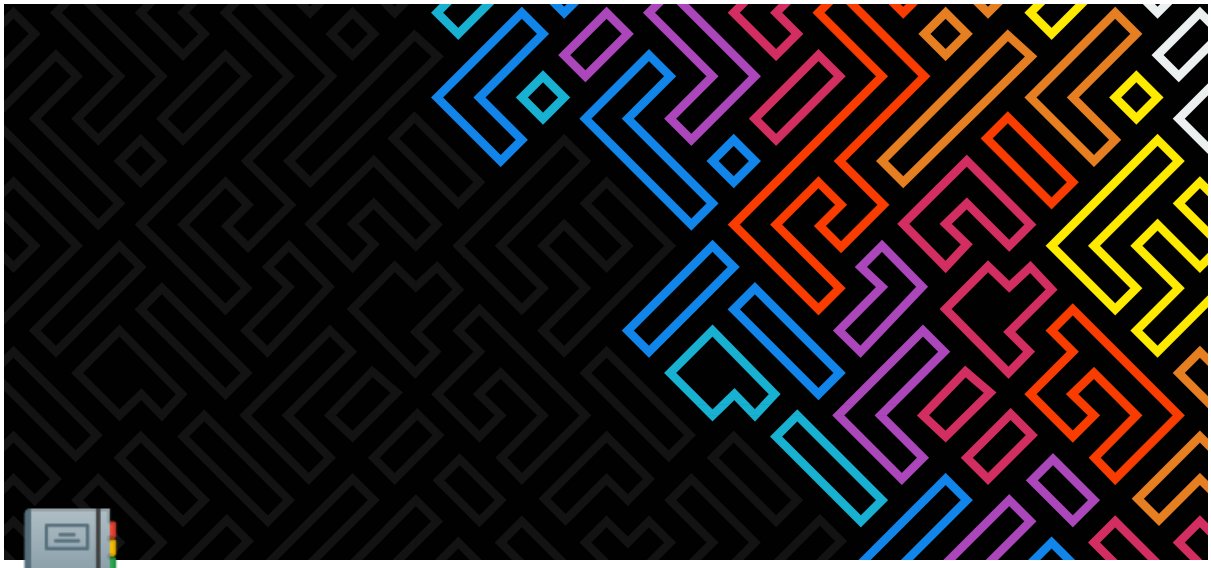


Casamento de Cadeias

🎓 **Aluno**: Lucas de Araújo
Matrícula: 18.2.4049
Professor Doutor: Guilherme Tavares de Assis

Introdução

O **Casamento de Cadeias** consiste, de maneira resumida, em uma busca de uma cadeia de caracteres dentro de um texto. Os caracteres são escolhidos de um conjunto chamado de alfabeto.

- Cadeia de Bits
  - Alfabeto: $\{0, 1\}$

O problema de casamento de cadeias consiste em encontrar todas as ocorrências de determinado padrão em um determinado texto

Como exemplos de aplicação, podemos citar:

- Edições de Texto
  - Exemplo: $ctrl + f$ para se realizar a busca de um padrão em um texto e editá-lo
- Recuperação de Informação
  - Exemplo: Busca da localização de informações em repositórios com grandes quantidades de informações
- Estudo de Sequências de DNA na Biologia Computacional
  - Exemplo: Localização de padrão dentro de uma sequência de bases nitrogenadas de uma faixa de DNA

Formalização do Problema

Para formalizar problemas relacionados ao casamento de cadeias, podemos descrevê-los da seguinte maneira:

- **Texto**: Cadeia $T[0 \ldots n-1]$ de tamanho n
- **Padrão**: Cadeia $P[0 \ldots m-1]$ de tamanho $m \leq n$
- Os elementos P e T são escolhidos de um alfabeto (finito) $\sum$ de tamanho c
  - Exemplo: $\sum = \{0, 1\}$
  - Exemplo: $\sum = \{a, b, ..., z\}$
- **Casamento de Cadeias**: Dadas as cadeias P de comprimento m e T de comprimento n, onde $m \leq n$, deseja-se saber as ocorrências de P em T

Estrutura de Dados - Código Exemplo

```
#define MAXTAMTEXTO 1000

#define MAXTAMPADRAO 10

#define MAXCHAR 256

#define NUMMAXERROS 10

typedef char TipoTexto[MAXTAMTEXTO];

typedef char TipoPadrao[MAXTAMPADRAO];
```

Categoria de Algoritmos

Podemos separar em três categorias os algoritmos relacionados ao casamento de cadeias, sendo eles:

P e T não são pré-processados

- Padrão e texto não são conhecidos a priori, portanto não sofreram nenhuma ação sobre si
- Algoritmo sequencial, on-line e de tempo-real
- Complexidades:
  - Complexidade de Tempo: $O(mn)$
    - Quando o último caractere do padrão sempre acaba por ser diferente ao se realizar a leitura do texto, sendo necessário varrer várias vezes o padrão
  - Complexidade de Espaço: $O(1)$
    - Como não se realiza nenhum pré-processamento, temos apenas as variáveis armazenadas, sem nenhuma estrutura de dados adicional

P é pré-processado

- Padrão é conhecido a priori, logo é permitido seu pré-processamento. Dessa forma evitamos a varredura repetida do padrão
- Algoritmo sequencial
- Complexidades:

- Complexidade de Tempo: $O(n)$
  - Complexidade se torna exclusivamente dependente do texto
- Complexidade de Espaço: $O(m + c)$
  - Um espaço adicional é necessário para que se realize o pré-processamento do padrão, portanto a complexidade também se torna dependente da quantidade de caracteres do alfabeto (c)

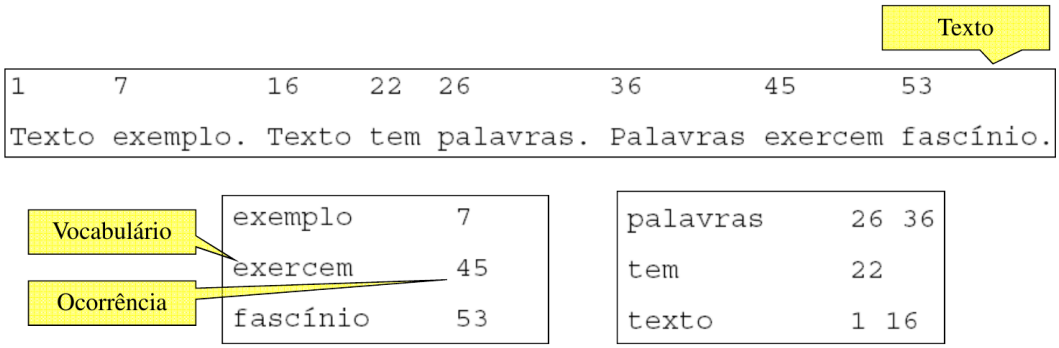
- Exemplo de Aplicação: Programas para edição de textos

P e T são pré-processados

- Padrão e Texto são conhecidos a priori
- Algoritmo constrói indice para o texto
  - Se torna interessante construir um índice quando a base de dados é grande e semi-estática, ou seja, possui atualizações em intervalos regulares
  - O tempo para geração do índice pode ser tão grande quanto $O(n)$ ou $O(n * \log n)$, mas é compensado por muitas operações de pesquisa no texto
- Tipos de Índices:
  - Arquivos Invertidos
    - Principal utilizado em arquivos grandes
  - Árvores TRIE e Árvores PATRICIA
  - Arranjos de Sufixos
- Complexidades:
  - Complexidade de Tempo: $O(\log n)$
  - Complexidade de Espaço: $O(n)$

Arquivo Invertido

É uma estrutura composta por vocabulário e ocorrências. O vocabulário é o conjunto das palavras distintas no texto e para cada palavra distinta, uma lista de posições onde ela ocorre no texto é armazenada. O conjunto de listas é chamado de ocorrências



Exemplo retirado dos slides

As ocorrências tendem a consumir normalmente (na maioria dos casos), mais espaço na memória que o vocabulário. Por conta disso, as ocorrências normalmente serão armazenadas em memória secundária (arquivos) e o vocabulário em memória principal

A previsão que determina o crescimento do tamanho do vocabulário é conhecida como **Lei de *Heaps***

- O vocabulário de um texto em linguagem natural contendo n palavras tem tamanho $V = Kn^{\beta} = O(n^{\beta})$, onde K e β dependem das características de cada texto
- K geralmente assume valores entre 10 e 100, e β é uma constante entre 0 e 1 (na prática, entre $0,4$ e $0,6$)
- Na prática, o vocabulário cresce com o tamanho do texto em proporção perto de sua raiz quadrada

Como dito, as ocorrências ocupam bem mais espaço. Podemos dizer também que:

- O espaço necessário $O(n)$ já que cada palavra é referenciada uma vez na lista de ocorrências
- Na prática, o espaço fica entre 30% e 40% do tamanho do texto

Pesquisa

A pesquisa é realizada em três passos:

1. **É feito a pesquisa no vocabulário**: palavras da consulta são isoladas e pesquisadas no vocabulário
2. **Recuperação das ocorrências**: as listas de ocorrências das palavras encontradas no vocabulário são recuperadas
3. **Manipulação das ocorrências**: as listas de ocorrências são processadas para tratar frases, proximidade e/ou operações lógicas

Como a pesquisa em um arquivo invertido sempre começa pelo vocabulário, é interessante mantê-lo em um arquivo separado. Normalmente, esse arquivo irá caber em memória principal

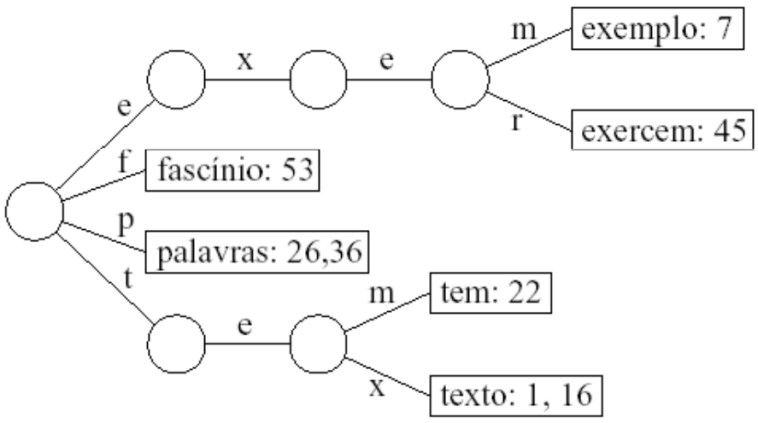
A pesquisa por palavras simples pode ser realizada usando qualquer estrutura de dados que torne a pesquisa eficiente.

- Árvore TRIE e Hashing: tem custo $O(m)$, onde m é o tamanho da consulta, independente do tamanho do texto
- Árvore B: possui custo $O(\log n)$ pois guardar as palavras na ordem lexicográfica é barato em termos de espaço e competitivo em desempenho

A pesquisa por frases através de índices é mais difícil, pois cada palavra da frase deve ser pesquisada separadamente no intuito de recuperar suas listas de ocorrências. Em seguida, as listas devem ser percorridas de forma sincronizada no intuito de encontrar as ocorrências nas quais todas as palavras aparecem em sequência

Exemplo Ilustrado - Utilizando Árvore TRIE

- Texto: "Texto tem palavras. Palavras exercem fascínio."
- Arquivo invertido usando uma árvore *TRIE*:



O vocabulário lido do texto é colocado em uma árvore *TRIE,* armazenando uma lista de ocorrências para cada palavra.

Exemplo retirado dos slides

Cada palavra lida no texto é pesquisada na TRIE:

- Se a pesquisa não tiver sucesso, a palavra é inserida na árvore e uma lista de ocorrências é inicializada com a posição de tal nova palavra no texto
- Caso a pesquisa tenha sucesso (palavra já se encontra presente na árvore), a nova posição é inserida ao final da lista de ocorrências da mesma

Observação: Em repositórios com capacidade de gerar um vocabulário muito extenso, é preferível a árvore TRIE em relação a árvore PATRICIA

Tipos de Casamentos de Cadeia

Casamento Exato

O problema de casamento exato de cadeias consiste, em suma, encontrar as ocorrências exatas de um padrão em um texto. Os algoritmos podem ser categorizados em relação à forma como o padrão é pesquisado no texto

- Leitura dos caracteres do texto um a um, no intuito de identificar a ocorrência possível do padrão.
  - Exemplos: Força Bruta e Shift-And
- Pesquisa do padrão em uma janela que desliza ao longo do texto, procurando por um sufixo da janela (texto) que casa com um sufixo do padrão, mediante as comparações realizadas da direita para a esquerda
  - Exemplos: Boyer-Moore, Boyer-Moore-Horspool e Boyer-Moore-Horspool-Sunday

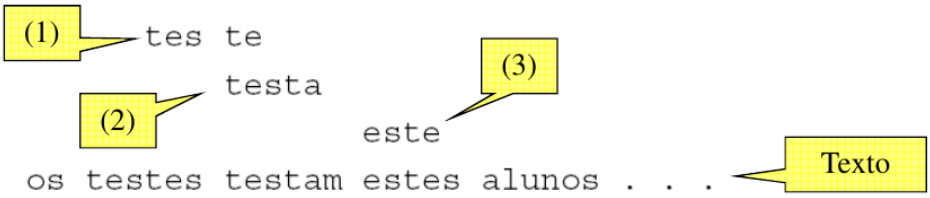
Casamento Aproximado

O problema de casamento aproximado de cadeias consiste em encontrar as ocorrências aproximadas de um determinado padrão no texto. Por conta disso, seremos apresentados as novos três tipos de operações sobre os caracteres do padrão: **Inserção**, **Substituição** e **Retirada**.

Exemplo

No exemplo a seguir, aparecem três ocorrências aproximadas do padrão $\{teste\}$:

1. **Inserção**: Espaço inserido entre o 3° e 4° caractere do padrão
2. **Substituição**: último caractere do padrão substituído pelo caractere a
3. **Retirada**: Primeiro caractere do padrão retirado



Exemplo de casamento aproximado retirado dos slides

Detalhes do Problema

Distância de edição entre duas cadeias P e P', denotada por $ed(P, P')$ é o menor número de operações para converter uma cadeia em outra.

- Por exemplo: $ed(teste, estende) = 4$
  - Valor obtido por meio da retirada do primeiro t de P e a inserção dos três caracteres nde ao final do mesmo

Formalmente, o problema do casamento aproximado de cadeias é o de encontrar todas as ocorrências de P' no texto tal que $ed(P, P') \leq k$, onde o valor de k representa o número limite de operações de inserção, substituição e retirada de caracteres necessárias para transformar o padrão P em P'

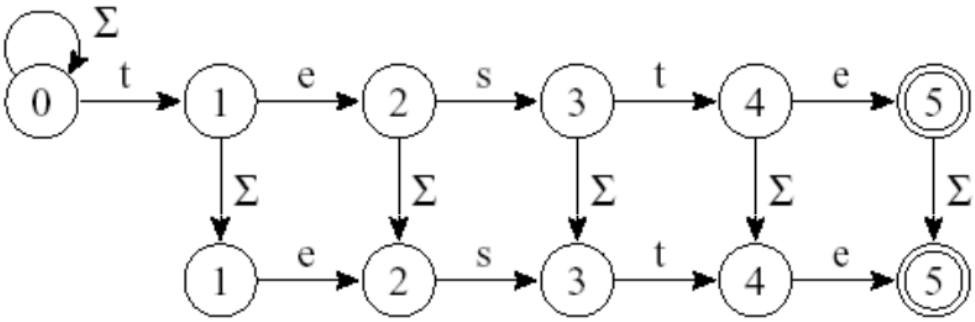
O casamento aproximado só faz algum sentido quando este valor de k se encontra entre 0 e m, pois caso $k = m$, podemos acabar convertendo todos caracteres e encontrando padrões sem nenhuma relação com nosso padrão inicial desejado (Ex: $\{teste\}$ e $\{papel\}$). Caso o $k = 0$, teremos um casamento exato de cadeias

Uma medida da fração do padrão que pode ser alterada é dada pelo nível de erro $\alpha = \frac{k}{m}$. Em geral, $a = \frac{1}{2}$ para a maioria dos casos

A pesquisa com o casamento aproximado é modelado por autômatos não-deterministas utilizando paralelismo de bit. Este conceito será apresentado no futuro deste mesmo PDF

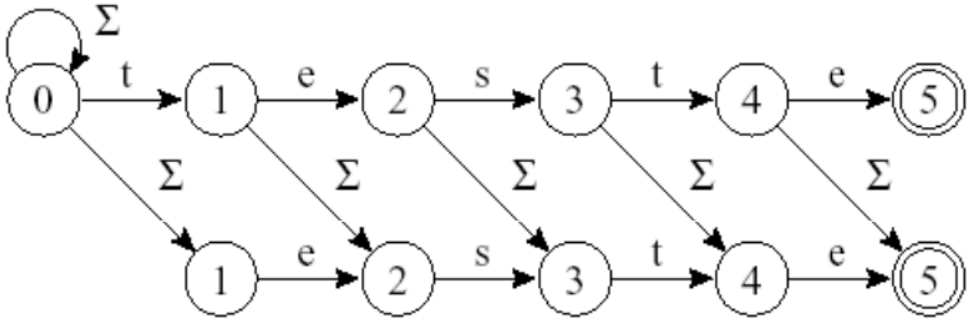
Autômatos Aplicados a Casamento Aproximado - Individual

Autômato que reconhece o padrão $P = \{teste\}$, permitindo uma inserção:



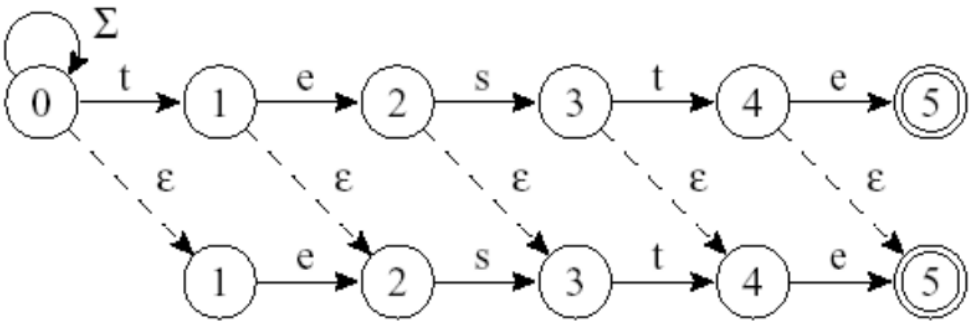
- Uma aresta horizontal representa um casamento de caractere, avançando-se no texto e no padrão
- Uma aresta vertical insere um caractere no padrão, avançando no texto **mas não** no padrão

Autômato que reconhece o padrão $P = \{teste\}$, permitindo uma substituição:



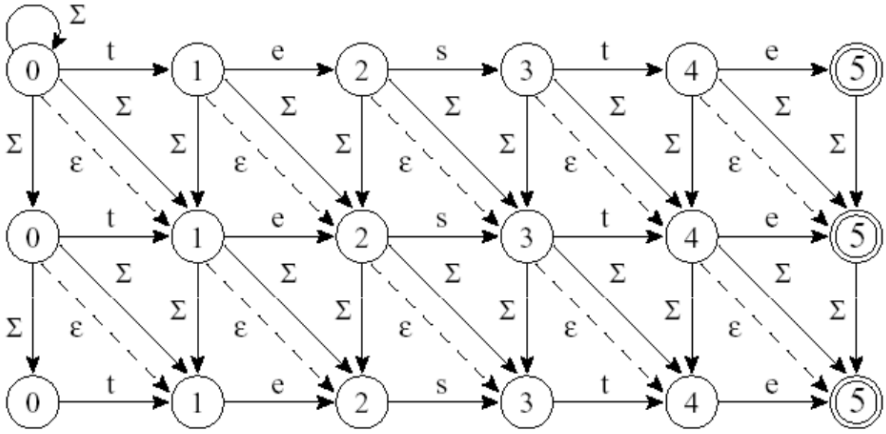
- Uma aresta horizontal representa um casamento de caractere, avançando-se no texto e no padrão
- Uma aresta diagonal substitui um caractere no padrão, avançando no texto **e** no padrão

Autômato que reconhece o padrão $P = \{teste\}$, permitindo uma retirada:



- Uma aresta horizontal representa um casamento de caractere, avançando-se no texto e no padrão
- Uma aresta diagonal retira um caractere no padrão, avançando no padrão **mas não** no texto (transição vazia)

Autômatos Aplicados a Casamento Aproximado - Conjunto

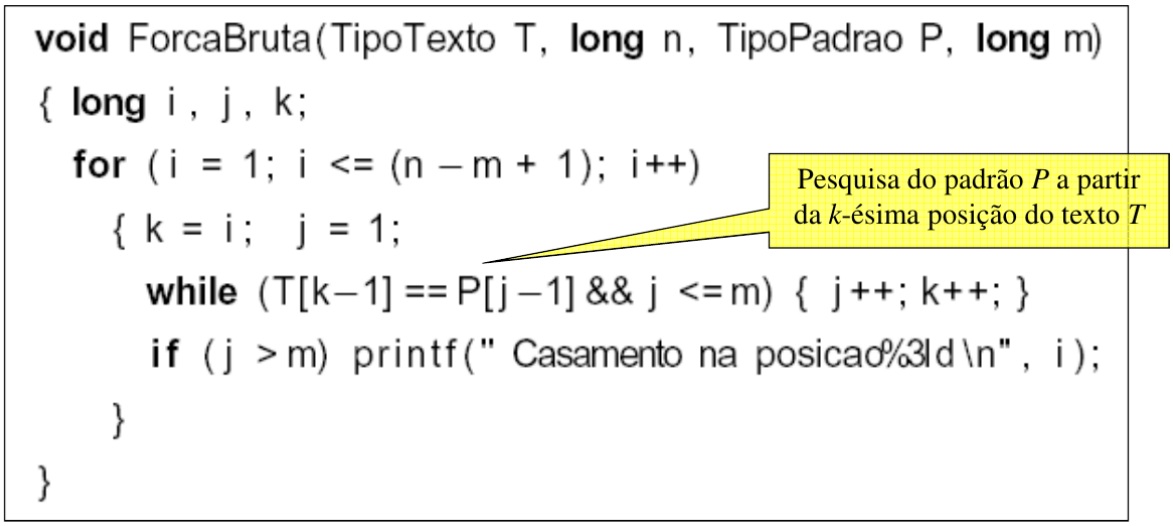


- O autômato reconhece $P = \{\text{teste}\}$ para $k = 2$.
  - Linha 1: casamento exato ($k = 0$).
  - Linha 2: casamento aproximado permitindo um erro ($k = 1$).
  - Linha 3: casamento aproximado permitindo dois erros ($k = 2$).

Algoritmos (Primeira Parte)

Algoritmo de Força Bruta

O algoritmo de força bruta é o algoritmo mais simples para casamento exato de cadeias. A ideia consiste em tentar casar qualquer subcadeia de comprimento m no texto com o padrão desejado



Análise do Algoritmo

- **Pior Caso**: $C_n = m * n$
  - Exemplo: $P = aab$ e $T = aaaaaa$
- **Caso Esperado**: $C_n = \frac{c}{c-1}(1 - \frac{1}{c^m})(n - m + 1) + O(1)$
  - O caso esperado é bem mais eficiente quando comparado ao pior caso

Algoritmo de Boyer-Moore

Surge em 1977 o algoritmo de casamento de cadeias conhecido como algoritmo Boyer-Moore (BM).

Seu funcionamento segue os seguintes passos:

- O BM pesquisa o padrão P em uma janela (do tamanho do padrão) que desliza ao longo do texto T
- Para cada posição desta janela, o algoritmo pesquisa por um sufixo da mesma que casa com um sufixo do padrão P, com comparações realizadas no sentido da direita para a esquerda
  - Se **não** ocorrer desigualdade, uma ocorrência de P em T foi localizada
  - Caso ocorra uma desigualdade, o algoritmo calcula o deslocamento que a janela deverá deslizar para direita antes que uma nova tentativa de casamento se inicie

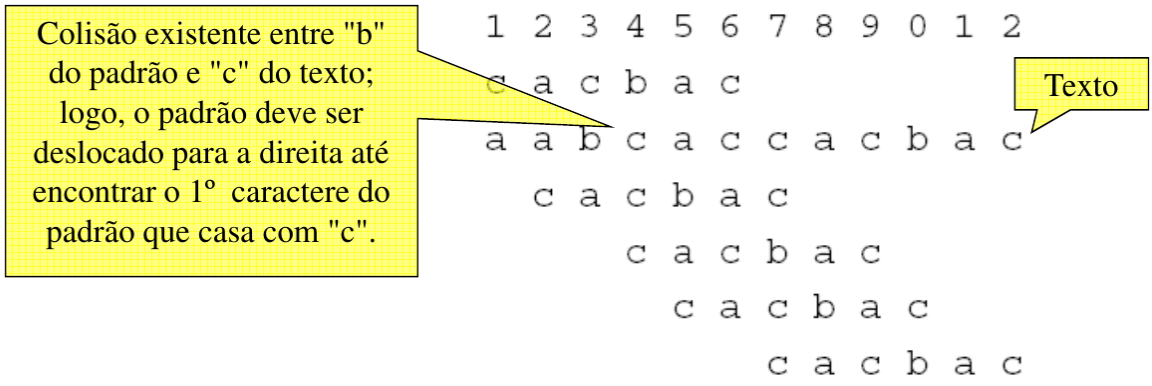
- São propostas duas heurísticas pelo BM para calcular este deslocamento: Ocorrência e Casamento

> 📌 As heurísticas tem como objetivo aumentar a eficiência do algoritmo de Boyer-Moore, pois se não teríamos apenas um algoritmo similar a força bruta (movendo-se a janela um a um caractere), porém sendo realizado ao contrário

Heurística Ocorrência

A heurística ocorrência alinha o caractere no texto que causou a colisão com o primeiro caractere no padrão, a esquerda do ponto de colisão, que casa com ele.

$$P = \{cacbac\}$$
$$T = \{aabcaccacbac\}$$



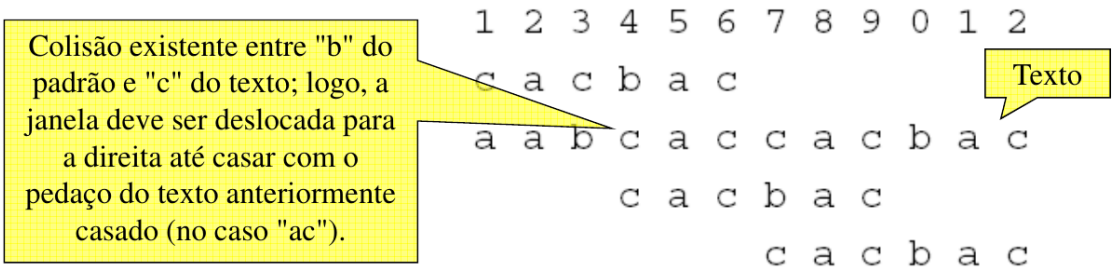
Caso o caractere não seja encontrado no restante do padrão após colisão, então o tamanho caminhado pelo deslocamento da janela será igual o tamanho do padrão.

Observação: Quanto maior o tamanho do alfabeto, melhor a heurística ocorrência se torna quando comparada a força bruta em termos de eficiência

Heurística Casamento

A heurística casamento faz com que, ao mover o padrão para direita, a janela em questão casa com o pedaço do texto anteriormente casado

$$P = \{cacbac\}$$
$$T = \{aabcaccacbac\}$$



O algoritmo Boyer-Moore decide qual das duas heurísticas deve seguir, escolhendo aquela que provoca o maior deslocamento do padrão. Esta escolha implica em realizar comparações para cada colisão que ocorrer, penalizando o desempenho do algoritmo com relação a tempo de processamento

Ao longo dos anos, várias propostas de simplificação surgiram, sendo que os melhores resultados foram obtidos por aquelas que consideraram apenas a heurística ocorrência

Algoritmo de Boyer-Moore-Horspool

Em 1980, Horspool apresentou uma simplificação no algoritmo BM, que o tornou mais rápido. Com isso, surge o algoritmo de **Boyer-Moore-Horspool** (BMH)

Pela extrema simplicidade de implementação e comprovada eficiência, o BMH deve ser escolhido em aplicações de uso geral que necessitam realizar o casamento exato de cadeias.

Sua simplificação pode ser descrita pelos seguintes pontos:

- Parte da observação de que qualquer caractere já lido do texto a partir do último deslocamento pode ser usado para endereçar uma tabela de deslocamentos
- Com isso, Horspool propôs deslocar a janela de acordo com o valor da tabela de deslocamento relativo ao caractere no texto correspondente ao último caractere do padrão

Tabela de Deslocamentos

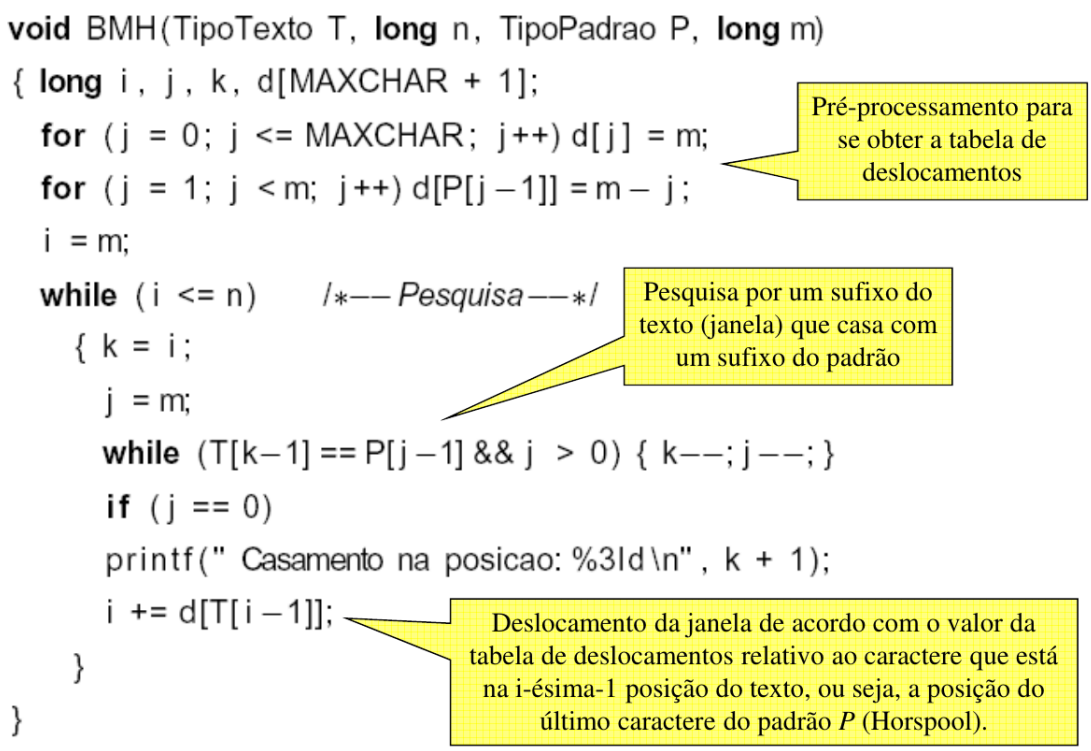
Para definirmos a tabela de deslocamentos, consideremos:

- O valor inicial do deslocamento para todo os caracteres do texto é igual a m
- Em seguida, para os $m - 1$ primeiros caracteres do padrão P, os valores do deslocamento são calculados pela seguinte regra:

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

- **Exemplo**: Para o padrão $P = \{teste\}$, os valores da tabela são:
  - $d[\text{"}t\text{"}] = 1,$
    $d[\text{"}e\text{"}] = 3,$
    $d[\text{"}s\text{"}] = 2;$
  - $d[x] = 5$ (valor de m) para todo caractere x do texto que não faça parte do padrão

Código Exemplo



Algoritmo Boyer-Moore-Horspool-Sunday

Em 1990, Sunday apresentou uma simplificação importante para o algoritmo BMH, tornando o conhecido agora como **Boyer-Moore-Horspool-Sunday** (BMHS). A diferença notável no algoritmo vai se dar em relação ao pré-processamento durante a criação da tabela de deslocamentos

Sua simplificação pode ser descrita pelos seguintes pontos:

- Corresponde a uma variante do algoritmo BMH
- Sunday propôs deslocar a janela de acordo com o valor da tabela de deslocamento relativo ao caractere no texto correspondente ao caractere após o último caractere do padrão

Tabela de Deslocamentos

Para definirmos a tabela de deslocamentos, consideremos:

- O valor inicial do deslocamento para todos os caracteres do texto agora é igual a $m + 1$
- Em seguida, para os m primeiros caracteres do padrão P, os valores do deslocamento são calculados pela nova regra:

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

- **Exemplo**: Para o padrão $P = \{teste\}$, os valores da tabela são:
  - $d[\text{"}t\text{"}] = 2,$
    $d[\text{"}e\text{"}] = 1,$
    $d[\text{"}s\text{"}] = 3;$
  - $d[x] = 6$ (valor de $m + 1$) para todo caractere x do texto que não faça parte do padrão

  Podemos notar uma melhora na posição dos caracteres através do acréscimo de uma unidade em seu valores (com exceção de um único caractere). Com isto, é perceptível uma redução de colisões no BMHS quando comparado ao BMH em alguns casos (BMHS se destaca quando executado em textos grandes)

Autômatos

Para se ter o entendimento melhor do algoritmo shift-and, será feita uma breve apresentação do conceito de autômatos. Um autômato, neste cenário, serve para reconhecer a cadeia ou padrões em um determinado texto.

O autômato é , em suma, um modelo de computação muito simples com as seguintes características:

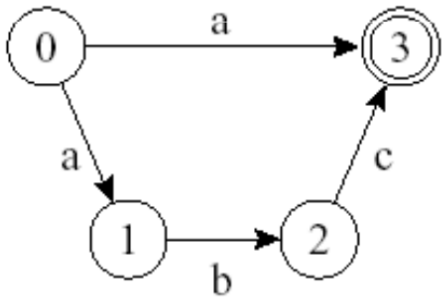
- Como exemplo, considere um autômato finito que é definido pela tupla $(Q, I, F, \sum, T)$, onde:
  - Q: Conjunto finito de estados
  - I: Estado Inicial $(I \in Q)$
  - F: Conjunto de alfabeto finito de entrada
  - $\sum$: Alfabeto finito de entrada
  - T: Função que define as transições entre estados

- T associa cada estado $q \in Q$ um conjunto de estados $\{q_1, q_2, ..., q_k\} \subseteq Q$ para cada $\alpha \in (\sum \cup \{\epsilon\})$ onde ϵ é a transição vazia

---

Autômato Finito Não-Determinista

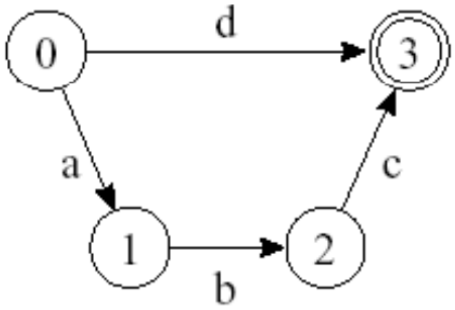
Ocorre quando a função T possibilita a associação de um estado q e um caractere α para mais de um estado do autômato (ou seja, $T(q, \alpha) = \{q_1, q_2, ..., q_k\}$ para $k > 1$), ou quando existe alguma transição rotulada por ϵ



No exemplo, a partir do estado 0, por meio do caractere de transição a, é possível atingir os estados 2 e 3.

Autômato Finito Determinista:

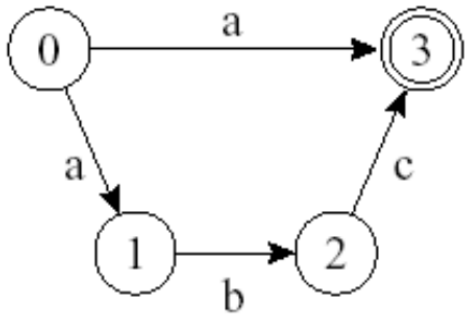
Ocorre quando a função T possibilita a associação de um estado q e um caractere α para apenas um estado do autômato, ou seja, $T(q, \alpha) = \{q_1\}$



Para cada caractere de transição, todos os estados levam a um único estado.

Uma cadeia é reconhecida pelo autômato $(Q, I, F, \sum, T)$ se o mesmo rotula um caminho, que vai do estado inicial até o um estado final, compreendendo a cadeia em questão. A linguagem reconhecida por um autômato é o conjunto de cadeias que o autômato é capaz de reconhecer, por exemplo:

- A linguagem reconhecida pelo autômato abaixo é o conjunto formado pelas cadeias $\{a\}$ e $\{abc\}$



Transições

Transições vazias são transições rotuladas com a cadeia vazia ε, chamadas de transições-ε em autômatos não-deterministas

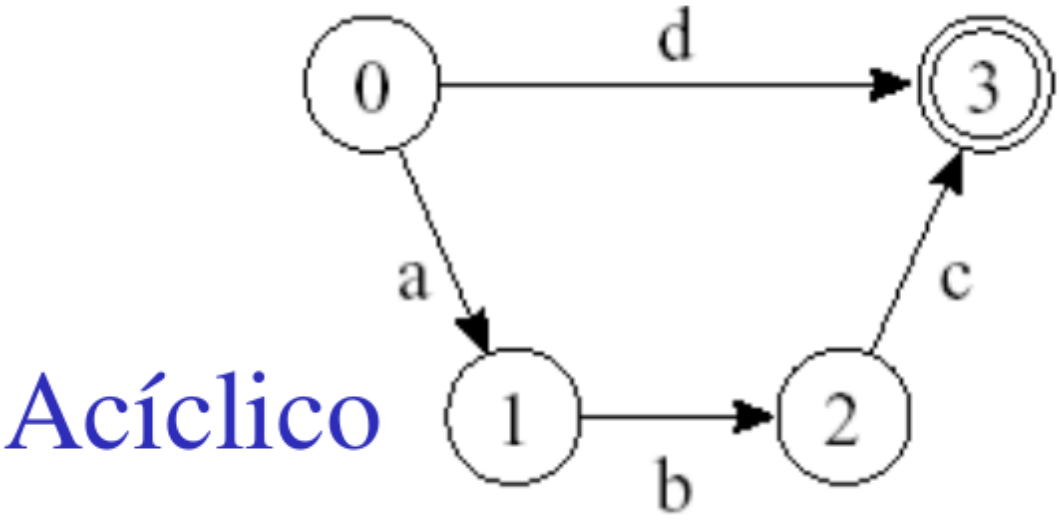
- Não há necessidade de se ler um caractere do alfabeto para se caminhar por meio de uma transição vazia
- As transições vazias simplificam a construção do autômato
- Sempre existe um autômato equivalente que reconhece a mesma linguagem sem transições vazias

Se uma cadeia x rotula um caminho de I até um estado q, então o estado q é considerado ativo depois de ler x

- Um autômato finito determinista possui, no máximo, um estado ativo em um determinado instante
- Um autômato finito não-determinista pode ter vários estados ativos em um determinado instante
    - Casamento aproximado de cadeias pode ser resolvido por meio de autômatos finitos não-deterministas

Autômatos Acíclicos

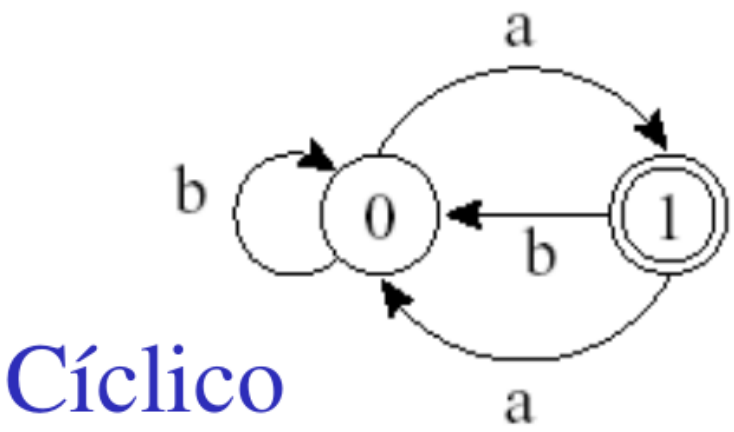
Autômatos acíclicos são aqueles cujas transições não formam ciclos



Autômatos Cíclicos

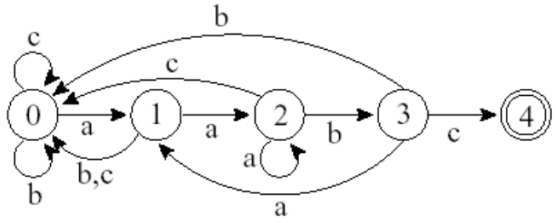
Autômatos cíclicos são aqueles que formam ciclos.

- Este tipo de autômatos (finitos cíclicos), são úteis para casamento de expressões regulares (regex).
- A linguagem reconhecida por eles pode ser infinita, por exemplo:
  - O autômato a seguir reconhece $\{ba\}, \{bba\}, \{bbba\}, \{bbbba\}, ...$



Exemplo de Autômato junto ao Texto

O autômato reconhece $P = \{aabc\}$



A pesquisa de P sobre um texto T com alfabeto Σ ={a, b, c} pode ser vista como a simulação do autômato na pesquisa P sobre T

1. No ínicio, cada estado ativo é o estado inicial 0
2. Para cada caractere lido do texto, a aresta correspondente a partir do estado ativo é seguida, ativando o estado destino
3. Se o estado 3 estiver ativo e um caractere c é lido, o estado final se torna ativo (ou seja, o padrão foi encontrado), resultando em um casamento de $\{aabc\}$ com o texto

Como cada caractere do texto é lido uma vez, a complexidade de tempo é $O(n)$. A complexidade de espaço é $O(m+1)$ para vértices e $O(|\sum| * m)$ para arestas

Algoritmos (Segunda Parte)

Algoritmo *Shift-And* Exato

> ❗ Para entender de maneira apropriada este algoritmo, é necessário realizar a leitura do tópico de Autômatos!

O algoritmo Shift-And é responsável por utilizar o conceito de paralelismo de bit. Este técnica tira proveito de paralelismo das operações sobre bits dentro de uma palavra de computador, sendo possível empacotar muitos valores em uma única palavra e atualizar todos estes valores em uma só atualização

Uma sequência de bits $(b_1 ... b_c)$ é chamada de mascára de bits de comprimento c, e é armazenada em alguma posição de uma palavra w no computador.

É importante destacar algumas operações possíveis sobre os bits de uma palavra:

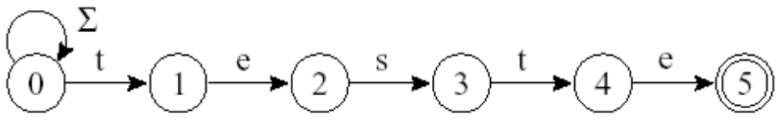
- Repetição de bits: exponenciação
  - Exemplo: $01^3 = 0111$
- Operador lógico **or**: ”$|$”
- Operador lógico **and**: ”$\&$”
- Operador que move bits para a direita e entra com zeros à esquerda: ” $>>$ ”
  - Exemplo: $b_1\ b_2\ ...\ b_{c-1}\ b_c >> 2 = 00 b_1\ ...\ b_{c-2}$

O algoritmo Shift-And irá manter o conjunto de todos os prefixos do padrão P que casam com o texto já lido. Este conjunto será representado por uma mascára de bits $R = (b_1 b_2 ... b_m)$. O Algoritmo utiliza o paralelismo de bit previamente mencionado para atualizar esta máscara a cada caractere lido do texto

Relação com Autômatos

Este algoritmo corresponde à simulação de um autômato não-determinista que pesquisa pelo padrão P no texto T

- Exemplo: Autômato que reconhece os prefixos de $P = \{teste\}$



O valor 1 é colocado na j-ésima posição da máscara de bits se e somente se $(p_1 ... p_j)$ é um sufixo de $(t_1 ... t_i)$, onde i corresponde à posição corrente no texto.

- Nesse caso, a j-ésima posição de R é dita estar ativa
- b_m ativo significa um casamento exato do padrão (fim do autômato)

R^1 (novo valor da máscara de bits) é calculado na leitura do próximo caractere t_{i+1} do texto

- A posição $j + 1$ em R^1 ficará ativa se e somente se a posição j estiver ativa em R, ou seja, $(p_1 ... p_j)$ é sufixo de $(t_1 ... t_i)$, e t_{i+1} casa com p_{j+1}
- Com o uso de paralelismo de bit, é possível computar a nova máscara com custo $O(1)$

Pré-processamento

O pré-processamento se dá pelos seguintes passos:

- Construção de uma tabela M que armazena uma máscara de bits para cada caractere do padrão P
  - 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

A máscara em $M[t]$ é 10010, pois o caractere t aparece nas posições 1 e 4 do padrão P

Regras do Algoritmo

- Algoritmo:
  - A máscara de *bits* R é inicializada como $R = 0^m$.
  - Para cada novo caractere t_{i+1} lido do texto, o valor da máscara R' é atualizado pela expressão:

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

    - A operação (R >> 1) desloca as posições para a direita no passo i+1 para manter as posições de P que eram sufixos no passo i.
    - A operação $((R >> 1) \mid 10^{m-1})$ retrata o fato de que a cadeia vazia ε é também marcada como um sufixo do padrão, permitindo um casamento em qq posição corrente do texto.
    - Para se manter apenas as posições que t_{i+1} casa com p_{j+1}, é realizada a conjunção (operador &) entre $((R >> 1) \mid 10^{m-1})$ e a máscara M relativa ao caractere lido do texto.

Exemplo de Funcionamento

- Exemplo de funcionamento do algoritmo:
  - Pesquisa do padrão P ={teste} no texto T ={os testes ...}.

Texto	$(R >> 1)\|10^{m-1}$					R'				
o	1	0	0	0	0	0	0	0	0	0
s	1	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
t	1	0	0	0	0	1	0	0	0	0
e	1	1	0	0	0	0	1	0	0	0
s	1	0	1	0	0	0	0	1	0	0
t	1	0	0	1	0	1	0	0	1	0
e	1	1	0	0	1	0	1	0	0	1
s	1	0	1	0	0	0	0	1	0	0
	1	0	0	1	0	0	0	0	0	0

Casamento exato

Código Exemplo (Portugol)

Shift–And $(P = p_1 p_2 \cdots p_m, T = t_1 t_2 \cdots t_n)$

```
{ /*—Préprocessamento—*/
  for (c∈ Σ) M[c] = 0^m;
  for (j = 1; j <= m; j++) M[p_j] = M[p_j] |0^{j-1}10^{m-j};
  /*—— Pesquisa ——*/
  R = 0^m;
  for (i = 1; i <= n; i++)
    { R = ((R >> 1|10^{m-1}) & M[T[i]]);
      if ( R & 0^{m-1}1 ≠ 0^m) 'Casamento na posicao i − m + 1';
    }
}
```

Análise do Algoritmo

O custo do algoritmo Shift-And é $O(n)$, desde de que as operações sobre os bits possam ser realizadas em tempo linear $O(1)$ e o padrão caiba em poucas palavras do computador que

estiver executando o algoritmo.

Algoritmo *Shift-And* Aproximado

> ❗ Este algoritmo trabalha com o conceito de casamento aproximado, conceito este mencionado anteriormente neste mesmo PDF.

O algoritmo Shift-And aproximado simula um autômato não-determinista, utilizando 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

Para cada novo caractere lido do texto, todas as transições do autômato são simuladas usando operações entre as $k + 1$ máscaras de bits:

- R_0 (casamento exato), R_1 (um erro), R_2 (dois erros), ..., R_k (k erros)

Regras do Algoritmo

- Algoritmo:
  - A máscara R_0 (casamento exato) é inicializada como $R_0 = 0^m$.
  - Para $0 < j \leq k$, Rj é inicializada como R$j = 1^j 0^{m-j}$.
  - Considerando M a tabela do algoritmo *Shift-And* para casamento exato, para cada novo caractere t_{i+1} lido do texto, as máscaras são atualizadas pelas expressões:
    - $R'_0 = ((R_0 >> 1) \mid 10^{m-1}) \ \& \ M[T[i]]$
    - Para $0 < j \leq k$,
      $R'_j = ((R_j >> 1) \ \& \ M[T[i]]) \mid R_{j-1} \mid (R_{j-1} >> 1) \mid (R'_{j-1} >> 1) \mid 10^{m-1}$
  - 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 ($R_{j-1} >> 1$);
    - diagonais tracejadas, indicando retirada ($R'_{j-1} >> 1$).

Exemplo de Funcionamento

- Exemplo de funcionamento do algoritmo:
  - Pesquisa do padrão P ={teste} no texto T ={os testes testam}.
  - Possibilidade de um erro de inserção (k = 1).

- Expressões para atualização das máscaras de bits:
  - $R'_0 = ((R_0 >> 1) \mid 10^{m-1}) \ \& \ M[T[i]]$
  - $R'_1 = ((R_1 >> 1) \ \& \ M[T[i]]) \mid R_0 \mid 10^{m-1}$

Texto	$(R_0 >> 1) \mid 10^{m-1}$	R'_0	$R_1 >> 1$	R'_1
o	1 0 0 0 0	0 0 0 0 0	0 1 0 0 0	1 0 0 0 0
s	1 0 0 0 0	0 0 0 0 0	0 1 0 0 0	1 0 0 0 0
	1 0 0 0 0	0 0 0 0 0	0 1 0 0 0	1 0 0 0 0
t	1 0 0 0 0	1 0 0 0 0	0 1 0 0 0	1 0 0 0 0
e	1 1 0 0 0	0 1 0 0 0	0 1 0 0 0	1 1 0 0 0
s	1 0 1 0 0	0 0 1 0 0	0 1 1 0 0	1 1 1 0 0
t	1 0 0 1 0	1 0 0 1 0	0 1 1 1 0	1 0 1 1 0
e	1 1 0 0 1	0 1 0 0 ***1***	0 1 0 1 1	1 1 0 1 ***1***
s	1 0 1 0 0	0 0 1 0 0	0 1 1 0 1	1 1 1 0 ***1***
	1 0 0 0 0	0 0 0 0 0	0 1 1 1 0	1 0 1 0 0
t	1 0 0 0 0	1 0 0 0 0	0 1 0 1 0	1 0 0 1 0
e	1 1 0 0 0	0 1 0 0 0	0 1 0 0 1	1 1 0 0 ***1***
s	1 0 1 0 0	0 0 1 0 0	0 1 1 0 0	1 1 1 0 0
t	1 0 0 1 0	1 0 0 1 0	0 1 1 1 0	1 0 1 1 0
a	1 1 0 0 1	0 0 0 0 0	0 1 0 1 1	1 0 0 1 0
m	1 0 0 0 0	0 0 0 0 0	0 1 0 0 1	1 0 0 0 0

Casamento exato

Casamento aproximado

Casamento aproximado

Códigos Exemplo (Portugol e C)

```
void Shift-And-Aproximado (P = p1 p2 ... pm, T = t1 t2 ... tn, k)
{ /*-- Préprocessamento --*/
  for (c ∈ Σ) M[c] = 0^m;
  for (j = 1; j <= m; j++)  M[pj] = M[pj] | 0^(j-1) 10^(m-j);
  /*-- Pesquisa --*/
  for (j = 0; j <= k; j++) Rj = 1^j 0^(m-j);
  for (i = 1; i <= n; i++)
    { Rant = R0;
      Rnovo = ((Rant >> 1) | 10^(m-1)) & M[T[i]];
      R0 = Rnovo;
      for (j = 1; j <= k; j++)
        { Rnovo = ((Rj >> 1 & M[T[i]]) | Rant | ((Rant | Rnovo) >> 1));
          Rant = Rj;
          Rj = Rnovo | 10^(m-1);
        }
      if (Rnovo & 0^(m-1) 1 ≠ 0^m) 'Casamento na posicao i';
    }
}
```

Código em Portugol

```
void ShiftAndAproximado(TipoTexto T, long n, TipoPadrao P, long m, long k)
{ long Masc[MAXCHAR], i, j, Ri, Rant, Rnovo;
  long R[NUMMAXERROS + 1];
  for (i = 0; i < MAXCHAR; i++) Masc[i] = 0;
  for (i = 1; i <= m; i++) { Masc[P[i-1] + 127] |= 1 << (m - i); }
  R[0] = 0;    Ri = 1 << (m - 1);
  for (j = 1; j <= k; j++) R[j] = (1 << (m - j)) | R[j-1];
  for (i = 0; i < n; i++)
    { Rant = R[0];
      Rnovo = ((((unsigned long)Rant) >> 1) | Ri) & Masc[T[i] + 127];
      R[0] = Rnovo;
      for (j = 1; j <= k; j++)
        { Rnovo = ((((unsigned long)R[j]) >> 1) & Masc[T[i] + 127])
                  | Rant | (((unsigned long)(Rant | Rnovo)) >> 1);
          Rant = R[j];    R[j] = Rnovo | Ri;
        }
      if ((Rnovo & 1) != 0) printf(" Casamento na posicao %12ld\n", i + 1);
    }
}
```

Código em C