Algoritmos e Estrutura de Dados III (2022-1)

Trabalho Prático 2

Luís Arthur de Assis Moraes

Luiz Filipe Magalhães Freitas

1 Introdução

Desde a criação da internet e a acessibilidade da mesma, nunca se foi visto um volume tão grande de dados como atualmente, sendo que em 2016, 90% dos dados existentes haviam sido criados em cerca de apenas 2 anos (IBM). Sendo assim, faz se necessária a criação de métodos cada vez mais eficientes para o tratamento dos dados, com eficiência e qualidade de acordo com os conceitos dos 5V's do BigData. Sabe-se que grande parte destes dados estão nas mais diversas formas, desde imagens, vídeos e outras estruturas de dados, e em meio a essas variadas formas, estão os textos, e para o tratamento destes verifica-se a necessidade de bons algoritmos de processamento de cadeias de caracteres, de forma a filtrar os mais diversos padrões e informações que podem ser retiradas dos mesmos.

Para tal, iremos apresentar um algoritmo eficiente, que vise mostrar alguns dos métodos já existes de processamento de cadeias de caracteres, na detecção de padrões em textos. O trabalho a seguir tem como objetivo demonstrar alguns dos fundamentos por trás deste algoritmo, e fazer algumas comparações entre os resultados apresentados pelo mesmo.

2 Problema Proposto

O trabalho aqui proposto, tem como motivação o entendimento posto em prática sobre o processo de reconhecimento de padrões em uma certa entrada de texto. O problema a ser discutido consiste em tratar um arranjo de notas musicais e buscar semelhanças intrínsecas com um outro arranjo de notas, no caso algo que seria descrito como um plágio, tal busca se conceitua em alguns aspectos e características das notas musicais. A música no geral possui 7 notas principais, e suas variações, no caso sustenidos e bemóis, destacados por um '#' ou 'b', essas notas podem ser postas na forma de algumas letras do alfabeto (A, B, C, D, E, F e G), essas notas possuem frequências distintas até certo ponto, contudo algumas possuem frequências semelhantes, e podem ser tocadas de formas diferentes, desde que elas

tenham essa determinada ressonância, que pode ser dada por potências de 2, em relação a suas distâncias na escala musical, que podem ser visualizadas na figura 1.

Notas	A-B	B-C	C-D	D-E	E-F	F-G	G-A
Número de meios-tons	2	1	2	2	1	2	2

(figura 1)

Portanto notas como B, C# e G podem ser usadas para tocar a nota A, logo uma mesma música pode ser tocada de várias formas diferentes. No presente trabalho, serão explorados alguns métodos para verificar essas semelhanças como o força bruta, BMH, ShiftAndExato e o KMP.

3 Solução

A solução então aqui proposta, fora o uso de 4 métodos distintos para a captação dos padrões, sendo elas os algoritmos de forçam bruta, o KMP (Knuth-Morris-Pratt), o BMH (Boyer–Moore–Horspool) e o ShiftAndExato, estes serão explicados com mais profundidade nas seções a seguir, com exceção do KMP, que será descrito na secção 7.1.

Primeiramente, o algoritmo recebe alguns parâmetros, sendo eles o arquivo de formato .txt a ser lido e um número referente ao método a ser utilizado para o processamento, o arquivo terá suas entradas lidas e convertidas para números, referentes a suas posições na escala principal, e então esses valores são registrados em duas STRUCTs, uma para o arranjo principal e uma para o arranjo que contém o padrão, essa STRUCT é chamada de STRUCT *sequencia*, e essa possui dois tipos de variáveis em seu escopo, do tipo int, sendo uma o ponteiro nota, responsável por reter um array contendo as notas, e a variável tamanho, para receber o tamanho do respectivo arranjo, o processo de registro é realizado até que todas as entradas tenham sido lidas.

A partir daí esses arranjos são analisados por um dos métodos existentes, que buscam e retornam a posição da primeira ocorrência do padrão no arranjo principal, caso contrário o mesmo retorna um valor indicando que não houve ocorrência, esses resultados são registrados em um documento de saída .txt. ao final da leitura completa das entradas os dados dos arranjos são liberados da memória.

3.1 Força Bruta

O algoritmo do tipo força bruta, realiza uma busca da forma mais simples possível, sem filtros ou métodos de encurtamento do processo de busca do padrão. O método consiste em buscar na primeiro nota do arranjo principal, e então a compara com o primeira nota do arranjo padrão, caso ocorra o casamento das notas, pela frequência semelhante, então ele realiza a comparação das próximas notas dos arranjos até que o final do arranjo padrão seja alcançado, e o mesmo tenha encontrado semelhança com a nota do arranjo principal, contudo no caso de uma dissemelhança o arranjo principal volta para primeira nota comparada da vez e avança para próxima, e o processo se repete, até o fim do arranjo principal seja alcançado, retornando que o padrão não ocorre no mesmo, ou até que o padrão seja encontrado, então ele retorna a posição onde o próprio ocorre.

3.2 BMH

O algoritmo de BMH possui dois diferenciais, o primeiro é que as comparações são feitas de trás para frente, a segunda é que o pré-processamento cria uma tabela com o tamanho do alfabeto e armazena o tamanho dos deslocamentos que devem ser realizados ao decorrer de colisões. Para realizar o pré-processamento a função *pre_proces* é chamada, sendo passado apenas o endereço de uma tabela e o arranjo das notas do padrão como parâmetros. Realiza então a primeira parte que consiste em preencher toda a tabela com o tamanho do arranjo para que não haja posições inválidas. Logo em seguida, preenche as devidas posições na tabela apenas com base no padrão e no alfabeto seguindo o seguinte cálculo:

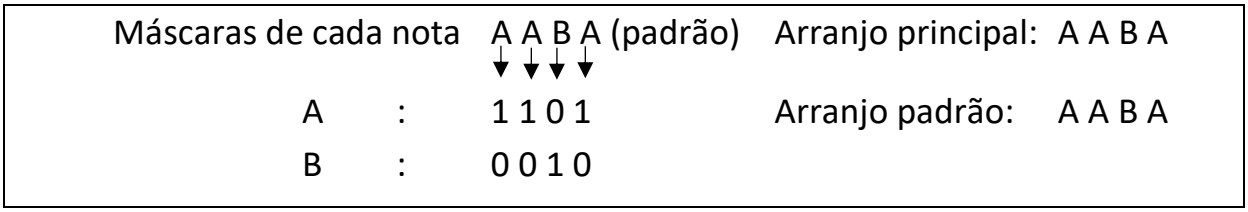
$$\text{tabela}[x] = \min \{ j \text{ tal que } j = m \mid (1 \leq j < m \ \& \ P[m - j] = x) \}$$

Após o pré-processamento, a primeira iteração compara a última posição do padrão com a posição (tamanho do padrão - 1) do texto, e enquanto houver a semelhança, o ponteiro para as posições a serem comparadas “andam” para esquerda até passar por todo o padrão e obtivermos uma ocorrência do padrão no texto ou acontecer uma colisão. Neste último, o loop é quebrado, fazendo com que o ponteiro para a posição do padrão seja resetado para a última posição e o ponteiro para o texto é recalculado baseado na tabela de deslocamento que fora pré-processada. Isto tudo dentro de um loop que limita as comparações para que o ponteiro para o texto não aponte para além do mesmo, gerando assim problemas.

3.3 ShiftAndExato

O algoritmo do tipo ShiftAndExato, diferente dos demais citados, possui outros métodos para facilitação do tratamento do arranjo, de forma a realizar menos processos de comparação, para tal o mesmo utiliza de mascaras para filtrar os dados. As máscaras usadas pelo algoritmo foram implementadas a partir de uma STRUCT *mascaras*, que é constituída em seu escopo por duas variáveis do tipo int, sendo uma o ponteiro *potencias*, que a de ser alocado para um array, e a variável *masc*, onde vai ficar guardado o dado da máscara da nota.

O procedimento para criação das máscaras das notas do arranjo principal é baseado no arranjo padrão, logo após a comparação da nota principal com cada nota do padrão é atribuído um valor a máscara, que no binário terá um bit 1 para correspondência e um bit 0 para notas distintas. Exemplo de um texto comum:



(Figura 2)

Ademais, durante o processo de formação de cada máscara, é alocado um array na variável potencias da STRUCT onde é guardada o valor das potências das distâncias entre a nota e as notas do padrão.

Após isso a busca é iniciada no arranjo principal, o sistema de busca do ShiftAndExato é atrelado a operadores bit a bit. A busca é linear através do arranjo principal, e faz uso de dois vetores, R e R', além das máscaras. Dessa forma o vetor R sempre terá 1 em seu bit mais à esquerda, logo o método se inicia na primeira nota do arranjo 1, fazendo uma comparação entre a máscara da respectiva nota com o vetor R, no caso da dupla ocorrência do bit 1 em posições semelhantes entre a máscara e R, é retornado 1, caso não é retornado 0, nas posições das ocorrências, o resultado é guardado em R'. Em seguida R recebe R', e os bits de valor 1 são deslocados um bit a esquerda, após isso a busca avança para próxima nota do arranjo principal e o processo é repetido em cima do R anteriormente criado, e o processo se repete até que o vetor R' tenha seu primeiro bit preenchido por um bit 1, onde ocorre o casamento.

(Figura 3)

Acima pode ser visto um exemplo do funcionamento do ShiftAndExato, até a ocorrência do mach (casamento), as máscaras usadas no exemplo são baseadas na Figura 2.

Mas além disso também ocorre um subprocesso no caso do mach, no qual é realizada uma verificação para saber se o arranjo encontrado tem distâncias de frequências semelhantes entre si para ocorrer o padrão de fato, nesse processo é analisado o vetor potências das máscaras envolvidas no mach, nas posições em relação ao mach de cada um para o final ocorrido, caso esteja correto é retornada a posição do padrão no vetor, caso não é retornada a negação, após isso as máscaras são liberadas da memória.

4 Análise de Complexidade

Para realização da análise de complexidade foram utilizados 2 parâmetros, sendo *n* o número de elementos do texto principal (arranjo principal) e *m* como o número de elementos do padrão (arranjo padrão).

leitor_de_arquivo: O(n) – Essa função realiza a alocação de dois ponteiros para array, no caso o arranjo principal e o padrão, em O(1), e em seguida passa por cada elemento das entradas, as convertendo para números, em um processo O(1), e as guarda em seu array especifico, esse processo é realizado para o texto em O(n) e para o padrão em O(m).

$$\max(O(1) + O(n)*O(1) + O(m)*O(1)) = O(n)$$

limpeza: O(1) – Essa função libera um ponteiro para array, em tempo O(1).

gravador: O(1) – Essa função realiza a gravação dos resultados de uma entrada em um documento de saída em O(1).

comparador: O(1) – Essa função compara dois elementos dos arranjos musicais, no caso as notas, retornando a distância entre eles, esse processo é realizado em O(1).

força_bruta : O(n*m) – Passa por cada um dos *n* elementos do arranjo principal, comparando-os com seus m – 1 posteriores com o arranjo padrão em O(m), até que todo o vetor seja percorrido ou o padrão seja encontrado, esse processo ocorre em O(n*m)

cria_mascaras & cria_dfas: O(n*m) – Estas funções passam por cada um dos *n* elementos do arranjo principal, comparando cada um com os elementos *m* do arranjo padrão, criando suas respectivas mascaras/dfas, esse processo ocorre em O(n*m).

shift_and_exato & KMP: O(n*m) – Essas funções percorrem todos os *n* elementos do arranjo principal, comparando os com suas respectivas mascaras/dfas, esse processo ocorre em O(n), no caso do casamento dos elementos, ocorre uma verificação nos *m* elementos em que ocorreu o casamento, verificando a paridade entre suas distâncias, esse processo ocorre em O(m), no melhor caso esse processo ocorre somente no final, levando O(n) o processo no todo, no pior caso, o casamento ocorre várias vezes de forma errônea, levando O(n*m).

limpeza_de_mascaras & limpeza_de_dfas: O(n) – Essas funções passam pelos *n* elementos do arranjo principal liberando suas máscaras/dfas, isso ocorre em O(n).

main: O(m*n) – Obs.: Pra o cálculo aqui feito foi padronizado o uso de uma única entrada e um padrão com uma variação mínima com exceção da última nota, o mesmo padrão se repete por todo o arranjo principal, com tudo a última nota só existe no final. Exemplo de arranjo padrão: “A A A B”. Exemplo de arranjo principal: “A A A A A A B”. Sabe-se que A e B tem equivalência, com tudo devido a distância do casamento fica inválida nesse caso, tornando-o o pior caso. A complexidade estabelecida aqui é referente aos métodos KMP e ShiftAndExato, já que ambos possuem 2 funções extras semelhantes em seus loops, enquanto o Força Bruta e o BMH não. Contudo isso não altera suas complexidades finais, que são iguais aos dos anteriores.

Essa função deve receber os parâmetros iniciais e constatar se estes estão dentro do esperado, esse processo ocorre em O(1), após isso a função realiza o processo especificado na entrada, durante o mesmo ocorre um loop até que o número de entradas tenha sido processada, nesse loop os arranjos são lidos a partir da função ***leitor_de_arquivo*** em O(n), após isso as máscaras/dfas são criadas em O(n*m), em seguida o método usado realiza sua busca em O(n*m), então os resultados são gravados em O(1), e os arranjos são liberados em

O(1) cada, e então as máscaras/dfas são liberadas da memória também, em tempo O(n), após o fechamento do loop a função é encerrada.

$$\max(O(1) + O(n) + O(n*m) + O(n*m) + O(1) + O(1) + O(n)) = O(n*m)$$

pre_proces: O(n) – O pré-processamento é feito baseado apenas no padrão e no alfabeto, o que o torna muito eficiente. Feito em duas etapas, temos que a primeira é um loop do tamanho do alfabeto, sendo nosso alfabeto sempre 12 temos que a primeira etapa é O(1). A segunda etapa é o cálculo dos deslocamentos, sendo em seu pior caso O(n).

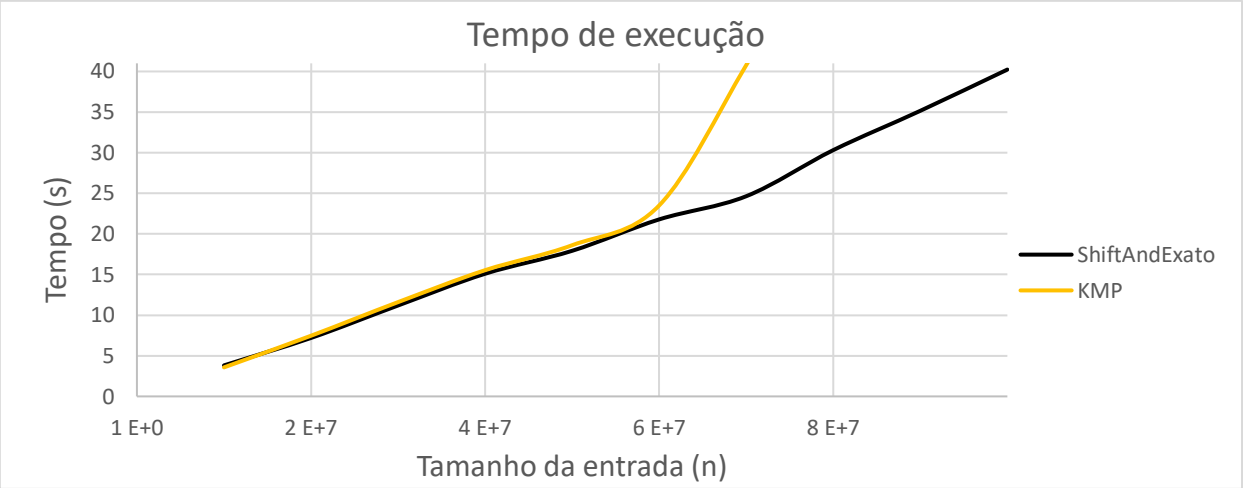
BMH: O(n*m) – realiza a comparação de trás para frente, além de um pré-processamento, definido em ***pre_proces***. Ainda com o pré-processamento, há casos que o algoritmo irá percorrer todo o texto e irá comparar todo o padrão dando colisão apenas na primeira posição do padrão, por isso em seu pior caso trata-se de uma complexidade de O(m*n). Entretanto, devido ao pré-processamento, conseguimos obter que o caso esperado é O(n/m).

5 Análise de Resultados

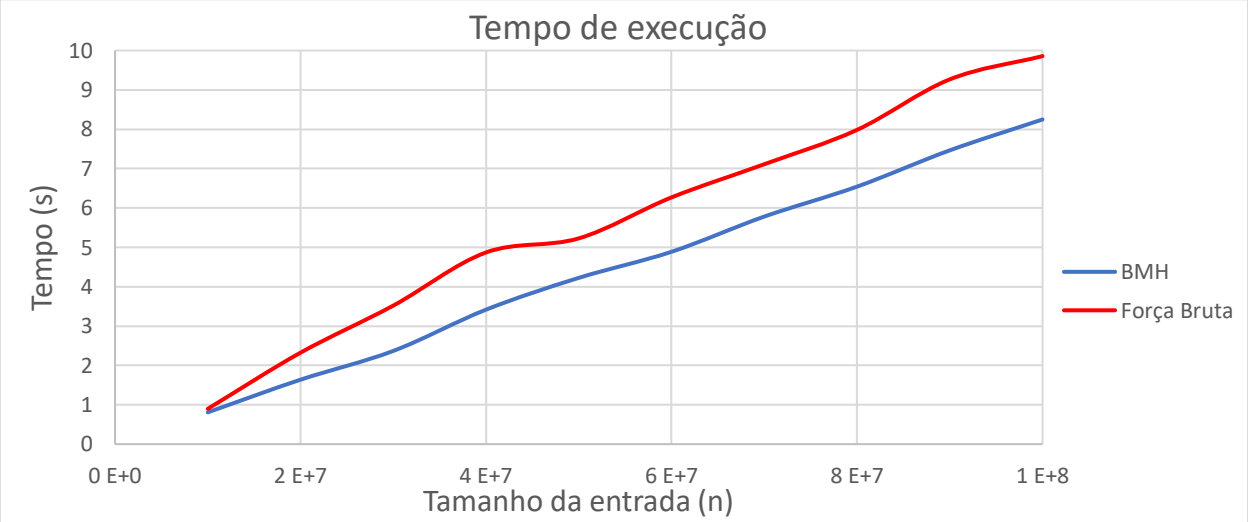
5.1 Análise Quantitativa

Para a análise aqui apresentada serão utilizados os mesmos arquivos de caso teste utilizados anteriormente, variando-se o tamanho do texto, sendo que haja ocorrência sempre nas últimas posições do texto.

Será alterado apenas o tamanho do texto. Com base nos dados derivados dos testes, percebemos que o tempo de execução aumenta conforme o tamanho do texto de forma linear. Este comportamento é esperado, e explica-se relembrando que a ocorrência ocorre somente no final, necessitando de realizar comparações até esta posição.



(Figura 1)



(Figura 2)

A figura 1 e 2 nos traz informações interessantes. Como base temos que os testes foram realizados com textos de tamanho $1 * 10^7$ até $1 * 10^8$ e mantendo o padrão com tamanho constante 5, além do nosso alfabeto ter tamanho 2.

Analisando as figuras 1 e 2 observamos que todos os métodos estão relacionados com seus respectivos cálculos de complexidade teóricos, apresentados anteriormente. Ou seja, como é de se esperar de tais algoritmos, eles portam um comportamento assintótico com tendência linear. A criação de dois gráficos se deu pela necessidade de apresentar duas escalas de tempo diferentes, a figura 1 variando de 0 a 40 segundos, enquanto a figura dois varia de 0 a 10 segundos. Isso ocorre devido à diferença de tempo de execução entre os algoritmos KMP, ShiftAndExato, BMH e força bruta. Vale destacar que o drástico aumento no tempo de execução do KMP e ShiftAndExato em relação aos

demais vem da necessidade de verificar a paridade das distâncias entre as notas de um padrão que foi encontrado quando há ocorrência de um casamento (match).

5.2 Análise Qualitativa

Qualquer um dos métodos já discutidos irá gerar respostas corretas para quaisquer que sejam as entradas. O que difere eles são as complexidades de execução, cada qual com seu "ponto fraco", isto é, para uma dada entrada o algoritmo BMH pode cair no seu pior caso enquanto o ShiftAndExato irá executar como caso médio, ou o caso esperado.

O algoritmo de força bruta possui a implementação mais simples e uma lógica de fácil entendimento, principalmente porque ela é intuitiva. Entretanto o algoritmo força bruta irá executar em seu pior caso quando a entrada tiver o texto com alfabeto pequeno e o padrão ser semelhante a sequencias de tamanhos menores que o tamanho do padrão, fazendo com que o método execute várias comparações várias vezes.

O algoritmo BMH possui uma implementação um pouco mais complexa, e uma lógica de atuação mais refinada, o que dificulta seu entendimento. Ademais, sua parte de pré-processamento garante um resultado mais eficiente em relação à força bruta, como provado anteriormente.

O algoritmo ShiftAndExato possui uma implementação mais complexa em relação ao BMH e o força bruta. É notado um aumento significante de tempo de execução deste método para os anteriores, devido a existência de vários elementos que podem ser considerados semelhantes nessa variação do algoritmo, sendo assim esse aumento no tempo se explica pela necessidade de verificar a paridade sempre que houver ocorrência aparente do padrão no texto, que ocorre no pior caso várias vezes.

O algoritmo KMP é muito semelhante ao ShiftAndExato, tanto na questão da complexidade de implementação que é mais complexo em relação aos dois primeiros, quanto no tempo de execução, que possui um valor elevado também devido a necessidade de verificar a paridade como no anterior.

6 Conclusão

O algoritmo desenvolvido nesse trabalho apresenta bons resultados, com uma boa qualidade de tempo como pode ser visto nos testes. Portanto torna-se interessante debater a respeito dos métodos, e a viabilização deles. Como pode-se notar houveram algumas discrepâncias entre as formas de se processar os arranjos. O método KMP é bom nos casos médios, e sendo até melhor de ser aplicado em outros problemas, contudo devido a existência das múltiplas identidades possíveis para um mesmo termo ele acaba por ter mascaras um tanto problemáticas, sendo isso um problema causado pelas próprias características do problema apresentado, já que em casos de longas cadeias acaba por demandar muitas verificações secundarias, o que por si só demanda muito tempo como pode ser visto, isto também se aplica ao ShiftAndExato, que possui uma implementação e processamento semelhante ao KMP, por esses motivos talvez não seja tão interessante o uso destes para esse problema em específico. Contudo os métodos que até então eram mais simples como o Força Bruta e o BMH, apresentaram uma eficiência notavelmente melhor, isso se dá pelo fato de ambos não necessitarem tanto de um pré-processamento complexo, como pelo fato também destes realizarem comparações únicas, podendo ter a verificação da paridade das distâncias mais facilmente verificadas. Logo pode-se concluir, que mesmo algoritmos que são mais complexos e eficientes, tem seus casos e momentos, e pelo menos a partir dos dados aqui encontrados, esse não é o momento deles, por fim pode-se finalizar que o BMH aqui implementado pode ser uma boa pedida para esse problema no final.

7 Extra

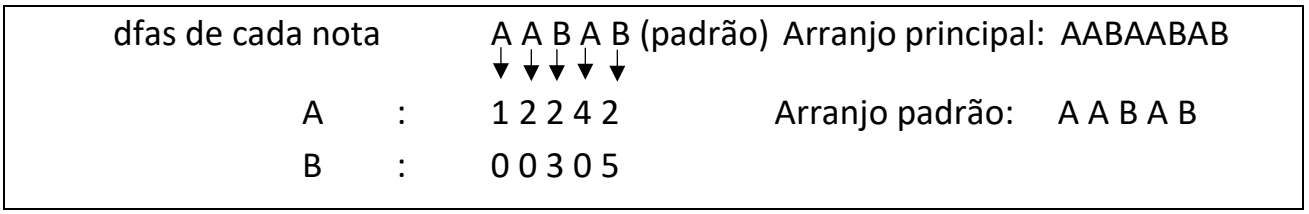
A seguir está descrito o método de casamentos de caracteres extra especificado no trabalho.

7.1 KMP

O algoritmo KMP tem várias semelhanças com um algoritmo aqui já implementado, o ShiftAndExato, possuindo uma implementação mais complexa que os demais, devido a necessidade da criação de mascaras chamadas de dfas (deterministic finite automaton), que o torna mais eficiente nos casos gerais, tais máscaras possuem em si um fator de deslocamento, fazendo com que em casos de repetição de termos no padrão o algoritmo no caso de erro não perca parte do processo já realizado, como é o caso do força bruta que

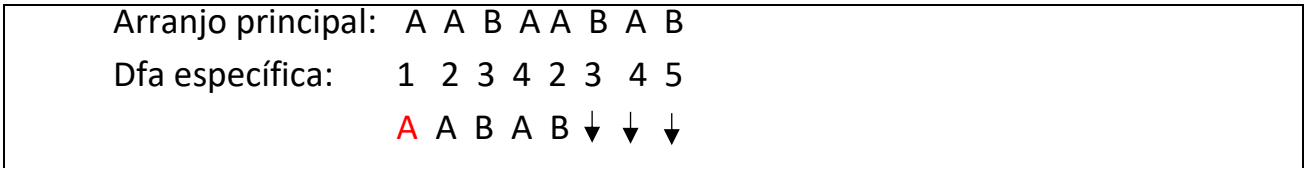
começa do 0 novamente. Para a implementação dessas mascaras foi utilizado uma STRUCT *mascaras*, que é composta de dois ponteiros para int, posteriormente alocados para arrays, sendo eles o ponteiro *masc* e o *potencias*, sendo esses com o mesmo intuito do ShiftAndExato, explicado na secção 3.3, o ponteiro masc, tem função de alocar os respectivos valores de cada termo em relação ao padrão, para possíveis deslocamentos.

O funcionamento desse processo ocorre, em primeiro lugar, em seu pré-processamento, onde as dfa são alocadas, uma para cada elemento do arranjo principal, em seguida cada uma delas tem suas masc alocadas para arrays de tamanho m (tamanho do arranjo padrão), logo após isso é calculado o pivô, que servirá como orientador do deslocamento do padrão durante a comparação com o texto, essa conta é bem focada em resolver problemas de repetição de termos no padrão, como por exemplo o padrão "AABAB" e o arranjo principal "AABAABAB", as dfas estão representadas na Figura 1. Ao chegar no 5º termo do texto ocorre uma divergência com o texto, porém ao invés da comparação reiniciar do primeiro termo do padrão ela segue do 3º termo, já que o cálculo já mostra a ocorrências dos dois primeiros. A alocação do ponteiro potências é idêntica ao do ShiftAndExato, e está explicada na secção 3.3.



(Figura 1)

Após isso ocorre o processamento final, que vale salientar que é linear durante a busca, isso só possível devido as dfas. As verificações ocorrem através das dfas, e o elemento comparado na próxima busca é definido pela posição definida pela mascara anterior, como pode ser observado na Figura 2.



A A B A B (mach)

(Figura 2)

Após o casamento ocorrer como na Figura 2, ocorre uma análise na paridade da distância entre as frequências dos elementos que encaixam com o padrão, da mesma forma que foi descrito no ShiftAndExato. Caso ocorra de fato o padrão é retornada a posição da ocorrência, caso não é retornada a negação. Após isso as dfas são liberadas da memória.

8 Referências

DATA SCIENCE – 90% DOS DADOS FORAM GERADOS NOS ÚLTIMOS 2 ANOS. **jorgeaudy**, 2016. Disponível em: < https://jorgeaudy.com/2016/07/18/data-science-90-dos-dados-foram-gerados-nos-ultimos-2-anos/>. Acesso em: 2, jul. de 2022.

CASAMENTO DE CADEIAS DE CARACTERES. **leanndroluiz.blogspot**, 2010. Disponível em: < http://leanndroluiz.blogspot.com/2010/01/casamento-de-cadeias-de-caracteres.html>. Acesso em: 24, jun. de 2022.

ALGORITMO KMP PARA BUSCA DE SUBSTRING. **ime.usp**, 2018. Disponível em: < https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/kmp.html#:~:text=O%20algoritmo%20KMP%20simula%20o,reconheceu%20ou%20aceitou%20o%20padr%C3%A3o.>. Acesso em: 30, jun. de 2022.