Anaísa Forti da Fonseca
11811ECP012

Trabalho 01 - Autômatos Finitos

Neste primeiro laboratório, foi solicitada a implementação de um Simulador Universal de Autômatos Finitos Determinísticos (AFDs) e Não-Determinísticos (AFNs). O programa aceita a especificação de um AFD ou AFN (sem transições vazias) e, dentre uma lista de cadeias de caracteres dada, diz quais pertencem e quais não pertencem à linguagem reconhecida pelo autômato finito.

As entradas consistem nas informações necessárias para montar o autômato, tais como o número de estados, os símbolos terminais (alfabeto), os estados de aceitação e as transições. Juntamente às especificações do autômato, são fornecidas as cadeias de entrada. A saída trata de dizer se tais cadeias pertencem ou não à linguagem do autômato, imprimindo "aceita" ou "rejeita" para cada cadeia que é analisada.

O código interpreta as entradas fornecidas pelo usuário com as informações para montar o autômato e armazena estas informações em variáveis e em estruturas de dados como a lista (list). Cada transição do autômato é uma lista na qual a primeira posição é o estado de origem, a segunda posição é o símbolo e a terceira posição é o estado de destino, Todas as transições são armazenadas em uma lista. O mesmo raciocínio vale para as cadeias: cada uma é uma lista e todas são armazenadas em uma lista (lista de listas).

Por sua simplicidade e rapidez em desenvolvimento, a solução foi implementada em python. O programa possui a função "percorreCadeia" que recebe como parâmetros uma cadeia dentre as fornecidas pelo usuário e o estado inicial do autômato que, neste laboratório, é sempre o 0. Esta é uma função que, para cada caractere da cadeia sendo percorrida, verifica todas transições possíveis por meio de recursividade. Por poder ter mais de uma transição possível, a cadeia pode percorrer sequências de transições diferentes e, portanto, ter múltiplos estados finais quando é completamente percorrida.

Quando atinge o fim da cadeia, a função compara seus estados finais com os estados de aceitação do autômato. Caso algum estado final seja de aceitação, a função retorna "true" e, posteriormente, imprime que a cadeia pertence à linguagem do autômato ("aceita"). Em contrapartida, caso não haja nenhum estado final que seja de aceitação, o programa retornará "false" e imprimirá "rejeita", mostrando que a cadeia não pertence à linguagem do autômato. Para analisar o retorno da função "percorreCadeia", foi utilizado um laço de repetição "for" verificando se, para cada cadeia, o retorno foi "true" ou "false". Com relação ao funcionamento, a solução apresentou-se eficiente retornando o resultado esperado para 100% dos casos de teste, sendo estes alguns retirados da internet e os do próprio sistema de correção.

Para que fosse possível analisar a eficiência do programa em termos de espaço e tempo, utilizou-se duas bibliotecas: Time e Memory Profiler. Para a primeira, basta importá-la no código, declarar variáveis de início e fim (iguais a "$time.time()$") antes e depois da função principal e realizar a subtração $fim - início$. Essa operação resulta no

tempo que a função levou para ser executada, sem levar em conta o tempo necessário para o usuário inserir as entradas. Já para a segunda biblioteca, é necessário importa-la no código e adicionar "$@profile$" logo antes da definição da função da qual deseja-se obter as informações de memória. Como esta funcionalidade precisa obrigatoriamente ser usada em uma função, para fim de testes, o laço de repetição "for" que verifica o retorno de "percorreCadeia" para cada cadeia foi transformado em uma função chamada "verificaAceitacao".

Dois autômatos finitos foram usados como base para fazer esses testes: um AFD e um AFN. As entradas usadas para montar cada um estão definidas a seguir:

```
# AFD
3
2 a b
1 2
6
0 a 0
0 b 1
1 a 2
1 b 0
2 a 1
2 b 2
5
aabab
aaba
bbab
abb
babaa
```

```
# AFN
4
2 a b
1 2
9
0 a 0
0 b 0
0 a 1
1 a 1
1 b 1
1 a 2
0 b 3
3 b 3
3 b 2
5
aababa
bbab
aaab
ab
ba
```

Executando o programa várias vezes, percebe-se que o tempo a cada execução variava para um mesmo autômato. Então, fez-se uma média do tempo que o programa levou para cada autômato acima. O AFD levou, em média, $4{,}21 * 10^{-5}$ segundos e o AFN $4{,}37 * 10^{-5}$ segundos, tendo uma diferença de $0{,}16 * 10^{-5}$ segundos entre eles. Conclui-se, portanto, que o fato de o AFN ter que conferir todas transições para todos os caracteres da cadeia faz com que ele leve mais tempo que um AFD, que possui no máximo uma transição possível para cada caractere da cadeia.

No que diz respeito à memória, executando o programa com os mesmos autômatos finitos definidos acima, percebeu-se uma média de variação de 150MB entre o AFD e o AFN. A biblioteca Memory Profiler também possui a funcionalidade de gerar um gráfico tempoXespaço. A seguir, são apresentados gráficos gerados para o AFD e para o AFN,

respectivamente, em uma das vezes que o programa foi executado (os gráficos não representam a média, apenas mostram as informações da execução em que ele foi gerado):