# *PMT*

***IF767*** *(Processamento de Cadeias de Caracteres)*

Vinícius Carneiro Pereira Souza (vcps)

# 1. Implementação

A ferramenta foi escrita na linguagem de programação C++ (versão 11), contando com scripts de instalação e de execução de testes escritos em Bash e sendo voltada à plataforma Linux.

Cada algoritmo é declarado e implementado em seus respectivos arquivos *.h* e *.cpp*, fazendo uso de funções especificadas e implementadas por um módulo *utils*.

O programa principal está contido no arquivo *main.cpp*, o qual importa todos os demais módulos e permite a utilização de argumentos de linha de comando que alteram o funcionamento do programa. Tais argumentos são obtidos a partir do uso das funções contidas na biblioteca optionparser, a única biblioteca externa utilizada na ferramenta.

Os seguintes algoritmos foram implementados:

- *Casamento Exato:*
    - *Força Bruta* (para fazer comparações de performance)
    - *Knuth-Morris-Pratt*
    - *Shift Or*
- *Casamento Aproximado:*
    - *Sellers (usando distância de Levenshtein)*
    - *Wu-Manber*

O programa escolhe que algoritmo utilizar por meio do argumento de linha de comando *edit*: se *edit* for fornecido com um valor maior que 0, então é escolhido o algoritmo *Wu Mamber*, caso contrário, o escolhido é o algoritmo *KMP*.
Se o usuário desejar é possível fornecer o algoritmo a ser utilizado pela ferramenta através do argumento *--algorithm/-a*, o que é descrito em mais detalhes pela mensagem de ajuda da ferramenta, acessada via *pmt --help*.

## Detalhes e Falhas

Uma estrutura de dados específica denominada *PatternOccurrence* também foi implementada, a qual guarda os valores relativos à linha e à coluna do texto contido em um arquivo onde um casamento foi detectado.

## 2. Testes e Resultados

Os dados utilizados durante o experimento foram gerados por um script próprio - data_gen.py -, enviado junto com a ferramenta. O script aceita os seguintes argumentos:

- ***alphabet_size***: O tamanho do alfabeto a ser utilizado. Pode ter valores *sm* (para o uso do alfabeto A, C, G, T) ou *lg* (o alfabeto de todas as letras minúsculas e maiúsculas).
- ***pattern_min*** e ***pattern_max***: Delimitam o tamanho dos padrões a serem gerados.
- ***edit_distance***: Determina o tamanho da distância de edição usada.
- ***dataset_name***: O nome do conjunto de dados sendo criado. Se a opção 'auto' for fornecida, o nome é criado a partir dos outros argumentos fornecidos.

O script gera um arquivo de padrões com um padrão para cada tamanho, de *pattern_min* até *pattern_max*, e então gera um arquivo de texto de 10000 linhas, cada qual com 10000 caracteres - em média cada arquivo tinha um tamanho de 100MB. Em cada linha é colocada uma quantidade aleatória de padrões, e tais padrões também são escolhidos aleatoriamente, em posições aleatórias da linha. O resto dos caracteres da linha também são determinados aleatoriamente. Se uma distância de edição *k* - maior que 0 - for fornecida, o script vai trocar *k* posições de cada padrão a ser inserido por outros caracteres do alfabeto.

O ambiente de testes consistiu numa máquina pessoal com as seguintes configurações:

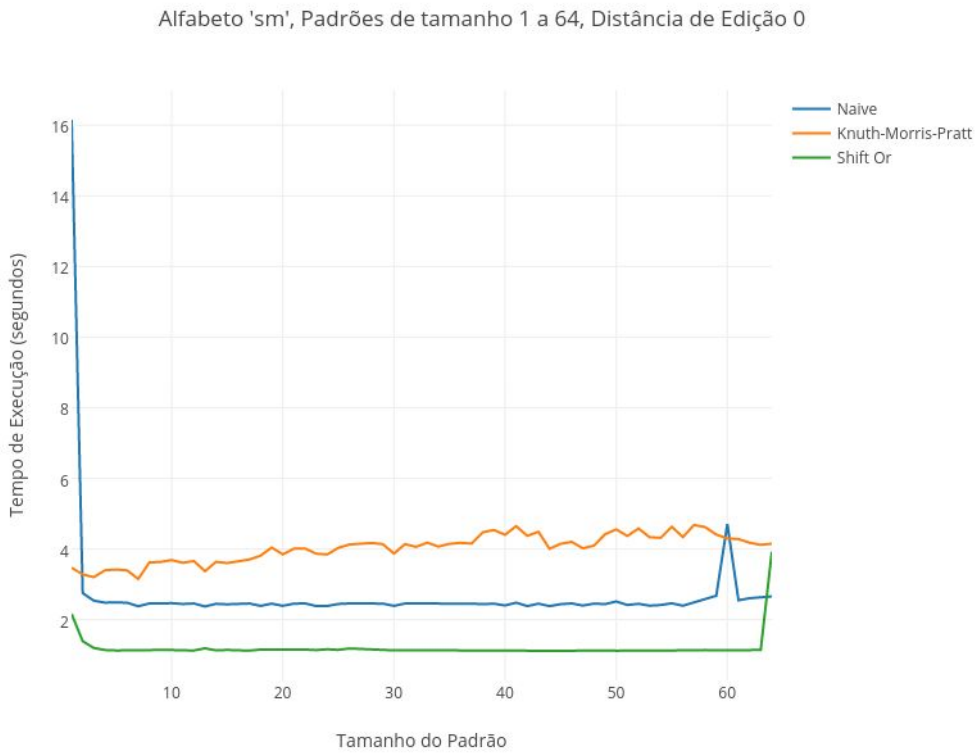
- Processador Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz
- Dois pentes de memória RAM DDR3, totalizando 6GB

O experimento consistiu em gerar 8 conjuntos de dados, variando-se *tamanho do alfabeto*, *distância de edição* e *tamanho dos padrões*:

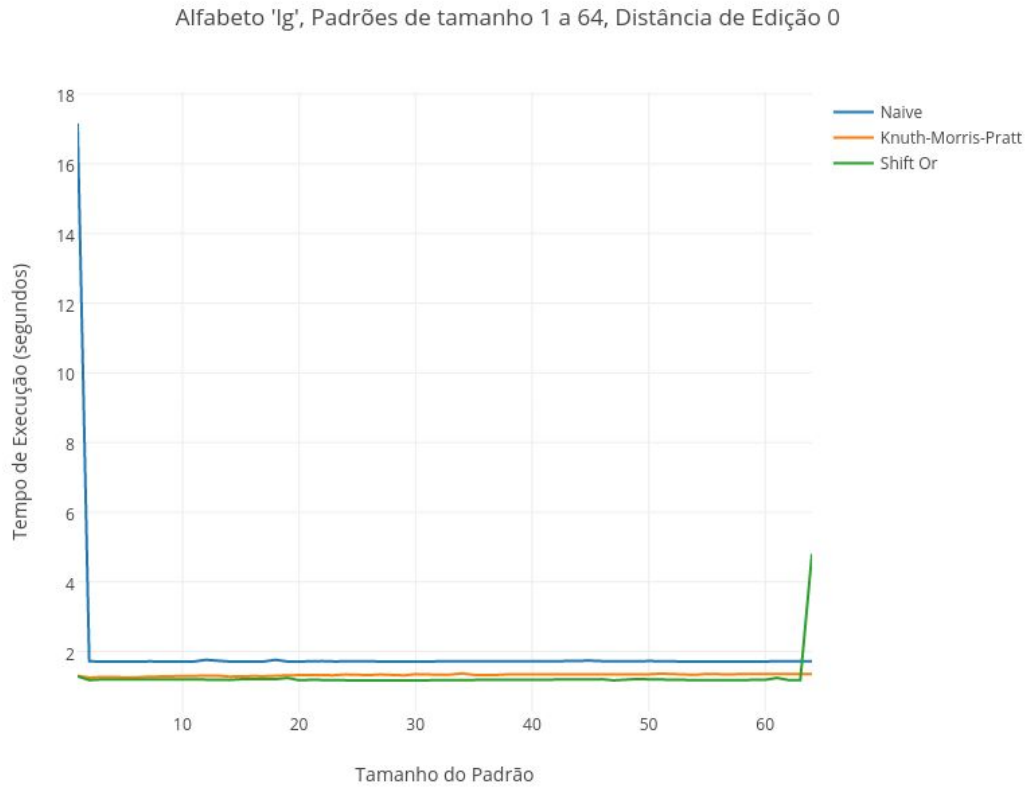
| Nome | Alfabeto | Tamanho dos Padrões | Distância de Edição |
|---|---|---|---|
| AsmP1_64_E0 | Nucleotídeos do DNA | 1 a 64 | 0 |
| AsmP1_10_E1 | Nucleotídeos do DNA | 1 a 10 | 1 |
| AsmP2_10_E2 | Nucleotídeos do DNA | 2 a 10 | 2 |
| AsmP3_10_E3 | Nucleotídeos do DNA | 3 a 10 | 3 |
| AlgP1_64_E0 | Letras maiúsculas e minúsculas | 1 a 64 | 0 |
| AlgP1_10_E1 | Letras maiúsculas e minúsculas | 1 a 10 | 1 |
| AlgP2_10_E2 | Letras maiúsculas e minúsculas | 2 a 10 | 2 |
| AlgP3_10_E3 | Letras maiúsculas e minúsculas | 3 a 10 | 3 |

Os algoritmos de casamento exato (força bruta, KMP e shit or) foram executados no conjunto de dados com distância de edição 0 e os algoritmos de casamento aproximado (wu-manber e sellers) foram executados no conjunto de dados com distância de edição maior que 0. Tais execuções foram feitas por meio do script run_tests.sh, e os tempos de execução capturados pela própria ferramenta usando a opção *--report-runtime*.

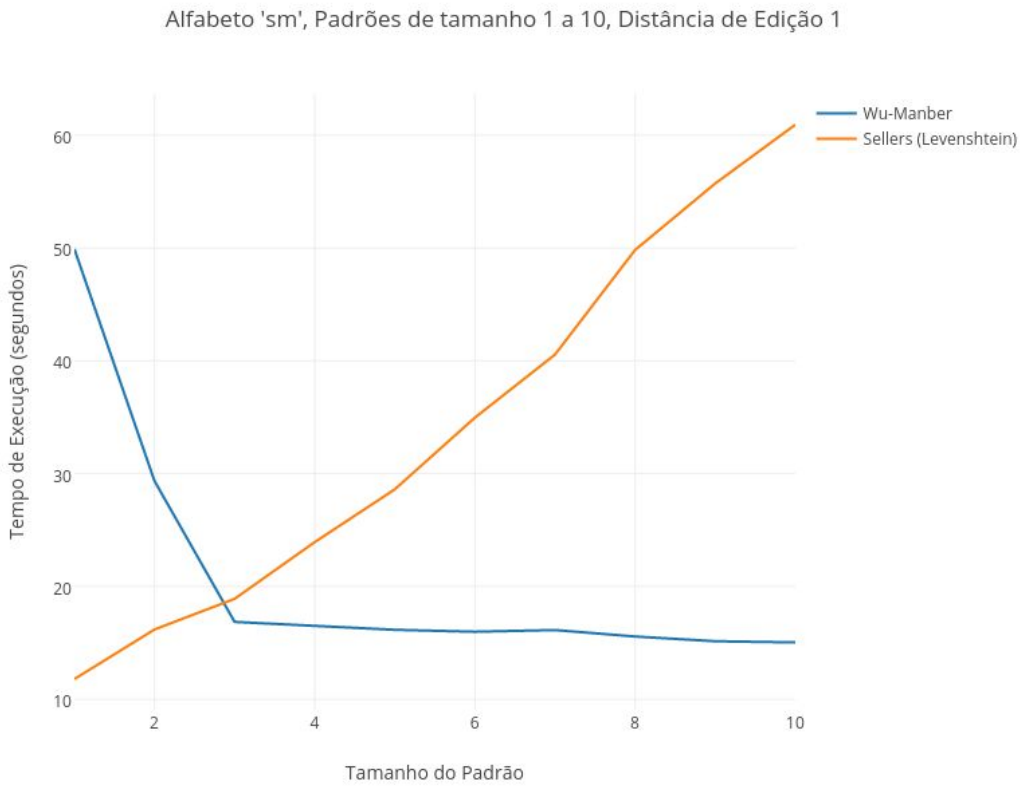
Abaixo seguem os dados obtidos:




Aqui é possível constatar que o algoritmo *KMP* realmente não tem uma performance adequada quando o alfabeto sendo utilizado é pequeno: na maior parte das vezes seu desempenho é pior do que o da abordagem de força bruta, dado que seu pré-processamento não é de grande auxílio para evitar comparações fazendo que o tempo gasto nessa tarefa não seja traduzido numa melhoria de seu desempenho. O algoritmo *shit or*, no entanto, demonstra um desempenho superior a ambas as abordagens, principalmente pelo fato do alfabeto utilizado ser pequeno. É também possível perceber que no padrão de tamanho 64 o súbito aumento do tempo de execução do algoritmo *Shift Or* se deve ao fato do dado estar corrompido: o algoritmo usa um inteiro de 64 bits em sua implementação, e durante a execução ocorreu um overflow por conta do tamanho do padrão.



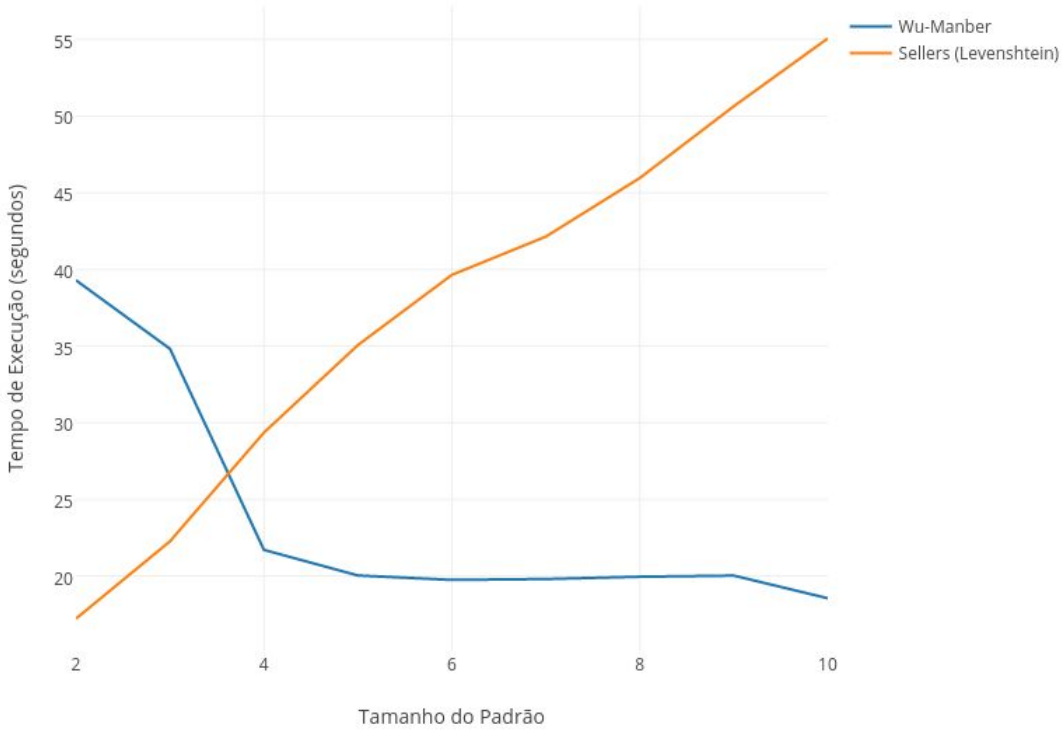

Com o aumento do alfabeto sendo utilizado, é possível perceber uma aproximação entre o desempenho dos algoritmos *KMP* e *Shift Or*. O mesmo problema de overflow pode ser observado no padrão de tamanho 64 executado pelo *Shift Or*.



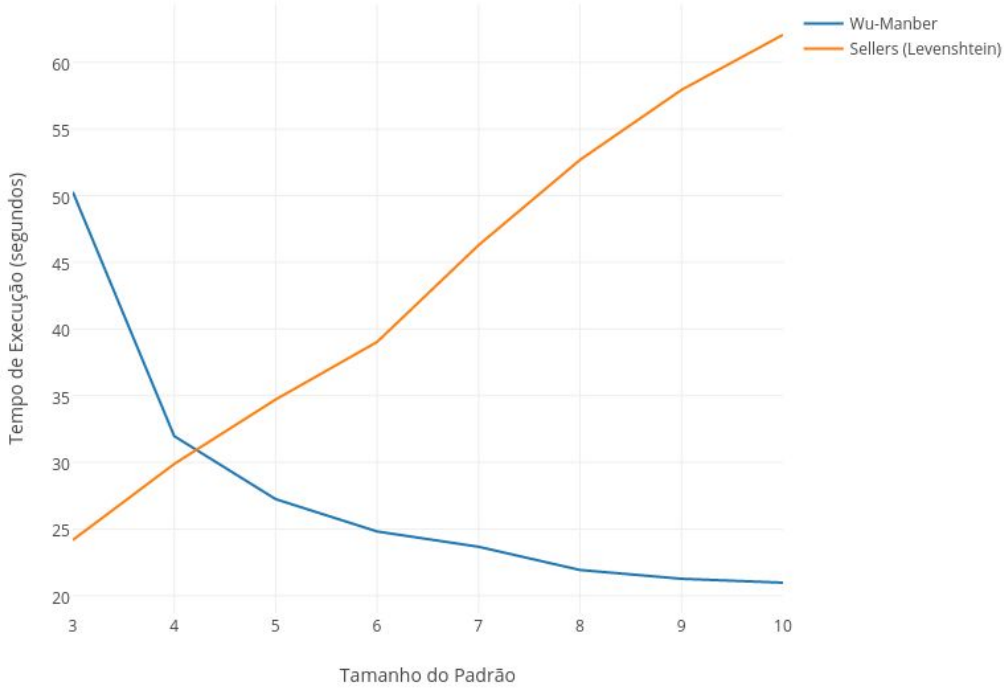

O desempenho do algoritmo *Sellers* (usando distância de *Levenshtein*) claramente não escala bem com o aumento do padrão, enquanto o algoritmo Wu-Manber melhora o seu desempenho pouco a pouco, à medida que o tamanho do padrão aumenta, até ultrapassar a performance do algoritmo *Sellers*.

A piora no desempenho do algoritmo Sellers se dá principalmente pelo fato de que cada novo caractere no padrão significa uma nova linha na tabela da programação dinâmica, a qual tem um número de colunas igual ao número de caracteres do texto de entrada.

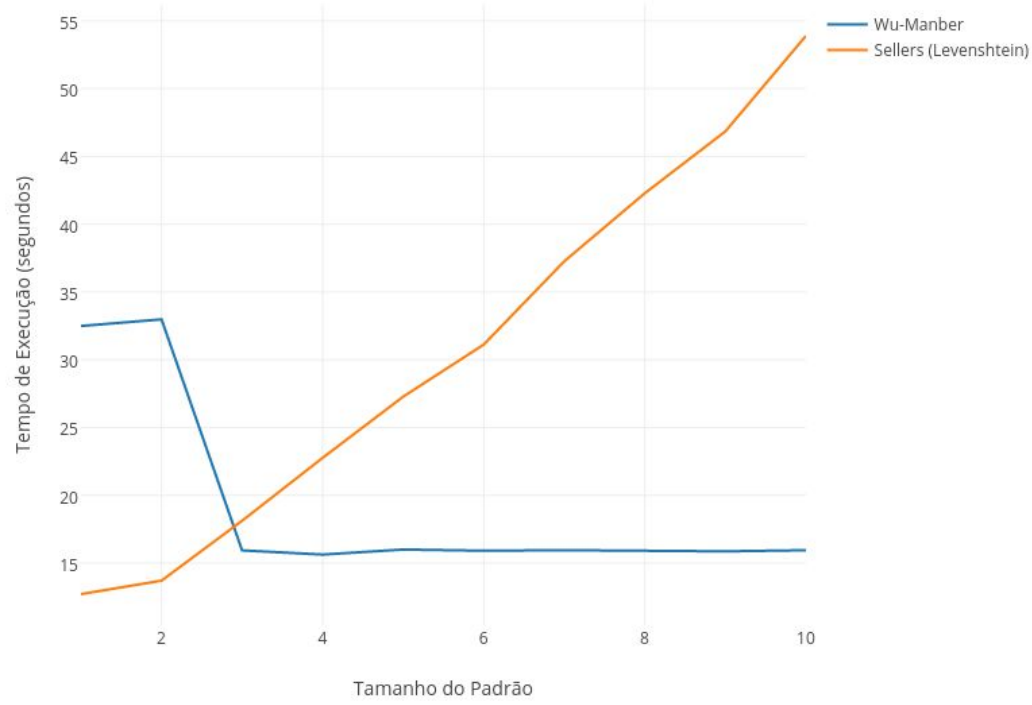
Alfabeto 'sm', Padrões de tamanho 2 a 10, Distância de Edição 2



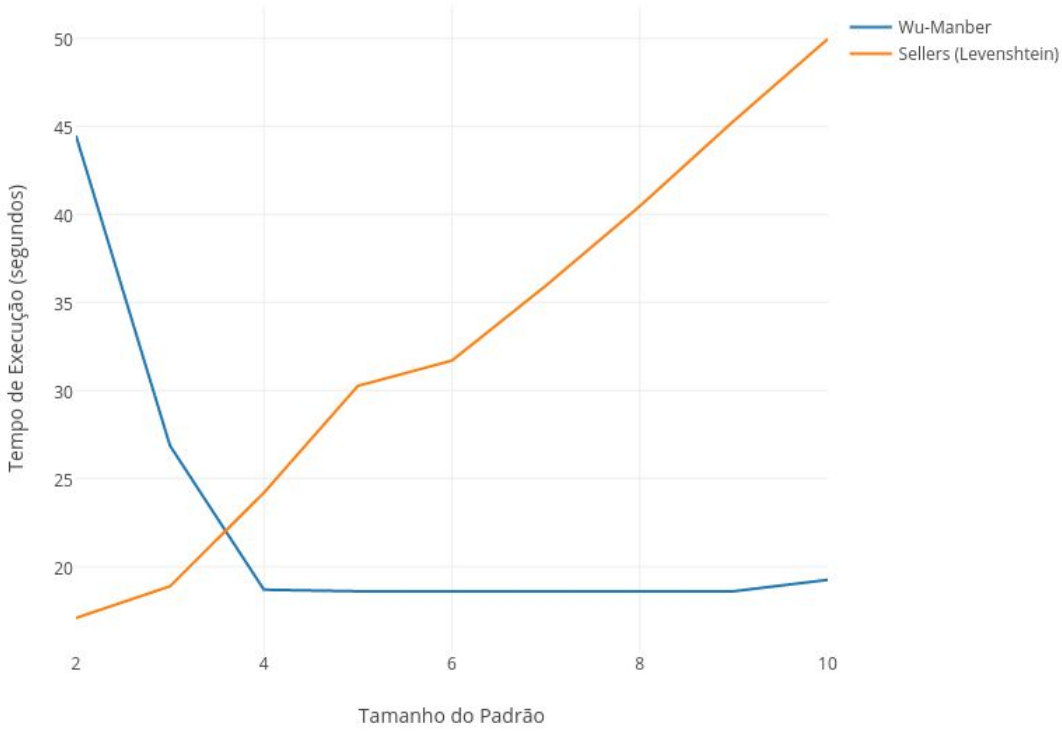

Alfabeto 'sm', Padrões de tamanho 3 a 10, Distância de Edição 3

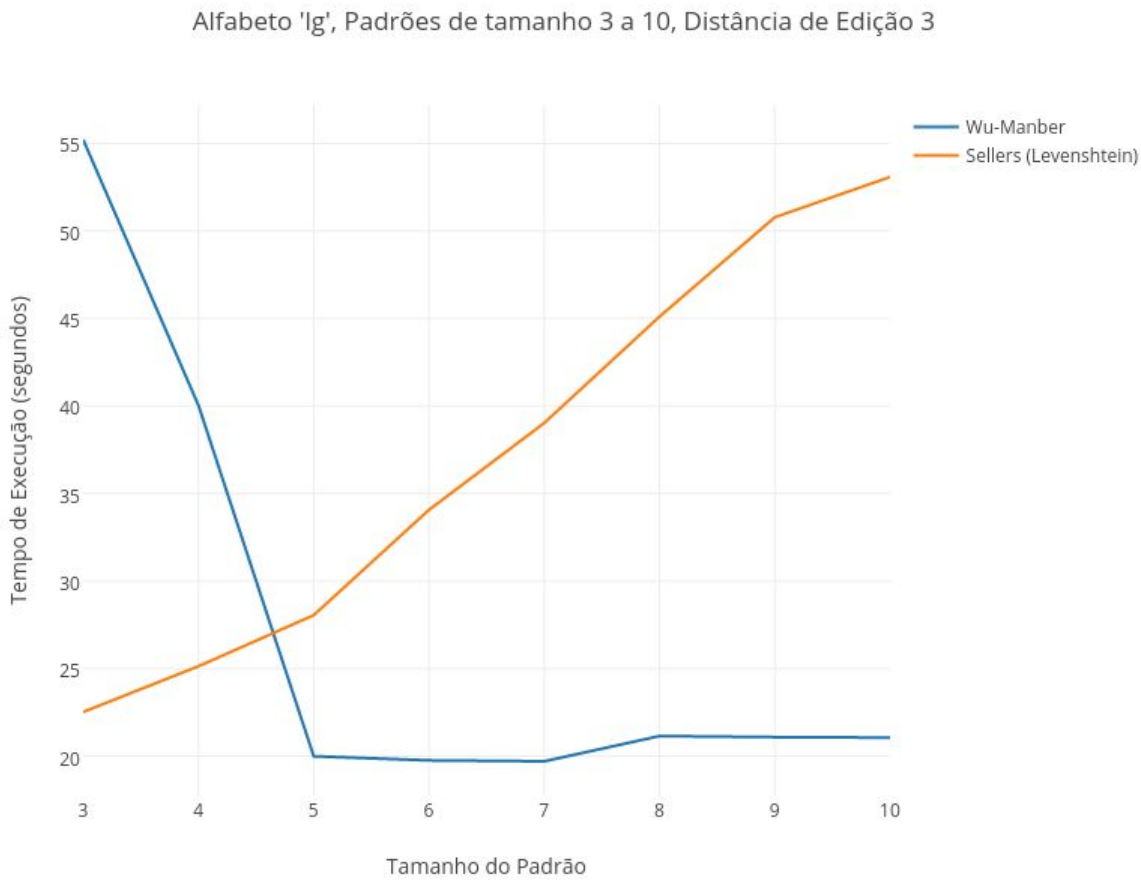


Alfabeto 'lg', Padrões de tamanho 1 a 10, Distância de Edição 1




Alfabeto 'lg', Padrões de tamanho 2 a 10, Distância de Edição 2






De acordo com as imagens acima, a relação entre os algoritmos *Sellers* e *Wu-Manber* permaneceu semelhante mesmo com as alterações de distância de edição e de tamanho do alfabeto. Conclui-se que a alteração no tamanho do padrão é significativa o suficiente para ocultar as demais alterações, o que poderá ser verificado num experimento mais extensivo.

## Conclusão

Foi interessante constatar através da implementação e posterior experimentação a veracidade das afirmações feitas acerca dos algoritmos implementados em aula. Com experimentos mais extensivos e rigorosos será possível perceber outras relações mais sutis entre os algoritmos, e com a implementação de mais algoritmos vistos em sala a comparação será ainda mais rica.