Simulador de Autômatos Finito Não-Determinístico

Um Autômato Finito não-Determinístico (AFND) é uma máquina de estados finita onde para cada par de estado e símbolo de entrada pode haver vários próximos estados possíveis, ou nenhum estado possível. Isso o distingue do autômato finito determinístico (AFD), onde o próximo estado possível é univocamente determinado, com isso o AFND tem o poder de estar em vários estados ao mesmo tempo. Essa habilidade é expressa com frequência como capacidade de adivinhar algo sobre entrada.

Como em um AFD, o AFND tem um conjunto finito de estados, um conjunto finito de símbolos de entrada, um estado inicial e um conjunto de estados finais, e a função de transição. Na função de transição está a diferença entre um AFD e AFND. Para um AFND, δ é uma função que recebe um estado e um símbolo de entrada como argumentos (igual o AFD), mas retorna um conjunto de zero, um ou mais estados (em vez de retornar um estado como no AFD). Considere a **Figura 1** que apresenta o diagrama de estados de AFND M1 que reconhece todas as palavras sobre o alfabeto $\Sigma=\{a, b\}$ que terminam com *ab*.

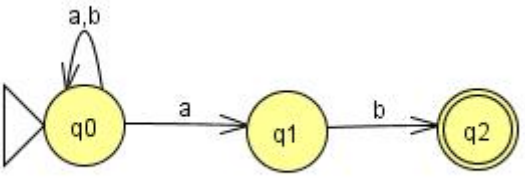

Figura 1: Diagrama de Estados para um automâto finito não-determinístico

A seguir é apresentado a descrição formal (quíntupla) do AFND M1:

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b\}$

$q_0 = q_0$

$F = \{q_2\}$

δ	*a*	*B*
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
q_2	Ø	Ø

A linguagem aceita pelo AFND M1 é dado por:

$L(M1) = \{ W \in \Sigma^* \mid \delta^*(q0, W) \cap F \neq \emptyset\}$

Isto é, L(M1) é o conjunto de todas as palavras W tal que o resultado da intersecção de $\delta^*(q_0, W)$ com F é diferente de vazio, pois estamos comparando conjuntos.

Objetivo

O objetivo deste trabalho é escrever um programa que tenha como entrada um AFND especificado em um arquivo texto, o programa deve simular qualquer AFND informado pelo arquivo e testar várias palavras sobre alfabeto do AFND. O arquivo texto deverá ter a seguinte configuração:

1. Na primeira linha, os símbolos do alfabeto de entrada (Σ), com letras sempre em minúsculo, em ordem alfabética sempre começando por **a**, e sem espaços em branco entre os símbolos, o tamanho do alfabeto é limitado a **10 símbolos na primeira linha**. Por exemplo se quiser um alfabeto com 5 símbolos como teríamos (abcde) tudo junto.
2. Na segunda linha, o número de **Q** estados, para facilitar a implementação o **estado inicial** será sempre o primeiro estado (q0) e qualquer AFND terá no **máximo 20 estados** (q0, q1, ... q19), ou seja, |Q|<=20.
3. Na terceira linha temos um número F indicando a quantidade de estados finais e na linha seguinte (quarta linha) a lista dos estados finais separados por espaço em branco na mesma linha, considere que **F <= Q**.
4. Na quinta linha temos o número de **N** transições do AFND, e nas **N** linhas seguintes as transições especificadas no seguinte formato, separadas por espaço em branco:

   <estado corrente> branco <símbolo em Σ > branco <estado chegada>
5. Ao final das **N** transições, teremos uma linha com um inteiro **T** especificando o número de palavras que deverão ser testadas no AFND, as palavras estarão cada uma em uma linha contendo somente os símbolos do alfabeto de entrada (Σ). Considere também, que cada palavra terá **no máximo 100 caracteres**.

Saída

O programa deverá, conforme especificado no arquivo de entrada, processar a as palavras, usando AFND especificado, e depois, escrever o resultado na tela com as seguintes sentenças: palavra processada e a informação "OK" caso do AFND tenha parado o processamento no estado de aceitação e "not OK" caso contrário.

Considere que termos as seguintes restrições para cada conjunto:

Exemplo de entrada

```
ab
3
1
2
4
0 a 0
0 a 1
0 b 0
1 b 2
5
aab
aababab
ababb
bbaa
b
```

Exemplo de saída

```
1: aab OK
2: aababab OK
3: ababb not OK
4: bbaa not OK
5: b not OK
```

Observações importantes:

O programa deve estar bem documentado e pode ser feito em grupo de até **2 alunos**, não esqueçam de colocar o **nome dos integrantes** do grupo no arquivo fonte do trabalho. O trabalho será avaliado de acordo com os seguintes critérios:

- Funcionamento do programa;
- O trabalho deve ser desenvolvido na **linguagem C** e será testado usando o compilador do **CodeBlocks**, ao final o programa deve ser finalizado com retorno igual a 0.
- O quão fiel é o programa quanto à descrição do enunciado;
- Indentação, comentários e legibilidade do código;
- Clareza na nomenclatura de variáveis e funções;

Para auxiliar na documentação do código e entendimento do que é um programa com boa legibilidade siga as dicas apresentadas nas páginas abaixo para:

- http://www.ime.usp.br/~pf/algoritmos/aulas/layout.html
- http://www.ime.usp.br/~pf/algoritmos/aulas/docu.html

Como este trabalho pode ser feito em **grupo**, evidentemente você pode "*discutir*"" o problema dado com outros **grupos**, inclusive as "*dicas*"" para chegar às soluções, mas você deve ser responsável pela solução final e pelo desenvolvimento do seu programa.