Universidade Federal
de Ouro Preto

ENYA LUÍSA GOMES DOS SANTOS
19.2.4201

RESUMO III - CASAMENTO DE CADEIAS

Resumo apresentado por exigência da disciplina BCC203 - Estrutura de Dados II, da Universidade Federal de Ouro Preto.
Professor: Guilherme Tavares Assis

Enya Luísa Gomes dos Santos
19.2.4201

CASAMENTO DE CADEIAS

O que é?

Segue a ideia de localizar, de maneira eficiente, uma cadeia de caracteres, chamado de padrão, dentro de um texto ou dentro de um volume grande de textos. Existe o casamento exato e o casamento aproximado de cadeias.

É de extrema importância a definição do alfabeto para estratégia de casamento de cadeias.

Exemplos de aplicações: edição de texto, recuperação de informação, estudo de sequência de DNA em biologia computacional.

Formalização do problema:

- Texto: cadeia T[0..n-1] de tamanho n.
- Padrão: cadeia P[0..m-1] de tamanho m ≤ n.
- Os elementos de P e T são escolhidos de um alfabeto finito Σ de tamanho c. Exemplos: Σ = {0, 1} ou Σ = {a, b, ... , z}.
- Casamento de cadeias: dadas as cadeias P (comprimento m) e T (comprimento n), onde m ≤ n, deseja-se saber as ocorrências de P em T.

Estrutura de dados:

```
#define MAXTAMTEXTO 1000
#define MAXTAMPADRAO 10
#define MAXCHAR 25
#define NUMMAXERROS 10
typedef char TipoTexto[MAXTAMTEXTO];
typedef char TipoPadrao[MAXTAMPADRAO];
```

Existem três tipos de categorias de algoritmos, sendo eles:

- P e T não são pré-processados:
  Varre todo o padrão a cada caractere do texto. Logo a complexidade de tempo é **O(*mn*)** e complexidade de espaço é *O(1)*, pois como não há um pré-processamento, não exige uma memória extra.

```
T(n) = aaaaaaaaaaab
```

Enya Luísa Gomes dos Santos
19.2.4201

```
P(n) = aaab
```

- P pré-processado:

  Evita a varredura de todos os caracteres do padrão. Logo a complexidade se faz **O(*n*)**, já a complexidade de espaço é **O(*m*+*c*)**.

  ```
  T(n) = aaaaaaaaaaaab
  P(n) = aaab
  P'(n) = *padrão pré-processado*
  ```

- P e T são pré-processados:

  Para aplicar esse tipo de algoritmo é necessário ter conhecimento do texto. A complexidade de pesquisa depende do método utilizado para pré-processar o texto e a complexidade de espaço geralmente é O(n).

  ```
  T(n) = os testes testam
  P(n) = teste
  T'(n) = *texto pré-processado* Exemplo: criar uma árvore B com o texto. Nesse caso a complexidade se torna O(log n).
  P'(n) = *padrão pré-processado*
  ```

  Alguns tipos de índices são:

  - Arquivos invertidos.
  - Árvores TRIE e árvores PATRICIA.
  - Arranjos de sufixos.

ARQUIVOS INVERTIDOS

É uma estrutura composta por duas partes, vocabulário e ocorrências.

Vocabulário: todas as palavras que formam parte do vocabulário do texto.

Ocorrência: Lista de posições onde as palavras aparecem no texto.

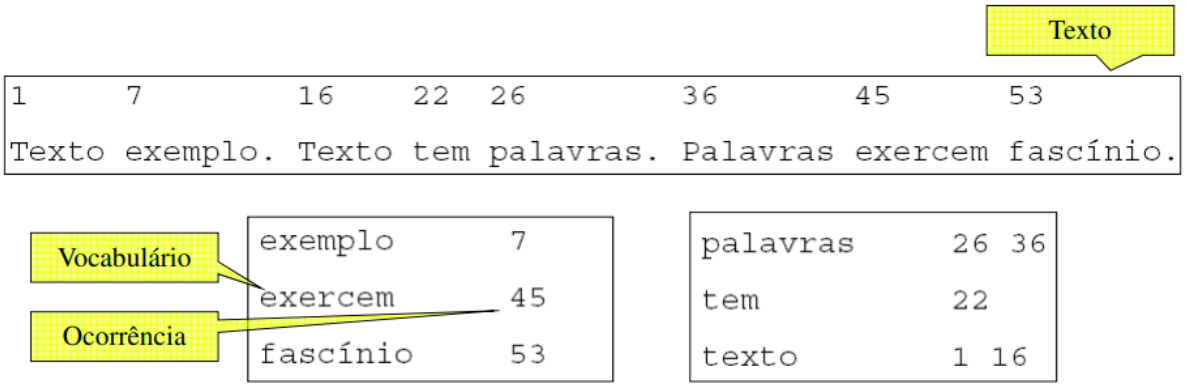

Figura 1 - Representação do arquivo invertido

Enya Luísa Gomes dos Santos
19.2.4201

A maneira de implementar um arquivo invertido vai de escolha do desenvolvedor. Por exemplo, usar uma árvore B, ou um hash, ou outros.

De modo geral as ocorrências ocupam mais espaço na memória em relação aos vocabulários, pois, imaginando uma quantidade *n* muito grande de documentos, cada palavra gera apenas um vocabulário já às ocorrências, ou seja, quantas vezes ela aparecerá nos n documentos, podem ser muitas ocorrências. Geralmente o vocabulário cabe em memória principal, já as ocorrências não.

A pesquisa é realizada em três passos:

1. Pesquisa no vocabulário: as 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.

Lembrando que o tempo de busca pela palavra depende da estrutura de dados utilizada.

A pesquisa por frases usando índices é mais difícil.

Uma das formas de representar o vocabulário do arquivo invertido é em cima de uma árvore digital, ou árvore TRIE. Como apresentado no exemplo abaixo:

Texto: “Texto tem palavras. Palavras exercem fascínio.”

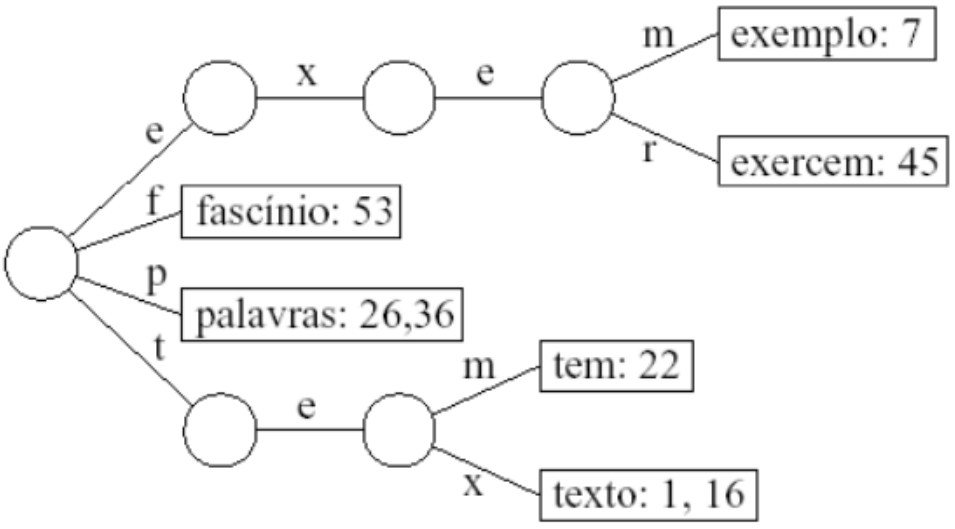

Figura 2 - Árvore TRIE. O vocabulário lido do texto é colocado em uma árvore TRIE, armazenando uma lista de ocorrências para cada palavra.

Cada nó folha guarda o vocabulário e as ocorrências, já os nós internos servem para direcionar a pesquisa. A árvore TRIE é formada pelos caracteres constituidores de uma palavra. Essa estrutura tem uma complexidade O(m) onde m é o tamanho do padrão.

ALGORITMOS BM, BMH E BMHS

Chamados de Boyer-Moore (BM), Boyer-Moore-Horspool (BMH) e Boyer-Moore-Horspool-Sunday (BMHS).

São algoritmos onde o padrão é pré-processado, a pesquisa do padrão P acontece de forma a utilizar uma janela que desliza ao longo do texto, procurando por um sufixo de janela que saca com o sufixo de P, ou seja, a pesquisa ocorre de trás para frente.

ALGORITMO BOYER-MOORE (BM)

Sobre o funcionamento

- Pesquisa o padrão P em uma janela que desliza ao longo do texto T.
- Para cada posição desta janela, o algoritmo pesquisa por um 16 sufixo da mesma que casa com um sufixo de P, com comparações realizadas no sentido da direita para a esquerda.
    - Se não ocorrer uma desigualdade, uma ocorrência de P em T foi localizada.
    - Caso contrário, o algoritmo calcula um deslocamento que a janela deve ser deslizada para a direita antes que uma nova tentativa de casamento se inicie.

Processo da janela

A janela virtual possui o tamanho do padrão, a janela desliza da esquerda para a direita e a comparação é feita da direita para a esquerda. No exemplo abaixo, primeiro se compara C com C, em seguida, como são iguais, A com A, C com B,

Enya Luísa Gomes dos Santos
19.2.4201

nesse caso como C é diferente de B, logo o padrão não foi localizado dentro da janela e a janela é caminhada para a direita e uma posição, e o processo se repete.

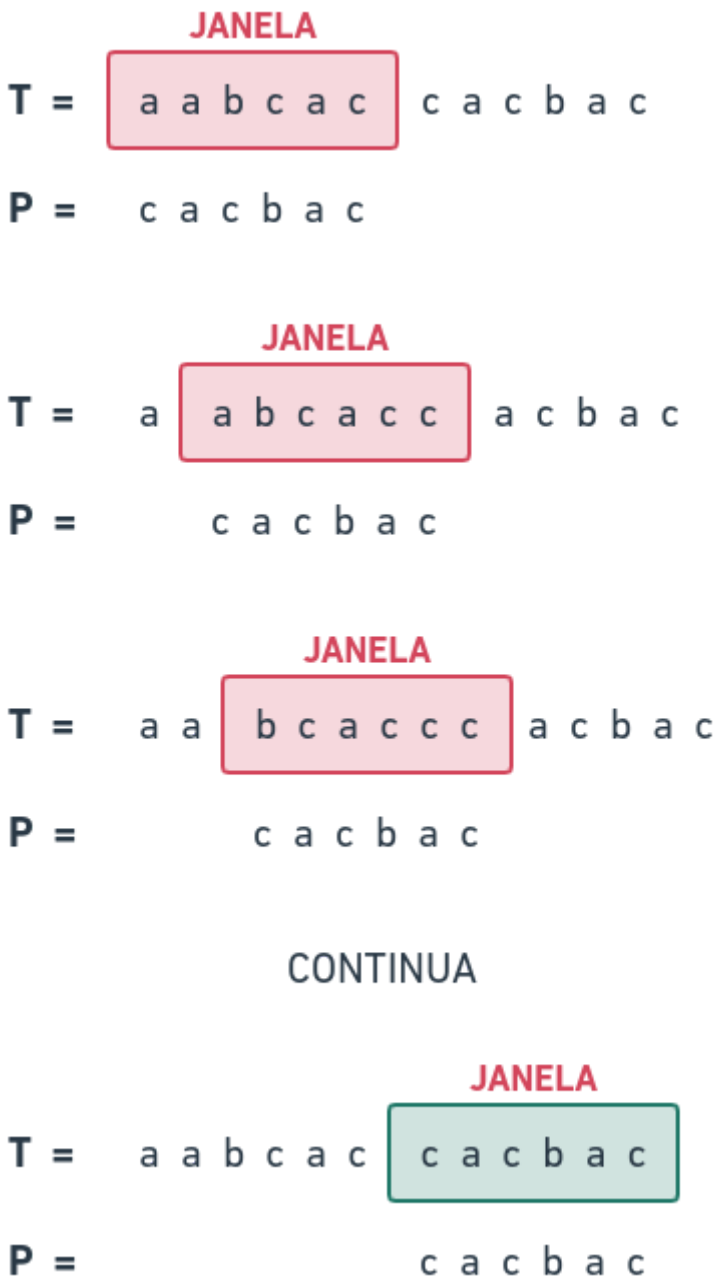

Figura 3 - Exemplificação do funcionamento da janela virtual do método BM

Pode-se perceber que esse método segue a ideia do método da força bruta, para que isso não aconteça, é necessário deslizar a janela não somente de um em um e existem duas heurísticas pelo método BM para isso, são elas ocorrência e casamento. É possível utilizar qualquer uma delas.

A heurística ocorrência alinha o caractere no texto que causou a colisão com o 1º caractere no padrão, à esquerda do ponto de colisão, que casa com ele. Ou seja, alinha a posição que causou a colisão no texto com o primeiro caractere no padrão que casa com este caractere.

Ex.: P ={cacbac}, T ={aabcaccacbac}.

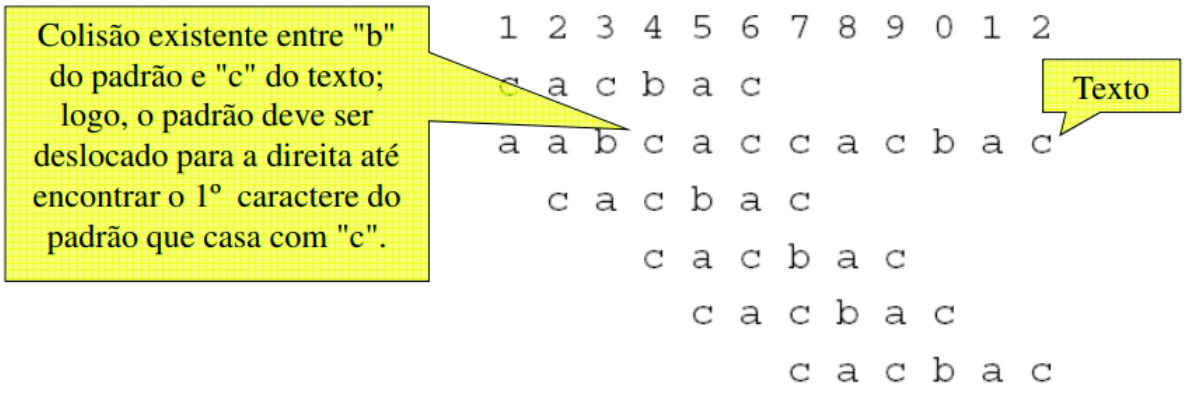

Figura 4 - Representação da heurística de ocorrência

Caso o caractere que gerou a colisão não esteja no restante do padrão, o padrão é empurrado por inteiro a partir do caractere da colisão.

Quanto maior o tamanho do alfabeto melhor será o algoritmo de BM com a ocorrência, em relação a força bruta, pois os deslocamentos da janela serão maiores.

A heurística casamento faz com que, ao mover o padrão para a direita, a janela em questão casa com o pedaço do texto anteriormente casado. Ou seja, ao mover o padrão para a direita, faça-o casar com o pedaço do texto anteriormente casado.

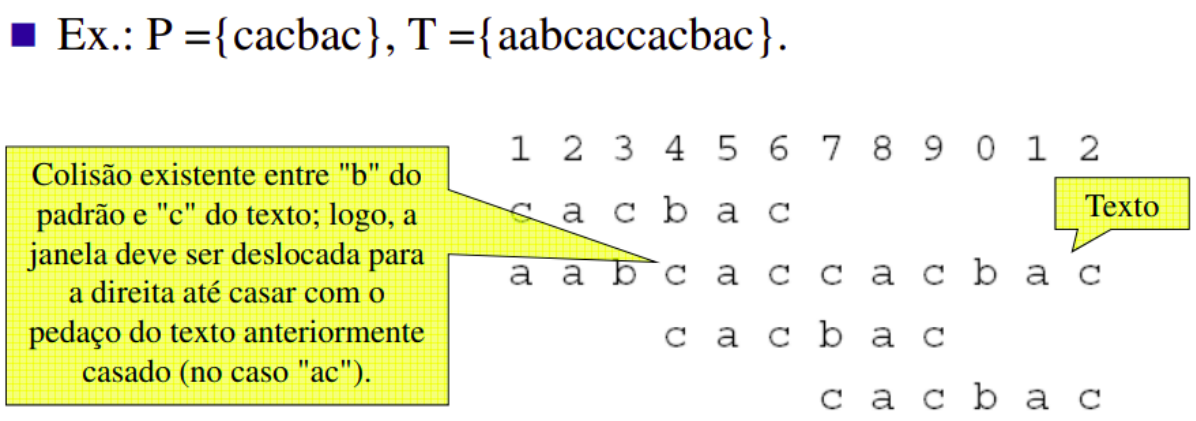

Figura 5 - Representação da heurística de casamento

Caso haja a colisão logo de cara, ou seja, na primeira comparação, não há o que buscar no padrão, o padrão será deslocado apenas uma posição, o que será o pior caso. O que faz com que esse heurística não seja muito legal para uma quantidade grande de alfabeto

De modo geral, a heurística ocorrência apresenta melhores resultados, principalmente quando se trata de um algoritmo com maior quantidade de elementos.

ALGORITMO BOYER-MOORE-HORSPOOL (BMH)

Simplificação do algoritmo BM, o que fez com que o algoritmo BMH fosse mais eficiente em relação ao BM. Essa simplificação diz respeito ao descolamento da janela.

Com a geração de uma **tabela de deslocamento**, para cada caractere do alfabeto é estabelecido um valor numérico, 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.

Definição da tabela

Existem duas regras:

1 - O valor inicial do deslocamento para todos os caracteres do alfabeto é igual a m, onde m é o tamanho do padrão.

2 - Logo, para os m-1 primeiro caracteres do padrão P, os valores do deslocamento são calculados pela regra: **d[x] = min{j tal que (j = m) | (1 ≤ j < m & P[m-j] = x)}.** Informalmente falando, d[x] diz respeito a para cada caractere do padrão, qual a distância desse caractere em relação ao último caractere do padrão.

Exemplo:

Imaginando que o padrão seja `P={teste}`:

- Todos os caracteres do meu alfabeto serão inicializados com valor 5, pois esse é o tamanho do padrão.
- Calculando para os m-1 primeiro caracteres a distância em relação ao último caractere do padrão `d["t"] = min(d["t"] = 4, d["t"] = 1), ou seja,` **`d["t"] = 1, d["e"] = 3, d["s"] = 2;`**
- d[x] = 5 (valor de m) para todo caractere x do texto que não faça parte do padrão.

Exemplificação:

Considerando o alfabeto *a b c d* temos o seguinte cenário:

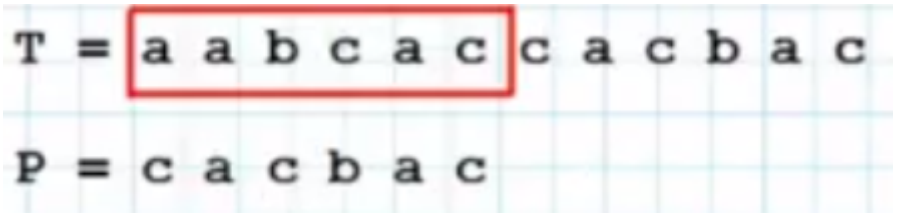

Figura 6 - Casamento de Cadeias com BMH

Inicialmente tem-se a seguinte tabela de descolamento:

```
d["a"] =  6
d["b"] =  6
d["c"] =  6
d["d"] =  6
```

Onde 6 é o tamanho do padrão **P = c a c b a c**

Agora analisa-se os **m - 1** primeiros caracteres do padrão, onde essa análise se trata da distância mínima em relação ao último caractere:

```
d["a"] =  min(d["a"]=4;d["a"]=1)  =  1
d["b"] =  2
d["c"] =  min(d["c"]=5;d["c"]=3)  =  3
```

Enya Luísa Gomes dos Santos
19.2.4201

```
d["d"] =  6
```

Agora, tenta-se fazer o casamento dos caracteres, no exemplo, ocorre a colisão entre o ***c*** e ***b,*** logo pega-se o caractere do texto alinhado com o último padrão, no exemplo é o ***c***, sabemos que `d["c"] = 3`, portanto, a janela se desloca 3 posições, como mostrado abaixo.

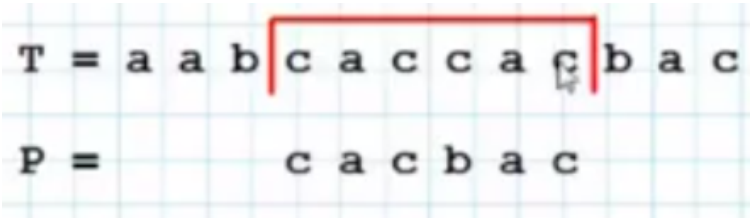

Figura 7 - Deslocamento da janela

Logo o processo se repete até encontrar o padrão no texto.

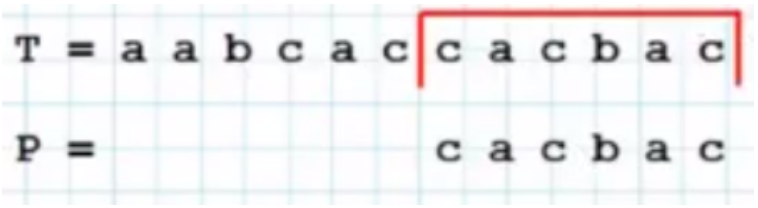

Figura 8 - Deslocamento da janela

A ideia é alinhar o caractere que causou o deslocamento com um caractere no padrão que case com ele, e caso o caractere não exista dentro do padrão, a janela descola o tamanho m deste padrão.

```
void BMH(TipoTexto T, long n, TipoPadrao P, long m)
{ long i, j, k, d[MAXCHAR + 1];
  for (j = 0; j <= MAXCHAR; j++) d[j] = m;
  for (j = 1; j < m; j++) d[P[j-1]] = m - j;
  i = m;
  while (i <= n)      /*-- Pesquisa --*/
    { k = i;
      j = m;
      while (T[k-1] == P[j-1] && j > 0) { k--; j--; }
      if (j == 0)
      printf(" Casamento na posicao: %3ld\n", k + 1);
      i += d[T[i-1]];
    }
}
```

Pré-processamento para se obter a tabela de deslocamentos

Pesquisa por um sufixo do texto (janela) que casa com um sufixo do padrão

Deslocamento da janela de acordo com o valor da tabela de deslocamentos relativo ao caractere que está na i-ésima-1 posição do texto, ou seja, a posição do último caractere do padrão *P* (Horspool).

Figura 9 - Algoritmo referente ao BMH

ALGORITMO BOYER-MOORE-HORSPOOL-SUNDAY (BMHS)

Simplificação importante para o algoritmo BMH. Sunday propõe 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**. Ou seja, o último caractere do padrão é considerado e não desconsiderado com no algoritmo de BMHS.

Definição da tabela

Existem duas regras:

1 - O valor inicial do deslocamento para todos os caracteres do alfabeto é igual a m + 1, onde m é o tamanho do padrão.

2 - Logo, para os m primeiro caracteres do padrão P, os valores do deslocamento são calculados pela regra: **d[x] = min{j tal que (j = m+1) | (1 ≤ j ≤ m & P[m+1-j] = x)}.** Informalmente falando, d[x] diz respeito a para cada caractere do padrão, qual a distância desse caractere em relação ao caractere **seguinte** ao último do padrão, ou seja, em relação ao `\0`.

Exemplo:

Imaginando que o padrão seja `P={teste}`:

- Todos os caracteres do meu alfabeto serão inicializados com valor 6, pois esse é o tamanho do padrão + 1.
- Calculando para os m primeiro caracteres a distância em relação ao último caractere do padrão `d["t"] = min(d["t"] = 5, d["t"] = 2), ou seja, **d["t"] = 2,** d["e"] = min(d["e"] = 4, d["e"] = 1), ou seja, **d["e"] = 1, d["s"] = 3;**`
- d[x] = 6 (valor de m + 1) para todo caractere x do texto que não faça parte do padrão.

Exemplificação:

Considerando o alfabeto *a b c d*. Teremos a tabela:

```
d["a"] =  2
d["b"] =  3
d["c"] =  1
d["d"] =  7
```

Enya Luísa Gomes dos Santos
19.2.4201

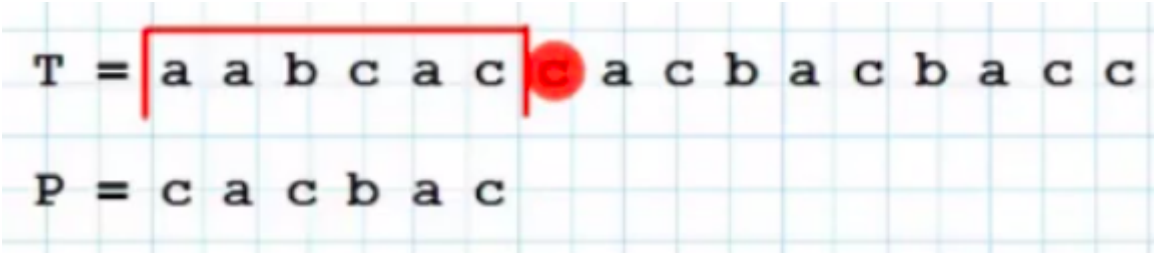

Figura 10 - Casamento e cadeia com BMHS

Na figura acima, como ocorreu a colisão em `b` e `c` a janela é deslocada em relação ao caractere seguinte ao último da janela, que nesse caso é o `c` marcado com uma bola vermelha. E o processo se repete.

```
void BMHS(TipoTexto T, long n, TipoPadrao P, long m)
{ long i, j, k, d[MAXCHAR + 1];
  for (j = 0; j <= MAXCHAR; j++) d[j] = m + 1;
  for (j = 1; j <= m; j++) d[P[j-1]] = m - j + 1;
  i = m;
  while (i <= n) /*-- Pesquisa --*/
    { k = i;
      j = m;
      while (T[k-1] == P[j-1] && j > 0) { k--; j--; }
      if (j == 0)
      printf(" Casamento na posicao: %3ld\n", k + 1);
      i += d[T[i]];
    }
}
```

Pré-processamento para se obter a tabela de deslocamentos

Pesquisa por um sufixo do texto (janela) que casa com um sufixo do padrão

Deslocamento da janela de acordo com o valor da tabela de deslocamentos relativo ao caractere que está na i-ésima posição do texto, ou seja, a posição seguinte ao último caractere do padrão *P* (Sunday).

Figura 11 - Algoritmo referente ao BMHS

AUTÔMATOS

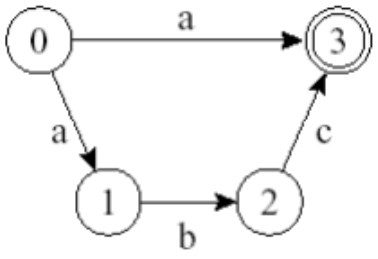

Figura 12 - Exemplo de um autômato

Enya Luísa Gomes dos Santos
19.2.4201

Um autômato finito é definido pela tupla (Q, I, F, Σ, T), onde:

- Q é um conjunto finito de estados;
- I é o estado inicial (I ∈ Q);
- F é o conjunto de estados finais (F ⊆ Q);
- Σ é o alfabeto finito de entrada;
- T é a função que define as transições entre os estados.
    - T associa a cada estado q ∈ Q um conjunto de estados {q1, q2, . . . , qk} ⊆ Q, para cada α ∈ (Σ ∪ {ε}), 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. No exemplo abaixo, a partir do estado 0, por meio do caractere de transição a, é possível atingir os estados 2 e 3

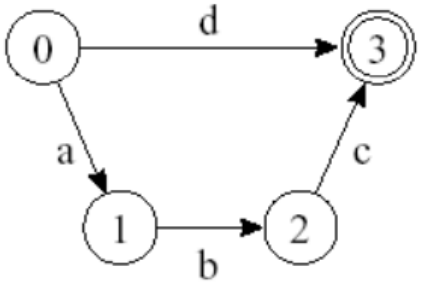

Figura 13 - Exemplo de um autômato finito não-determinista

Autômato finito determinista: ocorre quando a função T permite a associação de um estado *q* e um caractere α para apenas um estado do autômato. Para cada caractere de transição, todos os estados levam a um único estado.

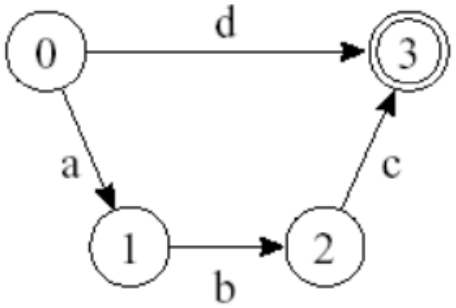

Figura 14 - Exemplo de um autômato finito determinista

Cadeia: um caminho que vai do seu estado inicial até o 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}.

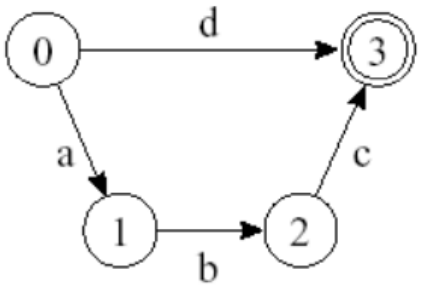

Figura 15 - Exemplo de um autômato

Autômato Cíclico: Um autômato cíclico ocorre quando suas transições formam ciclos. A linguagem reconhecida por um autômato cíclico pode ser infinita. Por exemplo, no autômato abaixo a direita reconhece `{ba}, {bba}, {bbba}, {bbbba}, ...`

Autômato Acíclico: Um autômato acíclico ocorre quando suas transições não formam ciclos.

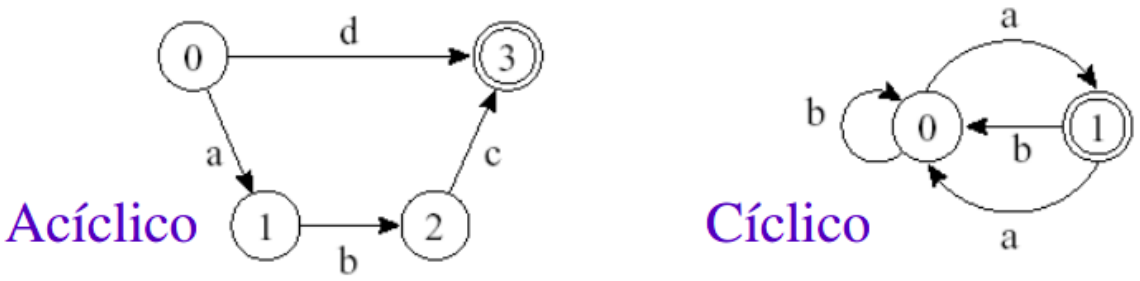

Figura 16 - Exemplo de um autômato acíclico e cíclico

ALGORITMO SHIFT-AND EXATO

É, ainda, um algoritmo que realiza o pré-processamento no padrão. No Shift-And a ideia dos autômatos são usadas para reconhecer os padrões desejados.

Enya Luísa Gomes dos Santos
19.2.4201

Imaginando que iremos representar o padrão `P = {aabc}`, o alfabeto é Σ = `{a, b, c}`; temos o seguinte autômato.

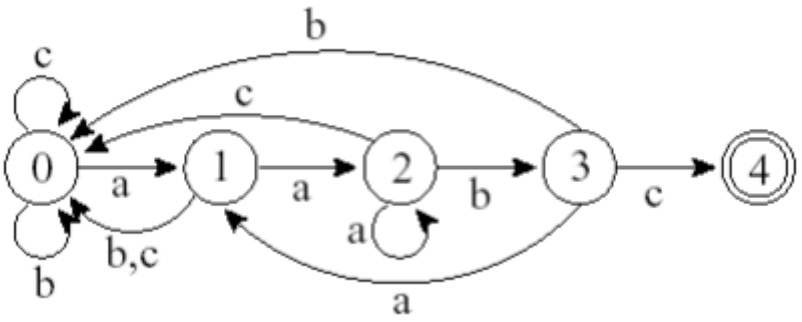

Figura 17 -Representação do autômato do padrão P={aabc}

Como há cinco caracteres no padrão, o autômato terá seis **(m + 1)** vértices/estados , as arestas horizontais representam o padrão `{aabc}`. As demais arestas representam o que irá acontecer à medida que outros caracteres são lidos dentro do texto. Já a quantidade de arestas depende do alfabeto **(|Σ| x m)**.

Em suma, o padrão P é representado por meio de um autômato, onde, conforme percorre-se o texto, há um caminhamento dentro desse autômato.

Analisando o método, se o alfabeto for grande e o padrão também for grande, a construção do autômato pode ser uma construção demorada, o que torna um custo adicional muito grande.

No Shift-End o autômato não é de fato construído, e sim simulado de uma forma mais simples por meio de uma sequência binária de bits.

O Shift-End exato

Usa o conceito de paralelismo de bit. O padrão é representado por meio de um autômato onde esse autômato é uma sequência binária que será manipulada pelas operações sobre os bits, operações como:

- Repetição de bits: exponenciação (ex.: 013 = 0111);
- "|": operador lógico or;
- "&": operador lógico and;
- ">>": operador que move os bits para a direita e insere zeros à esquerda (ex.: b1 b2 ... bc−1 bc >> *2* = **00**b1 ... bc-2).

Tais operações têm ordem de complexidade constante, ou seja, 1.

O padrão P que seria representado em um autômato é representado por uma máscara de bits R = (b1 b2 ... bm). De tamanho m, onde m é o tamanho do padrão. Cada bit representa um estado do autômato.

```
0 -> Estado inativo
1 -> Estado ativo
```

Sabe-se que o padrão foi encontrado quando o último bit de R tiver o valor 1. A máscara é 'atualizada' a cada leitura de um caractere do texto.

Exemplificação:

```
T = os testes testam ...
P = teste
```

A cada leitura de um caractere a máscara R é convertida em outra máscara, R', e verifica em R' se o 1 aparece no final, caso isso aconteça, o padrão foi localizado. Esse processo se repete até encontrar o 1 no final ou o texto finalizar. A transformação da máscara é feita pelas operações de bits.

```
R = 00000 => R' = 00000 => R'' = 00000 => ...
```

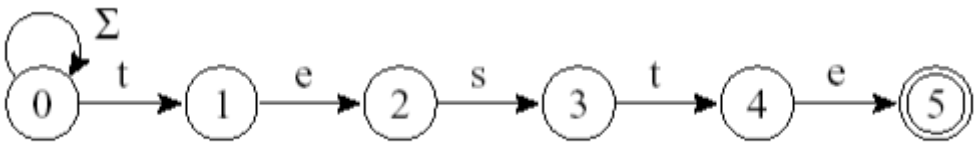

Figura 18 -Representação básica do autômato do padrão P={teste}

Antes, ocorre um pré-processamento, onde se constrói uma tabela M para armazenar uma máscara de bits (b1 b2 ... bm) para cada caractere do padrão, essa máscara independe da máscara R. Para cada caractere do padrão se tem uma máscara. Exemplo: `P = {teste}`

Enya Luísa Gomes dos Santos
19.2.4201

	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

Figura 19 -Representação da máscara de bits M

É colocado o bit 1 se na posição, representada por `1 2 3 4` no exemplo, do padrão o caractere estiver presente. A máscara em M[t] é 10010, pois o caractere t aparece nas posições 1 e 4 do padrão P.

Para os demais caracteres que não existem dentro do padrão a representação em bits será uma sequência de zeros.

Fórmula de transformação da máscara que representa o caminha dentro o autômato:

- A máscara de bits R é inicializada como R = 0^m.
- Para cada novo caractere ti+1 lido do texto, o valor da máscara R' é atualizado pela expressão: **R' = ((R >> 1) | 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) | 10m−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 ti+1 casa com pj+1, é realizada a conjunção (operador &) entre ((R >> 1) | 10m−1) e a máscara M relativa ao caractere lido do texto.

Representação por um exemplo:

```
T = os testes testam ...
P = teste
R = 00000 =>
R' {o} = (00000(R >> 1)  `or`  10000(10^m-1))
       = 1000 `and`  00000(M[T[o]])  =  00000
```

Nesse ponto, quer dizer que o estado ativo do autômato continua sendo o estado inicial.

R' {s} = 00000

R' { } = 00000

R' {t} = 10000 & 10010 = 10000 (o estado indicado pelo 1 é ativo, ou seja, corresponde ao caractere do padrão).

R' {e} = 10000 >> 1 -> 01000 (|) 10000 -> 11000 & 01001 = 01000 (indica que o casamento ocorreu com a letra **t** e a letra **e**).

R' {s} = 01000 >> 1 -> 00100 (|) 10000 -> 10100 & 00100 = 00100 (indica que o casamento ocorreu com a letra **t,e** e **s**).

R' {t} = 00100 >> 1 -> 00010 (|) 10000 -> 10010 & 10010 = 10010 (**t,e,s** e **t,** também temos que há dois estados ativos, o que quer dizer que foi encontrado dois ***t***).

R' {e} = 10010 >> 1 -> 01001 (|) 10000 -> 11001 & 01001 = 01001 (1 está no final, ou seja o último estado está ativo, além disso dois ***e*** foram encontrados).

R' {s} = 01001 >> 1 -> 00100 (|) 10000 -> 10100 & 00100 = 00100

R' { } = 00000 (voltou para o estado inicial).

```
Shift-And (P = p1 p2 ··· pm, T = t1 t2 ··· tn)
{ /*---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 ---*/
  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';
    }
}
```

Figura 20 - Algoritmo do Shift-End exato

Análise: O custo do algoritmo Shift-And é O(n), desde que as operações sobre os bits possam ser realizadas em O(1) e o padrão caiba em umas poucas palavras do computador.

CASAMENTO APROXIMADO

O casamento aproximado de cadeias consiste em encontrar ocorrências aproximadas de um padrão no texto. Permite encontrar ocorrências de cadeias similares aos padrão por meio de operações de **inserção, substituição e retirada** de caracteres **do padrão**.

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.

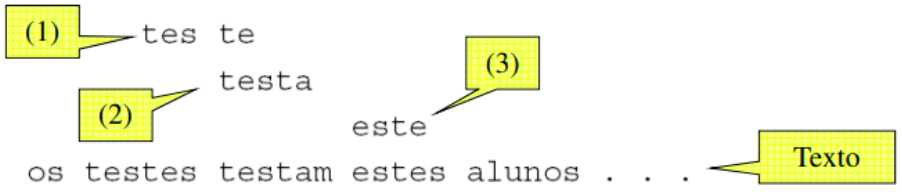

Figura 21 - **Inserção, substituição e retirada** de caracteres **do padrão**.

A **distância de edição** entre duas cadeias P e P' é a quantidade de operações de inserção, substituição e/ou retirada necessárias para converter P em P' ou vice-versa. Como por exemplo: `ed(teste, estende) = 4:` valor obtido por meio da retirada do primeiro ***t*** de P e a inserção dos caracteres ***nde*** ao final de P.

Para casamento aproximado de cadeias há um limite máximo para o valor da distância de edição, esse limite pode ser chamado de ***k*** onde para uma cadeias ser aproximada do pradão $ed(P,P') \leq k$. O valor máximo atribuído a ***k*** deve ser bem pensado, pois caso contrário, se por exemplo k > m as palavras aceitas serão muitas que podem nem ter similaridade com o padrão, logo $0 < k < m$. Um valor interessante para k é que não ultrapasse a metade do tamanho do padrão.

Nível de erro α = k/m. Em geral, α < 1/2 para a maioria dos casos.

Enya Luísa Gomes dos Santos
19.2.4201

A pesquisa com casamento aproximado é modelada por **autômatos não-deterministas** (é possível ter várias estado ativos no mesmo instante de tempo). Os algoritmos usam paralelismo de bit.

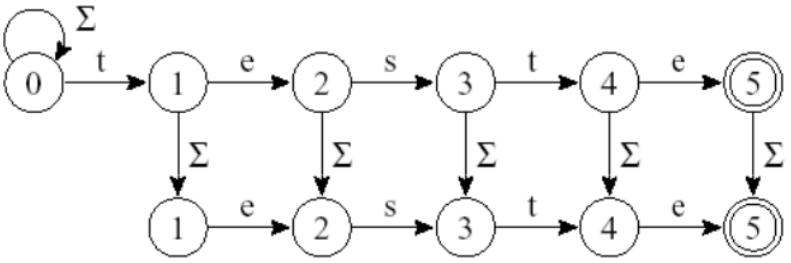

Figura - Autômato que reconhece P ={teste}, permitindo uma inserção

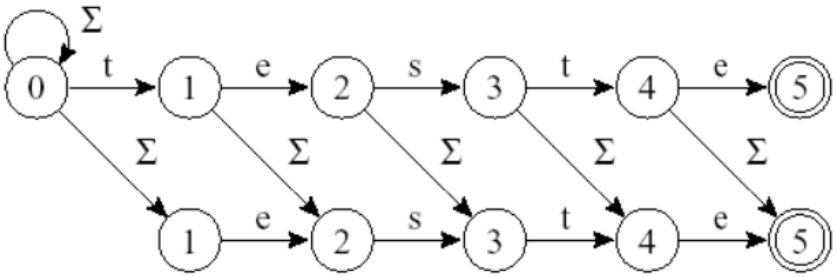

Figura - Autômato que reconhece P ={teste}, permitindo uma substituição

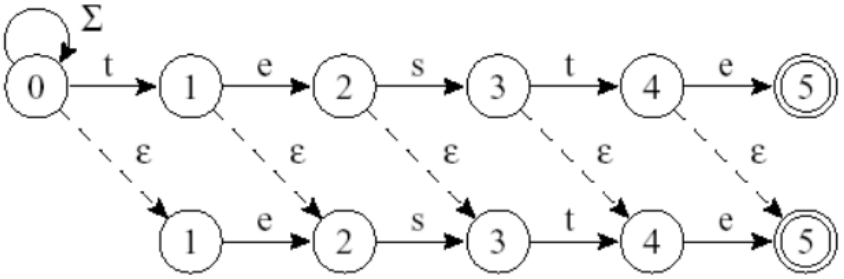

Figura - Autômato que reconhece P ={teste}, permitindo uma retirada

Enya Luísa Gomes dos Santos
19.2.4201

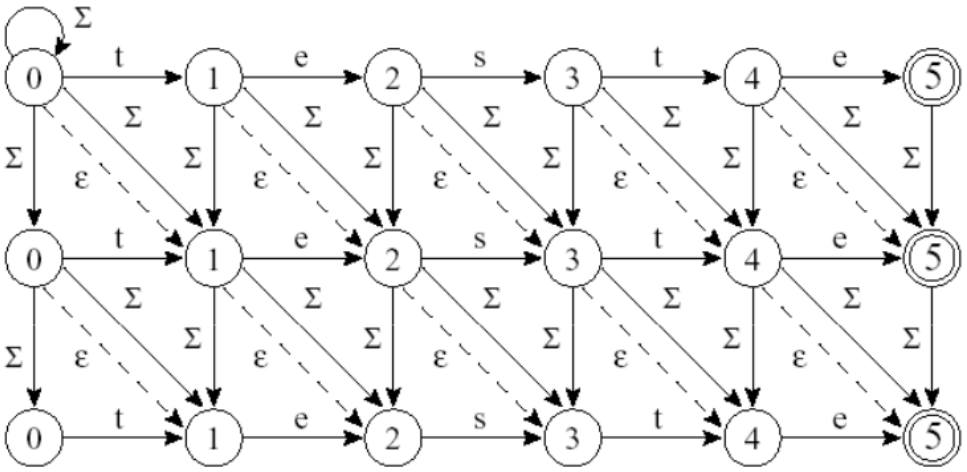

Figura - Autômato reconhece P ={teste} para k = 2
Linha 1: casamento exato (k = 0).
Linha 2: casamento aproximado permitindo um erro (k = 1).
Linha 3: casamento aproximado permitindo dois erros (k = 2).

ALGORITMO SHIFT-AND APROXIMADO

Shift-And aproximado simula um autômato não-determinista (é possível ter várias estado ativos no mesmo instante de tempo) , utilizando paralelismo de bit.

Assim como no Shift-And exato teremos as máscaras que representarão os estados do autômato, com a diferença de que não haverá apenas uma máscara e sim uma para cada linha do autômato, ou seja, terá k máscaras de bits:

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

- A máscara R_0 (casamento exato) é inicializada como $R_0 = 0^m$.
- Para $0 < j \leq k$, Rj é inicializada como $Rj = 1^j0^{(m-j)}$, pois assim considera-se que um erro pode ser encontrado logo de cara.
- Considerando M a **tabela do algoritmo** Shift-And para casamento exato, para cada novo caractere ti+1 lido do texto, as máscaras são atualizadas pelas expressões:

  R'_0 = $((R_0 >> 1) \mid 10^{(m-1)}) \& M[T[i]]$

  Para $0 < j \leq k$,

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

  - A operação (Rj >> 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 ((Rj >> 1) & M[T[i]]) um `and` com a máscara do caractere na tabela de máscaras.
- … **| Rj-1 | (Rj-1 >> 1) | (R'j-1 >> 1) | 10^(m-1)** onde:
  - (Rj-1): verticais, indicando erro de inserção.
  - (Rj-1 >> 1) diagonais cheias, indicando erro de substituição.
  - (R'j-1 >> 1) diagonais tracejadas, indicando erro de retirada.

- Assim como no Shift-End exato, a cada leitura de caractere do texto, verifica-se se o último bit é 1.

Exemplificação:

A tabela M utilizada será a mesma do exato

	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

```
T = os testes testam ...
P = teste
K = 1
Possibilidade de um erro de inserção
R'0 = ((R0 >> 1) | 10m−1) & M[T[i]]
R'1 = ((R1 >> 1) & M[T[i]]) | R0 | 10m−1

R'0 {o} = 00000 => 00000
R'1 {o} = 10000 => (01000(R1 >> 1) `and` 00000(M[T[o]]))
        = 00000 `or` 00000(R'0) `or` 10000(10^(m-1)) = 10000
R'0 {s} = 00000
R'1 {s} = 10000 => (01000(R1 >> 1) `and` 00100(M[T[s]]))
        = 00000 `or` 00000(R'0) `or` 10000(10^(m-1)) = 10000
R'0 { } = 00000
R'1 { } = 10000 => (01000(R1 >> 1) `and` 00000(M[T[ ]]))
        = 00000 `or` 00000(R'0) `or` 10000(10^(m-1)) = 10000
R'0 {t} = 10000
R'1 {t} = 10000 => (01000(R1 >> 1) `and` 10010(M[T[t]]))
        = 00000 `or` 10000(R'0) `or` 10000(10^(m-1)) = 10000
R'0 {e} = 01000
R'1 {e} = 10000 => (01000(R1 >> 1) `and` 01001(M[T[e]]))
        = 01000 `or` 10000(R'0) `or` 10000(10^(m-1)) = 11000
R'0 {s} = 00100
```

```
R'1 {s} = 11000 => (01100(R1 >> 1)  `and`  00100(M[T[s]]))
        = 00100 `or`  11000(R'0)  `or`  10000(10^(m-1)) = 11100
```

.
.
.

Continuação do exemplo 01

Exemplo 01:

P ={teste} e T ={os testes testam} com um erro de inserção e k = 1. As expressões para atualização das máscaras de bits serão:

- **R'0 = ((R0 >> 1) | 10m−1) & M[T[i]]**
- **R'1 = ((R1 >> 1) & M[T[i]]) | R0 | 10m−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

Figura - Tabela de “execução” do exemplo 01

Enya Luísa Gomes dos Santos
19.2.4201

Exemplo 02:

P ={teste} e T ={os testes testam} agora com um erro de inserção, um erro de substituição e um erro de retirada e k = 1. As expressões para atualização das máscaras de bits serão:

- **R'0 = ((R0 >> 1) | 10m−1) & M[T[i]]**
- **R'1 = ((R1 >> 1) & M[T[i]]) | R0 | (R0 >> 1) | (R'0 >> 1) | 10m−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 1 0 0 0	
e	1 1 0 0 0	0 1 0 0 0	0 1 1 0 0	1 1 1 0 0	Casamento aproximado R
s	1 0 1 0 0	0 0 1 0 0	0 1 1 1 0	1 1 1 1 0	
t	1 0 0 1 0	1 0 0 1 0	0 1 1 1 1	1 1 1 1 ***1***	Casamento exato
e	1 1 0 0 1	0 1 0 0 ***1***	0 1 1 1 1	1 1 1 1 ***1***	
s	1 0 1 0 0	0 0 1 0 0	0 1 1 1 1	1 1 1 1 ***1***	Casamento aproximado I
	1 0 0 0 0	0 0 0 0 0	0 1 1 1 1	1 0 1 1 0	
t	1 0 0 0 0	1 0 0 0 0	0 1 0 1 1	1 1 0 1 0	Casamento aproximado I
e	1 1 0 0 0	0 1 0 0 0	0 1 1 0 1	1 1 1 0 ***1***	
s	1 0 1 0 0	0 0 1 0 0	0 1 1 1 0	1 1 1 1 0	Casamento aproximado R
t	1 0 0 1 0	1 0 0 1 0	0 1 1 1 1	1 1 1 1 ***1***	
a	1 1 0 0 1	0 0 0 0 0	0 1 1 1 1	1 1 0 1 ***1***	Casamento aproximado S
m	1 0 0 0 0	0 0 0 0 0	0 1 1 0 1	1 0 0 0 0	

Figura - Tabela de “execução” do exemplo02

Enya Luísa Gomes dos Santos
19.2.4201

Algoritmo do Shift-And aproximado

```
void Shift-And-Aproximado (P = p1p2...pm, T = t1t2...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';
    }
}
```

```
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);
    }
}
```

Enya Luísa Gomes dos Santos
19.2.4201