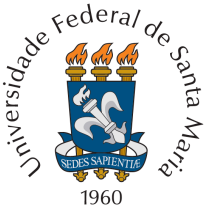

UNIVERSIDADE FEDERAL DE SANTA MARIA

CENTRO DE TECNOLOGIA

ELC1008 - TEORIA DA COMPUTAÇÃO

Guilherme Fereira Da Silva

Leandro Brum da Silva Lacorte

DOCUMENTAÇÃO TÉCNICA DA IMPLEMENTAÇÃO DA MÁQUINA DE TURING QUÂNTICA (MTQ)

Santa Maria, RS

2025

SUMÁRIO

1 INTRODUÇÃO

Esta documentação descreve a implementação de uma simulação de Máquina de Turing Quântica (MTQ) em C, incluindo:

- Algoritmo de **Deutsch** (para funções booleanas simples), e
- Reconhecimento da **linguagem clássica A = {0^n1^n}**, utilizada como exemplo de linguagem formal.

O projeto demonstra conceitos fundamentais de computação quântica, como **superposição**, **amplitudes complexas**, **transições reversíveis** e **medição probabilística**.

2 ESTRUTURA GERAL DO PROJETO

O projeto é organizado nas seguintes pastas e arquivos:

```
/include
    mtq.h
    quantum.h
    util.h
    deutsch.h
    langA.h
    linguagem_a.h

/src
    mtq_deutsch.c
    quantum.c
    util.c
    deutsch.c
    langA.c
    linguagem_a.c

Makefile
```

- **include/**: Contém headers com definições de structs, funções e constantes.

- **src/**: Contém implementações das funções e simuladores.
- **Makefile**: Gerencia compilação e execução.

3. PRINCIPAIS CONCEITOS COMPUTACIONAIS IMPLEMENTADOS

3.1. Máquina de Turing Clássica

A MTQ é baseada na Máquina de Turing clássica:

- **Fita infinita (ou de tamanho fixo para simulação)**: cada célula contém um símbolo do alfabeto.
- **Cabeça de leitura/escrita**: lê e escreve símbolos e move-se para esquerda ou direita.
- **Estados**: conjunto finito, com estado inicial e final.
- **Função de transição δ**: determina próximo estado, símbolo a escrever e direção do movimento.

3.2. Máquina de Turing Quântica

A MTQ estende a MT clássica para permitir **superposição de estados**, sendo cada configuração clássica ponderada por uma **amplitude complexa**:

- **Configuração clássica (`config_t`)**:

```
int estado;      // índice do estado
int cabeca;      // posição da cabeça
char fita[FITA_TAM]; // conteúdo da fita
```

- **Configuração quântica (`quantum_config_t`)**:

```
config_t config;
double complex amplitude; // amplitude complexa da configuração
```

- **Função de transição δ**: implementada como regras (`rule_t`) que atualizam a configuração e multiplicam a amplitude complexa.
- **Superposição linear**: estado quântico da MTQ é representado por um vetor de `quantum_config_t` com amplitudes complexas.
- **Medição**: escolhe aleatoriamente uma configuração com probabilidade proporcional ao quadrado da magnitude da amplitude ($|\psi|^2$).

4. ESTRUTURAS DE DADOS E CONSTANTES

4.1. Constantes e enums

- `FITA_TAM = 100` → tamanho da fita simulada.
- `MAX_CONFIGS = 200` → número máximo de configurações simultâneas.
- `BLANK = 'b'` → símbolo em branco.
- `MAX_TRANS = 4` → máximo de transições por par (estado, símbolo).

Estados Deutsch e Linguagem A definidos como `enum`:

```
enum { Q0, Q1, Q2, Q3, Q4, QF };
enum { QA0, QA1, QA2, QA3, QA4, QAF };
```

4.2. Structs principais

Regra de transição quântica (`rule_t`)

```
 int estado_in;
char simbolo_in;
int estado_out;
char simbolo_out;
int move;              // -1=L,0=N,+1=R
double complex amp;    // amplitude complexa
```

Transição genérica (`Transicao`): usada para MTQ clássica ou exemplos de linguagem.

MTQ genérica (`MTQ`):

```
Transicao *delta[10][10][MAX_TRANS];
int num_trans[10][10];
char alfabeto[10];
int num_estados, num_simbolos;
int qi, qf; // estado inicial/final
```

MTQ Linguagem A (`MTQ_A`): versão específica com suporte a superposição parcial.

5. PRINCIPAIS FUNÇÕES IMPLEMENTADAS

5.1. Normalização de amplitudes

```
void normalize(quantum_config_t *configs, int n);
```

- Calcula norma L2 das amplitudes $|\psi|^2$.
- Divide todas amplitudes pela raiz quadrada da soma de quadrados, garantindo que a soma das probabilidades seja 1.

5.2. Mesclar configurações iguais

```
int merge_equal_configs(quantum_config_t *src, int nsrc,
quantum_config_t *dst);
```

- Combina configurações idênticas, somando suas amplitudes.
- Evita duplicação e mantém simulação eficiente.

5.3. Transição quântica

```
int transicao(quantum_config_t *configs, int numConfigs, rule_t
```

```
*rules, int numRules, quantum_config_t *outConfigs);
```

- Aplica todas as regras relevantes a cada configuração.
- Cria novas configurações multiplicando amplitude pelas regras.
- Normaliza e retorna número de configurações resultantes.

5.4. Medição

```
quantum_config_t medir(quantum_config_t *configs, int n);
```

- Escolhe aleatoriamente uma configuração com probabilidade proporcional a |amplitude|².
- Simula colapso quântico da superposição.

5.5. Impressão de estado

```
void print_state(quantum_config_t *configs, int n);
```

- Exibe todas configurações, amplitudes e probabilidades associadas.

5.6. Funções auxiliares

- `configs_equal()` → compara duas configurações clássicas.
- `copiarConfig()` → cria cópia de configuração.
- `print_config_brief()` → imprime a fita e posição da cabeça.

6. IMPLEMENTAÇÃO DE EXEMPLOS

6.1. Algoritmo de Deutsch

- Simula superposição inicial `(1/√2)(|0⟩ + |1⟩)`.
- Aplica oráculo `Uf` com regras baseadas na função f(x) escolhida.
- Aplica Hadamard final e mede a configuração resultante.
- Determina se a função é **constante** ou **balanceada**.

6.2. Linguagem A = {0^n1^n}

- Configuração inicial com fita preenchida com `entrada`.
- Regras implementam marcação de zeros e busca do 1 correspondente.
- Simulação quântica permite múltiplos caminhos simultâneos via superposição.
- Medição retorna configuração final determinística ou probabilística, mostrando se a entrada é aceita.

7. FLUXO DE EXECUÇÃO DO SIMULADOR

1. Leitura de parâmetros (`argv`): tipo de simulação e entrada.
2. Inicialização da fita e do estado inicial.
3. Criação das regras (`rule_t`) de acordo com Deutsch ou Linguagem A.
4. Inicialização da superposição quântica com uma configuração inicial.
5. Loop de execução:
    - Aplica `transicao()` para gerar novas configurações.
    - Combina configurações iguais.
    - Normaliza amplitudes.
    - Imprime estado atual.

    - Verifica se todas amplitudes estão no estado final.
6. Medição final:
    - `medir()` seleciona configuração com probabilidade proporcional a $|\psi|^2$.
    - Exibe resultado e decisão final.

8. CONSIDERAÇÕES TÉCNICAS E COMPUTACIONAIS

- **Superposição**: Permite representar simultaneamente múltiplos caminhos computacionais.
- **Reversibilidade**: Cada regra é projetada para preservar amplitudes, garantindo evolução unitária.
- **Complexidade**: O número de configurações cresce exponencialmente com passos e superposição, limitado por `MAX_CONFIGS`.
- **Generalidade**: Estruturas `rule_t` e `MTQ` permitem extensão para outras linguagens e algoritmos quânticos.