UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS

Relatório TP03: Casamento de Padrões

Disciplina: Algoritmos e Estruturas de Dados II

Milliany Rodrigues de Morais

João Monlevade, fevereiro de 2026

Sumário

1 Introdução

A busca de padrões em strings é um problema clássico da Ciência da Computação, com aplicações que variam desde editores de texto e bioinformática até a detecção de plágio. Formalmente, dado um texto T de tamanho n e um padrão P de tamanho m, o objetivo é encontrar todos os índices i onde ocorre o casamento exato $T[i..i+m-1] = P[0..m-1]$.

Este trabalho prático tem como objetivo o estudo, a implementação e a comparação de diferentes algoritmos de busca, analisando seus comportamentos teóricos e empíricos.

2 Metodologia

A metodologia deste trabalho consistiu na implementação sistemática de seis algoritmos de casamento de padrões e na realização de testes para coleta de dados de desempenho. O desenvolvimento foi realizado na linguagem Python, utilizando uma abordagem modular para facilitar a comparação entre os métodos.

2.1 Algoritmos Implementados

- **Força Bruta**: Implementado como o método de referência, realizando o deslocamento de uma posição a cada iteração. Sua complexidade de pior caso é $O(n \cdot m)$.
- **Rabin-Karp**: Utiliza uma função de *hash* e o operador MOD com um número primo grande para evitar colisões.
- **Knuth-Morris-Pratt (KMP)**: Utiliza a Função Prefixo (π), pré-calculada em tempo $O(m)$, permitindo "pular"casas no texto sem retrocessos.
- **Boyer-Moore (BM)**: Implementação completa utilizando as heurísticas de mau-caractere e bom-sufixo.
- **Boyer-Moore-Horspool (BMH)**: Simplificação do BM que utiliza apenas a heurística do mau-caractere para determinar os saltos.

- **Boyer-Moore-Horspool-Sunday (BMHS)**: Variante que considera o caractere imediatamente após a janela de comparação para determinar o deslocamento.

3 Experimentos e Resultados

Abaixo será exposto os resultados comparativos obtidos através da execução dos algoritmos. Os testes foram estruturados para observar o comportamento de cada método em relação ao tempo de processamento e ao esforço computacional.

3.1 Comparativo de Desempenho: Cenário DNA

O teste utilizou um texto de 100.000 caracteres (n) composto por um alfabeto reduzido ($\Sigma = \{A, T, G, C\}$) e um padrão de 8 caracteres (m).

Algoritmo	**Tempo (ms)**	**Comparações**	**Ocorrências**
Força Bruta	33,22	274.986	24.999
Rabin-Karp	42,58	199.992	24.999
Knuth-Morris-Pratt (KMP)	16,83	100.000	24.999
Boyer-Moore (BM)	27,31	199.992	24.999
Boyer-Moore-Horspool (BMH)	27,01	199.992	24.999
Boyer-Moore-Horspool-Sunday	25,87	199.992	24.999

Tabela 1: Comparativo de desempenho dos algoritmos no cenário de alfabeto reduzido.

3.2 Análise Crítica dos Resultados

A análise dos dados experimentais permite verificar as vantagens e limitações de cada abordagem frente ao cenário de alfabeto reduzido ($\Sigma = \{A, T, G, C\}$) e padrões repetitivos.

- **Eficiência Linear do KMP**: O algoritmo Knuth-Morris-Pratt (KMP) apresentou o melhor desempenho geral, com exatamente 100.000 comparações. Isso reflete sua complexidade $O(n + m)$ no pior caso, onde

a Função Prefixo (π) evita que o ponteiro do texto retroceda, aproveitando o conhecimento prévio do padrão. Em alfabetos pequenos como o DNA, essa característica o torna superior às heurísticas de salto.

- **Limitações da Família Boyer-Moore**: Observa-se que BM, BMH e Sunday apresentaram um número idêntico de comparações (199.992). Embora tenham reduzido o esforço em relação à Força Bruta, a eficácia de suas heurísticas de salto é mitigada pela baixa variabilidade do alfabeto. Em alfabetos curtos, a regra do "mau-caractere"gera deslocamentos menores, pois os mesmos caracteres aparecem frequentemente no padrão.

- **Desempenho Prático do Sunday**: Apesar de realizar o mesmo número de comparações que o BMH, o algoritmo Sunday foi marginalmente mais rápido em termos de tempo de execução (25.87 ms). Isso demonstra que a estratégia de observar o caractere imediatamente após a janela atual pode oferecer vantagens de velocidade prática em implementações reais.

- **Ineficiência do Rabin-Karp**: O algoritmo Rabin-Karp apresentou o maior tempo de execução (42.58 ms). Embora o número de comparações de caracteres tenha sido reduzido, o custo computacional de calcular funções de *hash* e gerenciar operações aritméticas a cada iteração introduz um *overhead* significativo em Python. Além disso, alfabetos pequenos aumentam a probabilidade de colisões de *hash*, exigindo verificações manuais frequentes.

- **Referencial de Força Bruta**: O método de Força Bruta, como previsto, foi o menos eficiente em termos de comparações (274.986). O deslocamento unitário gera um esforço redundante de $O(n \cdot m)$, servindo como base para quantificar o ganho de eficiência dos demais métodos.

O custo de pré-processamento de cada método foi isolado, confirmando que, para textos de grande escala ($n = 100.000$), o investimento inicial na construção de tabelas e funções de falha é amplamente compensado pela economia no processo de busca.

4 Link para o GitHub

https://github.com/MilliMoraes/Casamento-de-padroes.git

5 Conclusão

A análise comparativa permite concluir que a eficiência de um algoritmo de busca de padrões é fortemente influenciada pela estrutura do alfabeto e do padrão escolhido. O algoritmo de Força Bruta, como observado, apresentou um número de comparações significativamente superior ao tamanho do texto ($n = 100.000$), devido ao seu deslocamento sequencial unitário que ignora casamentos parciais.

Em contrapartida, os testes demonstraram que algoritmos como o Knuth-Morris-Pratt (KMP) são ideais para alfabetos pequenos (DNA), pois garantem complexidade linear ao aproveitar o conhecimento prévio do padrão para evitar retrocessos. Já a família Boyer-Moore, apesar de eficiente, teve seu potencial de salto limitado pela baixa variabilidade do alfabeto testado.

O trabalho evidenciou que o investimento computacional no pré-processamento de tabelas e funções de falha é uma estratégia fundamental para otimizar buscas em grandes volumes de dados, tornando o custo inicial desprezível diante do ganho de velocidade na fase de busca.