实验 04 最短路径中文文本分词

罗毅凡

学号：2024202715

2025 年 12 月 20 日

1 需求分析

1. **核心功能**：输入一段中文文本，对其进行分词，并将分词结果输出。
2. **交互设计**：设计命令行交互界面或演示程序，便于用户操作。
3. **拓展功能**：实现 N-最短路径（N-Shortest Path）分词，提供多种可能的切分结果。

2 概要设计

为实现上述中文文本分词功能，系统流程设计如下：

1. **词典加载**：读取词表，构建哈希表存储词典。
2. **图构建**：对输入文本进行处理，构建有向无环图（DAG）结构。
3. **路径搜索**：使用 Dijkstra 算法或动态规划算法计算从句首到句尾的最短路径（或 Top-N 最短路径），得到分词结果并输出。

2.1 数据结构定义

2.1.1 1. 哈希表数据类型

用于高效存储和检索词典数据：

```
typedef struct WordNode {
    char word[MAX_WORD_LEN];
    int freq;
    char pos[MAX_POS_LEN];
    struct WordNode *next;
} WordNode;
```

```

typedef struct {
    WordNode* buckets[TABLE_SIZE];
    int total_count;
    long total_freq;
} WordDict;
```

2.1.2 2. 图的数据类型

用于表示文本的切分路径：

```
typedef struct Graph{
    double **adj;        // 邻接矩阵存储边权重
    int nodes_num;       // 节点数量
    char *source;        // 源文本
} Graph;
```

2.2 模块划分

本程序包含以下四个主要模块：

- **(a) 主程序模块**：设计命令行交互，接收中文文本，输出分词结果。
- **(b) 哈希表词典构建模块**：读取外部词表文件，构建内存中的哈希表。
- **(c) 文本图结构构建模块**：解析输入文本，根据词典匹配构建图结构。
- **(d) 最短路径查找分词模块**：实现核心算法（Dijkstra/DP），计算最优切分路径。

3 详细设计

3.1 哈希表词典构建

将词表中的词及其频率依次读取插入到哈希表中。哈希函数选用经典的 **djb2 算法**，并使用链地址法处理哈希冲突。

Listing 1: djb2 哈希算法实现

```
unsigned long hash_func(const char *str) {
    unsigned long hash = 5381;
    int c;
    while ((c = *str++))
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
```

```
    return hash % TABLE_SIZE;
}
```

3.2 文本图结构构建

3.2.1 实现逻辑

1. 图的节点数设置为“输入中文文本字数 + 1”，包含开始和结尾的间隔。
2. 遍历输入文本的所有子串：若词典中包含该子串，则在对应的起始位置 i 和终止位置 j 的节点间添加一条有向边。
3. **权重计算**：为了将“最大概率乘积”问题转化为“最短路径”问题，边的权重定义为：

$$w = -\ln(P(w_i)) \approx \ln(\text{TotalFreq}) - \ln(\text{Freq}(w_i)) \tag{1}$$

3.2.2 代码实现

Listing 2: 图构建核心代码

```
Graph* build_graph(char* s, WordDict* dict) {
    int len = strlen(s);
    Graph* G = (Graph*)malloc(sizeof(Graph));
    G->nodes_num = len + 1;
    G->source = strdup(s);

    // 初始化邻接矩阵
    G->adj = (double**)malloc(sizeof(double*) * G->nodes_num);
    for (int i = 0; i < G->nodes_num; i++) {
        G->adj[i] = (double*)malloc(sizeof(double) * G->nodes_num);
        for (int j = 0; j < G->nodes_num; j++) {
            G->adj[i][j] = INFINITY_WEIGHT;
        }
    }

    char temp_word[MAX_WORD_LEN];
    long total_freq = dict->total_freq > 0 ? dict->total_freq : 1;

    for (int i = 0; i < len; i++) {
        int char_len = get_utf8_len(s[i]);
        // 默认单字路径，防止断路
```

```
        if (i + char_len <= len) { G->adj[i][i + char_len] = 20.0; }

        // 匹配词典中的词
        for (int j = i + 1; j <= len && (j - i) < MAX_WORD_LEN; j++)
           {
            int sub_len = j - i;
            if (sub_len < char_len) continue;
            strncpy(temp_word, s + i, sub_len);
            temp_word[sub_len] = '\0';

            int freq = get_word_freq(dict, temp_word);
            if (freq > 0) {
                G->adj[i][j] = log((double)total_freq) - log((double)
                   freq);
            }
        }
    }
    return G;
}
```

3.3 最短路径查找分词

3.3.1 Top-1 分词：Dijkstra 算法

使用 Dijkstra 算法查找单条最短路径。

Listing 3: Dijkstra 分词伪代码

```
Algorithm Dijkstra_Segmentation(Graph G, int N):
    // 1. 初始化
    Create PriorityQueue Q    // 优先队列 (cost, node_index)
    Array dist[0...N]         // 存储最小代价
    Array parent[0...N]       // 记录路径来源

    Initialize all dist[] = INFINITY, parent[] = -1
    dist[0] = 0
    Q.push(0, 0)

    // 2. 主循环
    While Q is not empty:
        u = Q.pop_min_node()
```

```
        If u is visited: continue
        Mark u as visited

        // 松弛操作：遍历 u 的所有邻居 v（即后续节点）
        For each neighbor v of u in G:
            weight = G.adj[u][v]
            If dist[u] + weight < dist[v]:
                dist[v] = dist[u] + weight
                parent[v] = u
                Q.push(v, dist[v])

    // 3. 回溯 (Backtrack)
    path = []
    curr = N
    While curr != 0:
        prev = parent[curr]
        Add segment (prev -> curr) to path
        curr = prev
    Reverse path
    Return path
```

3.3.2 Top-N 分词：动态规划 (DP)

使用动态规划实现 N-最短路径查找。

Listing 4: N-最短路径 DP 伪代码

```
Algorithm N_Shortest_Path_DP(Graph G, int N, int K):
    // dp[i] 存储到达节点 i 的 Top-K 个路径状态
    Array dp[0...N] of List
    dp[0].add({cost: 0, parent: -1, rank: -1})

    // 1. 线性扫描 (拓扑序)
    For i from 0 to N-1:
        For each neighbor j of i:
            weight = G.adj[i][j]
            // i 的每一条路径都可以延续到 j
            For rank_k from 0 to dp[i].size() - 1:
                prev_path = dp[i][rank_k]
                new_cost = prev_path.cost + weight

```

```
                dp[j].add_candidate({
                    cost: new_cost,
                    parent: i,
                    rank: rank_k
                })

        // 2. 排序与截断
        Sort dp[j] by cost ascending
        Keep only top K elements in dp[j]

    // 3. 回溯
    Return Backtrack(dp, N, K)
```

4 使用手册

4.1 概述 & 核心功能

本系统是一个基于 **N-最短路径 (N-Shortest Path)** 算法的高效中文分词工具。它通过构建有向无环图 (DAG)，利用动态规划思想寻找句子中概率最高的 N 种切分方式，能够有效解决中文分词中的歧义问题。

4.2 操作说明

程序提供交互式的命令行界面 (CLI)。用户启动程序后，即可实时输入待分词的中文文本，程序将即时计算并输出 Top-1 及 Top-N 的分词结果。

5 功能测试报告

5.1 测试概述

本次测试旨在验证算法在处理长句、成语、口语俗语以及歧义文本时的准确性与鲁棒性，并观察 Top-N (N=3) 结果在解决歧义上的作用。

5.2 综合测试记录

表 1: 综合测试用例记录表

ID	测试文本	Top-1 分词结果 (最优解)	判定	备注
T01	曾经有一份真诚的爱情...	曾 经/有/一 份/真诚/的/爱情...	Pass	长句切分准确；“一万年”识别为整体
T02	说好了一辈子，少一年...	说好/了/一辈子，少/一年...	Pass	准确识别“一辈子”、“一个/时辰”
T03	以前我没得选...	以前/我/没/得/选...	Pass	口语“没得选”切分合理
T04	站着把钱挣了！	站/着/把/钱/挣/了/！	Pass	动词“站着”切分为单字（视词典而定）
T05	做人如果没梦想...	做人/如果/没/梦想...	Pass	准确识别“做人”、“咸鱼”等词汇

5.3 详细测试案例分析

5.3.1 案例 T01：《大话西游》经典台词

- **输入**：... 如果非要给这份爱加上一个期限，我希望是：一万年
- **分析**：
  1. **最优解**：... 加上/一个/期限/，/我/希望/是/：/一万年。
  2. **整体识别**：系统成功将“一万年”识别为固定词组，而非“一/万年”，说明在词典中其联合概率更高。
  3. **标点处理**：省略号被逐字节切分，逻辑符合未登录词处理机制。
  4. **歧义处理**：Result 1 选择了“非/要”，表明统计上单字组合优于双字词“非要”。

5.3.2 案例 T02：《霸王别姬》台词

- **输入**：说好了一辈子，少一年、一个月、一天、一个时辰...
- **分析**：

1. **量词处理**：“一辈子”、“一年”、“一个月”均正确识别。
2. **细微差别**：“一个/时辰”被分开，因为“一个”作为高频修饰语独立成词概率极高。
3. **歧义消除**：Result 1 (说好/了) 优于 (说/好/了)，动词短语语义更贴切。

5.3.3 案例 T03：《无间道》台词

- **输入**：以前我没得选，现在我想做个好人
- **分析**：
  - Result 1: 没/得/选
  - Result 3: 没得/选

  此处展示了 N-最短路径的威力。口语“没得”虽然在 Top-1 中被切开，但在 Top-3 列表中被保留，为后续语义分析提供了纠错可能。

5.4 测试结论与评价

本次测试表明，实现的 N-最短路径分词系统功能完备，性能良好：

1. **准确性**：Top-1 结果中，绝大多数常用词、成语和时间短语切分正确。
2. **鲁棒性**：能够稳定处理标点、ASCII 字符及未登录词，未出现崩溃或乱码。
3. **多结果召回**：Top-N 列表成功捕获了细微的歧义（如“没/得” vs “没得”），提升了系统的适用性。

改进建议：在预处理阶段加入对特殊标点符号（如……、——）的规则处理，避免将其切分为单个 ASCII 字符。