



Trabalho Final

DDC146 Aspectos Teóricos da Computação 2022-1º

Alunos:

Denner Efisio Emanuel Reis - 201735008

Thomas Santos - 201776034

Igor Westermann Lima - 201876021

Professor:

Gleiph Ghiotto Lima de Menezes

Instalação e execução do programa




Node.js

Certifique-se de que tem instalado o Node.js no seu computador. Caso não tenha, faça o download da versão do seu sistema operacional em <https://nodejs.org/en/download/>.

Downloads

Latest LTS Version: **16.16.0** (includes npm 8.11.0)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features	
 Windows Installer node-v16.16.0-x64.msi	 macOS Installer node-v16.16.0.pkg	 Source Code node-v16.16.0.tar.gz

Windows Installer (.msi)

Windows Binary (.zip)

macOS Installer (.pkg)

macOS Binary (.tar.gz)

Linux Binaries (x64)

Linux Binaries (ARM)

Source Code

32-bit	64-bit
32-bit	64-bit
64-bit / ARM64	
64-bit	ARM64
64-bit	
ARMv7	ARMv8
node-v16.16.0.tar.gz	

Instalando dependências com o npm

Após a instalação, acesse via terminal (Powershell) o local do repositório em seu computador e execute o comando “*npm install*”, conforme imagem abaixo.

```
PS C:\Users\Thomas\Desktop\dcc146-202201\js\src> npm install
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.

added 52 packages, and audited 53 packages in 3s

14 packages are looking for funding
  run 'npm fund' for details

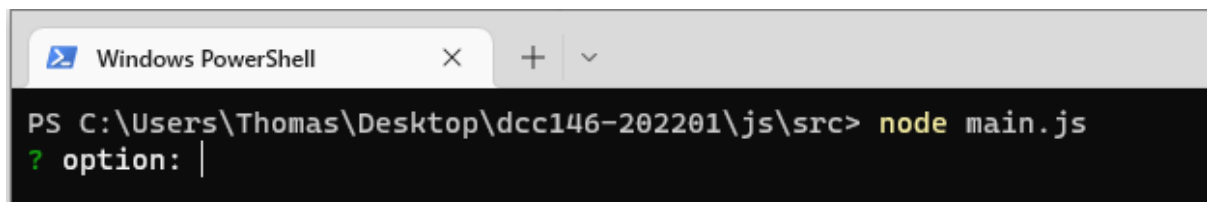
1 high severity vulnerability

To address all issues, run:
  npm audit fix

Run 'npm audit' for details.
PS C:\Users\Thomas\Desktop\dcc146-202201\js\src> |
```

Executando o programa

Após a instalação do Node.js e executar o comando para instalar as dependências via npm, basta acessar o diretório do programa e executar o comando “*node main.js*”



```
Windows PowerShell
PS C:\Users\Thomas\Desktop\dcc146-202201\js\src> node main.js
? option: |
```

Segue abaixo os comandos que poderão ser utilizados e suas devidas funções:

Comando	Descrição	Exemplo
:d	realiza a divisão em tags da string do arquivo informado	:d input.txt
:c	carrega um arquivo com definições de tags	:c tags.lex
:o	especifica o caminho do arquivo de saída para a divisão em tags	:o output.txt
:p	realiza a divisão em tags da entrada informada	:p x=1037
:a	Lista as definições formais dos autômatos em memória	:a
:l	Lista as definições de tag válidas	:l
:q	sair do programa	:q
:s	salvar as tags	:s file.txt

Sobre o programa

O código foi escrito na linguagem Javascript, fazendo uma adaptação da primeira parte que estava em C++. A mudança foi feita devido a dificuldade da equipe de implementar o código em C++ devido a nossa não familiarização com a linguagem e seu uso de ponteiros e dificuldade na criação e utilização dos arrays. O código em C++ se encontra numa branch no repositório Github do trabalho

Após finalizado o trabalho, a equipe tentou reescrever o código em Python, porém devido a falta de tempo devido ao fim do período e dificuldade de adaptação do código, mantivemos em Javascript. O editor de código fonte utilizado para montar o programa e debugar foi o Visual Studio Code da Microsoft.

Estrutura do código

O código foi dividido entre quatro arquivos, sendo eles: **automato.js**, **main.js**, **tags.js** e **transicao.js**.

O arquivo **main.js** é responsável por ler e coordenar os comandos executados pelo usuário.

O arquivo **tags.js** é responsável pelas funções relacionadas a tags, como por exemplo:

- Salvar tags em um arquivo;
- Adicionar tags;
- Validar as tags;
- Listar as tags;
- Dividir as tags

O arquivo **transicao.js** é responsável pelas classes Transicao e Transicoes, que são utilizadas ao longo do programa para a criação de novas transições e inserções na lista de transições.

O arquivo **automato.js** pelas funções relacionadas ao autômato, como por exemplo:

- Montar os autômatos conforme a operação necessária;
- Criar um autômato simples;
- Efetuar a união de dois autômatos;
- Efetuar a concatenação de dois autômatos;
- Criar um fecho de Kleene;

- Retornar o último estado do autômato atual;
- Criar o fecho Lambda;
- Transformar um AFN Lambda para um AFN normal;
- Criar um alfabeto conforme as transições passadas como parâmetro;

Funções e suas utilidades

Funções referentes ao Main

main

Essa é a função principal do programa, responsável por coordenar os comandos executados. Checa se o primeiro caractere é ":" e dependendo do segundo comando direciona para a função correta. Caso tenha algo antes dos ":" é considerado como o nome da TAG e então é adicionada uma nova TAG.

ListarAutomatos

Essa função é responsável por listar na tela os autômatos gerados através das TAGs criadas. É gerado todos os autômatos apresentando a TAG, seus estados iniciais, finais, alfabeto e transições (origem -> destino - símbolo da transição).

ImprimirArquivo

Essa função é responsável por imprimir em um arquivo especificado o resultado das Tags divididas.

Funções referente a Tags

ValidarTag

Essa função é responsável por validar a tag, fazendo uso de uma variável auxiliar para checar se já existe uma tag com o mesmo nome e uma pilha para armazenar os caracteres. A função também diferencia os caracteres especiais, adicionando "\\" quando aparecem.

ListarTagValidas

Essa função é responsável somente por listar no terminal as Tags adicionadas, ou seja, que válidas.

AdicionarTag

Essa função é responsável por adicionar novas tags, checando se existe algum caractere andar do “:” (Nome da Tag) e então escrevendo a nova tag (Descrição da Tag). Por fim, é responsável por chamar a função **MontarAutomato**, que será explicada posteriormente.

SalvarTags

Essa função é responsável por dado um nome de arquivo por parâmetro, escrever nele as tags válidas que foram adicionadas na execução do programa.

ValidarSeConsome

Esta função auxiliar da divisão tem como objetivo percorrer todas as tags já cadastradas e retornar o maior valor dos elementos que cada TAG consegue consumir. Para isto, a função faz análise se o primeiro elemento passado como parâmetro é o elemento inicial no autômato, se é o elemento final do autômato ou um dos demais elementos. Quando não for possível finalizar a divisão o algoritmo retornará -1 e quando for possível consumir a palavra ou parte dela o valor retornado será o índice.

DividirTags

Esta função utiliza o retorno da função auxiliar *ValidarSeConsome* e uma função nativa do JavaScript para remover o número de caracteres de uma palavra tendo como base um valor que representa o total de palavras removidas. O processo será encerrado somente se a palavra for totalmente consumida ou se não for possível consumir a mesma com as tags disponibilizadas como critério.

Funções e Classes referentes a Transição

Classe Transicao

Essa classe é responsável pelo construtor de novas Transições que recebe como parâmetro a Origem, Destino e Símbolo da transição.

Classe Transicoes

Essa classe é responsável por criar um Array de transições em seu construtor. Possui dentro dela a função **newTransicao**. Essa função auxiliar utilizada durante o código em diversas ocasiões é responsável pela criação de uma nova transição e inserção da mesma em uma lista de transições existentes no autômato.

Funções referentes ao Autômato

MontarAutomato

Essa função é responsável por coordenar a montagem dos autômatos de acordo com as operações a serem realizadas, sendo elas: **União**, **Concatenação**, Criação de **autômato simples** e criação do **fecho de Kleene**. A função faz o uso de uma pilha que é responsável por armazenar os autômatos e dependendo da operação realizada tem um funcionamento diferente.

Caso 1: Autômato Simples

É feita a chamada da função **AutomatoSimples** que recebe a tag e o número total de estados do autômato existente, por fim é aumentado o número total de estados e empilhado na Pilha o novo autômato.

Caso 2: União

Os dois últimos elementos da pilha são retirados, é feito um cálculo que retorna o último estado do autômato, que será utilizado como parâmetro para chamar a função **AutomatoUniao**. Por fim, o novo autômato retornado pela função é inserido novamente na pilha.

Caso 3: Fecho de Kleene

É desempilhado o autômato no topo da pilha e passado como parâmetro para a função **CriaFechoKleene**. Por fim, é empilhado na pilha o novo autômato gerado.

Caso 4: Concatenação

Os dois últimos elementos da pilha são retirados e passados como parâmetro na função **AutomatoConcatenacao**. Por fim, o novo autômato é empilhado novamente.

UltimoEstado

Função auxiliar responsável por calcular o último estado do autômato atual, esse valor é utilizado na criação do próximo autômato ou operação. A função ordena de forma crescente as origens e destinos e retorna o maior valor absoluto, ou seja, o último estado do autômato. Feito isso, o resultado é passado como parâmetro para as demais funções de criação do Autômato.

AutomatoSimples

Essa função é responsável por montar um autômato simples, recebendo como parâmetro o símbolo da transição e o último estado do autômato anterior. O estado inicial então se torna o próximo e é criado um novo estado final. É criada uma lista de transições que tem adicionada a transição atual e o alfabeto é atualizado com o símbolo da transição.

AutomatoUniao

Essa função é responsável por fazer a união entre dois autômatos. É criada uma lista de transições que irão absorver as transições de ambos os autômatos a serem unidos. É criado um novo alfabeto que irá absorver o alfabeto de ambos os autômatos.

Para todos os estados finais de ambos autômatos é são criadas transições lambda partindo do estado final atual para um novo estado final passando o símbolo "Lambda".

Para todos os estados iniciais de ambos os autômatos são criadas transições lambda partindo do NOVO estado final com destino ao estado inicial atual. É criado um novo alfabeto retirando os símbolos duplicados. Por fim, é criado um novo autômato com as novas transições, estados e alfabeto.

CriaFechoKleene

Essa função é responsável por criar o fecho de Kleene em um autômato. É armazenado o estado inicial do autômato e é criada uma lista de transições. Para todo estado final é criada uma transição lambda partindo do estado final com destino ao estado inicial passando o símbolo lambda. Esse processo ocorre para todos os estados iniciais do autômato. Por fim é criado um novo autômato com a nova lista de transições, estados onde o estado final é igual ao estado inicial e um novo alfabeto retirando os símbolos duplicados.

AutomatoConcatenacao

Essa função é responsável por fazer a concatenação entre dois autômatos passados por parâmetro. É criada uma nova lista de transições que recebe todas as transições presentes nos dois autômatos. É criado um novo alfabeto que contém o alfabeto de ambos os autômatos. Para todos os estados iniciais do primeiro autômato são criadas transições partindo de cada estado final do segundo autômato para os estados iniciais do primeiro autômato com o símbolo lambda. Por fim, é criado um novo autômato contendo as novas transições, os alfabetos e estados iniciais e finais atualizados.

FechoLambda

Essa função é responsável por criar o fecho lambda a partir do autômato passado como parâmetro. É criada uma lista Fecho que armazena os fechos que serão criados. Os estados repetidos são removidos e ordenados. É feito um filtro das transições que possuem o símbolo Lambda e é criada uma variável auxiliar que irá receber essa lista filtrada. É criada outra variável transições, que recebe um filtro de transições que possuem sua origem igual ao estado em questão.

Para cada estado, é verificado se o tamanho da lista de transições é superior a zero. Cada transição que possui origem igual ao estado é armazenada em uma variável `aux2` e enquanto isso for verdadeiro percorre o `aux2` para cada valor e faz um novo filtro na `aux1` para pegar quando o valor for igual a origem. Para cada valor é procurado o índice cujo valor é igual a -1 e se for o estadoFecho recebe o destino e `aux2` também. Por fim, salva os estados e transição no array `fecho` e passa como parâmetro para a função `afnLambdaParaAfn`.

afnLambdaParaAfn

Essa função é responsável por transformar o AFN Lambda em um AFN. Cria um array a partir dos estados iniciais do autômato recebido como parâmetro e salva na variável `aux1`. Após isso, atualiza o array para um novo array que possui os estados do fecho igual a cada valor do antigo array e retorna um novo array com as transições e remove os subarrays e faz a mesma coisa para os estados finais.

Remove os valores repetidos das variáveis auxiliares de início e fim, cria um array de transições que vai receber as transições AFN. Verifica se as transições já existentes ao filtrar aquelas que possuem origem, destino e símbolo diferentes da próxima transição e para cada valor insere no array transição sua origem, destino e símbolo. Por fim, cria um objeto com os estados, transições e alfabeto e insere na lista de AFNs.

CriarAlfabeto

Essa função é responsável por criar um alfabeto a partir das transições passadas como parâmetro. Para cada transição, insere no alfabeto os símbolos da transição passada como parâmetro e retorna um array sem nenhum símbolo repetido.

Testes executados

- INT: $9+4+2+5+4+*$ (**não será aprovado**) - Ao tentar remover da pilha dos elementos no primeiro "+", só terá o 9, fazendo com que não seja possível a operação
- VAR: $ab+cd+a+dc+$ (**não será aprovado**) - A pilha irá terminar com mais de 1 elemento
- VAR2: $ae+ea+dd+ca.*$ (**não será aprovado**) - A pilha irá terminar com mais de 1 elemento
- VAR3: $ae+ea+dd+ca.*$ (**não será aprovado**) - A pilha irá terminar com mais de 1 elemento
- INT2: $01+2+3+4+5+6+7+8+9+01+2+3+4+5+6+7+8+9+*$.
- A: a^*
- PAR: $ab.ba.+aa.+bb.+*$
- AB: $ab.$