Analysis of fsm-lite.cpp

Your Name

2025-08-01

Contents

1 Analysis of `fsm-lite.cpp`

This document provides a line-by-line analysis of `fsm-lite.cpp`, a frequency-based string mining implementation.

1.1 Overview

The code implements a single-core frequency-based substring mining algorithm using:

- **SDSL (Succinct Data Structure Library)** for compressed suffix trees (`cst_sct3`) and wavelet trees (`wt_int`)
- **FASTA** input format support
- Configurable parameters for minimum/maximum pattern length and frequency thresholds

1.2 Includes and Namespace

```
#include "default.h"
#include "configuration.h"
#include "input_reader.h"
#include <sdsl/suffix_trees.hpp> // TODO: replace with csa+lcp array
```

```
#include <sdsl/wt_algorithm.hpp>
#include <iostream>
#include <vector>
#include <cstdlib> // std::exit()
using namespace std;
```

Key includes:

- Project headers (`default.h`, `configuration.h`, `input_reader.h`)
- SDSL components for suffix trees and wavelet trees
- Standard C++ libraries for I/O and data structures

1.3 Type Definitions

```
typedef sdsl::cst_sct3<> cst_t;
typedef sdsl::wt_int<> wt_t;
typedef sdsl::bit_vector bitv_t;
typedef cst_t::char_type char_type;
typedef cst_t::node_type node_type;
typedef wt_t::size_type size_type;
```

Shorthand types for:

- Compressed suffix tree (`cst_t`)
- Wavelet tree (`wt_t`)
- Bit vector (`bitv_t`)
- Character and node types

1.4 3. `wt_init()` Function

Initializes the wavelet tree for sequence labels:

```
void wt_init(wt_t &wt, bitv_t &separator, cst_t &cst, input_reader *ir, configuration &config)
```

1.4.1 Key Steps:

1. **Initialization**:

```
uint64_t n = cst.csa.size();
sdsl::int_vector<DBITS> labels(n, ~0u);
separator = bitv_t(n, 0);
```

2. **Backward Labeling**:

```
uint64_t k = ir->size()-1;
uint64_t j = cst.csa.wavelet_tree.select(1, 0);
```

3. **Validation**:

```
if (j != bwtendpos || k != 0)
```

4. **Wavelet Tree Construction**:

```
wt = wt_t(text_buf, labels.size());
```

1.5 4. `main()` Function

1.5.1 Configuration Setup

```
configuration config(argc, argv);
if (!config.good) config.print_short_usage();
```

1.5.2 Input Reading

```
input_reader *ir = input_reader::build(config);
```

1.5.3 Data Structure Construction

```
cst_t cst;
construct(cst, config.tmpfile + ".tmp", 1);
wt_t label_wt;
bitv_t separator;
wt_init(label_wt, separator, cst, ir, config);
```

1.5.4 Main Processing Loop

```
while (!buffer.empty()) {
    node_type const node = buffer.back();
    buffer.pop_back();
    // ... processing logic ...
}
```

1.5.4.1 Loop Steps:

1. Node processing (depth-first traversal)
2. Length filtering (`minlength`/`maxlength`)
3. Weiner link checks
4. Support calculation
5. Frequency filtering
6. Pattern output

1.5.5 Output Format

```
<pattern> | <file_id1>:<count1> <file_id2>:<count2> ...
```

1.6 5. Key Algorithms

1. **Compressed Suffix Trees**: Efficient pattern storage
2. **Wavelet Trees**: Label management
3. **Backward Traversal**: Initial labeling
4. **Weiner Links**: Tree navigation
5. **Frequency Filtering**: Threshold enforcement

1.7 6. Dependencies

- Requires SDSL-lite v2.0.3
- GNU GPL v3 licensed

1.8 7. TODOs (From Code Comments)

1. Optimize time/space usage
2. Add multi-threading
3. Support gzip compressed input
4. Replace suffix tree with CSA+LCP array