Universidade Federal
de São João del-Rei

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROJETO E ANÁLISE DE ALGORITMO

TRABALHO PRÁTICO 3

Diogo Augusto Martins Honorato

Guilherme Garcia de Oliveira

São João del-Rei

2025

Sumário

1 Introdução

Este é um trabalho prático da disciplina de Projeto e Análise de Algoritmos no curso de Ciência da Computação na UFSJ, tendo como docente o professor Leonardo Rocha.

1.1 Objetivo

Neste trabalho prático, temos como objetivo ampliar e exercitar os conceitos aprendidos sobre processamento de cadeia de caracteres, podendo compreender na prática a implementação de quatro algoritmos diferentes para tal propósito.
Desta forma, fomos apresentados ao problema, onde, nesse cenário hipotético, somos imcubidos de ajudar nossa vizinha que estuda Música na UFSJ; para ajudá-la, precisamos desenvolver um algoritmo que receba uma determinada sequência de notas da melodia de uma música, e uma outra sequência de notas de melodia suspeito, onde assim nosso algoritmo deve averiguar se o trecho suspeito ocorre, em algum tom, na música dada.
Para isso, iremos implementar quatro métodos distintos de busca de caracteres, e comparar seus desempenhos; tais métodos sendo Força Bruta, KMP, Boyer-Moore(BMH) e Shift-And Exato.

1.2 Motivação

Com base na descrição do problema, e no requerimento para sua resolução, podemos concluir tratar-se de um problema que explorará os conceitos estudados de processamento de cadeia de caracteres e implementação dos métodos KMP, BMH, Shift-And Exato e Força Bruta.

2 Abordagem do Problema

Após analisar o problema, optamos por começar analisando as notas com alteração cromática e convertê-las em caracteres normais para facilitar a comparação e o armazenamento.
Com isso, criamos uma estrutura para armazenar os textos, padrões e seus respectivos tamanhos chamada PlagiChecker. Tendo esta estrutura pronta, só restou implementar os algoritmos requeridos.

Tendo todos os algoritmos implementados, desenvolvemos uma função separada, com nome de Benchmark, que é responsável por executar o algoritmo que o usuário escolheu, calcular o tempo de execução do algoritmo e o número de comparações que são realizadas.

3 Introdução aos Algoritmos

Para compreender melhor as análises deste trabalho, iremos, de maneira breve, explicar um pouco do funcionamento dos algoritmos implementados, sendo eles Boyer-Moore-Horspool(BMH), Knuth-Morris-Pratt(KMP), Shift-And Exato e Força Bruta.
Para todas as análises, consideramos n como o tamanho do texto e m o tamanho do padrão.

3.1 Força Bruta

O algoritmo Força Bruta é a abordagem mais simples para buscar um padrão dentro de um texto. Ele compara cada posição possível do texto com o padrão caractere por caractere, sem utilizar otimizações.

Funcionamento da Força Bruta

1. Pesquisa no Texto (O(nm) no pior caso)

Alinha o padrão com cada posição possível no texto.

Compara cada caractere do padrão com o texto, avançando até encontrar um descasamento.

Se todos os caracteres coincidirem, retorna a posição do casamento.

Caso contrário, move o padrão apenas uma posição para a direita e repete o processo.

3.2 Boyer-Moore-Horspool (BMH)

O BMH (Boyer-Moore-Horspool) é um algoritmo de busca de padrões otimizado para eficiência, baseado no Boyer-Moore, mas simplificado. Ele usa uma tabela de deslocamento para pular múltiplas posições no texto, evitando verificações desnecessárias.

Funcionamento do BMH

1. Pré-processamento (O(m))

Cria uma tabela de deslocamento (d[]) baseada nos últimos caracteres do padrão.

Para cada caractere do alfabeto, define quantas posições pular se houver um descasamento.

2. Pesquisa no Texto (O(n/m) no melhor caso, O(nm) no pior)

Alinha o padrão com o texto.

Compara da direita para a esquerda.

Se houver um descasamento, usa a tabela de deslocamento para pular posições.

Se houver casamento completo, retorna a posição da ocorrência.

3.3 Knuth-Morris-Pratt (KMP)

O KMP (Knuth-Morris-Pratt) é um algoritmo eficiente de busca de padrões que evita comparações redundantes ao usar uma tabela de prefixos. Isso permite que ele processe o texto em tempo linear O(n + m).

Funcionamento do KMP

1. Pré-processamento (O(m))

Constrói a tabela de prefixo (tabela[]), que armazena informações sobre repetições dentro do padrão.

Essa tabela indica quantos caracteres do padrão podem ser reutilizados após um descasamento.

2. Pesquisa no Texto (O(n))

Compara o padrão com o texto da esquerda para a direita.

Se houver um descasamento, usa a tabela de prefixo para evitar reavaliações desnecessárias.

Se houver um casamento completo, retorna a posição da ocorrência.

3.4 Shift-And Exato

O Shift-And é um algoritmo de busca de padrões baseado em operações bitwise, tornando-o extremamente eficiente para padrões curtos (até 128 caracteres). Ele repre-

senta o padrão como máscaras de bits e usa deslocamentos binários para processar o texto rapidamente.

Funcionamento do Shift-And Exato

1. Pré-processamento (O(m))

Cria uma tabela de bitmasks, onde cada caractere do alfabeto é representado por um número binário.

Cada bit da máscara indica a presença do caractere em uma posição do padrão.

2. Pesquisa no Texto (O(n))

Mantém um registrador de estado R para rastrear correspondências.

Para cada caractere do texto, atualiza R com uma operação bitwise (AND e SHIFT).

Se o bit menos significativo for 1, encontrou uma ocorrência do padrão.

4 Análise de Complexidade

4.1 Força Bruta

O algoritmo de Força Bruta compara o padrão com todas as possíveis posições dentro do texto, caractere por caractere.

1. Complexidade no Melhor Caso - O(n)

O melhor caso ocorre quando o padrão é encontrado logo na primeira posição do texto.
O algoritmo faz apenas m comparações e retorna a posição do casamento.
Complexidade: O(m) $\rightarrow$ Considerando que m é menor que n, podemos dizer que no melhor caso ele roda em O(n).

2. Complexidade no Pior Caso - O(nm)

O pior caso ocorre quando não há casamento ou há muitos falsos positivos iniciais (exemplo: padrões repetitivos no texto).
O algoritmo verifica cada posição do texto (n - m + 1 vezes) e compara até m caracteres em cada tentativa.
Complexidade: O(nm).

3. Complexidade no Caso Médio - O(nm)

Se o padrão ocorre poucas vezes no texto, o algoritmo pode parar cedo, reduzindo o número de comparações.

No entanto, em textos aleatórios, ele ainda pode executar O(nm) comparações.

Complexidade: O(nm) em geral, mas pode ser melhor dependendo do padrão e do texto.

Conclusão - Complexidade Total

Melhor caso: O(n) (casamento na primeira posição). Caso médio: O(nm) (depende do conteúdo do texto).

Pior caso: O(nm) (padrões repetitivos ou texto sem casamento).

4.2 Boyer-Moore-Horspool (BMH)

O algoritmo Boyer-Moore tem dois estágios principais:

1. Pré-processamento do padrão = O(m)
2. Pesquisa no texto = O(n/m) no melhor caso e O(nm) no pior caso

Complexidade do Pré-processamento

O primeiro loop percorre MaxTamAlfabeto (127 no código), mas isso pode ser considerado O(1), pois o tamanho do alfabeto é constante.

O segundo loop percorre tamPadrao - 1, que equivale a O(m)

Assim, o pré-processamento tem complexidade O(m)

Complexidade da Pesquisa

Melhor caso - O(n/m)

Se os deslocamentos forem grandes, o número de iterações do while (i $\leq$ tamTexto) será aproximadamente n/m.

Como há poucas comparações no while (j $\geq$ 0 && texto[k - 1] == padrao[j]), o tempo total será aproximadamente O(n/m).

Pior caso - O(nm)

Se o texto for composto de repetições da parte final do padrão, pode haver m comparações antes de cada deslocamento.

Nesse caso, o loop aninhado pode executar O(m) vezes para cada deslocamento, resultando

em O(nm) no pior caso.

Complexidade Geral

Melhor caso: O(n/m)

Pior caso: O(nm)

Caso médio: O(n) (se o padrão for razoavelmente discriminativo e os deslocamentos forem grandes).

4.3 Knuth-Morris-Pratt (KMP)

O algoritmo KMP (Knuth-Morris-Pratt) tem dois estágios principais:

1. Pré-processamento do padrão → O(m)
2. Pesquisa no texto → O(n)

Complexidade do Pré-processamento

O loop while (j < tamanhoPadrao) percorre no máximo m vezes.

Cada iteração avança j ou reduz indice, garantindo que o tempo total é O(m).

Não há loops aninhados significativos que aumentem a complexidade além de O(m).

Complexidade da Pesquisa

O loop while (i < tamanhoTexto && j < tamanhoPadrao) percorre no máximo n vezes.

Cada iteração:

Avança i se houver correspondência.

Retrocede j para evitar reavaliação desnecessária de caracteres.

Como cada caractere de texto é comparado no máximo uma vez, a complexidade é O(n).

Complexidade Geral

Pré-processamento: O(m)

Pesquisa: O(n)

Complexidade total: O(n + m)

Este é um dos principais benefícios do KMP: evita comparações desnecessárias e mantém eficiência mesmo em casos ruins.

4.4 Shift-And Exato

Assim como os anteriores, esse algoritmo possui dois estágios principais:

1. Pré-processamento do padrão → O(m)
2. Pesquisa no texto → O(n)

Complexidade do Pré-processamento

O loop for (int i = 0; i < tamPadrao; i++) percorre m vezes.

Como bitMasks tem um tamanho fixo de TAM_ALFABETO, a inicialização pode ser considerada O(1).

A complexidade total do pré-processamento é O(m).

Complexidade da Pesquisa

O loop for (int i = 0; i < tamanhoTexto; i/++) percorre no máximo n vezes.

Em cada iteração, ocorre uma atualização bitwise em tempo constante O(1).

A complexidade da pesquisa é O(n).

Complexidade Geral

Pré-processamento: O(m)

Pesquisa: O(n)

Complexidade total: O(n + m)

O Shift-And é mais eficiente para padrões curtos ($\leq$ 64 ou $\leq$ 128 bits), pois pode ser implementado com operações bitwise rápidas.

5 Testes

Diversos testes foram realizados com o objetivo de reunir resultados para comparação, sendo obtidos através das funções "gettimeofday".

Para compararmos a eficiência entre os algoritmos, analisamos seus tempos de execução em métricas diferentes e seus números de comparações.

Para todos os gráficos, executamos 100.000 testes, com tamanho do texto sendo 1000 e tamanho do padrão sendo 128 caracteres.

Na figura abaixo, comparamos apenas os números de comparações que os algoritmos realizam até encontrar o padrão, demarcando o número máximo de comparações alcançado com cada um deles, o menor número e a média de comparações.

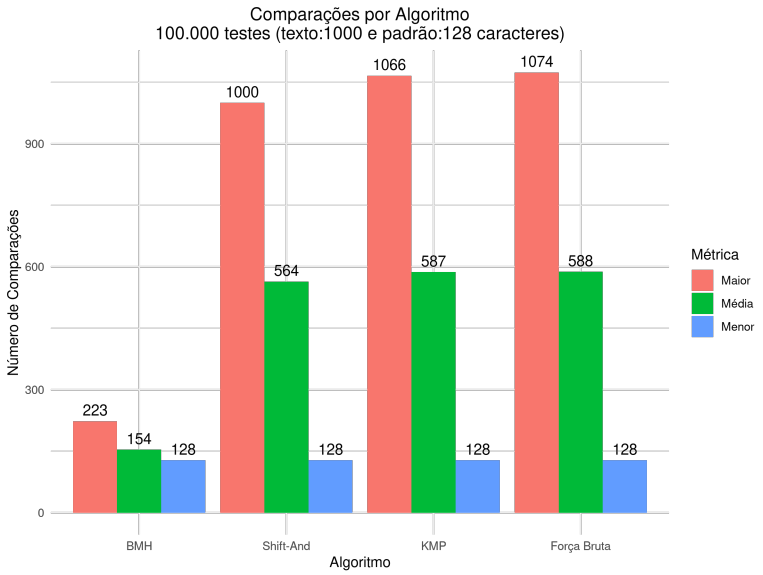

Figura 1: Número de comparações por cada algoritmo

Como pode ser observado, quando se trata do menor número de comparações, todos os algoritmos seguiram a mesma faixa; as diferenças aparecem a partir dos valores médios e máximos de comparações.

Ao analisarmos a média e a máxima, percebemos que, obviamente, o força bruta executa o maior número de comparações, com os outros algoritmos sendo mais eficientes, com destaque para o BMH que, dentre todos, sempre executa o menor número de comparações.

Neste próximo gráfico, buscamos colocar a posição de cada teste no plano ao relacionarmos o número de comparações ao tempo de execução. Para análise, as regiões mais escuras mostram o acúmulo maior de testes naquela localidade, sendo que cada teste é um número n de comparações necessários para encontrar a solução.

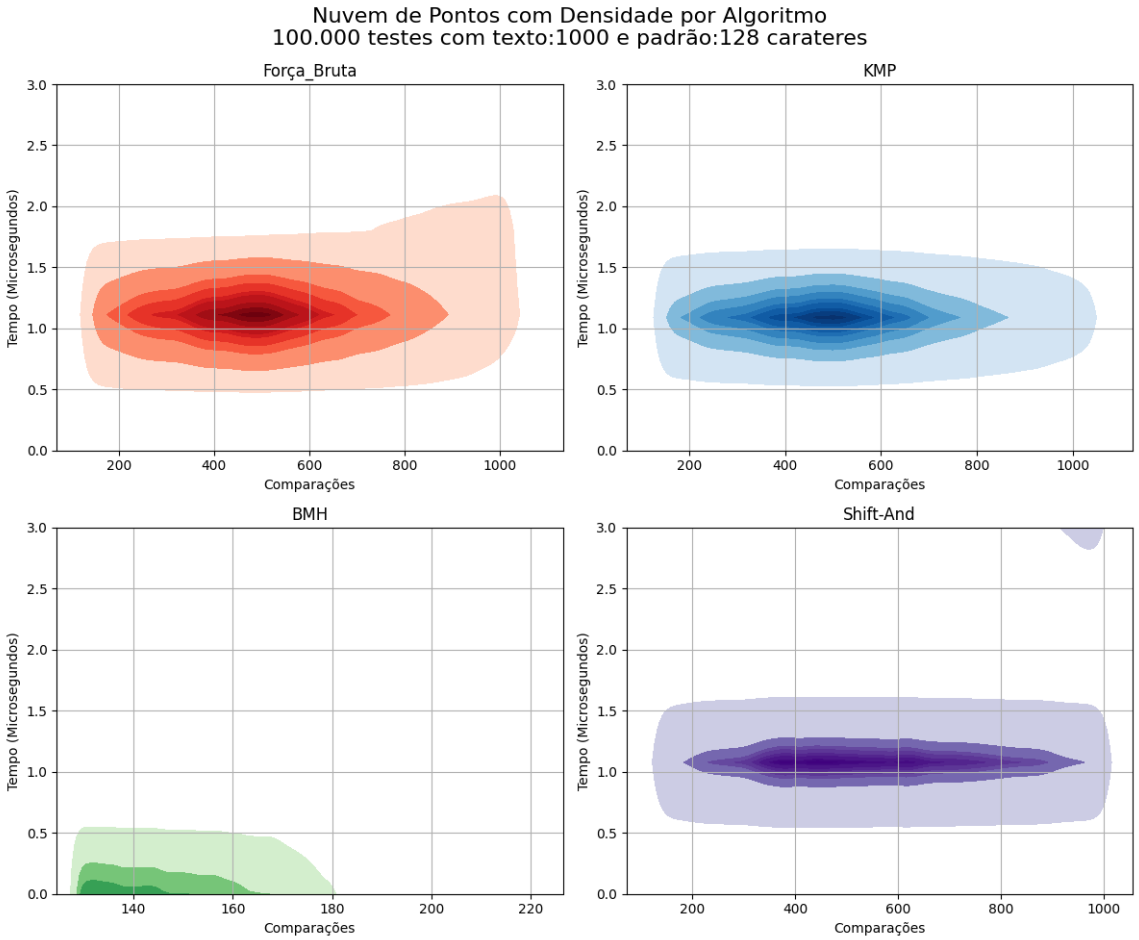

Figura 2: Densidade de testes

Observando o gráfico, podemos analisar que, a média do força bruta para encontrar a solução é entre 400 e 600 comparações; porém, conforme o número de comparações subia o tempo de execução tende a subir também. Já nos algoritmos de KMP e Shift-And, eles tendem a manter uma média de tempo mais estável independente da quantidade de comparações necessárias. É possível analisar também que, no Shift-And, a região mais escura é mais "alongada"em relação aos outros algoritmos, o que demonstra uma maior faixa de cobertura da solução mantendo o mesmo tempo de execução.

Em relação a todos os outros, o BMH concentrou seu número de comparações e tempo de execução em valores baixos, sem muitas alterações.

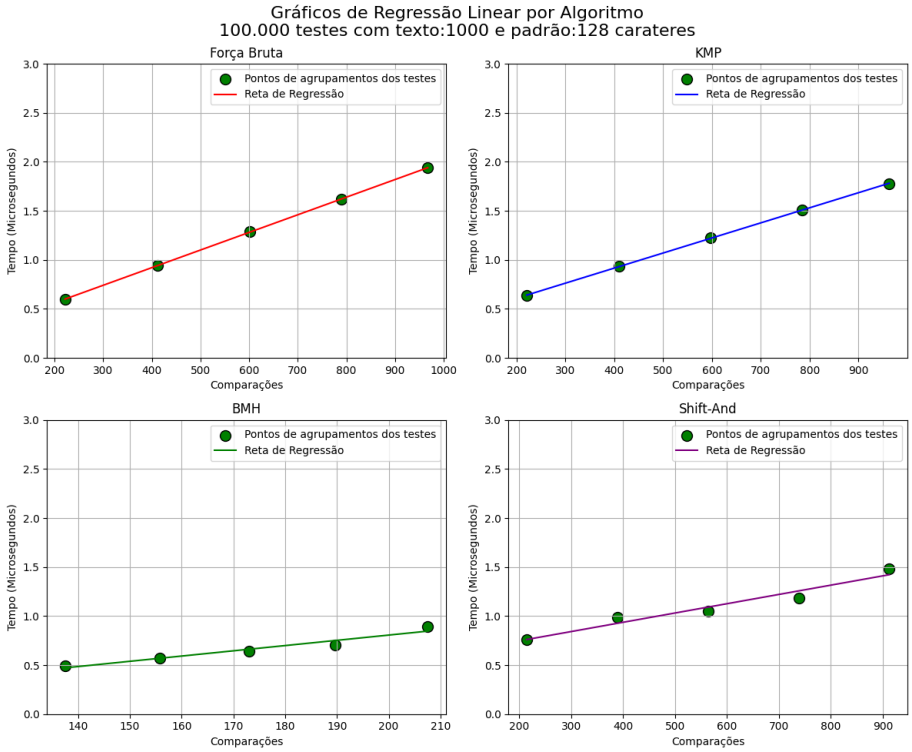

Figura 3: Gráfico de regressão

O gráfico acima representa a reta de regressão dos algoritmos, que mostra a tendência de comportamento dos métodos aplicados.

A inclinação das retas de regressão nos gráficos representa a taxa de variação do tempo em relação ao número de comparações realizadas pelos algoritmos. Em termos práticos, uma inclinação menor indica maior eficiência, pois o tempo de execução cresce mais lentamente com o aumento do número de comparações.

Como podemos observar, o Força Bruta possui uma inclinação maior na reta em relação aos outros algoritmos, demonstrando um crescimento maior no tempo em relação a quantidade de operações efetuadas.

Analisando o comportamento dos outros algoritmos, podemos observar que em aplicações onde o padrão é pequeno e o texto é grande, o BMH deve ser preferido devido à sua menor inclinação. Shift-And é um boa alternativa, mas apresenta um crescimento do tempo de execução ligeiramente mais rápido com o aumento do número de comparações.

6 Conclusão

Primeiramente, é fundamental destacar a importância de uma análise cuidadosa das especificações iniciais do trabalho.

Concluímos que, pelas análises durante os testes, o algoritmo BMH se destacou devido ao seu tempo de execução e número de comparações extremamente baixo em relação aos outros algoritmos. Isso se deve ao seu funcionamento com uma tabela de processamento, que dita como a busca deve se locomover pelo texto de maneira mais eficiente, minimizando o número de comparações.

7 Referências

[CORMEN et al., 2012] CORMEN, T. H., Leiserson, C., Rivest, R., and Stein, C (2012). Algoritmos: teoria e prática. LTC.