INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
USP SÃO CARLOS
SISTEMAS DE INFORMAÇÃO
INTRODUÇÃO A TEORIA DA COMPUTAÇÃO

DOCUMENTAÇÃO

Simulador Universal de Autômatos Finitos

Geraldo Murilo Carrijo Viana Alves da Silva, Lucas Caetano Procópio, Iara Duarte Mainates.

São Carlos, Maio de 2021

RESUMO

Este documento tem como objetivo apresentar os raciocínios empregados na construção da solução “Simulador Universal de Autômatos Finitos”, bem como discutir a qualidade da solução proposta e expor a eficiência da solução em termos de espaço e tempo.

Alunos:

Geraldo Murilo Carrijo Viana Alves da Silva, nUSP: 11849306
Lucas Caetano Procópio, nUSP: 11831338
Iara Duarte Mainates, nUSP: 11816143.

Professor:

João Luís Garcia Rosa.
Departamento de Ciências de Computação - ICMC - USP.

SUMÁRIO

RESUMO **2**

SUMÁRIO **2**

SOBRE A SOLUÇÃO **3**

Validação de Cadeias em Autômatos Finitos Determinísticos (AFD) 5

Validação de Cadeias em Autômato Finitos Não-Determinísticos (AFN) 5

QUALIDADE DA SOLUÇÃO **6**

EFICIÊNCIA DA SOLUÇÃO **7**

REFERÊNCIAS **8**

SOBRE A SOLUÇÃO

A solução proposta foi pensada utilizando os conceitos das aulas da disciplina Introdução à Teoria da Computação e programada utilizando a linguagem C, pois era a que os integrantes do grupo possuíam mais afinidade.

Para a representação dos autômato, foi utilizada uma estrutura chamada *automata_t,* que contém todos os dados necessários para a execução do programa, entre eles:

- Tipo do autômato (representado por 0 em caso de AFD e 1 em caso de AFNs)
- Quantidade de estados, variando de 1 a 10
- Quantidade de estados iniciais, variando de 1 a 10. Com a entrada 1 entende-se que o autômato será um AFD de estado inicial q0, já nos outros casos o autômato será um AFN com estados iniciais indo de q0 até o número digitado subtraído de 1.
- Quantidade de estados finais e um vetor para armazenar todos os respectivos estados
- Quantidade de transições do autômato
- Registro das transições
- Vocabulário que o autômato comporta

```c
//Definicao do tipo do Automato
#define AFD 0
#define AFN 1

typedef struct automata
{
    int tipoAutomato;
    int qntEstados;
    int estadosIniciais;
    int qntEstadosFinais;
    int *estadosFinais;
    int **qntTransicoes;
    int ***transicoes;
    char *vocabulario;
} automata_t;
```

Representação do Autômato

Já para a leitura das entradas, foram seguidas as instruções disponível para a descrição do trabalho onde as quatro primeiras linhas cuidam de dados inerentes ao autômato e as restantes cuidam das transições e das cadeias de teste.

```c
void leEntrada(automata_t *automata)
{
    int qntSimbolos;
    int qntTransicoes;

    // Estados
    scanf("%d", &automata->qntEstados);

    // Simbolos terminais
    scanf("%d", &qntSimbolos);
    automata->vocabulario = (char *) malloc(sizeof(char) * qntSimbolos);
    for (int i = 0; i < qntSimbolos; i++)
    {
        scanf(" %c", automata->vocabulario + i);
    }

    // Estados iniciais
    scanf("%d", &automata->estadosIniciais);
    if(automata->estadosIniciais==1)
    {
        automata->tipoAutomato=AFD;
    }
    else
    {
        automata->tipoAutomato=AFN;
    }

    // Estados finais
    scanf("%d", &automata->qntEstadosFinais);
    automata->estadosFinais = (int *) malloc(sizeof(int) * automata->qntEstadosFinais);
    for (int i = 0; i < automata->qntEstadosFinais; i++)
    {
        scanf(" %d", automata->estadosFinais + i);
    }
```

Leitura dos dados iniciais do autômato

```c
// Transicoes
// Alocando espaço para as transições
automata->transicoes = (int ***) malloc(sizeof(int **) * automata->qntEstados);
automata->qntTransicoes = (int **) malloc(sizeof(int *) * automata->qntEstados);
for (int i = 0; i < automata->qntEstados; i++)
{
    automata->transicoes[i] = (int **) malloc(sizeof(int *) * qntSimbolos);
    automata->qntTransicoes[i] = (int *) malloc(sizeof(int) * qntSimbolos);
}
// Quantidade de transições por par (estado, simbolo)
for (int i = 0; i < automata->qntEstados; i++)
{
    for (int j = 0; j < qntSimbolos; j++)
    {
        automata->transicoes[i][j] = NULL;
        automata->qntTransicoes[i][j] = 0;
    }
}
// Lendo as transições
for (int i = 0; i < qntTransicoes; i++)
{
    int q, qlinha;
    char simbolo;

    scanf("%d %c %d", &q, &simbolo, &qlinha);

    int indiceSimbolo;
    for (indiceSimbolo = 0; indiceSimbolo < qntSimbolos; indiceSimbolo++)
        if (automata->vocabulario[indiceSimbolo] == simbolo) break;

    int novaQnt = automata->qntTransicoes[q][indiceSimbolo] + 1;

    automata->qntTransicoes[q][indiceSimbolo] = novaQnt;
    automata->transicoes[q][indiceSimbolo] = (int *) realloc(automata->transicoes[q][indiceSimbolo], sizeof(int) * novaQnt);
    automata->transicoes[q][indiceSimbolo][novaQnt - 1] = qlinha;
}
```

Leitura das transições

Após a leitura destes dados já se tem os conhecimentos necessários a respeito do autômato e podemos iniciar a validação de cadeias. Para isso, é feita a

leitura da cadeia e a mesma será analisada dependendo do seu caso (autômato AFN ou AFD). Quando avaliada na devida função, seu resultado poderá ser 0 em caso de cadeia aceita, e -1 em caso de cadeias rejeitadas.

Após a avaliação, é chamada a impressão dos resultados que imprimem no formato: "*ID_da_Cadeia*. aceita" ou "*ID_da_Cadeia*. rejeita".

Validação de Cadeias em Autômatos Finitos Determinísticos (AFD)

A validação de cadeias em autômatos AFD é feita de forma recursiva. Isto é, na chamada inicial é passado o correspondente autômato e a cadeia a ser validada. Como o programa deixa claro que AFDs tem estado inicial na forma de q0 passamos 0 como valor do estado inicial.

Em cada iteração da função é obtido o símbolo atual, com ele é possível verificar a validade através da procura do próximo estado correspondente. Caso um próximo estado seja encontrado, o programa avança para o próximo símbolo contendo o próximo estado.

Ao final da recursão, é possível chegar a um dos estados finais e retornamos ao programa o valor 0 indicando que a cadeia é válida, ou a um estado intermediário que retorna valor -1 indicando que a cadeia é rejeitada pelo autômato.

```c
int avaliaAFD(automata_t *afd, char *str, int estadoAtual)
{
    char simboloAtual = *str;
    int indiceSimboloAtual;

    if (simboloAtual == '\0' || estadoAtual == -1)
    {
        for (int i = 0; i < afd->qntEstadosFinais; i++)
            if (afd->estadosFinais[i] == estadoAtual) return 0;
        return -1;
    }

    for (indiceSimboloAtual = 0; indiceSimboloAtual < MAX_SIMBOLOS; indiceSimboloAtual++)
        if (afd->vocabulario[indiceSimboloAtual] == simboloAtual) break;

    str++;
    return avaliaAFD(afd, str, afd->transicoes[estadoAtual][indiceSimboloAtual][0]);
}
```

Função de avaliação para cadeias AFD

Validação de Cadeias em Autômato Finitos Não-Determinísticos (AFN)

Já em casos de autômatos AFNs o raciocínio segue o mesmo, mas é necessário avaliar os vários caminhos. Para isso é feito um loop no programa principal, onde iremos executar a função de avaliação para cada estado inicial.

```
else if(automato.tipoAutomato==AFN)
{
    for (int estadoInicial = 0; estadoInicial < automato.estadosIniciais; estadoInicial++)
    {
        resultado = avaliaAFN(&automato, cadeia, estadoInicial);
        if (resultado == 0)
        {
            resultadosCadeia[i]=1;
            break;
        }
    }
}
```

Loop de estados iniciais no programa principal

Além disso,

```
int avaliaAFN(automata_t *afn, char *str, int estadoAtual)
{
    char simboloAtual = *str;
    int indiceSimboloAtual;

    if (simboloAtual == '\0')
    {
        for (int i = 0; i < afn->qntEstadosFinais; i++)
            if (afn->estadosFinais[i] == estadoAtual) return 0;
        return -1;
    }

    for (indiceSimboloAtual = 0; indiceSimboloAtual < MAX_SIMBOLOS; indiceSimboloAtual++)
        if (afn->vocabulario[indiceSimboloAtual] == simboloAtual) break;

    if (afn->qntTransicoes[estadoAtual][indiceSimboloAtual] == 0)
        return -1;

    for (int i = 0; i < afn->qntTransicoes[estadoAtual][indiceSimboloAtual]; i++)
    {
        if (avaliaAFN(afn, str + 1, afn->transicoes[estadoAtual][indiceSimboloAtual][i]) == 0)
            return 0;
    }
    return -1;
}
```

Função de avaliação para cadeias AFN

QUALIDADE DA SOLUÇÃO

Em termos de qualidade, a solução pode ser classificada como boa pois o código faz uso de alocações e loops dinâmicos, não rodando mais do que o necessário. Além disso, a estrutura de um autômato comporta todos os dados e tem tamanho entre 0 e 1kb (considerando todas as variáveis em seu máximo, estabelecido pela descrição do programa).

Já para a avaliação de qualidade de código, pode-se avaliar como uma boa escrita já que o código faz separação entre funções e as variáveis podem ser auto-explicadas.

EFICIÊNCIA DA SOLUÇÃO

Falando a respeito da eficiência, a solução cumpre eficiência de tempo média, e eficiência de espaço boa - pelos motivos mencionados acima.

Em geral, o programa poderia ser melhorado em termos de eficiência de tempo, pois as chamadas recursivas no tratamento de AFNs podem levar a exaustão. A solução mais recomendada nesse caso seria minimizar as recursões fazendo a transformação de autômatos não determinísticos em autômatos determinísticos (AFDs) para assim utilizar apenas uma função de validação.

Contudo, ao fazer essa transformação, o executor terá de tomar cuidado com o gerenciamento de espaços ao alocar vários estados novos que irão ser gerados na transformação.

REFERÊNCIAS

- **Normas ABNT para apresentação de trabalhos científicos.** Wikimedia. Disponível em: <http://pt.wikipedia.org/wiki/Normas_ABNT_para_apresentação_de_trabalhos_científicos>. Acesso em 25 de mai. de 2021.
- Rosa, J. L. G. Linguagens Formais e Autômatos. Editora LTC, 2010.
- Geeks for Geeks <https://www.geeksforgeeks.org/c-program-to-simulate-nondeterministic-finite-automata-nfa/>