最短路径中文文本分词实验报告

姓名：王松宸　学号：2024201594

2025 年 12 月 22 日

1　系统概述与功能亮点

1.1　简介

我设计并实现了一个高性能的中文分词系统，该系统基于大规模词典和有向无环图，采用 N-最短路径算法对中文句子进行切分。系统不仅能够输出最优的分词结果，还能根据用户需求输出概率最高的 N 条分词路径。

1.2　功能亮点

程序在完成所有基本功能的同时，**完整实现了实验规定的拓展内容**，亮点如下：

- **高效词典管理：**
  - 实现了基于链地址法的哈希表，采用 BKDR Hash 算法，支持动态扩容，能够高效加载和查询数十万条词条的大规模词典。
  - 支持词频和词性信息的存储，后续用于权重调整。
- **智能分词算法：**
  - **DAG 构建**：根据输入句子构建词语切分有向无环图，边权结合了词频概率与词性权重。
  - **N-最短路径**：实现了基于动态规划（DP）的 N-Best 算法，能够计算并回溯出前 N 条最优路径，而非仅仅是单条最短路径。
  - **词性加权**：设计了精细的词性权重调整策略，针对成语、专有名词、副词等进行降权（优先匹配），对虚词进行惩罚，有效解决了歧义切分问题。
- **美观的交互界面：**
  - 设计了基于 CLI 的图形化菜单，使用 ASCII 字符构建边框，提升了系统的专业感。
  - 引入 ANSI 颜色控制，对不同类型的输出（如菜单、提示符、错误信息、分词结果）进行颜色区分，极大地增强了可读性和用户体验。

1.3　输入与输出内容

- **输入**：用户交互式输入的中文句子及 N 值（路径数量）。
- **输出**：按概率从高到低排序的 N 条分词结果，词与词之间用“/”分隔。

2 程序架构

2.1 词典模块

该模块负责管理核心词库数据，主要包含初始化和查询两个阶段：

- **初始化**：读取 `dict_big.txt`，解析每行的词语、词频和词性，将其插入哈希表中。
- **查询**：提供高效的字符串匹配接口，用于在图构建过程中快速判断子串是否成词。

2.2 图构建模块

该模块负责将线性的句子转化为结构化的图模型：

- **节点映射**：将句子中的每个字符位置映射为图的一个节点。
- **边生成**：遍历句子的所有子串，若子串在词典中存在，则在对应起止位置之间建立一条有向边。
- **权重计算**：边的权重由词频概率（$-\ln P$）和词性系数共同决定，实现了语义感知的路径规划。

2.3 算法核心模块

该模块实现了 N-最短路径的核心逻辑：

- **拓扑排序与 DP**：利用 DAG 的拓扑特性，按顺序遍历节点，计算到达每个节点的前 N 个最短路径代价。
- **路径回溯**：根据记录的前驱节点信息，从终点反向回溯出完整的切分路径。

2.4 前端交互模块

该模块负责与用户进行交互，提供友好的操作界面：

- **界面渲染**：利用 ANSI 转义序列实现控制台文本的颜色高亮和样式控制（如加粗），绘制美观的菜单边框。
- **交互逻辑**：封装了清晰的主循环逻辑，支持菜单选择、输入验证、清屏刷新等功能，确保用户操作流畅。
- **结果展示**：将分词结果以高亮形式输出，清晰区分路径编号与分词内容。

3 核心设计

3.1 数据结构定义

```
// 哈希表节点
typedef struct DictNode {
    char *word;             // 词
    int frequency;          // 词频
```

```
    char *pos;                  // 词性
    struct DictNode *next;      // 链表指针
} DictNode;

// 图的边结构
struct Edge {
    int to;                 // 指向的节点索引
    double weight;          // 权重
    char word[128];         // 词的内容
    char pos[64];           // 词性
    struct Edge *next;      // 邻接表指针
};
```

3.2 关键算法实现

3.2.1 基于 BKDR Hash 的词典索引

为了在构建图时快速查找词语，在 AI 的建议下采用了经典的 BKDR Hash 算法：

- 选用 seed 为 131，能够极大地减少哈希冲突。
- 仿照教材采用哈希容量递增表，实现了自动扩容机制，当负载因子超过 0.95 时自动扩大哈希表容量并重哈希，保证查找效率维持在 $O(1)$ 水平。

3.2.2 N-最短路径算法

本系统利用了分词图的有向无环（DAG）特性，采用动态规划实现：

- **状态定义**：$D[i][k]$ 表示到达第 i 个字符位置的第 k 短路径的长度。
- **状态转移**：对于节点 i 的每一条出边 (i, to)，尝试将其扩展到 $D[to]$ 的候选路径集中。
- **有序插入**：在更新 $D[to]$ 时，采用插入排序的思想，维护一个大小为 N 的有序数组，始终保留当前最优的 N 个解。

3.2.3 基于词性的权重系数

为了解决中文分词中的歧义问题（如“的确”被切分为“的/确”），设计了 `GetPosMultiplier` 函数：

- **优先匹配**：对习用语（i）、成语（l）、专有名词（nr/ns/nt）赋予极低的权重系数，使其在路径竞争中占据优势。
- **歧义修正**：将副词（d）的权重系数设为 0.6，鼓励“的确”等副词作为一个整体被切分。
- **虚词惩罚**：对助词（u）、介词（p）等高频虚词赋予 1.2 的权重系数，防止长词被过度切碎。

4 调试分析

4.1 歧义切分问题的解决

- **问题**：在测试句子“这的确是一个好主意”时，系统倾向于将“的确”切分为“的/确”，因为“的”字频率极高，导致其路径权重较小。
- **解决**：引入了词性加权机制。通过查阅词典发现“的确”的词性为副词（d），因此在代码中特判了副词的权重（0.6），同时对单字虚词（u）施加惩罚（1.2）。调整后，系统成功输出了“这/的确/是/...”的正确结果。同时也借此对其他词性进行了合理的权重分配，提升了整体分词准确率。

4.2 malloc 与 new 的选择

- **问题**：在创建各种数据结构时，我更倾向于选择更熟练使用的 `malloc` 进行内存分配。然而，这也让我在释放内存部分中遇到了各种问题，导致好多次的异常报错。
- **解决**：
  - 二维及以上的动态数组，采用多重循环逐层释放，确保每一层的内存都被正确释放。
  - 下次最好使用 `new` 和 `delete`，以避免手动管理内存时的复杂性和潜在错误。

5 功能测试

5.1 测试结论

经过对多种类型的中文句子（包括日常用语、成语、长难句）进行测试，得出结论：

- 系统启动迅速，词典加载稳定。
- 分词准确率高，能够正确处理常见的歧义组合。
- N-最短路径功能正常，能够输出合理的备选分词方案。

5.2 功能测试报告

详细测试报告请参阅 *功能测试报告*`.pdf`。

5.3 使用手册

详细的操作指南请参阅 *使用手册*`.pdf`。

6 附录

- `lab4.cpp`
- 功能测试报告.pdf
- 使用手册.pdf