Construção e Análise de uma Máquina de Turing para Ordenação de Vetores Binários: Uma Abordagem para Disciplinas de Teoria da Computação

Maurilio Martins Campano Junior[1,2], **Linnyer Beatrys Ruiz Aylon**[1]

[1]Manna_Team
Universidade Estadual de Maringá (UEM)
Programa de Pós-Graduação em Ciência da Computação (PCC)
Maringá - PR - Brazil

[2]Engenharia de Software – UniCesumar
Maringá – PR – Brazil

maurilio.campanojr@gmail.com, lbruiz@uem.br

Abstract. *This paper presents a Turing Machine implementation of the Selection Sort algorithm for binary vectors, validated using the JFLAP simulator. Unlike arithmetic operations, data structure manipulation revealed severe non-linear computational costs: sorting just 127 elements required over 32 million transitions. Comprising 133 states and tape segmentation, the tool serves as a pedagogical resource to visually demonstrate the distinction between computability and efficiency, highlighting the practical limitations of sequential access compared to random-access memory models.*

Resumo. *Este artigo apresenta uma Máquina de Turing para a ordenação de vetores binários via Selection Sort, validada no simulador JFLAP. Diferente de operações aritméticas, a manipulação de estruturas de dados revelou um custo computacional não linear severo: a ordenação de apenas 127 elementos exigiu mais de 32 milhões de transições. A implementação, composta por 133 estados e segmentação de fita, atua como recurso pedagógico para demonstrar visualmente a distinção entre computabilidade e eficiência, evidenciando as limitações práticas do acesso sequencial em contraste com modelos de memória de acesso aleatório.*

1. Introdução

A área de Ciência da Computação enfrenta historicamente desafios significativos relacionados à retenção de estudantes nos primeiros anos de graduação. Disciplinas teóricas, fundamentais para a formação do cientista da computação, são frequentemente citadas como barreiras para o engajamento discente. Os trabalhos de Fukao et al. (2023) e Mundim et al. (2025) destacam a evasão em cursos de computação, indicando que a dificuldade de abstração em disciplinas de base matemática e teórica desempenha um papel crucial no desestímulo dos alunos.

Neste cenário, o ensino de Teoria da Computação, Linguagens Formais e Autômatos (LFA) torna-se um ponto focal para investigações pedagógicas. Junior et al. (2021), por meio de uma revisão sistemática sobre o ensino de computação teórica no

ensino básico, identificaram 17 trabalhos que incorporam recursos didáticos na área. Os autores observam que, embora conceitos como Máquinas de Estado Finito, Máquinas de Turing e Teoria da Complexidade sejam explorados, há uma escassez de pesquisas em computação teórica quando comparada à vasta literatura sobre o ensino de programação. As estratégias variam desde abordagens tradicionais e ferramentas digitais até jogos desplugados e robótica.

Corroborando com este cenário, a revisão sistemática realizada por Silva et al. (2024) focou especificamente na Teoria dos Autômatos, analisando 30 trabalhos que abordam o tema no processo de ensino-aprendizagem. A literatura aponta que o uso de ferramentas de apoio é essencial no ensino de conceitos abstratos. Souza et al. (2016), por exemplo, catalogaram 37 trabalhos descrevendo recursos computacionais para LFA, notando que a maioria (29%) foca em Autômatos Finitos, embora existam recursos que englobam a hierarquia de Chomsky até as Máquinas de Turing (deterministas e não deterministas).

Diversas ferramentas têm sido propostas para mitigar a dificuldade de abstração. Ferreira et al. (2023) utilizaram o *Google Colab* para implementar exemplos de Autômatos Finitos Determinísticos (AFD) e grafos, permitindo a manipulação de código em sala de aula. Em paralelo, Mioni e Barbosa (2022) e Paiva et al. (2023) apresentaram comparativos de simuladores voltados ao ensino de LFA. Paiva et al. (2023), especificamente, avançam ao propor uma recomendação de ferramentas baseada nos conteúdos comuns das ementas da disciplina, guiando docentes na escolha do recurso mais adequado para cada tópico, sendo que a ferramenta indicada para o ensino de Máquina de Turing é o JFLAP [JFLAP 2026].

Apesar da existência de simuladores, a implementação de algoritmos complexos em modelos teóricos ainda é um desafio didático. Trabalhos anteriores exploraram operações aritméticas em Máquinas de Turing como forma de ensinar lógica binária e manipulação de fita. Em Campano Junior et al. (2019), foi desenvolvida uma máquina para realizar a soma de dois números binários de qualquer magnitude. Evoluindo a complexidade, Campano Junior et al. (2022) implementaram a multiplicação binária, demonstrando o uso de "registradores" virtuais na fita.

No entanto, as operações aritméticas, embora fundamentais, não capturam a complexidade de manipulação de estruturas de dados lineares e algoritmos de ordenação, tópicos centrais na formação do cientista da computação. Assim, o presente trabalho visa preencher essa lacuna ao apresentar o desenvolvimento e a análise técnica de desempenho de uma Máquina de Turing que implementa o algoritmo *Selection Sort* para a ordenação de vetores binários. O objetivo é investigar o potencial da "explosão" de estados e transições, inerente a este processo, como um recurso pedagógico auxiliar para ilustrar, visualmente, os conceitos de complexidade computacional e os limites práticos do modelo de Turing. Ressalta-se que este estudo foca na construção da ferramenta e na validação via simulação, sendo a aplicação em sala de aula uma etapa subsequente desta pesquisa.

2. Fundamentação Teórica e Trabalhos Relacionados

A Máquina de Turing (MT) consolida-se como o modelo teórico fundamental da computação moderna. Conceitualmente, o modelo é composto por uma fita infinita

(dividida em células), uma unidade de controle finita e um cabeçote de leitura e escrita. A fita atua como o dispositivo de memória, na qual cada célula armazena um único símbolo de um alfabeto finito, podendo ser percorrida livremente para a esquerda ou para a direita pelo cabeçote, conforme comandado pela unidade de controle [Vieira 2006, Hopcroft et al. 2001].

Formalmente, uma MT é definida como uma 7-tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$, onde:

- Q é o conjunto finito de estados da unidade de controle;
- Σ é o alfabeto de entrada (símbolos que podem aparecer inicialmente na fita);
- Γ é o alfabeto da fita, tal que $\Sigma \subset \Gamma$ e $\square \in \Gamma$;
- δ é a função de transição, definida por $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, mapeando um estado e um símbolo lido para um novo estado, um símbolo a ser escrito e um movimento (Left ou Right);
- $q_0 \in Q$ é o estado inicial;
- $\square$ é o símbolo branco (vazio);
- $F \subseteq Q$ é o conjunto de estados finais ou de aceitação.

No contexto do ensino de Teoria da Computação, é crucial distinguir as MTs em duas categorias funcionais: Reconhecedoras e Transdutoras [Diverio and Menezes 2011]. As máquinas Reconhecedoras computam problemas de decisão, respondendo se uma palavra de entrada pertence ou não a uma determinada Linguagem Formal. Já as máquinas Transdutoras, foco deste trabalho, computam funções, transformando uma palavra de entrada em uma palavra de saída gravada na fita.

A representação gráfica padrão utiliza grafos dirigidos, na qual os nós representam os estados e as arestas representam as transições. No ensino destes conceitos, é comum apresentar o conceito com máquinas reconhecedoras de linguagens livres de contexto, como a linguagem $L = \{a^n b^n \mid n \geq 1\}$.

A compreensão destes mecanismos de leitura, escrita e movimentação é pré-requisito para o entendimento da implementação de algoritmos de ordenação na fita, na qual a MT deve comportar-se estritamente como uma transdutora, manipulando a fita como um vetor de dados.

Neste sentido, as estruturas de dados constituem a base para a organização e o armazenamento eficiente de informações em sistemas computacionais. O domínio destes conceitos abrange desde listas lineares, árvores e grafos até tipos abstratos mais complexos [Walker and Morrisett 2000, Sano et al. 2023]. Conforme destacam [Pardalos and Rajasekaran 1999] e [Cormen et al. 2012], tais estruturas desempenham um papel crucial no *design* de algoritmos, permitindo operações de inserção, exclusão e manipulação de dados sob regras padronizadas.

No contexto da educação em Ciência da Computação, o estudo de estruturas de dados e algoritmos de ordenação é identificado como um eixo de competência central [Oda et al. 2021]. Pesquisas empíricas, como as de [Zendler et al. 2013] e [Zendler et al. 2014], apontam que a combinação de conceitos de conteúdo (algoritmos, sistemas) com conceitos de processo (análise, classificação, resolução de problemas) é fundamental para o desenvolvimento cognitivo do estudante.

Dentre os algoritmos fundamentais para a manipulação de estruturas lineares (vetores e listas), destaca-se o *Selection Sort* (Ordenação por Seleção). Este algoritmo é frequentemente utilizado em disciplinas introdutórias devido à sua simplicidade lógica, embora não seja o mais eficiente para grandes conjuntos de dados.

O funcionamento do *Selection Sort* baseia-se na divisão da estrutura de dados em duas subestruturas: uma ordenada e outra não ordenada. O algoritmo busca iterativamente o menor elemento (ou maior, dependendo da ordem desejada) na subestrutura não ordenada e o move para o final da subestrutura ordenada.

Em termos de complexidade computacional, o *Selection Sort* realiza $O(n^2)$ comparações e $O(n)$ trocas, onde n é o número de elementos a serem ordenados. Uma característica didática importante deste algoritmo é sua ineficiência assintótica quadrática, o que o torna um excelente candidato para demonstrações de custo computacional em modelos teóricos, como a Máquina de Turing. Diferente de algoritmos como o *Merge Sort* ou *Quick Sort*, o *Selection Sort* opera *in-place* (utilizando espaço auxiliar constante $O(1)$), o que facilita sua representação em uma fita única de MT.

A compreensão profunda desses modelos estruturais e das transformações de dados exigidas por algoritmos de ordenação é vital para a formação do cientista da computação [Bressoud and Thomas 2019, Zendler et al. 2016]. No entanto, a abstração necessária para visualizar a execução desses algoritmos em modelos formais pode ser uma barreira de aprendizado. Para mitigar essa dificuldade, o uso de ferramentas e simuladores apresenta-se como uma estratégia pedagógica eficaz. A próxima seção descreve simuladores e ferramentas voltadas ao ensino de Estruturas de Dados e Teoria da Computação.

O uso de simuladores e técnicas visuais no ensino de Linguagens Formais e Autômatos tem sido amplamente explorado na literatura como forma de reduzir a abstração inerente à disciplina. No contexto específico de implementação de algoritmos clássicos em Máquinas de Turing, dois trabalhos prévios fundamentam a proposta atual, focando em operações aritméticas binárias.

O primeiro trabalho, desenvolvido por Campano Junior et al. (2019), implementa a soma de dois números binários de qualquer magnitude. A estratégia adotada utiliza o símbolo "#" como delimitador na fita. O algoritmo inicia inserindo um marcador final, valida a quantidade de *bits* dos operando para dimensionar o resultado e, sequencialmente, realiza a soma bit a bit com transporte (carry).

Elevando a complexidade das operações, Campano Junior et al. (2022) implementaram no simulador JFLAP a multiplicação de números binários de 8 *bits*. O algoritmo mimetiza o hardware real, operando com a lógica de multiplicação de inteiros sem sinal ($A \times Q$). A implementação simula "registradores" (C e A) na fita, realizando operações de deslocamento, soma e subtração condicionadas ao bit menos significativo do multiplicador.

Além da implementação técnica, este segundo trabalho destacou-se pela aplicação pedagógica interdisciplinar, sendo utilizado simultaneamente nas disciplinas de Linguagens Formais e Autômatos e Arquitetura e Organização de Computadores. A validação com 28 alunos de Ciência da Computação indicou que a visualização do algoritmo em baixo nível atuou como facilitadora no ensino de conceitos computacionais complexos.

O presente trabalho difere das abordagens citadas anteriormente ao deslocar o foco das operações aritméticas puras (processamento de dois escalares) para a manipulação de estruturas de dados lineares. A implementação do algoritmo de ordenação *Selection Sort* em uma Máquina de Turing exige não apenas a comparação de valores, mas o gerenciamento de um vetor desordenado de tamanho N, introduzindo desafios de indexação e movimentação de dados na fita que não estão presentes nas operações de soma e multiplicação.

3. Desenvolvimento da Máquina de Turing de Ordenação

A construção de uma Máquina de Turing (MT) para a ordenação de vetores impõe desafios significativos de gerenciamento de memória, visto que o acesso aos dados é estritamente sequencial. Diferente de arquiteturas de acesso aleatório (RAM), na qual índices de vetores são acessados em tempo constante ($O(1)$), na MT o custo de movimentação do cabeçote para comparações diretas entre posições distintas (como $A[i] > A[j]$) é elevado.

Nesta seção, detalha-se a estratégia de representação de dados, a lógica de comparação por exaustão e a manipulação da fita para a preservação da informação original. Para otimizar o conjunto de estados e transições, optou-se pela representação binária dos elementos. O vetor de entrada é disposto na fita separando cada elemento binário pelo símbolo auxiliar $\#$. Assume-se, para esta implementação, que o vetor de entrada não contém elementos repetidos ($v_i \neq v_j, \forall i \neq j$).

Considerando um vetor de exemplo $V = [3, 6, 2, 5, 4]$, a entrada inicial na fita é dada pela cadeia:

`11#110#10#101#100`

A estratégia central adotada para identificar o menor elemento do vetor, passo fundamental do algoritmo *Selection Sort*, difere da comparação tradicional par-a-par. Implementou-se uma lógica de decremento simultâneo:

1. A máquina percorre o vetor decrementando todos os números em uma unidade ($n \leftarrow n - 1$);
2. Após cada ciclo de decremento, verifica-se se algum número atingiu o valor zero;
3. O primeiro número a atingir zero é identificado como o menor número do conjunto remanescente.

No entanto, essa abordagem introduz um efeito colateral crítico: a perda da informação. Ao decrementar o valor binário `10` (2) até `00` (0) para descobrir que ele é o menor, o valor original é perdido, impossibilitando sua cópia para a posição ordenada final. Para solucionar o problema da destruição dos dados, foi desenvolvida uma estratégia de tripla segmentação da fita. A fita é expandida para conter três regiões distintas, separadas pelo marcador @:

- Região 1 (Entrada Original): Mantém o registro da entrada inicial para referência.
- Região 2 (Processamento): Onde ocorrem as operações de decremento. Nesta região, cada número recebe um identificador de posição (índice $1, 2, \ldots, n$).
- Região 3 (Armazenamento): Cópia de segurança contendo o índice e o valor original intacto, utilizada para recuperar o dado após a descoberta do menor elemento.

Para viabilizar o algoritmo de seleção sem a perda dos dados originais, a máquina inicia sua execução realizando um pré-processamento que expande a entrada. A fita é estruturada em regiões distintas, separadas pelo marcador @.

O exemplo a seguir ilustra o estado da fita imediatamente após o pré-processamento, utilizando cores para diferenciar as regiões lógicas:

`11#110#10#101#100@1#11#2#110#3#10#4#101#5#100@1#11#2#110#3#10#4#101#5#100@`

A segmentação da fita obedece à seguinte lógica funcional:

- Entrada Original (Azul): Preserva a sequência inicial fornecida à máquina. Esta região serve apenas como registro histórico da execução.
- Região de Processamento (Vermelho): Nesta área, os dados são convertidos para o formato (índice, valor). É sobre esta região que o algoritmo atua destrutivamente, decrementando os valores binários a cada ciclo para identificar o menor elemento.
- Região de Armazenamento (Verde): Contém uma cópia exata dos dados da Região 2 (índice, valor), porém estática. Esta região é utilizada para recuperar o valor original de um número uma vez que seu correspondente na região vermelha tenha sido reduzido a zero.
- Região de Saída (Não ilustrada): Uma quarta região será criada dinamicamente após o último marcador @ para armazenar os números, um a um, conforme forem ordenados.

O algoritmo opera ciclicamente sobre a Região 2 (Vermelha). A cada passada do cabeçote, todos os valores numéricos associados aos índices são decrementados em uma unidade binária. Considerando o exemplo anterior, na qual o menor valor é o binário `10` (decimal 2) localizado no índice 3, a fita apresentará o seguinte estado após a execução de dois ciclos de decremento:

`11#110#10#101#100@1#01#2#100#3#00#4#011#5#010@1#11#2#110#3#10#4#101#5#100@`

Note que na região de processamento (vermelha), o valor associado ao índice 3 atingiu `00`. Este evento dispara a transição de estado que leva o cabeçote à região de armazenamento (verde) para buscar o valor original do índice 3 e copiá-lo para a região de saída.

O processo repete-se até que todos os índices tenham sido processados. Ao final, a fita conterá, após o último marcador @, a sequência ordenada:

`@10#11#100#101#110`

Esta abordagem, embora custosa em termos de transições e espaço de fita, demonstra a capacidade da Máquina de Turing de simular manipulações complexas de memória auxiliar e indexação, conceitos fundamentais em algoritmos de ordenação. Para fins de reprodutibilidade e uso pedagógico, tanto a máquina de ordenação aqui descrita quanto as versões anteriores de operações aritméticas encontram-se disponíveis publicamente em repositório *online*[1].

A arquitetura de segmentação de fita e a lógica de decremento descritas acima, embora fundamentais para contornar as limitações de acesso sequencial e garantir a corretude do algoritmo, acarretam um custo operacional significativo. Para mensurar o impacto

[1] Acesse em: `https://surli.cc/ddsfsk`

dessa complexidade estrutural, a próxima seção apresenta os dados experimentais obtidos via simulador JFLAP, analisando o crescimento não linear do número de transições em função do tamanho da entrada e estabelecendo um comparativo de desempenho com as operações aritméticas desenvolvidas em trabalhos anteriores.

Visando a aplicação prática deste artefato em disciplinas de Teoria da Computação, propõe-se uma sequência didática baseada na progressão de complexidade algorítmica. O roteiro de aula sugerido inicia-se com a execução da Máquina de Soma [Campano Junior et al. 2019], permitindo aos discentes observar o comportamento linear do cabeçote, seguido pela Máquina de Multiplicação [Campano Junior et al. 2022], que aumenta a complexidade da operação matemática. A atividade finaliza com a experimentação da Máquina de Ordenação aqui descrita, na qual os alunos são instruídos a realizar testes comparativos com vetores de diferentes cardinalidades.

Importante ressaltar que, devido à densidade do diagrama de estados (133 nós), a mediação docente não deve focar na análise microestrutural das transições, mas sim direcionar a atenção dos discentes para o comportamento macroscópico da fita e a evolução do contador de passos. Esta abordagem prática tem por objetivo fazer com que o estudante vivencie empiricamente o salto de custo computacional, correlacionando as interrupções progressivas do simulador JFLAP com os conceitos teóricos de análise assintótica e ineficiência de acesso sequencial.

4. Análise de Complexidade e Resultados Experimentais

A validação da Máquina de Turing de ordenação foi realizada utilizando o simulador JFLAP. Diferente das implementações anteriores de operações aritméticas (soma e multiplicação), a máquina de ordenação apresentou uma complexidade estrutural significativamente maior, sendo constituída por 133 estados e um total de 1381 transições. Esta magnitude reflete a necessidade de gerenciar múltiplos marcadores, realizar a aritmética de decremento e controlar a movimentação de cópia de dados entre as regiões da fita.

4.1. Comportamento no Simulador e Custo Computacional

Durante a execução dos testes, observou-se um comportamento peculiar no JFLAP que serve como indicador visual da complexidade. O simulador interrompe a execução em intervalos de transições que seguem uma progressão geométrica $(500, 1.000, 2.000, \ldots)$ para confirmar a continuidade. Nos experimentos de ordenação, o tempo real de processamento entre essas confirmações aumentou sensivelmente, corroborando os dados quantitativos dos testes de estresse que revelaram o crescimento não linear do número de passos:

- **Pequena Escala (4 a 8 elementos):** Para vetores decrescentes pequenos, a máquina já exige entre 4.000 e 16.000 transições.
- **Média Escala (20 a 30 elementos):** O processamento escala rapidamente. Com 20 elementos, ultrapassa-se a marca de 250.000 transições. Ao atingir 31 elementos, a execução supera 500.000 transições.
- **Escala Extrema (127 elementos):** Em um teste de estresse massivo, a máquina ultrapassou a barreira de 32 milhões de transições para concluir a ordenação, mantendo, contudo, a corretude do resultado.

Esses resultados absolutos evidenciam a magnitude do esforço computacional exigido pela ordenação em fita única. No entanto, para contextualizar adequadamente esse custo e seu valor pedagógico, faz-se necessário contrastá-lo com o desempenho das operações aritméticas implementadas anteriormente, conforme detalhado na análise a seguir.

4.2. Análise Comparativa: Aritmética vs. Estrutura de Dados

A análise comparativa entre as máquinas desenvolvidas (Soma, Multiplicação e Ordenação) exige uma distinção crítica na natureza da entrada e na estratégia algorítmica. Enquanto o custo das operações aritméticas cresce estritamente em função da magnitude dos operandos (*bits*), a ordenação implementada apresenta um comportamento híbrido. Devido à estratégia de "busca pelo zero" via decremento sucessivo, a complexidade da máquina não é puramente dependente da cardinalidade do vetor (N), mas também sensível ao valor absoluto dos elementos (k). Esta característica classifica a implementação proposta como pseudo-polinomial, oferecendo uma rica oportunidade pedagógica para demonstrar aos alunos como decisões de implementação (decremento unário versus comparação binária) podem alterar drasticamente a classe de complexidade prática de um algoritmo, mesmo que a lógica de ordenação permaneça a mesma.

Conforme detalhado na Tabela 1, a Máquina de Soma apresenta crescimento linear eficiente: somar números de 30 *bits* exige pouco mais de 8.000 passos. A Máquina de Multiplicação, com arquitetura fixa para 8 *bits*, estabiliza-se entre 2.000 e 4.000 transições. Em contraste, a ordenação de uma quantidade similar de elementos (31 itens) é aproximadamente 64 vezes mais custosa que a soma de 30 bits, superando 512.000 transições.

Tabela 1. Comparativo de Transições: Aritmética vs. Estrutura de Dados

Cenário	**Soma (Qtd Bits)**	**Transições (Aprox.)**	**Ordenação (Tam. Vetor)**	**Transições (Aprox.)**
Pequena Escala	6 bits (Multiplicação 2 bits)*	< 500 (> 1.000)	4 elem.	> 4.000
Média Escala	10 bits (Multiplicação 8 bits)*	> 1.000 (> 2.000)	8 elem.	> 16.000
Alta Escala	30 bits	> 8.000	31 elem.	> 512.000
Escala Extrema	–	–	127 elem.	$> \mathbf{32.768.000}$

*Nota: A máquina de multiplicação opera com tamanho fixo de 8 bits na entrada.

4.3. Discussão Pedagógica

Para o aluno de Ciência da Computação, observar o simulador executar milhões de passos para ordenar uma lista que, para um humano, seria trivial, solidifica dois conceitos fundamentais:

1. **Complexidade de Algoritmos:** Visualiza-se na prática a diferença entre operações de custo linear e operações de custo quadrático ou superior.
2. **Computabilidade vs. Performance:** A experiência demonstra que a Máquina de Turing é um modelo poderoso de computabilidade, mas ineficiente como arquitetura prática. O custo de emular acesso aleatório em uma fita sequencial torna-se tangível.

Dessa forma, a ineficiência temporal observada transforma-se em um recurso didático intrínseco. Os dados quantitativos apresentados na Tabela 1 fundamentam a sequência didática sugerida na metodologia, pois a transição abrupta de magnitude, de milhares para milhões de passos, oferece o substrato empírico necessário para o roteiro de aula. Ao vivenciar o contraste entre a execução quase instantânea da soma e a longa espera exigida pela ordenação, o discente valida experimentalmente a hierarquia de complexidade, compreendendo que a possibilidade de resolver um problema não implica, necessariamente, em sua viabilidade prática sob certas arquiteturas.

5. Conclusão e Trabalhos Futuros

O ensino de Teoria da Computação e Linguagens Formais enfrenta o constante desafio de trabalhar conceitos abstratos. Enquanto autômatos finitos costumam ser intuitivos, a Máquina de Turing (MT) frequentemente é percebida pelos discentes apenas como um modelo matemático distante da realidade da programação prática.

Este trabalho apresentou o desenvolvimento e a análise técnica de uma Máquina de Turing capaz de executar o algoritmo *Selection Sort* em vetores binários. A implementação, composta por 133 estados e 1381 transições, expande o conjunto de ferramentas didáticas iniciado com as máquinas de soma [Campano Junior et al. 2019] e multiplicação [Campano Junior et al. 2022], elevando o nível de complexidade algorítmica disponível para exploração em sala de aula.

Os resultados experimentais demonstraram que, embora a MT seja teoricamente capaz de resolver qualquer problema computável, sua eficiência prática é severamente limitada pela natureza sequencial de sua memória (fita). A necessidade de estratégias complexas de manipulação de fita, como a segmentação em três regiões e a cópia para preservação de dados, evidenciou tecnicamente o "custo oculto" das operações que são triviais em linguagens de alto nível.

Do ponto de vista pedagógico, argumenta-se que a "explosão" no número de transições (superando 10^6 passos para vetores com $n = 30$) apresenta potencial para atuar como uma ferramenta visual de apoio. A hipótese deste trabalho é que, ao observar o simulador JFLAP solicitar confirmações repetidas para continuar o processamento, o aluno tenha a oportunidade de vivenciar empiricamente o conceito de complexidade de tempo. A máquina proposta, portanto, foi projetada não apenas para demonstrar como ordenar, mas para fornecer substrato visual que justifique o estudo de análise de algoritmos e arquitetura de computadores. Como desdobramentos desta pesquisa, os trabalhos futuros enquadram-se em três áreas de atuação:

- **Estudo Comparativo de Algoritmos na MT:** Implementação de outros algoritmos de ordenação $O(n^2)$, como o *Bubble Sort*, e algoritmos $O(n \log n)$, como o *Merge Sort*, para comparar o número de transições e movimentações do cabeçote, criando um *benchmark* de algoritmos clássicos rodando sobre o modelo de Turing.
- **Otimização via Múltiplas Fitas:** Desenvolvimento de uma versão da máquina de *Selection Sort* utilizando uma MT com múltiplas fitas (Multi-tape Turing Machine), visando demonstrar aos alunos como a adição de recursos de hardware (mais fitas) pode reduzir drasticamente a complexidade de movimentação, transformando o custo de cópia e comparação.

- **Validação em Sala de Aula:** Aplicação de um estudo de caso com turmas de graduação em Ciência da Computação, utilizando a máquina proposta para medir, através de pré e pós-testes, o impacto efetivo do uso do simulador na compreensão dos conceitos de decidibilidade e complexidade computacional.

A expectativa é que este trabalho contribua para o repertório didático da comunidade de Computação, oferecendo uma alternativa concreta para o ensino de complexidade. Ao permitir a visualização da execução mecânica de um algoritmo de ordenação, a ferramenta busca auxiliar na transposição das barreiras da notação matemática, com o intuito de tornar o estudo da computabilidade uma experiência mais experimental e intuitiva.

Declaração sobre uso de Inteligência Artificial

Este trabalho foi desenvolvido sem o uso de IA Generativa, sendo o texto e os elementos visuais de autoria exclusiva dos pesquisadores, que assumem total responsabilidade pela obra.

Agradecimentos

“Agradecimentos ao Manna Team, a Fundação Araucária de Apoio ao Desenvolvimento Científico e Tecnológico do Estado do Paraná (FA) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - Brasil (Processo nº 421548/2022-3) pelo apoio”.

Referências

Bressoud, T. C. and Thomas, G. (2019). A novel course in data systems with minimal prerequisites. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 15–21.

Campano Junior, M. M., Barbosa, C. R., Felinto, A., and Aylon, L. R. (2022). Multiplicando números binários com máquinas de turing: Interdisciplinaridade no ensino de computação. In *Anais do XXXIII Simpósio Brasileiro de Informática na Educação*, pages 1313–1323, Porto Alegre, RS, Brasil. SBC.

Campano Junior, M. M., Felinto, A. S., Sachs, C. R., and de Barbosa, C. (2019). Um merge entre máquina de turing e operações matemáticas em binário no ensino de linguagens formais e autômatos. In *Anais do XXIV Congresso Internacional de Informática Educativa. Nuevas Ideas en Informática Educativa, Volumen 15*, pages 78–83.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2012). Algoritmos-teoria e prática (3a. ediçao). *Editora Campus*.

Diverio, T. A. and Menezes, P. B. (2011). *Teoria da Computação: Máquinas Universais e Computabilidade Volume 5 da Série Livros Didáticos Informática UFRGS*, volume 5. Bookman Editora.

Ferreira, R., Canesche, M., and Penha, J. (2023). Google colab para ensino de computação. In *Anais Estendidos do III Simpósio Brasileiro de Educação em Computação*, pages 46–47. SBC.

Fukao, A. T., Colanzi, T. E., Martimiano, L. A., and Feltrim, V. D. (2023). Estudo sobre evasão nos cursos de computação da universidade estadual de maringá. In *Anais do III Simpósio Brasileiro de Educação em Computação*, pages 86–96. SBC.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65.

JFLAP (2026). Jflap version 7.1. https://www.jflap.org/ Acessado em março 2026.

Junior, B. A., Cavalheiro, S. A., and Foss, L. (2021). Theoretical computer science in basic education: A systematic review. In *Anais do VI Workshop-Escola de Informática Teórica*, pages 133–140. SBC.

Mioni, J. and Barbosa, C. (2022). Ferramentas para o aprendizado de linguagens formais e autômatos. In *Anais Estendidos do XXI Simpósio Brasileiro de Jogos e Entretenimento Digital*, pages 969–978, Porto Alegre, RS, Brasil. SBC.

Mundim, P., Silva, T., e Silva, G. B., and Barbosa, D. (2025). Evasão em cursos superiores na área de computação: Um mapeamento sistemático da literatura. In *Anais do XXXIII Workshop sobre Educação em Computação*, pages 1055–1067, Porto Alegre, RS, Brasil. SBC.

Oda, M., Noborimoto, Y., and Horita, T. (2021). International trends in k-12 computer science curricula through comparative analysis: Implications for the primary curricula. *International Journal of Computer Science Education in Schools*, 4(4):n4.

Paiva, P., Souza, M., and Terra, R. (2023). Ferramentas de apoio para a disciplina de linguagens formais e autômatos: uma proposta de uso. In *Anais do XXXIV Simpósio Brasileiro de Informática na Educação*, pages 1698–1709, Porto Alegre, RS, Brasil. SBC.

Pardalos, P. M. and Rajasekaran, S. (1999). *Data Structures and Algorithms*. John Wiley Sons, Ltd.

Sano, J., Yamamoto, N., and Ueda, K. (2023). Type checking data structures more complex than trees. *Journal of information processing*, 31:112–130.

Silva, J., Cavalheiro, S., and Foss, L. (2024). Automata theory in computing education: A systematic review. In *Anais do XXXV Simpósio Brasileiro de Informática na Educação*, pages 301–313, Porto Alegre, RS, Brasil. SBC.

Souza, , Matos, E., Santos, D., and Sousa, R. (2016). Recursos computacionais para suporte ao ensino de teoria da computação, linguagens formais e autômatos. In *Anais do XXIV Workshop sobre Educação em Computação*, pages 2373–2382, Porto Alegre, RS, Brasil. SBC.

Vieira, N. J. (2006). *Introdução aos fundamentos da computação: linguagens e máquinas*. Pioneira Thomson Learning.

Walker, D. and Morrisett, G. (2000). Alias types for recursive data structures. In *International Workshop on Types in Compilation*, pages 177–206. Springer.

Zendler, A., Klaudt, D., and Seitz, C. (2014). Empirical determination of competence areas to computer science education. *Journal of Educational Computing Research*, 51(1):71–89.

Zendler, A., Klaudt, D., Spannagel, C., and Reuter, T. (2013). Semantic categorization of content and process concepts relevant to computer science education. *International Journal of Research Studies in Computing*, 2(1):3–10.

Zendler, A., Seitz, C., and Klaudt, D. (2016). Process-based development of competence models to computer science education. *Journal of Educational Computing Research*, 54(4):563–592.