Universidade Federal da Paraíba
Centro de Informática
Disciplina: Teoria da Computação
Professor: Bruno Bruck

Projeto Final

O projeto final deve ser realizado em grupo de **2 integrantes** e vale 10 pontos, relativos à terceira nota da disciplina. Cada grupo deve escolher um dos temas disponíveis e implementar o projeto em uma das seguintes linguagens: **C/C++, Python, Julia**.

É possível também escolher outros temas caso o grupo tenha alguma ideia interessante. Porém, nesses casos, **é necessário primeiro consultar o professor via e-mail sobre a viabilidade do tema e como será executado**.

Cada grupo deverá gravar uma apresentação de no **máximo 15 min** demonstrando o funcionamento do programa e explicando brevemente o funcionamento do código. No início do vídeo, todos os membros do grupo deverão aparecer no vídeo e se apresentarem por nome. Além disso, é obrigatório que cada membro apresente uma parte do projeto, pois a nota é individual..

O projeto deve ser entregue via SIGAA **até o dia 11 de Dezembro de 2022 às 23:59**. Para a entrega, **somente um** dos integrantes do grupo deve submeter um arquivo .zip contendo o código-fonte do programa e um arquivo .txt com a lista de integrantes da equipe e o link para a apresentação.

Observações:

- Independentemente do tema escolhido, é necessário que seu programa faça uma **verificação de erros** nos dados de entrada;
- É importante que **todos** os integrantes do grupo participem da apresentação pois a nota será individual;
- Entregas fora do prazo ou via outra plataforma que não seja o SIGAA **não serão consideradas**;
- **A tolerância com plágio (cópia) é ZERO**. Quaisquer programas similares ou idênticos terão nota zero independentemente de qual for o original e qual for a cópia. O mesmo vale para códigos copiados de outras fontes.
- Evitem disponibilizar o código-fonte em repositórios públicos como o github ou gitlab.

Universidade Federal da Paraíba
Centro de Informática
Disciplina: Teoria da Computação
Professor: Bruno Bruck

Tema 1: Simulação de um Autômato Finito Não-determinístico

Nesse tema o grupo deve desenvolver um programa para simular o processamento de Autômatos Finitos Não-determinísticos (AFNs). O programa deve receber como entrada um arquivo contendo a descrição do AFN no seguinte formato:

```
alfabeto=a,b,c,d          # Lista de símbolos do alfabeto aceito pelo autômato
estados=q0,q1,q2          # Lista de estados no autômato
inicial=q0                # Indica qual é o estado inicial
finais=q1,q2              # Especifica os estados finais do autômato.
transicoes
q0,q1,a                   # Representa uma transição de q0 para q1 com o símbolo "a"
q1,q2,epsilon             # Transição de cadeia vazia de q1 para q2
...
```

Note que é possível que o autômato não tenha estados finais e o simulador deve ser capaz de levar isso em consideração. Além disso, é obrigatória a utilização do formato de arquivo especificado acima.

Após interpretar o arquivo de entrada, o programa deve perguntar ao usuário qual cadeia ele deseja utilizar como entrada para o autômato, realizar a simulação de todas as possíveis formas que aquela cadeia pode ser processada pelo autômato. Como saída, deve ser apresentado ao usuário todas as sequências de processamento testadas, assim como alguma indicação do resultado de cada uma (aceita ou rejeita). Essa simulação pode ser implementada utilizando ou não *threads*, ficando a critério do grupo.

Importante: Atentar para o fato de que o simulador deve ser capaz de lidar com autômatos que contenham *loops* compostos por transições de cadeia vazia sem entrar em loop infinito!

Universidade Federal da Paraíba
Centro de Informática
Disciplina: Teoria da Computação
Professor: Bruno Bruck

Tema 2: Gerador de cadeias para Gramáticas Livres de Contexto

O grupo que escolher esse tema deve implementar um gerador de cadeias para Gramáticas Livres de Contexto (GLCs). Nesse programa, dado uma GLC, o programa deve ser capaz de gerar cadeias (a pedido do usuário) que fazem parte da linguagem associada à gramática. A gramática deve ser fornecida ao programar por meio de um arquivo no seguinte formato:

```
terminais=a,b,c,d          # Lista de símbolos terminais
variaveis=S,A,B            # Lista de símbolos não-terminais (variáveis)
inicial=S                  # Indica o não-terminal inicial

producoes
S-> aA                     # Cada produção deve ser colocada em uma linha diferente
S-> bB
A-> epsilon
...
```

Note que só é permitido a utilização de símbolos terminais e não-terminais com um **único caractere**, e que a palavra reservada "epsilon" deve ser interpretada como cadeia vazia. Além disso, é obrigatória a utilização do formato de arquivo especificado acima.

O gerador deve possuir **2 modos de funcionamento**: rápido e detalhado. No modo **rápido**, à medida em que o usuário solicita, o programa gera uma nova cadeia e mostra a derivação *mais à esquerda* que foi utilizada para gerar a mesma. Note que, caso a gramática seja ambígua, é possível que uma mesma cadeia possa ser gerada por múltiplas derivações *mais à esquerda*. Entretanto, o programa não deve mostrar a mesma derivação mais de uma vez, o que implica que a geração da cadeia não pode ser feita de forma aleatória. Já no modo **detalhado**, o programa funciona quase como um jogo para realizar uma única derivação. Essa derivação inicia pelo não-terminal inicial e, a cada etapa, o programa mostra ao usuário quais produções podem ser utilizadas para a próxima substituição e o deixa escolher uma delas. Note que, a próxima substituição deve ser sempre a do não-terminal mais à esquerda.

Universidade Federal da Paraíba
Centro de Informática
Disciplina: Teoria da Computação
Professor: Bruno Bruck

Tema 3: Minimizador de Autômatos Finitos Determinísticos

Nesse tema o grupo deve implementar uma aplicação que inicialmente receba como entrada um arquivo .txt contendo a descrição de Autômato Finito Determinístico (AFD) no seguinte formato:

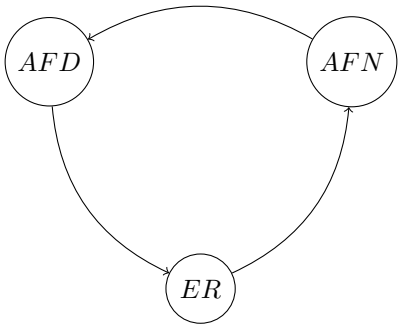
```
alfabeto=a,b,c,d          # Lista de símbolos do alfabeto aceito pelo autômato
estados=q0,q1,q2          # Lista de estados no autômato
inicial=q0                # Indica qual é o estado inicial
finais=q1,q2              # Especifica os estados finais do autômato.
transicoes
q0,q1,a                   # Representa uma transição de q0 para q1 com o símbolo "a"
q1,q2,b                   # Representa uma transição de q1 para q2 com o símbolo "b"
...
```

Em seguida, a aplicação deve utilizar o algoritmo de *Myhill Nerode* (ou *Table Filling Method*) para minimizar o AFD dado como entrada. O algoritmo deve mostrar na tela o passo-a-passo da sua execução e ao final utilizar alguma biblioteca ou software para exibir o diagrama de estados do AFD resultante.

Importante: É necessário verificar se o AFD passado como entrada é de fato válido antes de utilizar o algoritmo de conversão.

Tema 4: Conversor AFD/AFN/ER

Como sabemos, os Autômatos Finitos Determinísticos (AFDs), Autômatos Finitos Não-Determinísticos (AFNs) e as Expressões Regulares (ERs) são todos formalismos que representam o conjunto das Linguagens Regulares. Isso é comprovado pela existência de algoritmos de conversão que demonstram a equivalência de tais formalismos. Mais especificamente, o caminho mais fácil de conversão é como a seguir.



Sendo assim, o grupo deve implementar uma aplicação onde o usuário possa, inicialmente, passar como entrada qualquer um dos 3 formalismos. No caso de AFDs/AFNs o seguinte formato de arquivo deve ser utilizado:

```
alfabeto=a,b,c,d          # Lista de símbolos do alfabeto aceito pelo autômato
estados=q0,q1,q2          # Lista de estados no autômato
inicial=q0                # Indica qual é o estado inicial
finais=q1,q2              # Especifica os estados finais do autômato.
transicoes
q0,q1,a                   # Representa uma transição de q0 para q1 com o símbolo "a"
q1,q2,b                   # Representa uma transição de q1 para q2 com o símbolo "b"
...
```

Já para o caso em que o usuário queira passar como entrada uma ER, iremos utilizar o seguinte formato de arquivo:

```
alfabeto=a,b,c,d          # Lista de símbolos do alfabeto
expressao= a|bc*
```

Uma vez carregado o conteúdo do arquivo, o usuário poderá escolher em qual outro formalismo deseja converter. Sendo assim, o grupo deve implementar três algoritmos de conversão (i.e., $AFN \rightarrow AFD$, $AFD \rightarrow ER$ e $ER \rightarrow AFN$) e utilizá-los de acordo com a opção selecionada pelo usuário. Ao final deve ser exibido o resultado da conversão.