Relatório de Introdução a Teoria da Computação

Alexandre Lima | 11797038
Erick Barcelos | 11345562

Tech Stack

Para desenvolver o que foi proposto no enunciado do trabalho, utilizamos da linguagem Python, juntamente de suas bibliotecas padrões para fins de facilitar a implementação da máquina de estados e o processamento de cadeias.

Classe de teste (extra)

Além do código fonte, implementamos um código voltado para testagem de nosso trabalho, agilizando verificações de consistência dos métodos da classe `FiniteAutomaton`, usada para representar o autômato finito. Ambos estarão anexados ao trabalho no momento da entrega.

Métodos e Classes

Classe `FiniteAutomaton`:

Classe usada para representar o autômato finito, possuindo como atributos:

- `states`**:** Lista (string) de estados do autômato
- `alphabet`**:** Lista (string) do alfabeto
- `transitions`**:** Dicionário com as transições de cada estado e seu respectivo símbolo terminal
- `initial_state`**:** Estado inicial, que por default foi setado como ‘0’
- `final_states`**:** Lista (string) de estados finais

Métodos implementados na classe são os seguintes:

- `__init__`**:** Construtor da classe que define os atributos
- `transition(self, state, symbol)`**:** Método que performa a transição dos estados
- `accepts(self, string)`**:** Método que verifica se a cadeia é aceita pelo autômato finito

A escolha de usar uma classe foi feita a fim de que se tenha um melhor controle do domínio da abstração de um autômato finito, possuindo métodos e atributos que performam seu comportamento.

Método `read_automaton`:

O Método `read_automaton` tem a função de fazer a leitura dos dados necessários para construir o objeto FiniteAutomaton como retorno, seguindo todas as especificações definidas para o projeto.

Método `read_strings`:

O método `read_strings` tem a função de fazer a leitura da lista de cadeias que serão validadas pelo autômato finito.

Qualidade da solução implementada

- **Clareza e legibilidade:** O código é estruturado de forma clara e bem comentada, facilitando a compreensão do que cada parte faz.
- **Modularidade:** A implementação utiliza classes e funções bem definidas para separar as diferentes partes do código, como a definição da classe `FiniteAutomaton` para representar o autômato finito e funções separadas para a leitura do autômato e das strings de entrada.
- **Robustez:** Foram incluídos tratamentos de erro para lidar com entradas inválidas, como número inválido de estados, número de transições acima do limite, entre outros. Isso aumenta a robustez do programa e ajuda a evitar falhas inesperadas.
- **Eficiência:** A implementação é eficiente em termos de tempo e espaço, pois utiliza estruturas de dados adequadas para armazenar o autômato e realizar as transições.
- **Testes automatizados:** Para garantir o correto funcionamento do código em diferentes cenários.

Uso do espaço e tempo

Espaço

Armazenamento do autômato:

- Os estados do autômato são armazenados em uma lista `states`, que consome espaço proporcional ao número de estados.
- O alfabeto é armazenado em uma lista `alphabet`, com espaço proporcional ao tamanho do alfabeto.
- As transições são armazenadas em um dicionário `transitions`, onde as chaves são os estados e os valores são outros dicionários representando as transições. O espaço utilizado aqui é proporcional ao número de estados e ao tamanho do alfabeto.
- O estado inicial e os estados de aceitação são armazenados como strings ou em listas, ocupando um espaço constante.

Leitura das Strings de Entrada:

As strings de entrada são lidas e armazenadas em uma lista, onde cada string ocupa espaço proporcional ao seu tamanho. O espaço total utilizado depende do número e do tamanho das strings de entrada.

Variáveis temporárias e estruturas de controle:

Durante a execução do programa, são utilizadas variáveis temporárias para armazenar estados intermediários, símbolos de entrada, etc. O espaço utilizado por essas variáveis é mínimo e geralmente negligenciável em comparação com o restante da estrutura de dados.

Tempo

Leitura e Inicialização do autômato:

A leitura e inicialização do autómato têm uma complexidade de tempo de O(n), onde 'n' é o número total de estados, devido ao processamento necessário para ler e armazenar as informações do autómato.

Execução das Transições:

Para cada símbolo em cada string de entrada, é necessário realizar uma transição no autômato. Seja 'm' o número total de transições no autômato e 'k' o número total de símbolos em todas as strings de entrada. O tempo de execução é O(m * k).

Verificação de Aceitação:

Após executar todas as transições para uma determinada string de entrada, a verificação se o estado final está entre os estados de aceitação é uma operação de tempo constante, portanto, O(1).

Tratamento de Erros e Validação:

O tratamento de erros e validação das entradas do usuário adiciona um pequeno custo de tempo de execução, mas é insignificante em comparação com outras operações. Portanto, pode ser considerado como O(1).