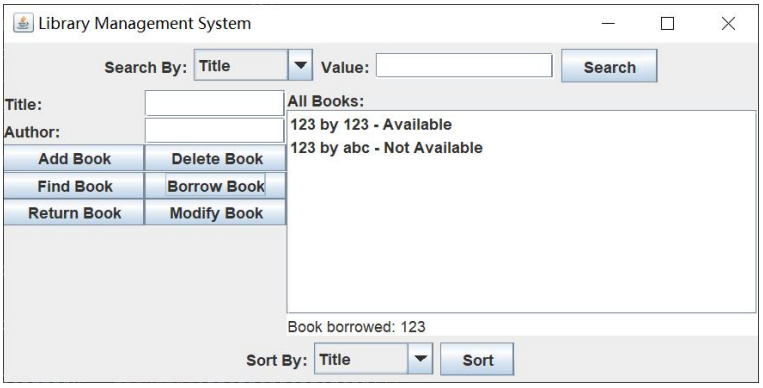
```
void printHuffmanTree(Node* root, int indent = 0) {
    if (root == nullptr) {
        return;
    }

    // 右子树，根，左子树的顺序
    printHuffmanTree(root->right, indent + 4);

    // 打印当前节点
    if (indent > 0) {
        cout << setw(indent) << " ";
    }
    cout << root->data << " (" << root->frequency << ")" << endl;

    // 打印左子树
    printHuffmanTree(root->left, indent + 4);
}
```

第三章　小型信息管理系统设计



添加书籍：
用户可以通过输入书籍的标题和作者信息，点击“Add Book”按钮来添加新的书籍到图书馆。

删除书籍：
用户可以在书籍列表中选择一本书籍，并点击“Delete Book”按钮来删除该书籍。系统会弹出确认对话框，确保用户的删除操作是有意的。

查找书籍：
用户可以通过输入书籍的标题，点击“Find Book”按钮来查找图书馆中是否存在该书籍。系统会在结果区域显示查找结果。

借阅书籍：
用户可以选择书籍列表中的一本可用书籍，点击“Borrow Book”按钮来借阅该书籍。系统会更新书籍的借阅状态，并在结果区域显示相关信息。

归还书籍：
用户可以选择已借阅的书籍，点击“Return Book”按钮来归还书籍。系统会更新书籍的借阅状态，并在结果区域显示相关信息。

修改书籍信息：
用户可以选择书籍列表中的一本书籍，点击“Modify Book”按钮，然后输入新的标题和作者信息来修改书籍的信息。

排序书籍列表：
用户可以通过选择排序方式（按标题、作者或可用性），然后点击“Sort”按钮，对书籍列表进行排序。系统会在结果区域显示排序后的书籍列表。

搜索书籍：
用户可以通过选择搜索条件（按标题、作者或可用性），输入搜索值，然后点击“Search”按钮，来搜索匹配的书籍。系统会在结果区域显示搜索结果。
总体而言，这个图书管理系统提供了基本的图书管理功能，包括添加、删除、查找、借阅、归还、修改、排序和搜索。用户通过简单的界面交互即可完成这些操作，使图书馆管理更加方便和高效。

第四章　总结

这三个主题的学习和实践为我提供了全面的计算机科学和信息技术知识，涵盖了数据结构、算法、面向对象编程和图形用户界面等方面。通过实际动手操作，我不仅提升了编程技能，还深化了对计算机科学领域的理解。这些知识和经验将在未来的学习和职业生涯中发挥重要作用。