# 数据结构lab4实验报告

## 最短路径中文文本分词

**作者**：卢虹宇-2023202269
**日期**：2024年12月9号

---

# 1. 需求分析

本实验的任务是实现一个基于最短路径匹配算法的中文分词程序。该程序能够使用前缀树解析词典，并且将连续的汉字序列切分成单独的词语，适用于中文文本分析和自然语言处理的基础工作。

## 1.1 输入的形式和输入值的范围

- **输入形式**：程序通过命令行界面进行交互式输入。用户可以在命令行中输入一段连续的中文文本进行分词，也可以选择重新加载词典。
- **输入值范围**：输入为任意长度的中文字符串，不包含形式上的词界符。词典文件`dict.txt`中的词汇包含汉字、词频和词性信息，词频为整数，词性为字符串标识。

## 1.2 输出的形式

- **输出形式**：程序在命令行中输出分词结果，每个词之间用“/”进行分隔。例如，输入“我是人大学生”，输出“我/是/人大/学生”。

## 1.3 程序所能达到的功能

- **基本功能**：
  - 实现基础的最短路径中文分词。
  - 提供命令行交互界面，支持用户输入文本进行分词。
  - 支持加载和重新加载词典文件`dict.txt`。
- **拓展功能**：
  - 实现N最短路径分词算法，输出前n短的分词路径。

# 2. 概要设计

本程序主要由词典加载模块、分词模块和用户交互模块组成。程序通过构建前缀树来存储词典中的词汇，并利用最短路径算法进行分词。

## 2.1 抽象数据类型定义

- **树节点结构（Tree_node）**：用于构建前缀树的节点，包含数据域和子节点链表。
- **前缀树类（Prefix_tree）**：基于树节点结构实现的前缀树，用于高效存储和查找词典中的词汇。
- **前缀节点结构（word_node）**：包含字符和权重，用于前缀树中的每个节点。

## 2.2 主程序流程

程序的主流程如下：

1. 程序启动，设置本地化环境以支持宽字符。
2. 用户输入词典文件的路径，程序加载词典并构建前缀树。
3. 进入操作选择界面，用户可以选择重新加载词典、输入句子进行分词或退出程序。
4. 若选择分词，用户输入句子，程序利用前缀树进行最短路径分词，并输出结果。
5. 用户可重复选择操作，直至选择退出程序。

# 3. 详细设计

本节详细描述了程序中各数据类型的实现方式以及主要算法的伪代码。

## 3.1 数据类型实现

- **树节点结构（Tree_node）**：包含数据域和子节点链表，用于构建前缀树。
- **前缀树类（Prefix_tree）**：实现词典的前缀树，支持添加词汇和获取词汇权重等操作。通过前缀树高效查找词汇，支持分词算法的需求。

## 3.2 核心算法

### 3.2.1 最短路径分词伪码（分词函数）

**输入**:

- `sentence`: 待分词的句子
- `tree`: 前缀树，提供子串的权重信息

**输出**:

- 分词后的最短路径

**算法流程**:

1. 初始化：
   - `min_weight`: 记录每个位置的最小路径权重，初始为正无穷。
   - `split_pos`: 记录每个位置的最佳切割点，初始为 `-1`。
   - `pos = 0`：开始处理的位置。
2. 动态规划求解最短路径：
   - 对于当前起始位置 `pos`，从 `pos` 开始尝试扩展到所有可能的结束位置 t：
     - 提取子串 `sub = sentence[pos:t]`。
     - 如果 `sub` 在词典中有效：
       - 计算新的权重 `tw`：
         - 如果 `pos` 是起始位置，则 `tw = weight(sub)`。
         - 否则，`tw = min_weight[pos-1] + weight(sub)`。
       - 如果新的权重 `tw` 比当前位置的最小权重更优，更新：
         - `min_weight[t] = tw`。

            - `split_pos[t] = pos`。
        - 记录当前最优权重的结束位置 `re_index`。
    - 更新 `pos = re_index + 1`。
3. 回溯路径：
    - 从末尾位置开始，根据 `split_pos` 逐步回溯到起始位置，构建分词结果。
4. 返回分词结果。

### 3.2.2 N最短路径分词伪码（分词函数）

**输入**：

- `sentence`：待分词的句子
- `tree`：前缀树，提供子串的权重信息
- `n`：返回的路径数量

**输出**：

- `n` 条不同的最短路径，每条路径为一个单词序列

---

**数据结构**

- `split_pos[i]`：保存所有到达位置 i的路径记录，每条记录包含：
    - `prior_ver`：上一个切分点的位置。
    - `weight_log *former`：对应上一个路径记录点
    - `weight`：从起点到位置 `i` 的累计权重。

**主要流程**

1. **初始化**：
    - 在位置 `0` 插入一个虚拟的初始记录，表示起点，权值为 `0`。
2. **动态规划**：
    - 遍历每个起始位置 `pos`：
        - 如果位置 `pos` 没有任何候选路径，跳过。
        - 对每个可能的结束位置 `t`(t > pos)：
            - 检查子串 `sentence[pos:t]` 是否为合法词。
            - 如果合法：
                - 从 `split_pos[pos]` 中的每条候选路径延伸到位置 `t`。
                - 计算新路径权重，并存储在 `split_pos[t]` 中。
3. **排序**：
    - 对 `split_pos[length]` 按权重升序排序。
4. **回溯路径**：
    - 从 `split_pos[length]` 的前 `n` 条路径中逐一回溯，构建最终的分词路径。
5. **返回结果**：
    - 返回 `n` 条不同的最短路径。

# 4. 调试分析

## 4.1 调试过程中遇到的问题及解决方法

在调试过程中，主要遇到了以下几个问题：

- **宽字符处理问题**：由于中文涉及宽字符，初期在文件读取和输出时出现乱码。通过设置全局本地化环境和使用宽字符流（`wcin`，`wcout`）解决了该问题。
- **文件编码转换问题**：词典文件采用UTF-8编码，而程序需要处理宽字符。使用`wstring_convert`和`codecvt_utf8`进行编码转换，确保正确读取和解析词典文件。
- **分词算法的边界情况处理**：在分词过程中，遇到一些无法通过词典匹配的字符序列，导致分词失败。通过在分词函数中增加对未匹配词语的处理，确保程序能够处理所有输入情况。
- **内存泄漏问题**：在动态分配内存的过程中，未正确释放节点和树结构，导致内存泄漏。通过完善析构函数，确保所有动态分配的内存能够被正确释放。

## 4.2 算法的时空分析

### 4.2.1 最短路径分词：

最短路径匹配分词算法的时间复杂度主要取决于输入文本的长度和词典的大小。具体分析如下：

- **时间复杂度**：假设输入文本长度为n，词典中最大词长度为。在最坏情况下，每个位置都需要检查最长m个字符的子串，因此时间复杂度为O(n×m)
- **空间复杂度**：前缀树的空间复杂度为词典中所有词汇的总长度，记为O(T)，其中T为词典中所有词汇字符数的总和。分词过程中使用的辅助数组`min_weight`和`split_pos`的空间复杂度为O(n)。

总体而言，算法在处理大规模文本和词典时，时间和空间的消耗较为合理，但在词典非常庞大或输入文本非常长时，可能需要进一步优化。

### 4.2.2 N最短路径分词：

- **时间复杂度**:
  - 动态规划阶段：`O(L^2 * K)`，其中 `L` 是句子长度，`K` 是单个位置的最大路径数。
  - 排序阶段：`O(K log K)`，每个位置可能排序 `K` 次。
- **空间复杂度**:
  - `O(L * K)`，用于存储 `split_pos`。

# 5. 用户使用说明

本节将详细说明如何运行编写的中文分词程序。

## 5.2 运行步骤

1. **编译程序**：
   - 打开终端或命令行窗口。
   - 进入程序文件所在的目录。
   - 输入编译命令，例如：

```
g++ Shortest.cpp -o s
```

2. **运行程序**：
    - 在命令行中输入运行命令：

```
./s
```

    - 按提示输入词典文件的路径，例如：

```
请输入词典地址：./dict.txt
```

    - 选择操作编号：

```
请选择以下操作：
 [1]: 重新选择词典
 [2]: 输入句子进行分词
 [3]: N最短路径分词
 [-1]: 退出程序
========================================
请输入操作编号：
```

    - 根据需要选择相应的操作，输入句子进行分词或重新加载词典。

# 6. 测试结果

以下列出了程序的测试用例，包括输入、预期输出和实际输出结果。

## 6.1 测试用例1

- **输入**：

```
我是人大学生
```

- **最短路径输出**：

```
我\是\人大\学生\
```

- **N-最短路径输出**：

```
我\是\人大\学生\
我\是\人大\学\生\
我\是\人\大学生\
```

- **结果**：通过

## 6.2 测试用例2

- **输入**：

```
这儿是我们祖先发祥之地今天我们又在这儿团聚民族的命运全担在我们双肩抗日救亡要我们加倍努力忠诚团结紧张活泼战斗的学习努力努力争取国防教育的模范努力努力锻炼成抗战的骨干我们要忠实于民族解放事业我们献身于新社会的建设昂头看那边胜利就在前面
```

- **最短路径输出**：

  这儿\是\我们\祖先\发祥之地\今天\我们\又\在\这儿\团聚\民族\的\命运\全\担\在\我们\双肩\抗日救亡\要\我们\加倍努力\忠诚\团结\紧张\活泼\战斗\的\学习\努力\努力争取\国防教育\的\模范\努力\努力\锻\炼成\抗战\的\骨干\我们\要\忠实\于\民族解放\事业\我们\献身\于\新\社会\的\建设\昂头\看\那边\胜利\就\在\前面\

- **N-最短路径输出**：

  这儿\是\我们\祖先\发祥之地\今天\我们\又\在\这儿\团聚\民族\的\命运\全\担\在\我们\双肩\抗日救亡\要\我们\加倍努力\忠诚\团结\紧张\活泼\战斗\的\学习\努力\努力争取\国防教育\的\模范\努力\努力\锻\炼成\抗战\的\骨干\我们\要\忠实\于\民族\解放事业\我们\献身\于\新\社会\的\建设\昂头\看\那边\胜利\就\在\前面\

  这儿\是\我们\祖先\发祥之地\今天\我们\又\在\这儿\团聚\民族\的\命运\全\担\在\我们\双肩\抗日救亡\要\我们\加倍努力\忠诚\团结\紧张\活泼\战斗\的\学习\努力\努力争取\国防教育\的\模范\努力\努力\锻\炼成\抗战\的\骨干\我们\要\忠实\于\民族解放\事业\我们\献身\于\新\社会\的\建设\昂头\看\那边\胜利\就\在\前面\

  这儿\是\我们\祖先\发祥之地\今天\我们\又\在\这儿\团聚\民族\的\命运\全\担\在\我们\双肩\抗日救亡\要\我们\加倍努力\忠诚\团结\紧张\活泼\战斗\的\学习\努力\努力争取\国防教育\的\模\范\努力\努力\锻\炼成\抗战\的\骨干\我们\要\忠实\于\民族\解放事业\我们\献身\于\新\社会\的\建设\昂头\看\那边\胜利\就\在\前面\

- **结果**：通过