UNIVERSIDADE DE SÃO PAULO
CAMPUS DE SÃO CARLOS
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO (ICMC)

EDUARDO MACIEL DE MATOS 12563821
JOÃO PEDRO RIBEIRO DA SILVA 12563727
JOÃO OTÁVIO DA SILVA 12563748
LEONARDO MINORU IWASHIMA 12534738
NICOLAS DE GÓES 12563780

INTRODUÇÃO A TEORIA DA COMPUTAÇÃO (SCC-0505)

TRABALHO 01: DOCUMENTAÇÃO

São Carlos
2022

1. Estruturação

O código foi todo modularizado em funções, de maneira a simplificar o entendimento e possíveis manutenções. No código em si, cada função foi nomeada e comentada a fim de esclarecer seu funcionamento.

2. Fluxo

A execução do programa segue o seguinte fluxo:

- Inclusão das bibliotecas utilizadas;
- Declaração das structs utilizadas;
- Declaração do cabeçalho das funções utilizadas;
- Criação do autômato;
- Inicialização e leitura das informações do autômato. Isso inclui:
    - Alocação de memória e inicialização dos estados;
    - Alocação de memória e leitura dos símbolos terminais;
    - Leitura dos estados finais; e
    - Alocação de memória e leitura das transições.
- Leitura das cadeias;
- Validação das cadeias através de recursão;
- Destruição do autômato, ou seja, liberação das alocações de memória.

3. Desempenho e alocações de memória

Na construção do projeto, o grupo procurou um meio termo entre desempenho, uso de memória e simplicidade do código. Os estados, os símbolos terminais e as cadeias foram salvos em arrays dinâmicos, que alocam a memória correta em tempo de execução dependendo do seu número de elementos. Por outro lado, os arrays de transições e as strings das cadeias possuem alocação estática, ou seja, com número fixo de elementos independente da execução do programa.

Nos casos onde foram utilizadas alocações dinâmicas, a implementação era simples de ser feita, além de não afetar o desempenho do programa. Já nos casos onde a alocação foi feita de maneira estática, realizar alocações dinâmicas teria uma implementação bem mais complexa, envolvendo diversas realocações de memória, que podem afetar o desempenho do programa se feitas em excesso. Como o enunciado do trabalho já informa o número máximo de transições e o comprimento máximo das cadeias, sendo ambos números pequenos, a utilização de alocação estática foi preferível pelo grupo.

Todas as alocações de memória possuem uma verificação em caso de erro na alocação, interrompendo a execução do programa. Além disso, todas as

alocações são liberadas ao final do programa, não havendo vazamentos de memória.

4. Structs

4.1. Transição:

```
typedef struct transicao_st
{
    char c;
    int indiceDestino;
} TRANSICAO;
```

char c: caractere que representa o símbolo de uma transição.
int indiceDestino: inteiro que representa o índice do estado para o qual a transição leva.

4.2. Estado:

```
typedef struct estado_st
{
    int ehFinal;
    int qtdTransicoes;
    TRANSICAO *transicoes[MAX_TRANSICOES_ESTADO];
} ESTADO;
```

int ehFinal: inteiro que sinaliza se é estado de aceitação;
int qtdTransicoes: inteiro que indica a quantidade de transições que o estado faz.
*TRANSICAO *transicoes[MAX_TRANSICOES_ESTADO]:* vetor de transições com máximo de 50 elementos, que indica todas as transições que este estado atual faz.

4.3. Autômato:

```
typedef struct automato_st
{
    int qtdEstados;
    int indiceEstadoAtual;
    char *simbolosTerminais;
    int qtdSimbolosTerminais;
    ESTADO **estados;
} AUTOMATO;
```

int qtdEstados: quantidade de estados que o autômato vai ler.

int indiceEstadoAtual: índice que indica o estado em que o autômato se encontra no momento.
*char *simbolosTerminais:* caracteres representando os símbolos terminais indicados.
*ESTADO **estados:* vetor de ponteiros para os estados do autômato.

5. Validação de Cadeia

A função de validação de cadeia é um algoritmo recursivo que recebe o autômato e a cadeia que será processada.

O caso base é quando o caractere atual é o *'\0'*, ou seja, a cadeia já foi inteiramente consumida, após a chegada nesse caso, é feita a verificação do estado atual, se ele for um estado de aceitação a cadeia é aceita pelo autômato.

Se a chamada da função não entrou no caso base, é feita a verificação se há alguma transição saindo do estado atual e com o mesmo símbolo que o caractere atual da cadeia, se sim, é feita uma chamada recursiva com o estado atual do autômato e cadeia atualizados (de acordo com a transição). Se não há nenhuma transição com o caractere atual, a função retorna 0.

Ao fim das recursões: Ou o autômato ficou preso em um estado, sem ter consumido a cadeia inteira. Ou a cadeia foi inteiramente consumida, porém o estado atual não é um estado de aceitação. Ou a cadeia foi inteiramente consumida e o estado atual é um estado de aceitação, somente nesse caso, a cadeia é aceita pelo autômato.

6. Discussões

O grupo ficou satisfeito com a solução desenvolvida. A organização e a modularização do código ajudaram no entendimento e na implementação de novas funcionalidades. Apesar de algumas alocações estáticas, a utilização de espaço do trabalho ainda é bastante otimizada, tendo em vista que as poucas informações que foram alocadas estaticamente tratam-se de arrays e strings pequenas. Em relação ao tempo, o desempenho do programa também é bastante otimizado, tendo em vista que ele foi codificado em C. Em caso de não-determinismo, a testagem de todos os caminhos possíveis tem uma complexidade um pouco alta, mas o grupo tentou ao máximo reduzir ciclos desnecessários.