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

DOCUMENTAÇÃO

Simulador Universal de Máquinas De Turing

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

São Carlos, Julho de 2021

RESUMO

Este documento tem como objetivo apresentar os raciocínios empregados na construção da solução “Simulador Universal de Máquinas de Turing”, 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

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 é a que os integrantes do grupo possuíam mais afinidade.

Representação da Máquina de Turing

Para representar a Máquina de Turing a ser simulada foi utilizado o recurso de *struct* de dados que a identificam, chamada de “dadosTuring”. Nessa struct são armazenados os dados de quantidade de estados, quantidade de símbolos terminais, quantidade de símbolos estendidos e o estado de aceitação qn - além disso, são armazenados também os símbolos terminais e estendidos em um vetor de char.

```c
/* As características gerais do automato são gravados no tipo
   dadosTuring(struct dados)
    Inteiros:
        Respectivamente quantidade total de estados,
        quantidade de símbolos terminais, quantidades de símbolos
        do alfabeto estendido e estado de aceitação.
    Vetores de Char:
        Respectivamente string que contém todos os símbolos
        terminais e String que contém todos os símbolos
        do alfabeto estendido.
*/
typedef struct dados
{
    int qtdEstados, qtdTerminais, qtdEstendidos, aceitacao;
    char *terminais, *estendidos;
}dadosTuring;
```

Struct de dados da máquina de Turing

Representação de Estados

Para a representação de cada estado e suas transições também é utilizada a estrutura de *struct* que contém os dados inerentes à um estado, isto é, sua leitura, sua resposta (reescrita) a direção de movimento e o destino.

```c
/* Cada estado é representado pelo tipo estado (struct estados_t)
    - count: Variável inteira que itera os vetores abaixo,
        também é responsável a relação entre eles.
    - leitura[10]: Qual caractere da fita é lido na transição.
    - reescrita[10]: Qual caractere é reescrito sobre o que é lido na transição.
    - direção[10]: Para qual direção se move a "cabeça do leitor",
        no caso desse programa é incremento ou decremento no index da cadeia.
    - destino[10]: Em qual estado a máquina se encontrará após a transição.
*/
typedef struct estados_t
{
    int count;
    char leitura[10];
    char reescrita[10];
    char direcao[10];
    int destino[10];
}estado;
```

Representação dos estados da máquina de Turing

Inicialmente, o programa faz a leitura e a alocação de todos os dados até o estado de aceitação e os direciona para a struct *dadosTuring dados*

```c
/* i e j são iteradores,
    **MatrizFunção é a matriz cujo par (x,y) identifica a
    existência de caminhos entre o estado Qx e Qy (posicao [i][j] == 1).
*/
int i, j, **MatrizTransicao;
/* total é um vetor do tipo estado que guarda todos o n estados do
automato (struct estados_t)*/
estado *total;
// Representação da máquina de turing
dadosTuring dados;

//Representação da quantidade de dados
scanf("%d", &dados.qtdEstados);
//Alocação dinâmica do vetor de estados.
total = (estado*)malloc(dados.qtdEstados*sizeof(estado));

//Preenchimento da Matriz de Estados
MatrizTransicao = preencheMatrizEstatal(dados);

//Esse laço inicializa o contador de controle dos estados como 0, note que cada estado tem um contador próprio
for(i = 0; i < dados.qtdEstados; i++)
{
    total[i].count = 0;
}

scanf("%d", &dados.qtdTerminais);

//Aloca dinamicamente o vetor de símbolos terminais e os lê um por um
dados.terminais = (char *)malloc(dados.qtdTerminais*(sizeof(char)));
for(i = 0; i < dados.qtdTerminais; i++){
    scanf(" %c", &dados.terminais[i]);

}

//Aloca dinamicamente o vetor de símbolos do alfabeto estendido e os lê um por um
scanf("%d", &dados.qtdEstendidos);
dados.estendidos = (char *)malloc(dados.qtdEstendidos * (sizeof(char)));
for(i = 0; i < dados.qtdEstendidos; i++){
    scanf(" %c", &dados.estendidos[i]);

}

scanf("%d", &dados.aceitacao);
```

Leitura de dados da máquina de Turing

Leitura de Transições

A leitura de cada transição é feita através chamada a função *leTransicoes*, chamadas na função main. Essa leitura é armazenada na struct de estados *total.*

```
leTransicoes(dados, total, MatrizTransicao);
```

```
void leTransicoes(dadosTuring dados,estado *total, int **MatrizTransicao)
{
    //Transicoes é a quatidade de transições a serem lidas, i é um iterador
    int transicoes, i, source, dest;
    char read, write, dir;

    scanf("%d", &transicoes);

    for(i = 0; i < transicoes; i++)
    {
        //estado atual da máquina
        scanf("%d", &source);
        //caractere atual lido da fita
        scanf(" %c", &read);
        //estado que esse caractere leva
        scanf("%d", &dest);
        //caractere a ser reescrito antes mover na fita
        scanf(" %c", &write);
        //direção a mover na fita
        scanf(" %c", &dir);

        //identifica a existência de um caminho no par [source][dest]
        MatrizTransicao[source][dest] = 1;

        /* Aqui começa a correlação, lendo "read", devo reescrever "write"
            e mover para "dir" na fita e entrar no estado "dest".
            Com o uso do mesmo iterador nos 3 vetores diferentes têm-se a
            relação de dados diferentes no mesmo índice.
            -> Count só incrementado após as inserções e é atrelado a cada estado.
        */
        total[source].leitura[total[source].count] = read;
        total[source].reescrita[total[source].count] = write;
        total[source].destino[total[source].count] = dest;
        total[source].direcao[total[source].count] = dir;
        total[source].count++;

    }
}
```

Leitura de Transições

Leitura de Cadeias

Ao finalizar essas leituras, o programa tem todos os dados necessários para simular uma Máquina de Turing. Para isso, temos a chamada da função de *lerCadeias* para ler as sequência inseridas pelo usuário.

```c
void lerCadeias(dadosTuring dados, int **MatrizTransicao, estado *total)
{
    /*qtdEntradas é a quantidade de entradas, i iterador,
    *result vetor de resultados da avaliação da cadeia.
    */
    int qtdEntradas, i, *result;

    //String que representa a cadeia de até 20 elementos.
    char cadeia[20];

    scanf("%d", &qtdEntradas);
    result = (int *)malloc(qtdEntradas*sizeof(int));
    for(i = 0; i < qtdEntradas; i++)
    {
        scanf("%s", cadeia);
        result[i] = testaCadeia(cadeia, dados, MatrizTransicao, total);
    }//o retorno da testaCadeia é carregada no vetor de resultados

    for(i = 0; i < qtdEntradas; i++)
    {
        if(result[i] == 1) //para os casos positivos
        {
            printf("aceita\n");
        }
        else //para os negativos
        {
            printf("rejeita\n");
        }
    }

    free(result);
}
```

Leitura de Cadeias e Validação via Máquina de Turing

Dentro dessa função, o programa faz a leitura da quantidade de entradas e as lê individualmente. Ao fazer a leitura, o programa redireciona a cadeia para a função *testaCadeia* que irá fazer a validação de acordo com os princípios da Máquina de Turing e retorna 1 em caso de sucesso (cadeia válida) ou 0 em caso de cadeias inválidas. Após a leitura e validação de todas as cadeias, o programa irá imprimir na tela o resultado como “aceita” em casos positivos e “rejeita” em casos negativos.

Validação de Cadeias

Já a função *testaCadeia* irá receber a cadeia a ser avaliada, a máquina de

Turing utilizada e os estados da máquina correspondente - e para validar a cadeia fará uso de um loop que é executado até que a cadeia atinja o estado final ou até ou até que não seja encontrado, no vocabulário do estado, determinado símbolo terminal.

Dentro desse loop, o programa vai de 0 até o tamanho da string “leitura” do estado verificado no momento, assim, o programa verifica se o atual símbolo da cadeia pode ser lido por tal estado. Caso o símbolo atual esteja nesse estado, o programa reescreve na “fita” o valor de reescrita e verifica sua orientação de direção (R para direita e L para esquerda).

Na situação de direcionamento a direita, o index de símbolos é incrementado e o programa passará a olhar para o próximo símbolo - e para garantir que a fita será infinita, quando passamos do index limite do vetor da cadeia analisada, assinalamos o símbolo atual com ‘B', significando um branco.

Na situação de direcionamento para esquerda, o index de símbolos é decrementado e o programa irá olhar para o símbolo anterior - e para garantir que a fita será infinita, quando passamos para um index negativo, atribuímos ao símbolo atual o caractere ‘B’, significando um branco.

Com a verificação, o estado atual é modificado para o índice j dentro do vetor de estados (controlado pelo loop for) e j é atualizado com o valor 200, nos casos em que j não é atualizado para este valor, sabe-se que o Símbolo atual não foi encontrado e portanto a máquina não consegue realizar nenhuma transição ocasionando a interrupção do loop mais externo.

Para o tratamento do caso em que a máquina não chega ao estado de aceitação, o programa irá verificar se há a possibilidade de ler lambda (cadeia vazia) para chegar ao estado final. Caso a verificação funcione, o estado é atualizado.

Ao final, verificamos se o estado atual corresponde ao estado de aceitação - em caso positivo, a função retornará 1, e em caso negativo irá retornar 0.

```c
int testaCadeia(char *cadeia, dadosTuring dados, int **MatrizTransicao, estado *total)
{
    // Caractere atual na fita
    char currChar;

    /* currState é o estado atual da máquina (iniciado em q0 por padrão),
        i e j iteradores,
        currCharIndex é o indíce na fita para atualizar currChar;
    */
    int currState = 0, i, j, currCharIndex = 0;

    //o mesmo que fazer currCharr = cadeia[0], inicia currChar para o caractere mais a esquerda da fita
    currChar = *cadeia;

    /* O loop é executado até atingir o estado final (a partir do qual não há transições)
    ou até que uma transição n seja possível(explicado posteriormente)
    */
    while(currState != dados.aceitacao)
    {
        for(j = 0; j < total[currState].count; j++)
        {
            /* Dentro do vetor de estados na posição do estado atual,
               busca-se valor do contador para evitar acessos desnecessários de memória.
            */
            if(total[currState].leitura[j] == currChar)
            {
                /* Dentro do vetor de estados, busca-se os caracteres lidos pelo estado e os compara com currChar,
                caso esse loop chege ao final sem que esse if seja satisfeito j não passará a verifivação.
                */
                cadeia[currCharIndex] = total[currState].reescrita[j];
                /* A posição atual na cadeia é reescrita com o elemento do vetor de símbolos estendidos de
                   mesmo índice encontrado no laço e aceito no if.
                */

                if(total[currState].direcao[j] == 'R')
                {
                    //Caso a direção seja direita o currCharIndex é incrementado uma unidade
                    currCharIndex++;

                    /*Para garantir que a fita é "infinita", quando o index extrapola os limites do vetor cadeia,
                    currChar é atualizado para 'B', que é o símbolo que representa o Branco
                    */
                    if(currCharIndex >= strlen(cadeia))
                    {
                        currChar = 'B';
                    }
                    else // Caso o índice esteja dentro do vetor
```

Função testaCadeia, parte 1.

```c
                    {
                        //currChar recebe o caractere na posição currCharIndex da fita.
                        currChar = cadeia[currCharIndex];
                    }

                }
                else
                {
                    /*Caso não vá para direita 'R', pode ir para esquerda 'L' decrementando uma unidade e as
                    intruções seguintes são espelhadas da direita com mudanças no limite do vetor
                    */
                    if(total[currState].direcao[j] == 'L')
                    {
                        currCharIndex--;

                        /*Para garantir que a fita é "infinita", quando o index extrapola os limites do vetor cadeia (<0),
                        currChar é atualizado para 'B', que é o símbolo que representa o Branco
                        */
                        if(currCharIndex < 0)
                        {
                            currChar = 'B';
                        }
                        else
                        {
                            currChar = cadeia[currCharIndex];
                        }

                    }

                    //Caso não vá nem para direita nem para esquerda quer dizer que não há movimento 'S', logo o currCharIndex não é alterado.
                }

                //Feitas as verificações e edições o estado atual é modificado para o destino de mesmo índice j dentro do vetor de estados.
                currState = total[currState].destino[j];
                /*J é reescrito para 200
                (valor inalcançável por j dentro do programa)
                caso passe o if para futura verificação.
                */
                j = 200;

                //Feitas todas as movimentações o loop interno é interrompido
                break;
            }
        }
    }
```

Função testaCadeia, parte 2.

```
//Mos casos em que a máquina termina fora do estado de aceitação (caso do automato de cadeias a^2n)
if(currState != dados.aceitacao)
{
    for(j = 0; j < total[currState].count; j++)
    {
        //É feita uma verificação onde busca-se a possibilidade de ler lambda ('-') para  chegar a outro estado
        if(total[currState].leitura[j] == '-')
        {
            //Caso a verificação seja bem sucedida o estado atual é atualizado
            currState = total[currState].destino[j];
            break;
        }
    }
}

/* Por fim, se o estado final do automato for o de aceitação o retorno é 1,
senão é 0
*/
if(currState != dados.aceitacao)
{
    return 0;
}
else
{
    return 1;
}
```

Função testaCadeia, parte 3.

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 e utilizando ciclos de acordo com o inserido pelo usuário - e não mais que isso. Além disso, a estrutura de uma Máquina de Turing têm no máximo tamanho de 36 bytes (16 bytes de inteiros relacionados aos controladores de quantidades e 20 bytes de char relacionados aos vetores de char), enquanto a estrutura de estados ocupa até 44 bytes (40 dos vetores de inteiros e 4 de inteiros).

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

EFICIÊNCIA DA SOLUÇÃO

Como visto anteriormente, em termos de memória, a representação das Máquinas de Turing e seus estados ocupam um espaço reduzido e a representação dos estados em vetor de structs, armazenados sequencialmente, reduz também a quantidade de acessos - que são feitos “sequencialmente”.

Quanto ao tempo de execução, o algoritmo empregado para o cálculo do resultado da Máquina de Turing pode ser traduzido para um laço while simples que possui um for que caminha por (no máximo) todo o vetor “leitura” de n estados

(sendo n a quantidade de estados visitados até atingir a última transição), assim sendo limitado apenas pela quantidade de transições multiplicadas pelo tamanho do vetor de leitura no pior caso (para cada cadeia), possuindo portanto crescimento linear.

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 maio de 2021.
- **Rosa, J. L. G. Linguagens Formais e Autômatos**. Editora LTC, 2010.