Implementação do Algoritmo de Ordenação Bubble Sort utilizando Máquina de Turing

Fernanda Sousa de Assunção Vale[1]; Jhones de Sousa Soares[2]
[1,2] Universidade Federal do Maranhão – Cidade Universitária Dom Delgado – São Luís – Maranhão
{fernanda.sav, jhones.sousa}@discente.ufma.br

Resumo
Este artigo apresenta a implementação do algoritmo de ordenação Bubble Sort utilizando uma Máquina de Turing, demonstrando a viabilidade de modelar algoritmos clássicos de ordenação. A Máquina de Turing é um modelo matemático fundamental que define a base da computação, sendo capaz de simular qualquer algoritmo computável. O Bubble Sort, embora ineficiente para grandes conjuntos de dados, é amplamente utilizado no ensino de algoritmos devido à sua simplicidade e fácil compreensão. Os resultados demonstram que, apesar das limitações da Máquina de Turing em termos de eficiência prática, o modelo é capaz de executar o Bubble Sort corretamente, reforçando sua aplicabilidade como ferramenta de ensino para conceitos fundamentais da computação.
Palavras-chave: Máquina de Turing. Bubble Sort. Algoritmo.

1. Introdução

Alan Turing revolucionou a computação ao criar a chamada Máquina de Turing, um modelo matemático capaz de representar qualquer cálculo computável. Esse modelo teórico serviu de base para o desenvolvimento da ciência da computação e demonstrou os limites do que é computacionalmente possível (Turing, 1936). A Máquina de Turing é uma ferramenta poderosa que permite a formalização de algoritmos e a verificação de sua viabilidade dentro de um contexto computacional universal. Desde sua concepção, tem sido utilizada para estudar problemas de decisão e complexidade computacional, sendo um marco fundamental para a teoria da computação (Sipser, 2018).

Dentre os diversos algoritmos de ordenação, o Bubble Sort destaca-se por sua simplicidade e fácil implementação. Ele funciona percorrendo repetidamente uma lista, comparando e trocando elementos adjacentes sempre que necessário, fazendo com que os maiores valores "subam" para o final da sequência. Apesar de ser ineficiente para grandes volumes de dados, sua lógica é intuitiva e serve bem para ensinar os conceitos básicos de ordenação e manipulação de dados (Cormen et al., 2009).

A relação entre o algoritmo Bubble Sort e a Máquina de Turing está na possibilidade de simulação desse algoritmo dentro desse modelo computacional teórico. A Máquina de Turing, ao ser programada para percorrer uma fita contendo os elementos a serem ordenados, pode realizar as operações de leitura, escrita e deslocamento para efetuar as trocas necessárias, simulando o funcionamento do Bubble Sort em um ambiente formal. Este estudo busca explorar essa implementação e analisar sua viabilidade e eficiência dentro desse contexto computacional.

2. Fundamentação Teórica

2.1 Máquina de Turing

A Máquina de Turing é um modelo matemático fundamental na teoria da computação, proposta por Alan Turing em 1936. A ideia central de uma Máquina de Turing é fornecer uma definição precisa de um processo computacional. Embora o modelo seja abstrato e simples, ele é poderoso o suficiente para simular qualquer computador moderno, o que a torna uma das bases da teoria da computação (Sipser, 2018). É definida formalmente como uma 8-tupla, conforme Figura 1 dada por:

$$M = \{\Sigma, Q, \delta, q_0, F, V, \beta, \circledast\}$$

Figura 1: Representação teórica da Máquina de Turing

Onde:

- Σ: alfabeto de símbolos de entrada;
- Q: conjunto finito de estados do autômato;
- δ: função programa da forma, conforme Figura 2:

$$\delta : Q \times (\Sigma \cup V \cup \{\beta, \circledast\}) \rightarrow Q \times (\Sigma \cup V \cup \{\beta, \circledast\}) \times \{E, D\}$$

Figura 2: Representação teórica da função programa

Em que:

- q0: estado inicial, tal que q0 ∈ Q;
- F: conjunto de estados finais, tal que F ⊂ Q;
- V: alfabeto auxiliar (pode ser vazio);
- β: símbolo especial branco;
- ⊛: símbolo ou marcador de início da fita.

É composta pelos seguintes elementos, que também estão representados na Figura 3:

- **Fita**: Uma sequência infinita de células, geralmente representada como uma linha contínua. Cada célula pode armazenar um símbolo de um conjunto finito e pode ser lida ou modificada pela máquina, permitindo que ela altere seu próprio estado;
- **Cabeça de leitura/gravação**: Um dispositivo que se move para a esquerda ou para a direita sobre a fita, lendo o símbolo de uma célula e escrevendo um novo símbolo quando necessário;
- **Estado interno**: A máquina sempre se encontra em um de um número finito de estados. O estado atual define como a máquina deve reagir ao símbolo lido na fita. Existem estados especiais, como o "estado de aceitação" e o "estado de rejeição", que determinam se o processamento termina com sucesso ou não;
- **Tabela de transições**: Um conjunto de regras que define o funcionamento da máquina com base no estado atual e no símbolo lido. Essas regras determinam:
  - O símbolo a ser escrito na fita;
  - A direção em que a cabeça se moverá (esquerda ou direita);
  - O próximo estado que a máquina assumirá.

- **Finalização:** A execução termina quando a máquina alcança um estado de aceitação ou rejeição, ou quando não há mais regras a serem aplicadas.

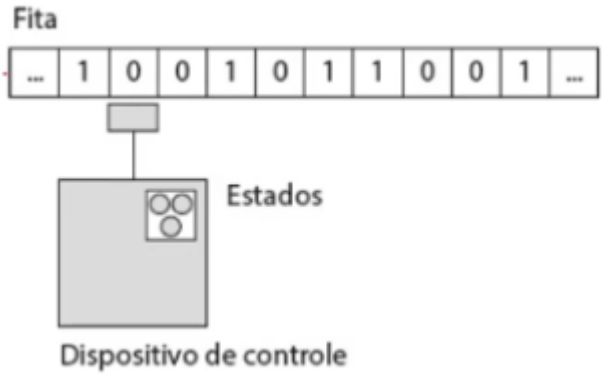

Figura 3: Elementos da Máquina de Turing

O conceito de Máquina de Turing está diretamente ligado à definição moderna de um computador. A noção de **equivalência de Turing** estabelece que qualquer modelo de computação com o mesmo poder de processamento de uma Máquina de Turing pode resolver os mesmos problemas computacionais. Isso inclui os computadores atuais, já que qualquer linguagem de programação pode ser expressa de forma equivalente a uma Máquina de Turing.

2.2 O Algoritmo Bubble Sort

O Bubble Sort é um dos algoritmos de ordenação mais simples e intuitivos na ciência da computação (Cormen et al., 2009). Apesar de sua simplicidade, ele não é considerado eficiente para grandes conjuntos de dados devido à sua complexidade de tempo, mas é útil em contextos educacionais para ilustrar conceitos básicos de algoritmos de ordenação. Seu funcionamento é dado da seguinte forma:

1. **Comparação e Troca**: O algoritmo começa comparando os primeiros dois elementos da lista. Se o primeiro for maior que o segundo, eles são trocados. O processo continua para os próximos pares de elementos adjacentes.
2. **Passagem Completa**: Após uma passagem completa pela lista, o maior elemento estará na última posição (daí o nome "Bubble", como uma bolha que sobe para o topo). Isso ocorre porque o maior elemento é movido para o final da lista a cada passagem.
3. **Repetição das Passagens**: O algoritmo repete esse processo para as n-1 primeiras posições (onde n é o número de elementos na lista). A cada nova passagem, a quantidade de elementos a serem comparados diminui, já que o maior elemento da lista já foi colocado na posição final.
4. **Condição de Parada**: O algoritmo termina quando uma passagem completa é feita sem nenhuma troca, indicando que a lista está ordenada.

Um exemplo do algoritmo é demonstrado na imagem a seguir, conforme Figura 4 a seguir.

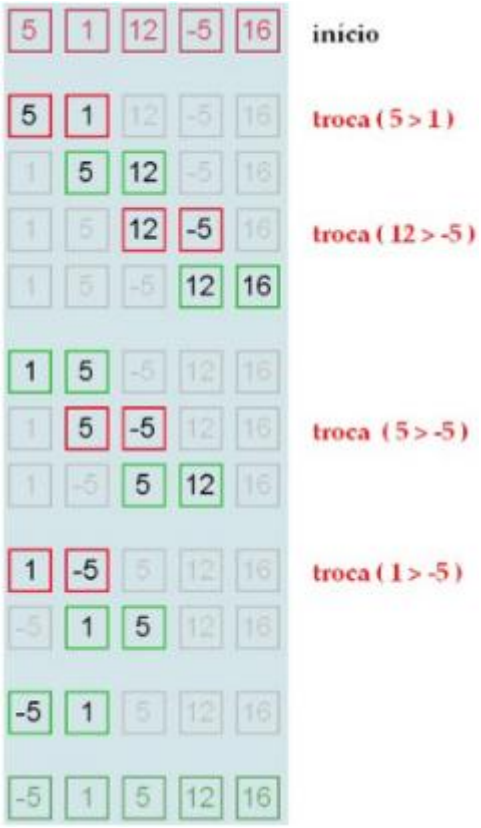

Figura 4: Funcionamento do Bubble Sort

Sua vantagem consiste na simplicidade, visto que é um algoritmo simples de entender e implementar. Sua desvantagem, como já citado, é que ele não é eficiente para grandes listas. No entanto, sua aplicação é valiosa em ambientes educacionais para ensinar os conceitos de algoritmos de ordenação e análise de complexidade computacional. Além disso, ele pode ser útil para ordenar listas pequenas ou em situações onde a simplicidade é mais importante que a eficiência.

3. Desenvolvimento

O primeiro passo no desenvolvimento da Máquina de Turing para Bubble Sort foi compreender a estrutura do algoritmo de ordenação e como ele poderia ser adaptado para uma representação baseada em estados. O Bubble Sort é um algoritmo iterativo que compara elementos adjacentes e os troca quando necessário, repetindo o processo até que a lista esteja ordenada. Para modelar essa lógica em uma Máquina de Turing, foi necessário definir os estados da máquina, as transições entre eles e a forma como a fita seria manipulada. Além disso, foi incluída uma funcionalidade de visualização para demonstrar a execução do algoritmo passo a passo. A representação do código está demonstrada conforme o Código 1 a seguir.

Código 1: `turing-machine.ipynb`

```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML

class MaquinaDeTuring:
    def __init__(self, fita):
        """
        Inicializa a Máquina de Turing com a fita de entrada.
        Converte os elementos da fita para inteiros e adiciona um espaço extra no final.
        """
        self.fita = [int(x) for x in fita] + [0]
        self.cabeca = 0  # Define a posição inicial da cabeça de leitura
        self.estado = 'inicio'  # Define o estado inicial da máquina
        self.flag_troca = 0  # Indica se houve troca na última passagem
        self.historico = []  # Registra o histórico de ações
        self.contador_passos = 0  # Contador de passos executados
        self.passos_executados = []  # Lista para armazenar os estados da fita para animação

    def registrar_acao(self, acao):
        """
        Registra uma ação da Máquina de Turing no histórico.
        Salva o estado atual da fita e da cabeça de leitura.
        """
        self.contador_passos += 1
        log = f"Passo {self.contador_passos}: {acao} | Flag de Troca: {self.flag_troca}"
        self.historico.append(log)
        self.passos_executados.append((self.fita[:], self.cabeca, self.estado, self.flag_troca, log))

    def passo(self):
        """
        Executa um único passo da Máquina de Turing.
        Realiza comparações, trocas e movimentação da cabeça de leitura.
        """
        if self.estado == 'parado':
            return

        simbolo_atual = self.fita[self.cabeca]  # Obtém o valor na posição atual da cabeça

        if self.estado == 'inicio':
            # Reinicia a passagem pela fita
            self.cabeca = 0
            self.flag_troca = 0
            self.estado = 'comparar'
            self.registrar_acao("INICIAR NOVA PASSAGEM | Cabeça na posição 0")

        elif self.estado == 'comparar':
            if self.cabeca >= len(self.fita) - 2:  # Se chegou ao final da fita
                self.estado = 'verificar_flag'
                self.registrar_acao("Chegou ao final | Verificar flag de troca")
            else:
                proximo_simbolo = self.fita[self.cabeca + 1]
                if simbolo_atual > proximo_simbolo:
                    # Identifica necessidade de troca
                    self.estado = 'trocar'
                    self.registrar_acao(f"COMPARAR [{simbolo_atual} >
```

```
    {proximo_simbolo}] | Necessário trocar")
                else:
                    # Move a cabeça para frente sem troca
                    self.cabeca += 1
            self.registrar_acao(f"COMPARAR [{simbolo_atual} ≤ {proximo_simbolo}] | Sem troca")

        elif self.estado == 'trocar':
            # Troca os valores adjacentes
            self.fita[self.cabeca], self.fita[self.cabeca + 1] = self.fita[self.cabeca + 1], self.fita[self.cabeca]
            self.flag_troca = 1  # Marca que houve troca
            self.registrar_acao(f"TROCAR [{self.fita[self.cabeca]} ⇄ {self.fita[self.cabeca+1]}] | Mover para {self.cabeca+1}")
            self.cabeca += 1
            self.estado = 'comparar'

        elif self.estado == 'verificar_flag':
            if self.flag_troca == 1:
                # Se houve trocas, reinicia a ordenação
                self.estado = 'inicio'
                self.registrar_acao("Flag = 1: Houve trocas | Reiniciar máquina")
            else:
                # Se não houve trocas, a ordenação está concluída
                self.estado = 'parado'
                self.registrar_acao("Flag = 0: Ordenação concluída")

    def executar(self):
        """
        Executa a Máquina de Turing até que o estado seja 'parado'.
        Retorna a fita ordenada e o histórico de ações.
        """
        while self.estado != 'parado':
            self.passo()
        return self.fita[:-1], self.historico  # Retorna a fita ordenada e o histórico de ações

# Definição da fita inicial (números a serem ordenados)
fita_inicial = ['1', '4', '4', '3', '2']
mt = MaquinaDeTuring(fita_inicial)
mt.executar()
quadros = mt.passos_executados  # Armazena os quadros para a animação

# Configuração da animação
fig, ax = plt.subplots()

def atualizar(frame):
    """
    Atualiza o gráfico da animação para representar um quadro específico do processo.
    """
    fita, cabeca, estado, flag_troca, log = quadros[frame]
    ax.clear()
    ax.set_title(f"Máquina de Turing - Bubble Sort - Passo {frame+1}")

    # Representação da fita como string
    fita_str = ' | '.join(map(str, fita[:-1]))
    estado_info = f"Estado: {estado}"
    flag_info = f"Flag de Troca: {flag_troca} (1 = Reinicia, 0 = Concluído)"

    # Exibição de informações na animação
```

```
111.     ax.text(0.5, 0.65, fita_str, ha='center', va='center', fontsize=16,
   family='monospace')
112.     ax.text(0.5, 0.45, estado_info, ha='center', va='center',
   fontsize=12, color='red')
113.     ax.text(0.5, 0.35, flag_info, ha='center', va='center', fontsize=12,
   color='green')
114.     ax.text(0.5, 0.2, log, ha='center', va='center', fontsize=10,
   color='blue')
115.
116.
117.     #x_pos = 0.28 + cabeca * 0.1  # Calcula a posição X da seta na tela
118.     #ax.annotate('↓', (x_pos, 0.75), textcoords="offset points",
   xytext=(0, 10), ha='center', fontsize=20, color='purple')
119.
120.     ax.axis("off")
121.
122. # Criar animação com base no número real de passos
123. num_frames = len(quadros)
124. animacao_mt = animation.FuncAnimation(fig, atualizar, frames=num_frames,
   interval=500, repeat=False)
125. HTML(animacao_mt.to_jshtml())
```

Após esses passos foi possível visualizar cada etapa da fita da Máquina de Turing. A estrutura de estados garante que cada passo do algoritmo seja fiel ao funcionamento de uma Máquina de Turing real, permitindo um entendimento profundo do processo de ordenação.

4. Resultados

Com os resultados obtidos durante a execução do algoritmo adaptado para uma máquina de Turing, pode-se observar a clara eficiência desta implementação. O algoritmo conseguiu ordenar com sucesso qualquer lista de número em que os elementos façam parte do alfabeto de entrada da fita. Inicialmente foram conduzidos vários testes com listas diferentes para avaliar a eficácia da implementação.
Um dos experimentos conduzidos, buscou ordenar a lista de número (1,4,4,3,2), onde o algoritmo necessitaria realizar as operações de troca e verificação.
A figura 4 demonstra uma representação em grafo de um dos resultados obtidos:

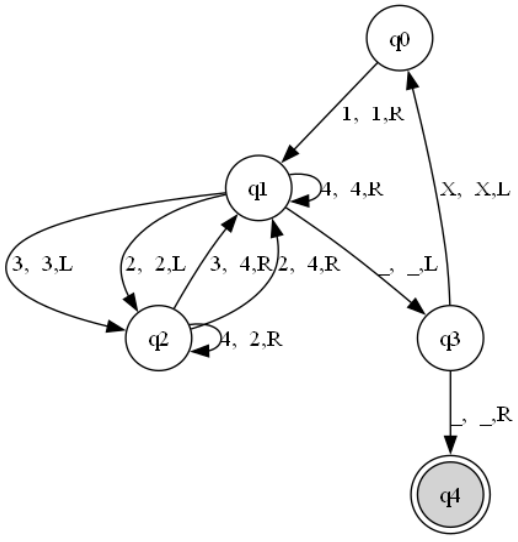

Figura 4: Diagrama de ordenação da lista (1,4,4,3,2)

Também podemos representar este resultado a partir da tabela de transição abaixo:

Estado Atual	Símbolo lido	Próximo estado	Símbolo escrito	Direção do movimento	Condição
q0	1	q1	1	Direita	Movimenta para o próximo
q1	4	q1	4	Direita	Compara com o próximo
q1	4	q1	4	Direita	Compara com o próximo
q1	3	q3	3	Esquerda	Troca para a esquerda
q2	3	q1	4	Direita	Troca para a direita
q1	2	q2	2	Esquerda	Troca
q2	4	q2	2	Direita	Troca para a esquerda
q2	2	q1	4	Direita	Troca para a direita
q1	_	q3	_	Esquerda	Fim da fica, checar a flag
q3	X	q0	X	Esquerda	Reiniciar se houver alguma troca
q3	_	q4	_	_	Fim da ordenação

Tabela 1: Tabela de transição

Como saída final do código, obtivemos a lista ordenada (1,2,3,4,4), comprovando assim a eficácia do algoritmo na tarefa de ordenação.

5. Conclusão

A implementação do algoritmo Bubble Sort em uma Máquina de Turing demonstrou-se viável e eficaz, cumprindo os objetivos propostos ao validar a capacidade de modelar algoritmos clássicos de ordenação em um contexto teórico. A aplicação também incorpora uma representação visual dinâmica por meio de animações geradas com matplotlib, tornando tangíveis os processos abstratos de leitura, escrita e movimentação da fita. Essa abordagem visual, aliada à estrutura de estados e transições da Máquina de Turing, facilitou a compreensão dos fundamentos computacionais.

Destaca-se que, embora a Máquina de Turing não seja prática para ordenação de grandes volumes de dados devido à sua natureza sequencial e teórica. Ao simular o Bubble Sort, ele ilustra como modelos computacionais abstratos podem ser aplicados a problemas reais, servindo como ponte entre a teoria e a prática. A visualização passo a passo da execução, com destaque para trocas de elementos e estados intermediários, permite que estudantes internalizem não apenas o funcionamento do algoritmo, mas também a lógica subjacente à computação universal.

Além disso, o trabalho estabelece uma base para expansões futuras. Possíveis melhorias incluem a adaptação para outros algoritmos de ordenação, como Insertion Sort ou Merge Sort, ou a incorporação de análises de complexidade em tempo real durante a execução. A estrutura modular do código também permite a integração com ferramentas interativas, potencializando seu uso em ambientes de ensino.

Referências

RAMOS, MARCUS V. M. **Linguagens formais: teoria, modelagem e implementação.** 1ª ed. Porto Alegre: Bookman, 2009.

CORMEN, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

SIPSER, M.. Introdução à Teoria da Computação. 1. ed. São Paulo: Editora Manole, 2018.

TURING, A. M. On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society. 1936

Reconhecimentos e Direitos Autorais

@autor: Jhones de Sousa e Fernanda Assunção

@data última versão: 20/02/2025
@versão: 1.0
@outros repositórios: https://github.com/Jhones257/turing-machine-sorting-algorith/tree/main
@Agradecimentos: Universidade Federal do Maranhão (UFMA), Professor Doutor Thales Levi Azevedo Valente, e colegas de curso.