数据结构 Lab2 实验报告

卢虹宇
2023202269

2024 年 11 月 4 日

1 整体介绍

本实验主要利用模板类实现了基于链表和栈的数据结构（`My_structure.h`），并构建了一个 HTML 解析器（`html_parser.cpp`）。在 HTML 解析中，采用嵌套链表作为数据结构，其中每个节点存储一个标签的名称、HTML 原文以及该标签的子标签链表。这种设计实际上形成了一个树形结构，经过与窦老师的讨论确认，该实现方式是合理的。

2 亮点

2.1 可扩展性

在实现链表和栈的数据结构时，使用了模板类，这大大提高了代码的可扩展性和复用性，使其能够支持多种数据类型的存储。

2.2 使用 <text> 标签来存储文本

使用 ‘<text>‘标签来存储文本是一种比较巧妙的设计，能够简化文本节点的管理，使得文本的寻址和提取变得更加方便。

2.3 实现了 URL 爬取并解析返回的 HTML 文件

利用 *curl* 库实现了 URL 爬虫功能，能够从指定的网络地址获取 HTML 文本并进行解析。

2.4 页面设计合理

在用户输入的 HTML 文件不合法或提供的文件地址与 URL 不正确时，程序不会立即退出。这一设计使得用户可以反复输入，直到提供有效的输入。在成功解析后，程序将弹出功能菜单，允许用户选择所需的操作。此外，用户可以在不退出程序的情况下，多次选择不同的功能，或重新选择文件进行解析。这种交互设计提升了用户体验，使得操作更加灵活便捷。

```
root@LAPTOP-IFFR0KNH:/home/课程资料/数据结构与算法/lab_2# ./parser
请选择输入方式:
[1]: 输入文件地址
[2]: 输入 URL
1
请输入文件地址
dwoidh
无法打开文件！请重新输入文件地址!
请输入文件地址:
/home/课程资料/数据结构与算法/lab_2/test_cases/emnlp_valid.html
文件解析成功
[1]:重新选择文件进行解析
[2]:判断该html文件合法性
[3]:输出对应路径下的文本
[4]:输出对应路径下的html代码段
[5]:打印解析出的html文件
[-1]:退出程序
请选择以上操作:
1
请选择输入方式:
[1]: 输入文件地址
[2]: 输入 URL
```

图 1: 用户交互界面示例

3 My_structure.h

3.1 模板类链表的实现

链表类通过模板实现，可以存储任意类型的数据。主要方法包括:

- `append(T value)`: 向链表末尾添加新节点。
- 析构函数: 释放链表内存，确保没有内存泄漏。

链表的基本结构如下所示:

```
class Link_list {
    Node<T> *head; // 链表头指针
    ...
    void append(T value);
```

```
    ~Link_list();
};
```

3.2 模板类栈的实现

栈类实现了基本的栈操作，如压栈、弹栈和查看栈顶元素。主要方法包括：

- `push(T value)`: 压入栈顶。
- `pop()`: 弹出栈顶元素。
- `top()`: 返回栈顶元素。

栈的基本结构如下所示：

```
class Stack {
    Node<T> *topNode; // 栈顶指针
    ...
    void push(T value);
    void pop();
};
```

4 html_parser.cpp

4.1 html_node 结构体

该结构体定义了 HTML 节点的基本信息，包括标签名和原始 HTML 内容。节点之间通过链表进行连接，便于存储和访问。

```
struct html_node {
    string tag_name; // 标签名
    string tag;      // tag的HTML原文
    Link_list<html_node *> list; // 子节点列表
};
```

4.2 parser 解析器的实现

4.2.1 核心逻辑

解析器的核心逻辑主要通过遍历 HTML 内容并根据标签类型进行处理。伪代码示例如下：

```
bool parse(string &htmlcontent) {
    Stack<html_node *> nodestack;
    size_t i = 0;

    while (i < htmlcontent.size()) {
        if (isDOCTYPE(htmlcontent, i)) {
            skipDOCTYPE();
        } else if (isComment(htmlcontent, i)) {
            skipComment();
        } else if (isScript(htmlcontent, i)) {
            skipScript();
        } else if (isOpeningTag(htmlcontent, i)) {
            string tag = extractTag(htmlcontent, i);
            if (isClosingTag(tag)) {
                handleClosingTag(tag);
            } else {
                handleOpeningTag(tag);
            }
        } else {
            handleTextNode(htmlcontent, i);
        }
    }
    return true;
}
```

4.2.2 错误处理

解析器会在遇到标签不匹配时输出错误信息，并清空解析过程中的数据。错误主要分为以下两类：

- **标签未闭合**
  - 在处理开始标签时，将其压入栈；遇到闭合标签时检查与栈顶的开始标签是否匹配。
- **标签包含问题**

- **特殊块级元素**（如 <h1>、<h2>、<p>、<dt>）只能嵌套行内元素：

```
if (is\_SpecialBlockTag(nodestack.top()->tag\_name)
                                    && !isInlineTag(tagname))
```

- **块级元素**不能嵌套在行内元素中（除了 <a>）：

```
if (isBlockTag(tagname) && isInlineTag(nodestack.top()->tag\_name)
                                    && nodestack.top()->tag\_name != "a")
```

- **特殊行内元素**（<a>）：可以嵌套块级和行内元素，但不能嵌套自身：

```
if (tagname == "a" && nodestack.top()->tag\_name == "a")
```

4.3 text 函数 & outerhtml 函数的实现

这两个函数用于返回指定路径下的所有文本内容和 html 原文。核心思路都是按照地址访问解析好的 html 树，当遇到地址的最后一个标签时，访问该标签下的所有子节点提取其中的 <text> 标签或者标签中的 html 内容