



Introdução, Arquivo Invertido e Força Bruta

<input checked="" type="checkbox"/> Column	<input type="checkbox"/>
Files	https://www.moodlepresencial.ufop.br/pluginfile.php/954476/mod_resource/content/1/bcc203_casamento-cadeias.pdf
Unit/Module	Casamento de cadeias
Video	https://www.youtube.com/watch?v=9H8nIm3thEQ
Week	

Clara Ribas - 20.1.4008

Livia Sousa - 20.1.4029

Introdução

- A ideia do casamento de cadeias é localizar de maneira eficiente todas as ocorrências de uma determinada cadeia de caracteres, chamada de **padrão**, dentro de um texto ou dentro de um grande volume de textos.
 - No **casamento exato de cadeias** deseja-se encontrar exatamente a palavra que foi fornecida na busca.
 - No **casamento aproximado de cadeias** localiza-se não apenas as ocorrências exatas daquela busca desejada, mas também ocorrências aproximadas/similar.
- **Cadeia de caracteres:** Sequência de elementos denominados caracteres. (*String*)
 - Os caracteres a serem buscados são escolhidos de um conjunto denominado **alfabeto**, a definição do alfabeto é de extrema importância. (Algumas estratégias de casamento de cadeia se tornam mais rápidas quanto maior for o tamanho do alfabeto.)
- Exemplos de aplicação:
 - Edição de texto: Todo editor de texto tem a opção de procura de palavras, onde digita-se a cadeia desejada e as ocorrências de tal cadeia dentro do texto são apresentadas.
 - Recuperação de Informação: Área voltada para máquinas de busca, por exemplo o Google, que localizam documentos que tenham a ver com a cadeia buscada dentro de repositórios gigantescos de documentos.
 - Estudo de sequências de DNA em biologia computacional: Localizar um padrão dentro de uma sequência de DNA.
- Formalização do problema:
 - **Texto:** Onde é feita a busca do padrão desejado. Este encontra-se dentro de um arquivo qualquer e deve ser transferido para a memória principal em uma cadeia $T[0...n-1]$ de tamanho n .
 - **Padrão:** É uma cadeia de caracteres $P[0...m-1]$ de tamanho m tal que $m \leq n$.

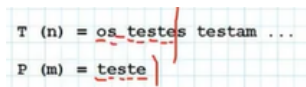
- Os elementos de **P** e **T** são escolhidos dentro do determinado alfabeto Σ de tamanho c .
- Casamento de cadeias:** Dadas as cadeias **P** e **T**, deseja-se saber as ocorrências de **P** em **T**.

Estrutura de Dados

```
#define MAXTAMTEXTO 1000
#define MAXTAMPADRAO 10
#define MAXCHAR 256 // Representa o tamanho do Alfabeto
#define NUMMAXERROS 10 // Quantidade máxima de erros a ser considerada em Casamento Aproximado de Cadeia
typedef char TipoTexto[MAXTAMTEXTO]; // Representa o Texto
typedef char TipoPadrao[MAXTAMPADRAO]; // Representa o Padrão
```

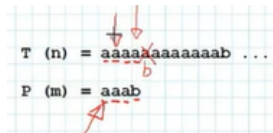
Categorias de Algoritmos

- Padrão e Texto não são pré-processados.
 - Também chamado de **força bruta**, compara-se caractere por caractere até que seja encontrado o Padrão desejado no Texto.



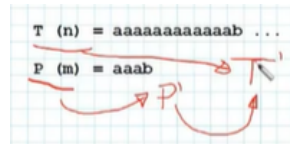
T (n) = os testes testam ...
P (m) = teste

- Quando não encontra-se o Padrão desejado, é preciso recomençar a busca pelo primeiro elemento do Padrão, mas a busca do Texto recomeça a partir do próximo caractere do Texto onde a primeira sequência tentou ser localizada. (*Primeira seta, não a segunda.*)



T (n) = aaaaaaaaaaab ...
P (m) = aaab

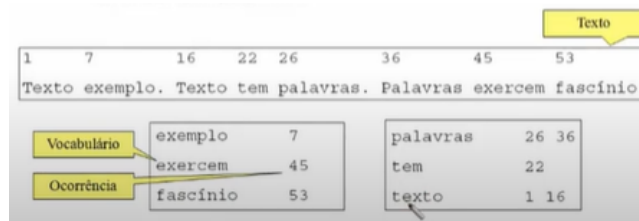
- A imagem acima ilustra o **pior caso** do algoritmo força bruta, porque encontra-se todos os caracteres com exceção do último, logo, para cada caractere do Texto é necessário varrer o padrão por completo, gerando uma complexidade de tempo **$O(m \cdot n)$** . E como não é construída estrutura de dado alguma para auxiliar no processo, tem-se complexidade de espaço **$O(1)$** .
- Padrão é pré-processado.
 - Categoria mais utilizada para edição de texto, no qual o Texto não é pré-processado pois ainda está em construção no momento de edição.
 - É criada uma estrutura de dado vinculada ao Padrão, e este, após ser processado, será buscado no Texto para que não seja necessário fazer a varredura de todos os caracteres do padrão sempre que for feita uma busca.
 - Desta forma, a complexidade de tempo passa a ser **$O(n)$** , pois depende puramente do Texto, e a complexidade de espaço **$O(m+c)$** , pois o padrão é transformado em outra estrutura, logo depende da quantidade de caracteres do Padrão + a do Alfabeto.
- Padrão e Texto são pré-processados.



- Algoritmo muito usado na área de recuperação de informação uma vez que se conhece o repositório de documentos a priori.
- O algoritmo constrói um índice para o texto, como é o caso da Árvore B. A construção desse índice demanda certo tempo, mas é compensado no tempo de pesquisa.
 - Alguns tipos de Índice são: Arquivos Invertidos, Árvores TRIE e PATRICIA, Arranjos de Sufixos.
- O pré-processamento do Texto permite que a busca seja feita de forma ainda mais rápida, com complexidade de tempo $O(\log n)$ e complexidade de espaço $O(n)$.

Arquivo Invertido

- Estrutura composta por duas partes:
 - **Vocabulário:** Conjunto de palavras distintas no texto.
 - **Ocorrência:** Lista de posições onde cada vocabulário aparece dentro do texto.



- A estrutura pode ser implementada de diversas formas.
- Uma vez fornecido o Padrão, este é buscado no Vocabulário. Quando este é localizado dentro do Vocabulário, pela lista de Ocorrência é possível acessar o arquivo na posição exata ou fornecê-la ao usuário.
- Para buscar uma **sequência** de palavras, busca-se cada um dos termos separadamente e, em seguida, é checado na lista de Ocorrência se estão na ordem desejada.
- Quanto mais novos documentos são acrescentados dentro do arquivo invertido, menos o vocabulário cresce, uma vez que será menos frequente o surgimento de novas palavras. Tal crescimento segue a **Lei de Heaps** e tem a complexidade dada por:

$$V = Kn^\beta = O(n^\beta).$$

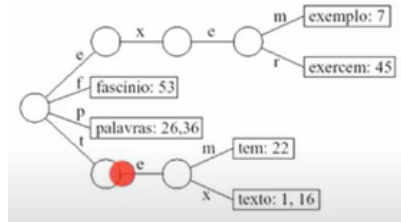
Sendo K entre 10 e 100 e Beta entre 0,4 e 0,6.

- Na prática, o vocabulário cresce com o tamanho do texto, em uma proporção perto de sua **raiz quadrada**.

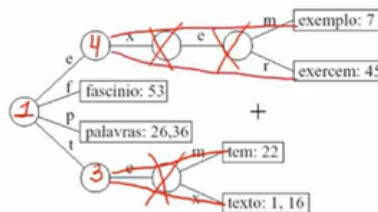
- As Ocorrências ocupam mais espaço que o Vocabulário já que a cada nova palavra inserida no arquivo o número de Ocorrências aumenta.

Exemplo

- Uma das formas de representar o Vocabulário do Arquivo Invertido é usando uma **Árvore de Pesquisa Digital**, como uma **Árvore TRIE**.
 - A Árvore é formada por páginas, **nós internos**, que não possuem **nada dentro**, servindo para **direcionar a pesquisa**, e por **nós externos**, folhas, que **guardam as palavras do Vocabulário e sua respectiva Ocorrência** (Na prática, o link de onde a Ocorrência se encontra).
 - Cada um dos nós internos contém vários apontadores, e cada apontador diz respeito a um caractere do Alfabeto.
 - A árvore é constituída de acordo com a composição dos caracteres. Cria-se um "galho" até a ramificação (caractere) da Árvore em que o Vocabulário difere.



- Desta forma, para realizar a pesquisa de um Padrão basta seguir os apontadores dos caracteres que o compõe.
 - No exemplo ilustrado, o Vocabulário "tem" constitui o **pior caso**, pois o último caractere é o que o diferencia, sendo necessário fazer comparações para todas as letras, ou seja, possui complexidade **O(m)**. **O pior caso é mais provável de acontecer quando tem-se muito Vocabulário.**
- Tem-se também a **Árvore PATRICIA**, que é uma melhora da anterior. Nela, invés de criar um galho grande, no **nó interno** coloca-se a **posição** do caractere que difere as duas palavras.



- Normalmente, utiliza-se com mais **frequência** a **Árvore TRIE** porque tem-se um Vocabulário muito **extenso**, fazendo com que as palavras se **diferenciem nos primeiros caracteres**. Logo, a PATRICIA não tem muito uso em encurtar a Árvore e gera um custo adicional para armazenar o posicionamento no nó interno.

Força Bruta

- O algoritmo de Força Bruta é a "base" para os próximos algoritmos.
- Compara-se os caracteres um a um.

```
void forcaBruta(tipoTexto T, long n, TipoPadrao P, long m){
    long i, j, k;
    // FOR realiza a leitura no texto
    // i representa o início do texto
    // até n-m+1, posição que antecede o número de caracter do padrão que se deseja buscar
    for ( i = 1; i <= (n-m+1); i++){
```

```
k = i; j = 1; // k representa a varredura
while (T[k-1] == P[j-1] && j <= m){ // enquanto os caracteres forem iguais até o j atingir o tamanho do padrão
    j++;
    k++;
}
if(j > m){ // casamento exato
    printf("Casamento na posição %3ld\n", i);
}
}
```



Algoritmos BM, BMH e BMHS

<input checked="" type="checkbox"/> Column	<input type="checkbox"/>
Files	
Unit/Module	Casamento de cadeias
Video	https://www.youtube.com/watch?v=_0832IC3qKA
Week	

- Algoritmos nos quais ocorre a pesquisa do padrão (P) em uma **janela** que desliza ao longo do texto (T), procurando por um **sufixo** da janela (*texto* T) que casa com um **sufixo** de P , mediante comparações realizadas da **direita para a esquerda**.
 - Caso não ocorra desigualdade, uma ocorrência de P em T foi localizada.
 - Caso contrário, o algoritmo calcula o deslocamento que a janela deve ser deslizada para a direita antes que uma nova tentativa de casamento se inicie.

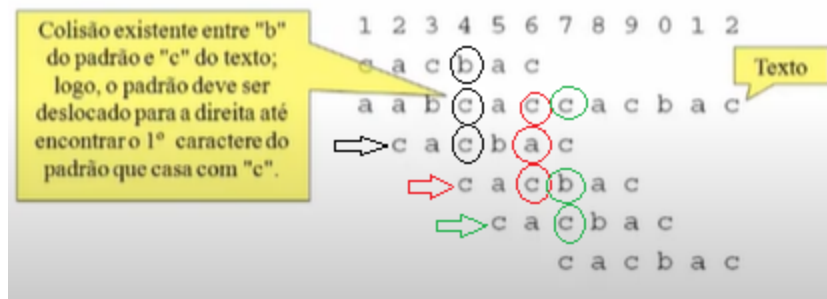
Algoritmo Boyer-Moore (BM)

- Cria-se uma "janela virtual" do tamanho do padrão que caminha para direita a medida que se tenta localizar o padrão dentro da mesma.

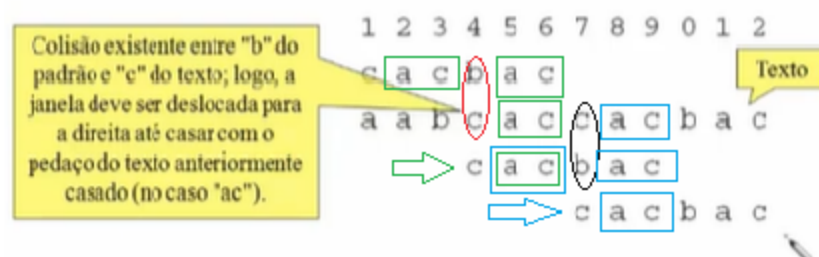
T = a a b c a c c a c b a c
P = c a c b a c

- Compara-se cada caractere do padrão com cada caractere dentro da janela, **sendo as comparações são feitas da direita para a esquerda**. (*Sufixo*)
- Havendo ou não o casamento, o processo continua da mesma forma.

- A grande diferença entre este algoritmo e o de força bruta é que a janela não necessariamente desloca de uma em uma posição. Depende da **heurística** utilizada.
- O BM propõe duas heurísticas para calcular o deslocamento:
 - **Ocorrência:** Pega-se o caractere do Texto que sofreu a colisão (*desigualdade*), este caractere é procurado no Padrão e, se encontrado, a janela é deslizada de forma que os dois caracteres iguais se alinhem.



- Quando o caractere que provocou a colisão não se encontra no padrão, não existe o que alinhar, logo, o restante do padrão é pulado todo. Comum na prática, quando a quantidade de caracteres do Alfabeto é grande.
- Quando tem-se um Alfabeto binário por exemplo (0 e 1), não se tem tanta vantagem em relação ao Força Bruta, assim que, **quanto maior o Alfabeto, mais vantajoso é o BM.**
- **Casamento:** Nessa heurística pega-se a parte do Texto que já casou com o Padrão, e tais caracteres são buscados dentro do Padrão a partir do ponto de colisão e, caso encontrados, são alinhados.



- Quando o caractere que provocou a colisão não se encontra no padrão, não existe o que buscar no padrão. Logo, cai-se no **pior caso** e o deslocamento acontece de um em um caractere.
- De forma geral, a **heurística casamento** é melhor para **Alfabetos maiores**, já a **ocorrência** para **Alfabetos menores**.

Algoritmo Boyer-Moore-Horspool (BMH)

- **Melhoria** do algoritmo BM quanto ao **deslocamento da janela**, o que fez com que o algoritmo apresentasse melhores resultados em termos de eficiência.
- O funcionamento do método se mantém, a mudança foi ao método utilizado para deslocar a janela, em que, ao invés das heurísticas, foi proposta uma **tabela de deslocamento**.
- Na tabela, para cada caractere do Alfabeto é estabelecido um **valor numérico**, e este valor indica o **deslocamento a ser utilizado no momento que o caractere em questão é o caractere do Texto alinhado com o último caractere do Padrão**.
- Havendo ou não casamento, pega-se o último caractere do Texto que está alinhado com o último caractere do Padrão e verifica-se o valor numérico associado a ele na tabela, tal valor será o tamanho do deslocamento.
- Para definir a tabela de deslocamentos tem-se:
 - O **valor inicial** do deslocamento para todos os caracteres do texto é igual a **m** (*tamanho do padrão*).
 - Em seguida, o valor dos **m-1** primeiros caracteres são calculados seguindo a fórmula:

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

O valor associado a x (um dos caracteres do Padrão) representa o menor valor de j tal que j = m, ou $1 \leq j < m$ e de tal forma que o caractere usado para o deslocamento esteja na posição m-j. Ou de forma informal, qual a menor distância de tal caractere em relação ao último caractere do Padrão.

T = a a b c a c c a c b a c
P = c a c b a c

Tabela de deslocamento:
d['a'] = 1
d['b'] = 2
d['c'] = 3

1. Inicialmente são atribuídos os valores à tabela e é feita a comparação.

T = a a b c a c c a c b a c
P = c a c b a c

2. Independente de qual caractere for, quando há colisão, desloca-se o valor associado ao último caractere. No caso 3 posições já que o último era o c.

T = a a b c a c c a c b a c
P = c a c b a c

3. O procedimento segue até encontrar o casamento.

- Quanto maior o tamanho do Alfabeto, a chance de o caractere do Texto alinhado com o Padrão não fazer parte do Padrão é grande. Logo, a chance de o tamanho do deslocamento ser o Padrão como um todo é grande. Como este caso pode acontecer várias vezes tem-se um ganho de eficiência.

Algoritmo Boyer-Moore-Horspool-Sunday (BMHS)

- Outra melhoria foi implementada gerando o BMHS, que ocasionou em resultados melhores dentro de Textos em linguagens naturais.
- No BMH, o cálculo do deslocamento se baseia no último caractere do Texto alinhado ao último caractere do Padrão. Dessa forma, na geração da Tabela, a última posição do Padrão não é considerada.
- A fim de considerar tal caractere, foi proposto que a janela se deslocasse de acordo com o valor da tabela de deslocamento relativo ao caractere no Texto correspondente ao caractere **após o último caractere do Padrão**. (/0)
- Dessa forma, as novas regras para construir a tabela, incluindo o último valor do Padrão são:
 - O valor inicial do deslocamento para todos os caracteres do texto é igual a **m+1**.
 - Em seguida, para obter os valores do deslocamento dos **m** primeiros caracteres do Padrão, usa-se a fórmula:

$$d[x] = \min\{j \text{ tal que } (j = m+1) \vee (1 \leq j \leq m \ \& \ P[m+1-j] = x)\}$$

- Dessa forma, para todos os caracteres, com exceção de um, há uma melhora no valor de deslocamento.

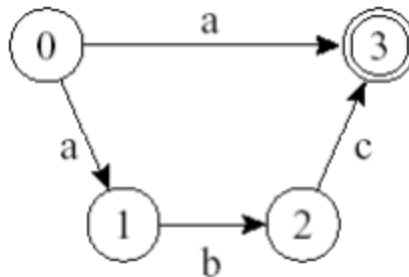


Algoritmo Shift-And exato

<input checked="" type="checkbox"/> Column	<input type="checkbox"/>
Files	
Unit/Module	Casamento de cadeias
Video	https://www.youtube.com/watch?v=3oMKhTrT17o
Week	

- A fórmula do Shift-And é baseada na utilização de **autômatos**.

Autômatos



- Estrutura de dados composta por:
 - Q é um conjunto finito de estados (*nódulos*). **No exemplo: 0, 1, 2, 3.**
 - I é o estado inicial ($I \in Q$). O **primeiro** estado, como a raiz de uma Árvore. **No exemplo: 0.**
 - F é o conjunto de estados finais ($F \subseteq Q$). Pode ser um ou mais. **No exemplo é representado pela linha dupla: 3.**
 - Σ é o alfabeto finito de entrada. **Responsáveis por rotular as arestas.**

- **T** é a função que define as **transições** entre os estados.
- No casamento de cadeias, o **Padrão** será representado como a sequência de caracteres que sai do estado inicial e chega no estado final.
- Um autômato pode ser classificado como:
 - **Determinista:** A partir de um determinado estado e um dado caractere só é possível atingir um estado.

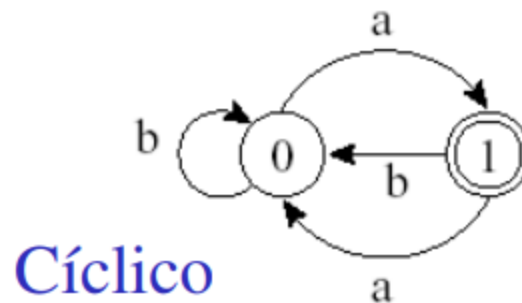
$$T(q, \alpha) = \{q_1\}$$

- **Não-determinista:** Um autômato no qual dentro da função de transições, dado um estado q e um caractere x qualquer de dentro do Alfabeto, **é possível atingir vários outros estados**.

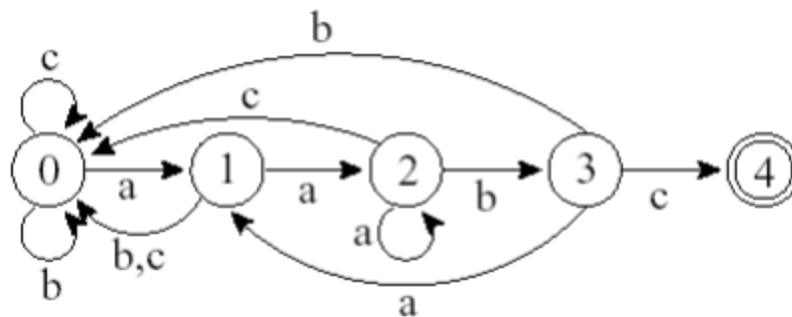
$$T(q, \alpha) = \{q_1, q_2, \dots, q_k\} \text{ para } k > 1$$

- Como no exemplo inicial, **a partir no estado 0, dado o caractere a é possível atingir tanto o estado 1 como o estado 3.**
- O conjunto de todas as cadeias que o autômato pode reconhecer é chamada de **linguagem**. No exemplo: **{a}** e **{abc}**.
- Dentro dos autômatos podem ser colocadas **transições vazias**, transições utilizadas em autômatos não deterministas que servem para **ligar dois estados sem a necessidade de ler um determinado caractere**.
- Um autômato é formado por vários estados pelos quais é possível "caminhar", o estado em que nos encontramos correntemente no caminhar é chamado de **estado ativo**. Isso significa que foram lidos no texto os caracteres que conectam os estados.
- Os autômatos podem ser:

- **Cíclicos:** São aqueles que formam ciclos, podendo-se **reconhecer inúmeras cadeias**.



- **Acíclicos:** Não existe formação de ciclo representada pelas arestas. Como o do exemplo inicial, que reconhece apenas 2 cadeias.
- No Shift-And, fazemos o uso de autômatos para reconhecer o padrão que desejamos. No exemplo, o autômato reconhece **P = {aabc}**.



- Tendo um Padrão de 4 caracteres, cria-se um autômato para representá-lo com 5 estados e 4 vértices. A quantidade de estados sempre é 1 a mais que a do Padrão.
- O **estado final** só é atingido caso seja lida a sequência desejada, ou seja, encontrou-se o Padrão desejado.
- O autômato possui **outras arestas** que representam o que deve ocorrer caso seja lido **outro caractere** (*Diferente do Padrão*) dentro do texto, ficando em **ciclo** até que o Padrão seja encontrado.

- Caso o Alfabeto seja muito grande, o custo de construção do autômato pode ser muito grande, deixando o método menos vantajoso. De forma que, o autômato não é construído e sim **simulado por uma cadeia binária de bits**, onde cada bit representa um estado.

Algoritmo Shit-And Exato

- O algoritmo utiliza o conceito de **paralelismo de bit**. Como cada estado é representado por um bit, a medida que os caracteres são lidos, **a sequência de bits é manipulada**. Essa manipulação utiliza **operações binárias**:
 - **Repetição de bits**: exponenciação (ex.: $01^3 = 0111$).
 - **|** : operador lógico **or**.
 - **&** : operador lógico **and**.
 - **>>** : operador **shift** que move os bits para a direita e entra com zeros à esquerda.
- O autômato é representado por uma sequência de bits **R = (b1, b2, ... ,bm)**. Iniciando da primeira posição, desconsiderando o estado inicial e sendo m o tamanho do Padrão. A sequência é **inicializada com 0**, que representa os **estados inativos**. Durante o caminhar os bits são manipulados para apresentar o **valor 1**, indicando **estados ativos**. Sabe-se que o Padrão foi encontrado quando último bit apresenta valor 1.
- A cada leitura, a máscara é convertida em uma nova máscara, e nela é buscado o bit 1 na última posição. Localizado ou não, o procedimento se repete.

```

T = os testes testam ...
P = teste

R = 00000 => R' = 00000 => 10000

```

- A fórmula de transformação representa o caminhar dentro do autômato que corresponde ao Padrão desejado.

Pré-processamento

- Construção de uma tabela **M** que representa **uma máscara de bits para cada caractere do Padrão**, representando qual a posição que aquele caractere do Padrão está dentro do próprio Padrão. Coloca-se o bit 1 se naquela posição do Padrão o caractere estiver presente. *No exemplo $P = \{teste\}$:*

	1	2	3	4	5
M[t]	1	0	0	1	0
M[e]	0	1	0	0	1
M[s]	0	0	1	0	0

Algoritmo

- A máscara **R** é inicializada com 0.
- Para cada caractere lido do texto, o valor da máscara **R'** é atualizada de acordo com a fórmula de transformação:

$$R' = ((R \gg 1) \mid 10^{m-1}) \& M[T[i]].$$

- O shift (\gg) representa uma **tentativa de caminhar** pelo autômato. Da mesma forma, o ou (\mid) é uma tentativa de localizar no Texto o Padrão desejado ou uma sequência vazia, para caminhar de um estado a outro.
- Com o resultado da tentativa de caminhamento é feito um e ($\&$) com a máscara do caractere que acaba de ser lido de dentro do Texto. Isso irá dizer se a tentativa de caminhamento **se tornará um caminhamento real**.
- Pode-se ter uma sequência com **mais de um estado estado ativo (1)**, porque as máscaras são construídas de tal forma a encontrar **não somente um único**

casamento exato, mas a prospecção de casamentos exatos.

Shift-And ($P = p_1p_2 \dots p_m, T = t_1t_2 \dots t_n$) Passando o Padrão e o Texto

```
{ /*—Préprocessamento—*/  
  for (c ∈ Σ)  $M[c] = 0^m$ ; Criação da máscara de Bits para todos os caracteres do Alfabeto e inicializa com 0  
  for (j = 1; j ≤ m; j++)  $M[p_j] = M[p_j] \mid 0^{j-1}10^{m-j}$ ; Aplicação da fórmula somente para os caracteres presentes no padrão  
  /*—Pesquisa—*/  
   $R = 0^m$ ;  
  for (i = 1; i ≤ n; i++) Para cada caractere do Texto é feita a leitura e aplicada a fórmula  
    {  $R = ((R \gg 1 \mid 10^{m-1}) \& M[t_i])$ ;  
      if (  $R \& 0^{m-1}1 \neq 0^m$  ) 'Casamento na posicao  $i - m + 1$ '; Verifica se o 1 aparece na última posição da sequência  
    }  
}
```

- **Análise:** A complexidade do algoritmo é $O(n)$ uma vez que é necessário ter todos os caracteres do texto um a um.
- O que torna o Shift-And melhor que a Força Bruta é que não há necessidade de fazer varredura do Padrão.



Algoritmo Shift-And aproximado

<input checked="" type="checkbox"/> Column	<input type="checkbox"/>
Files	
Unit/Module	Casamento de cadeias
Video	https://www.youtube.com/watch?v=4McQTiec2-A
Week	

Casamento Aproximado

O problema de casamento aproximado de cadeias consiste em encontrar as ocorrências aproximadas e exatas de um padrão P no texto T. Para que isso ocorra de forma correta, é necessário tratar operações de inserção, substituição e retirada de caracteres do padrão.

Exemplo para o padrão {teste}:

1. Inserção: espaço inserido entre o 3º e 4º caracteres do padrão
2. Substituição: último caractere do padrão substituído pelo **a**
3. Retirada: primeiro caractere do padrão retirado

1 → tes te (cadeia similar com um erro de inserção)

2 → testa (cadeia similar com um erro de substituição)

3 → este (cadeia similar com um erro de retirada)

$T(n)$ = os testes testam estes alunos

Exemplo

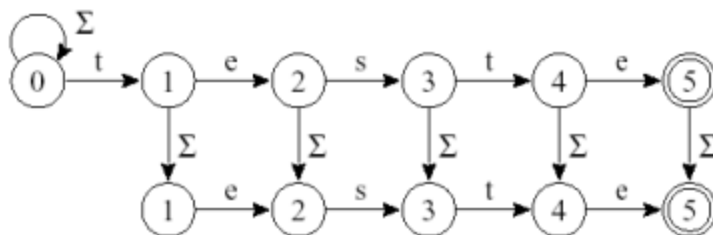
O conceito utilizado para definir o problema é chamado de **distância de edição** entre cadeias P e P' , denotada por $ed(P, P')$. É o **menor número de operações necessárias** para fazer com que P seja exatamente igual a P' ou vice-versa. O problema do casamento aproximado de cadeias é encontrar todas as ocorrências de P' no texto T tal que $ed(P, P') \leq k$, onde k representa o limite de operações para transformar o padrão P em P' .

Quanto maior o valor de k , maior o número de erros de P' , $0 < k < m$, pois para $k = m$, toda subcadeia de comprimento m pode ser convertida em P por meio da substituição de m caracteres. Uma medida de fração do padrão que pode ser alterada é dada pelo nível de erro $\alpha = k/m$. O ideal seria $\alpha < 50\%$ porque se o valor de k for muito alto, onde é atingido um nível de erro acima de 50% poderíamos começar a receber cadeias que não são muito similares a cadeia desejada.

A pesquisa do casamento aproximado é modelada por **autômatos não-deterministas** (a partir de um estado e um caractere podemos atingir vários outros estados. No mesmo espaço de tempo podemos ter vários estados ativos) e os algoritmos utilizam **paralelismo de bit** (ter uma determinada forma binária que tem o custo **extremamente rápido** de ser executado justamente para fazer com que o casamento aproximado seja **tão rápido quanto o casamento exato**).

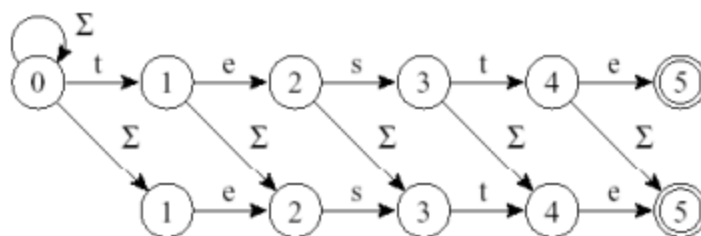
Operações com o Padrão

- O autômato abaixo permite que haja apenas um erro no texto. Onde as arestas horizontais representam um casamento de caracteres, avançando no texto T e no padrão P . As arestas verticais representam a inserção de um caractere no padrão P , avançando-se no texto T , mas não no padrão P . Caso eu queria permitir mais um erro, basta adicionar mais uma linha horizontal abaixo da segunda linha.



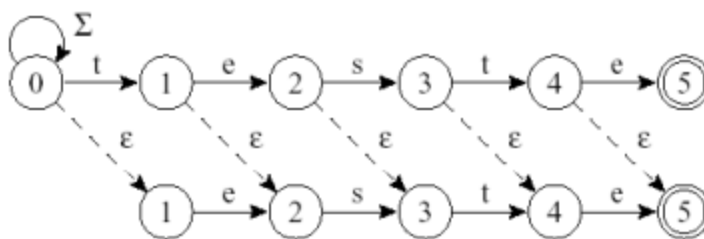
Inserção que permite apenas um erro

- O autômato abaixo permite apenas um erro de substituição. As arestas horizontais representam um casamento de caracteres, avançando no texto T e no padrão P. As arestas diagonais representam a substituição de um caractere no padrão P, avançando-se no texto T e no padrão P. Para permitir mais erros, basta adicionar mais uma linha horizontal abaixo.



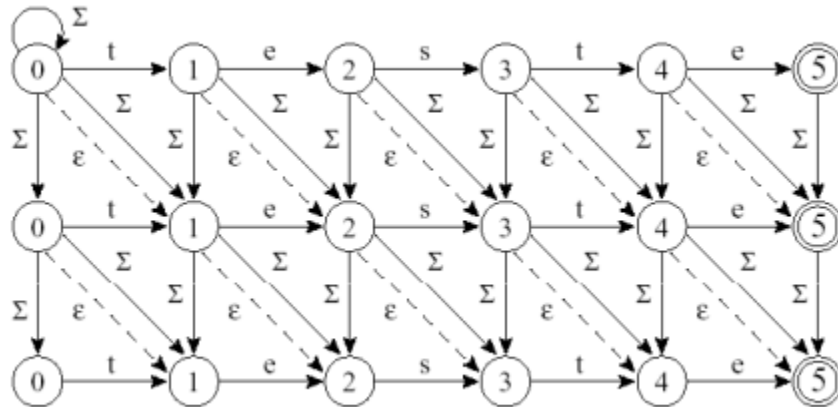
Substituição que permite apenas um erro

- O autômato abaixo permite apenas um erro de remoção. As arestas horizontais representam um casamento de caracteres, avançando no texto T e no padrão P. As arestas diagonais tracejadas representam a retirada de um caractere no padrão P, avançando-se no padrão P, mas não no texto T (transição vazia). Para permitir mais erros, basta adicionar mais uma linha horizontal abaixo.



Remoção que permite apenas um erro

O autômato não-determinista abaixo permite duas opções de retirada, duas de remoção e duas de inserção. Na linha 1 ocorre o casamento exato, então $k = 0$. Na linha 2, ocorre o casamento aproximado permitindo um erro e na linha 3 ocorre o casamento aproximado permitindo dois erros.



Remoção, Inserção e substituição permitindo de 0 a 2 erros.

Algoritmo SHIFT-AND aproximado

O algoritmo Shift-And aproximado simula o autômato não-determinista utilizando sequências binárias voltadas ao paralelismo de bit. O algoritmo empacota cada linha j ($0 < j \leq k$) do autômato não-determinista em uma palavra R_j diferente do computador. Os autômatos serão representados por várias máscaras de bits. A quantidade de máscaras que serão utilizadas depende do valor de k .

No funcionamento, o R_0 é inicializado com 0 e R_j com $1j0m-j$. A tabela de máscaras é a mesma para o casamento exato e aproximado. Uma vez inicializado as máscaras de bits R e a máscara M , nós vamos começar a ler caracteres do texto. A cada caractere lido, é aplicada a fórmula de transformação do R , transformando o R em R' . As máscaras são atualizadas pelas expressões:

$$R_0' = ((R \gg 1) | 10m-1) \& M[T[i]] \quad (\text{a mesma do casamento exato})$$

$$R_j' = ((R_j \gg 1) \& M[T[i]] | R_{j-1} | R_{j-1} \gg 1) | (R_{j-1}' \gg 1) | 10m-1 \quad \text{para } 0 < j \leq k$$

- Considerando o autômato, a fórmula para R' expressa as arestas:
 - Horizontais, indicando casamento de um caractere;
 - Verticais, indicando **inserção** (R_{j-1});

- Diagonais cheias, indicando substituição ($R_{j-1} \gg 1$);
- Diagonais tracejadas, indicando remoção ($R'_{j-1} \gg 1$);

Algoritmo:

Pesquisando pelo padrão de tamanho M no texto de tamanho N, considerando K erros:

```
void Shift-And-Aproximado ( $P = p_1 p_2 \dots p_m, T = t_1 t_2 \dots t_n, k$ )
{ /*— Préprocessamento —*/
  for ( $c \in \Sigma$ )  $M[c] = 0^m$ ;
  for ( $j = 1; j \leq m; j++$ )  $M[p_j] = M[p_j] | 0^{j-1} 10^{m-j}$ ;
  /*— Pesquisa —*/
  for ( $j = 0; j \leq k; j++$ )  $R_j = 1^j 0^{m-j}$ ;
  for ( $i = 1; i \leq n; i++$ )
  {  $R_{ant} = R_0$ ;
     $R_{novo} = ((R_{ant} \gg 1) | 10^{m-1}) \& M[T[i]]$ ;
     $R_0 = R_{novo}$ ;
    for ( $j = 1; j \leq k; j++$ )
    {  $R_{novo} = ((R_j \gg 1 \& M[T[i]]) | R_{ant} | ((R_{ant} | R_{novo}) \gg 1))$ ;
       $R_{ant} = R_j$ ;
       $R_j = R_{novo} | 10^{m-1}$ ;
    }
    if ( $R_{novo} \& 0^{m-1} 1 \neq 0^m$ ) 'Casamento na posicao i';
  }
}
```