Análise detalhada do código fsm-lite.cpp

Profa Dra Helena R. S. D'Espindula

2025-08-01

Contents

0.1 Introdução

O programa `fsm-lite.cpp` implementa uma solução para mineração de substrings frequentes em conjuntos de sequências (como genomas). Ele utiliza estruturas de dados comprimidas (Suffix Trees) da biblioteca **SDSL-lite** para identificar substrings que aparecem em múltiplas sequências.

0.2 Objetivo geral

Este programa lê várias sequências, constrói uma **árvore de sufixos comprimida (CST)** e percorre essa estrutura para encontrar **subsequências** que aparecem com uma frequência mínima (em número de amostras).

0.3 Análise linha a linha do código `fsm-lite.cpp`

0.3.1 Inclusão de bibliotecas

```
#include "default.h"
#include "configuration.h"
#include "input_reader.h"
#include <sdsl/suffix_trees.hpp>
using namespace std;
using namespace sdsl;
```

Essas linhas incluem os cabeçalhos do projeto e da biblioteca SDSL. Os namespaces `std` e `sdsl` são utilizados para evitar repetições como `std::string` ou `sdsl::cst_sada`.

0.3.2 Função principal

```
int main(int argc, char **argv)
```

Essa é a função que inicia o programa. `argc` e `argv` armazenam os argumentos passados pela linha de comando.

0.3.3 Leitura da configuração

```
configuration config(argc, argv);
```

Cria um objeto `config` que interpreta os parâmetros fornecidos pelo usuário.

0.3.4 Leitura de entrada

```
input_reader * reader = input_reader::build(config);
reader->read_input(config);
```

Instancia um leitor para ler os arquivos informados e prepara os dados para análise, concatenando as sequências.

0.3.5 Construção da CST

```
cst_sada<> cst;
construct_im(cst, reader->text, 1);
```

Constrói a árvore de sufixos comprimida (CST) a partir das sequências. Esta estrutura permite buscas rápidas por substrings.

0.3.6 Inicialização de variáveis

```
size_t num_seqs = reader->total_seqs();
size_t min_support = config.m;
```

Obtém o número de genomas e o número mínimo de genomas nos quais um k-mer deve aparecer.

0.3.7 Vetores auxiliares

```
vector<size_t> labels(cst.size(), num_seqs);
vector<bool> seen(num_seqs, false);
```

`labels`: armazena o rótulo (genoma) de cada sufixo.
`seen`: garante que cada genoma seja contado uma única vez por sufixo.

0.3.8 Etapa de marcação (labeling)

```
for (size_t j = 0; j < cst.size(); ++j) {
    size_t id = reader->seq_id(j);
    if (!seen[id]) {
        for (size_t i = j; i > 0; i = cst.parent(i)) {
            if (labels[i] != num_seqs) break;
            labels[i] = id;
        }
        seen[id] = true;
    }
}
```

Marca cada nó da árvore com o ID do genoma correspondente, propagando essa marcação para os nós pais.

0.3.9 Contagem de suporte (número de genomas por substring)

```
vector<size_t> support(cst.size(), 0);
for (size_t i = 0; i < cst.size(); ++i) {
    if (labels[i] < num_seqs)
        ++support[i];
    if (!cst.is_leaf(i)) {
        for (auto it = cst.children(i); it != cst.children_end(i); ++it)
            support[i] += support[*it];
    }
}
```

Calcula quantos genomas possuem cada substring (nó da árvore), somando os suportes das folhas e dos filhos.

0.3.10 Impressão dos k-mers válidos

```
for (size_t i = 0; i < cst.size(); ++i) {
    if (cst.depth(i) >= config.s && cst.depth(i) <= config.S && support[i] >= min_support) {
        string kmer;
        for (auto v = i; cst.depth(v) > 0; v = cst.parent(v)) {
            auto label = cst.edge(v);
            kmer = string(label.first, label.second) + kmer;
        }
        cout << kmer << endl;
    }
}
```

Para cada nó: - Verifica se está dentro do tamanho permitido. - Reconstrói a substring subindo na árvore. - Imprime a substring.

0.3.11 Finalização

```
delete reader;
return 0;
```

Libera recursos usados e encerra o programa.

0.4 Conclusão

O programa `fsm-lite` é eficiente para identificar substrings frequentes entre múltiplas sequências, usando estruturas de dados comprimidas. Ele é indicado para análise de grandes volumes de dados biológicos como genomas ou metagenomas.