Universidade Federal de Ouro Preto
Departamento de Computação – DECOM
Disciplina: BCC203 - Estrutura de Dados II
Professor: Guilherme Tavares de Assis
Resumo III – Casamento de Cadeias

Turma: 11
Nome: Pedro Lucas Damasceno Silva
Matrícula: 20.1.4003

- **INTRODUÇÃO**

O casamento de cadeias consiste em identificar, eficientemente, uma cadeia de elementos (também chamada de 'padrão') dentro de um grande volume de elementos. Há também a possibilidade do casamento aproximado de cadeias, permitindo a localização de ocorrências similares ao padrão. No caso de uma cadeia de caracteres, os elementos são escolhidos a partir de um conjunto (o alfabeto); já em uma cadeia de bits, esse conjunto será composto apenas entre os elementos {0,1}.

Em um texto de tamanho *n*, podemos buscar um padrão com uma cadeia de $m \leq n$, sendo todos os elementos de *P* e *T* pertencentes a um alfabeto finito Σ de tamanho *c*. A partir disso, dadas as cadeias P e *T*, deseja-se saber as ocorrências de *P* em *T*.

Alguns exemplos de aplicação do casamento de cadeias são: edição de texto; recuperação de informação; estudo de sequências de DNA em biologia computacional.

- **CATEGORIAS DE ALGORITMOS**

1. *P* e *T* não são pré-processados (força bruta)

Algoritmo sequencial, on-line e de tempo real. Ambos padrão e texto não são conhecidos a priori. A complexidade de tempo é O(*m* x *n*) no pior caso e a de espaço é O (1). Nesse algoritmo, são comparados os caracteres 1 a 1 e, uma vez que um casamento seja invalidado por algum caractere, a busca se resume a partir do caractere seguinte ao qual o casamento se iniciou. Por exemplo:

P: aaaa*aaaaaaaab
T: aaab

No instante em que o casamento falha (no caractere anterior ao *), a busca deve reiniciar a partir do segundo elemento (em azul) de P, e não do caractere seguinte a *. Essa é a representação do pior caso, no qual o casamento se dá quase por completo diversas vezes. Todavia, o caso esperado é bem melhor que o pior caso e, em experimentos realizados por Baeza-Yates com textos aleatórios e alfabeto de tamanho 4, o número esperado de comparações por caractere foi aproximadamente 1,3.

$$\overline{C_n} = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right)(n - m + 1) + O(1)$$

Figura 1 - Caso esperado do algoritmo força bruta

2. Apenas *P* é pré-processado

O padrão é conhecido a priori e, dessa forma, seu pré-processamento é permitido. É um algoritmo sequencial, possui complexidade de tempo O (*n*), dependendo apenas da dimensão do texto, e de espaço O (*m* + *c*). É aplicado em programas para edição de textos.

3. Ambos *P* e *T* são pré-processados

Ambos padrão e texto são conhecidos a priori. O algoritmo constrói um índice quando a base de dados é grande e semi-estática (atualizações em intervalos regulares) e o tempo para geração desse índice pode ser O (*n*) ou O (*n* log *n*). Um pré-processamento possível é a construção de uma estrutura de pesquisa. No caso de uma árvore B indexada pelas palavras do texto, por exemplo, a complexidade de pesquisa será logarítmica.

Os índices mais utilizados são os arquivos invertidos (principal estrutura de índice utilizada quando o repositório possui um volume de grandes proporções), árvores TRIE e árvores PATRICIA e arranjos de sufixos. Nessa categoria de algoritmos, a complexidade de tempo é, de uma forma geral, logarítmica, e a complexidade de espaços é linear de acordo com o tamanho do texto.

- **ARQUIVO INVERTIDO**

Composto por vocabulário (conjunto das palavras distintas no texto) e ocorrências (posições das palavras no texto). O arquivo pode ser implementado de diversas formas, como, por exemplo, por meio de uma árvore B ou uma tabela de *hash*.

Uma vez criado o índice, a palavra buscada é pesquisada no vocabulário e a lista de ocorrências associadas à palavra é retornada ao usuário, indicando a presença e a localização da palavra nos arquivos sobre o qual a pesquisa é efetuada. Para buscar termos sequenciais (uma frase, por exemplo), basta buscá-los separadamente e manipular a lista de ocorrências para determinar se estes termos estão na sequência correta.

Em índices criados sobre um grande repositório de documentos, a tendência é de que as ocorrências ocupem muito mais espaço do que o vocabulário, haja vista a recorrência de termos comuns como as palavras ‘de’, ‘que’ ou outros termos muito comuns para o tipo de texto analisado. O vocabulário comumente pode ser armazenado em memória principal, já as ocorrências são armazenadas em um arquivo em memória secundária quando há necessidade.

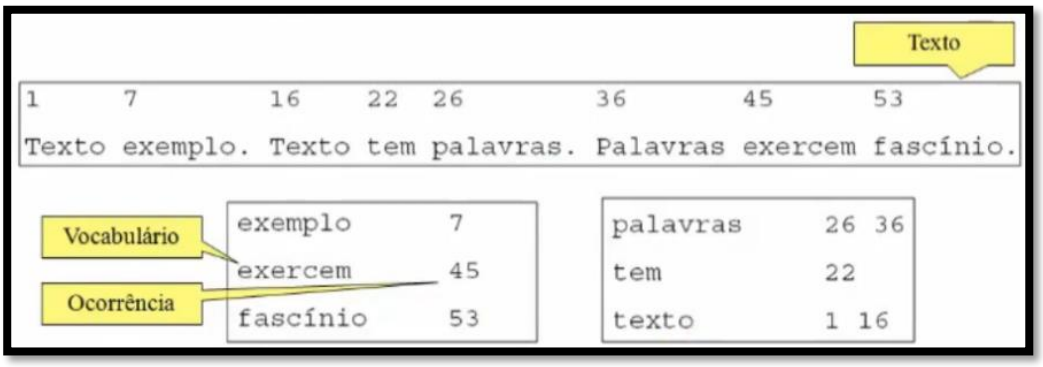

Figura 2 - Exemplo de vocabulário e ocorrências extraídos de um trecho

A previsão sobre o crescimento do tamanho do vocabulário é definido pela lei de Heaps:

$$V = Kn^{\beta} = O(n^{\beta})$$

n = nº de palavras
K = geralmente entre 10 e 100
β = constante entre 0 e 1 (geralmente entre 0,4 e 0,6)

Na prática, o vocabulário cresce de acordo com o texto em uma proporção próxima de sua raiz quadrada. Já as ocorrências ocupam muito mais espaço, sendo da ordem de O(n) e proporcional a 30~40% do tamanho do texto.

Na inserção de um novo trecho ou arquivo a um índice já existente, é provável que apenas suas ocorrências aumentem, principalmente se o índice prévio for proveniente de um repositório textual vasto.

A pesquisa nesse método é realizada em três passos: pesquisa no vocabulário, recuperação da lista de ocorrências e manipulação das ocorrências (para tratar frases, proximidades e/ou operações lógicas). O tempo de busca depende da estrutura de dados utilizada para implementar o índice.

- Árvore Trie

Utiliza índices de caracteres para realizar o caminhamento e os elementos são armazenados em nós folha juntos a um apontador para sua lista de ocorrências. A partir da raiz, comparam-se os primeiros caracteres e, em seguida, os seguintes são diferenciados a partir do caminhamento. A ordem de complexidade é, no pior caso, O(n), referente ao tamanho do padrão, e, quanto maior a árvore, maior a probabilidade de aumento do custo de pesquisa.

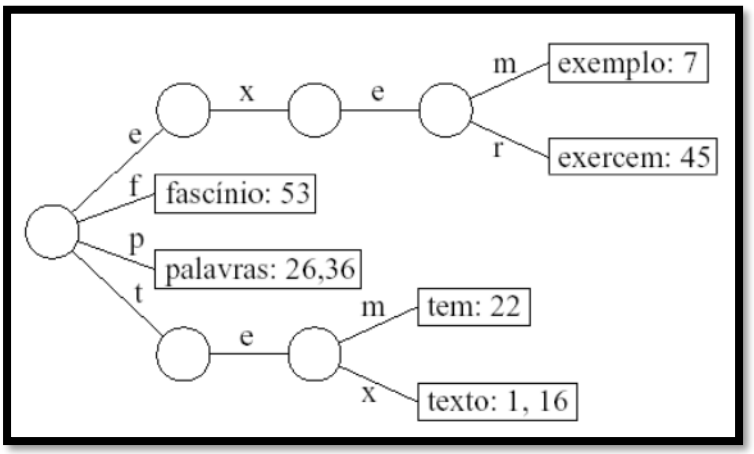

Figura 3 - Exemplo de árvore TRIE com poucos elementos

- Árvore Patricia

Dispensa nós de caracteres de acordo com o vocabulário existente. Na figura 2, por exemplo, é possível eliminar nós internos que possuem apenas um apontador para outro nó interno, como o ‘x’ e ‘e’ do caminho superior e o ‘e’ do caminho inferior. Dessa forma, encurtamos o caminho percorrido para atingir os nós folhas.

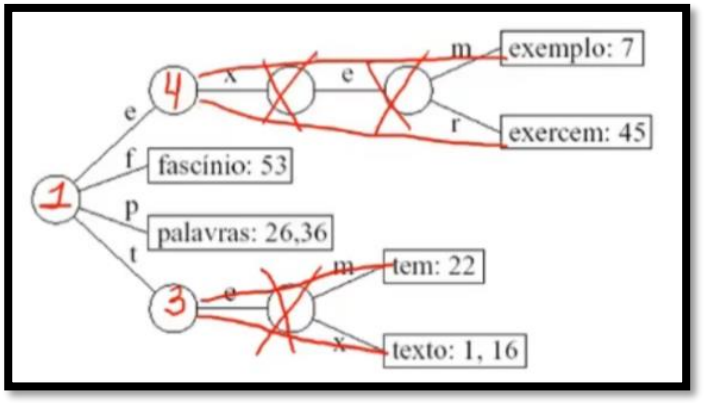

Figura 4 - Encurtamento da árvore Patricia

Em um repositório de documentos muito vasto e com vocabulários muito extensos, muitos elementos se diferenciam já nos primeiros caracteres, o que favorece a árvore Trie à medida que dispensa algumas etapas de pré-processamento existentes na árvore Patricia.

- **ALGORITMO BOYER-MOORE**

Pesquisa um padrão P em uma janela que desliza ao longo do texto. Para cada posição desta janela, o algoritmo pesquisa por um sufixo da janela que casa com um sufixo de P, com comparações realizadas da direita para a esquerda. Se não ocorrer uma desigualdade, uma ocorrência do padrão foi localizada. Caso contrário, o algoritmo calcula um deslocamento que a janela deve realizar para a direita antes que uma nova busca por casamento se inicie.

O que diferencia o algoritmo de uma força bruta é a criação de duas heurísticas que calculam o deslocamento da janela quando o casamento falha, substituindo o avanço singular de caracteres visto na

força bruta, podendo avançar até um máximo de caracteres proporcional ao tamanho do padrão de acordo com a heurística utilizada. Mais de uma heurística pode ser utilizada, propondo uma forma 'híbrida' que calcula os deslocamentos de cada uma e utiliza o que for maior. Todavia, isso implica em um incremento considerável da complexidade do algoritmo, algo não desejável. Ao longo dos anos, várias propostas de simplificação surgiram e os melhores resultados foram obtidos por métodos que consideraram apenas a heurística ocorrência.

- Heurística ocorrência

Alinha o caractere anterior ao causador da colisão com a primeira ocorrência dele no texto a partir da referência (da direita para a esquerda) de onde a pesquisa está sendo realizada. Quando o caractere causador da colisão não se encontra no restante do padrão, ele é pulado por inteiro. É bastante comum que o primeiro caractere comparado se diferencie do padrão logo de início e que este caractere causador da colisão não se encontre no restante do texto, fazendo com que grandes deslocamentos ocorram (principalmente considerando alfabetos mais diversificados), o que torna o algoritmo mais interessante do que a força bruta.

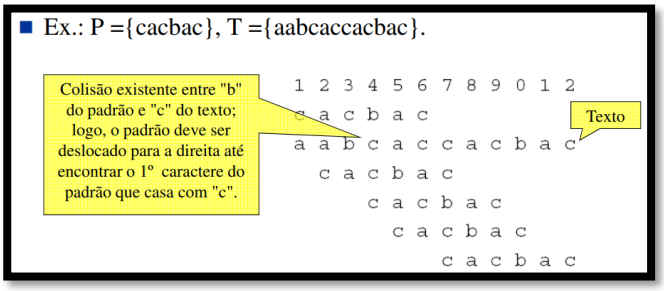

Figura 5 - Heurística ocorrência. Pequenos deslocamentos devidos ao alfabeto pouco diversificado

- Heurística casamento

Alinha os caracteres casados (seguintes ao causador da colisão no sentido da esquerda para a direita) com a primeira ocorrência deles no texto. O pior caso se dá quando a primeira comparação é imediatamente desigual, não havendo, portanto, caracteres casados para serem buscados no padrão e, portanto, o padrão é deslocado de apenas uma unidade. Em um alfabeto diversificado, esse fenômeno é bastante comum, desfavorecendo essa heurística em relação à heurística ocorrência. Já em alfabetos mais limitados, essa heurística se prova mais eficiente dada a maior probabilidade de casamentos parciais.

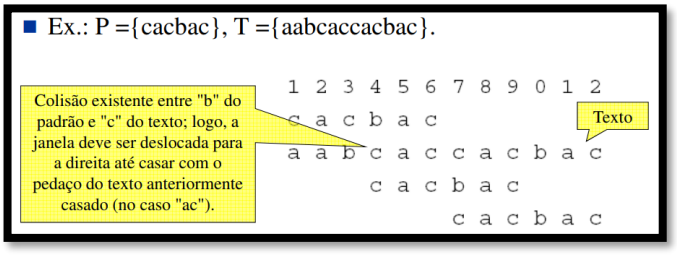

Figura 6 - Heurística casamento. Grandes deslocamentos devidos ao alfabeto pouco diversificado

- **ALGORITMO BOYER-MOORE-HORSPOOL**

Em 1980, Horspool apresentou uma simplificação do algoritmo BM, que o tornou mais rápido. Pela extrema simplicidade e comprovada eficiência, o BMH deve ser escolhido em aplicações de uso geral que necessitam realizar casamento exato de cadeias.

Consiste na geração de uma tabela de deslocamentos definida a partir de:

1. O valor inicial do deslocamento para todos os caracteres do texto é igual a *m.*
2. Para os *m*-1 primeiros caracteres do padrão P, os valores do deslocamento são calculados por:

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

Calcula a distância da última ocorrência de cada caractere do padrão em relação ao último e atribui o valor de m para caracteres do texto que não estejam presentes no padrão.

3. Quando o casamento falha, independente do caractere causador, o deslocamento é definido pelo valor do caractere do texto alinhado ao último do padrão na tabela de deslocamentos.
4. Na ocorrência de um caractere no texto que não existe no padrão, o deslocamento é de *m* posições.

- **ALGORITMO BOYER-MOORE-HORSPOOL-SUNDAY**

Em 1990, Sunday propôs deslocar a janela de acordo com o valor da tabela de deslocamento relativo ao caractere no texto correspondente ao caractere seguinte ao último do padrão. A nova implementação da tabela de deslocamentos se dá a partir de:

1. O valor inicial do deslocamento para todos os caracteres do texto é igual a *m+1*.
2. Para os m primeiros caracteres do padrão P, os valores do deslocamento são calculados por:

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

3. Para todos os caracteres (com exceção do último do padrão que não era considerado anteriormente e que possui ocorrência prévia no padrão), todos os deslocamentos foram incrementados, diminuindo a frequência destes deslocamentos em relação ao algoritmo BMH.

- **AUTÔMATOS**

Modelo matemático de uma máquina de estados finitos onde, no exemplo a seguir, são os nodos (0, 1, 2, 3), que são ligados por arestas (pertencentes a um alfabeto). Há também o estado inicial (0), onde o autômato inicia, e um conjunto de estados finais. No exemplo abaixo, o estado 3 é o final. Por fim, uma função define as transições entre os estados e o caminhamento pelas arestas. No contexto do casamento de cadeias, uma cadeia a ser reconhecida é qualquer sequência de caracteres que parte do estado inicial e chega ao final. A quantidade de nodos é igual a (n + 1), a complexidade de tempo é O(*n*) e a de espaço é (*m + 1*) para vértices e (*alfabeto x m*) para arestas.

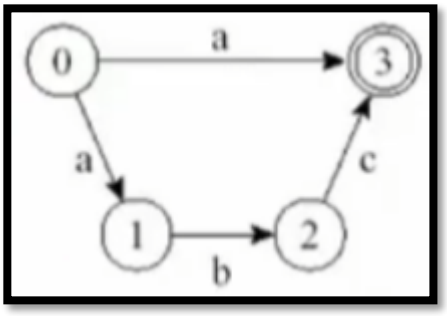

Figura 7 - Exemplo de autômato simples

1. Autômato finito não-determinista

Ocorre quando a função possibilita a associação de um estado e um caractere para mais de um estado do autômato. A figura 7 é um exemplo de autômato não-determinista pois a associação de 0 e ‘a’ conduz aos estados 1 e 3.

2. Autômato finito determinista

Ocorre quando a função permite a associação de um estado e um caractere para apenas um estado do autômato.

3. Transição vazia

Ocorre quando não há necessidade de ler o caractere para transitar de um estado para outro.

4. Estado ativo

Estado atual que representa a subcadeia de caracteres encontrada até então.

5. Autômatos cíclicos

Aqueles cujas transições formam ciclos. Especialmente úteis para casamento de expressões regulares.

- **ALGORITMO *SHIFT-AND* EXATO**

Simula um autômato não-determinista no qual o padrão desejado é representado por uma sequência de bits (uma máscara) e cada bit representa um estado do autômato. Baseia-se no conceito de paralelismo de bit, que utiliza operações (|, &, >>) sobre bits dentro de uma palavra. À medida que cada caractere da cadeia é encontrado, o estado do autômato simulado se torna ativo (o bit correspondente adquire valor ‘1’) e se o último estado for alcançado, então o padrão foi localizado no texto.

1. Pré-processamento

Construção de uma tabela para armazenar a máscara de bits de cada caractere do padrão.

	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 8 - Tabela referente à máscara da palavra 'teste'

2. Algoritmo

Para cada novo caractere, uma tentativa de caminhamento é realizada conforme a fórmula:

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

R >> 1 - shift à direita de 1 com a sequência binária inicial
10^{m-1} - utilizado para tentar caminhar no autômato (representa também uma transição vazia)
M[T[i]] - máscara do caractere lido e validação do 'ou' lógico à esquerda

À medida que a associação de operadores de R' seja verdadeira, o bit 1 caminha da esquerda para a direita entre zeros. No caso de padrões com letras repetidas, mais de um estado pode estar ativo simultaneamente, o que pode ser útil para um próximo casamento caso o 1 mais à direita seja invalidado posteriormente. Um casamento bem sucedido se dá quando o último bit é verdadeiro.

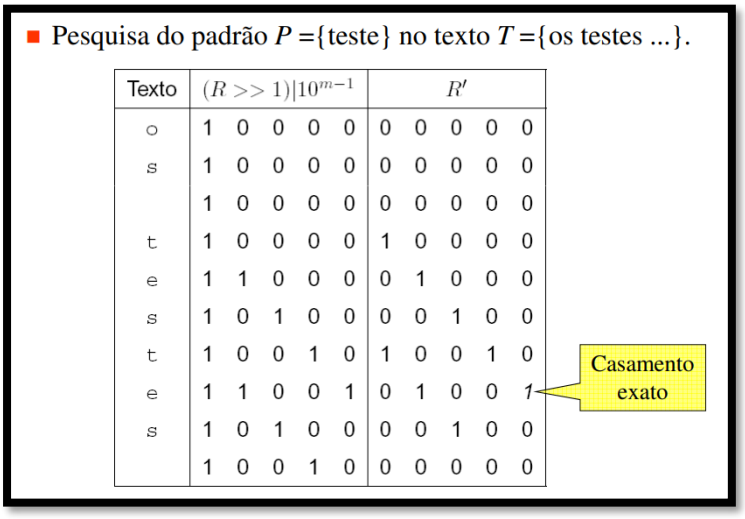

Figura 9 - Exemplo de execução do algoritmo shift-and exato

3. Complexidade

Custo O(*n*) desde que as operações sobre os bits possam ser realizadas em O(*1*) e que o padrão caiba em poucas palavras do computador.

- **CASAMENTO APROXIMADO**

Consiste em encontrar as ocorrências aproximadas de um padrão no texto, sob o modelo de um autômato não-determinista e utilizando algoritmos com paralelismo de bit, por meio de operações de:

1. Inserção (caractere inserido no meio do padrão)
2. Substituição (último caractere diferente do padrão)
3. Retirada (primeiro caractere do padrão)

O conceito de **distância de edição** entre duas cadeias, denotadas por *ed*, representa o menor número de operações necessárias para converter uma cadeia na outra ou vice-versa. É utilizado para formalizar o problema do casamento aproximado à medida que possibilita o estabelecimento de uma tolerância de desvio do padrão buscado. O valor de *k* deve estar entre 0 (casamento exato) e o tamanho do padrão, caso contrário diversos casamentos distantes do padrão serão encontrados. Por exemplo:

ed(teste, estendo) = 4, sendo uma retirada e 3 inserções (*ndo*) ao final do padrão. Esse casamento aproximado seria considerado se a tolerância (*k*) determinada fosse maior ou igual a 4.

Há também o conceito de **nível de erro** (*k / tamanho do padrão*), que para a maioria dos casos assume valores menores que ½.

Exemplos de autômatos para cada tipo de operação:

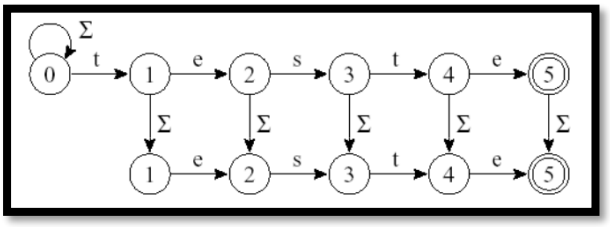

Figura 10 - autômato que reconhece 'teste' permitindo uma inserção. As arestas verticais admitem a transição para qualquer caractere extra fora do padrão e, para aumentar a quantidade de inserções, basta aumentar o número de linhas proporcionalmente.

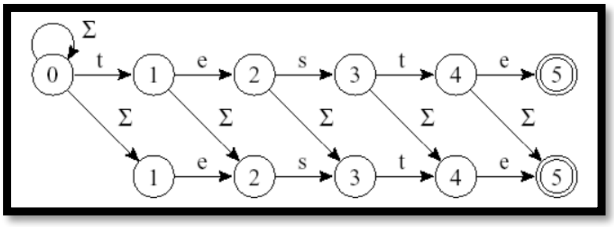

Figura 11 - autômato que reconhece 'teste' permitindo uma substituição. As arestas diagonais admitem a transição para caracteres diferentes do padrão.

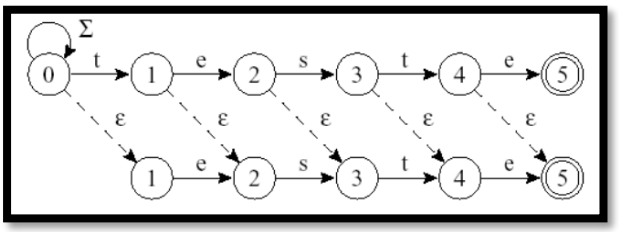

Figura 12 - autômato que reconhece 'teste' permitindo uma retirada. As arestas verticais (transições vazias) tornam ativos os estados correspondentes da linha inferior, possibilitando o caminhamento caso um caractere desejado não seja encontrado.

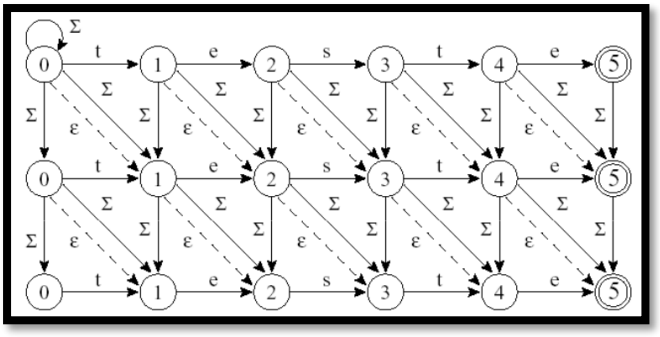

Figura 13 - autômato que reconhece 'teste' para até 2 erros tratados por quaisquer das 3 operações apresentadas.

- **ALGORITMO *SHIFT-AND* APROXIMADO**

Tal como o shift-and exato, também simula um autômato não-determinista que utiliza o paralelismo de bit. O autômato é representado por mais de uma máscara de bit, sendo R_0 (inicializada como 0^m) o casamento exato, R_1 o casamento com um erro e R_k para *k* erros.

1. Pré-processamento

As máscaras de R_1 a R_k são inicializadas como $R_j = 1^j0^{m-j}$, dados os estados ativos nas linhas inferiores do autômato simulado, e a tabela M do algoritmo é inicializada da mesma forma vista no algoritmo shift-and exato.

2. Algoritmo

Para cada novo caractere, uma tentativa de caminhamento é realizada conforme as fórmulas:

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

$\mid R_{j-1}$ – para erros de inserção
$\mid (R_{j-1} >> 1)$ – para erros de substituição
$\mid (R'_{j-1} >> 1)$ – para erros de retirada

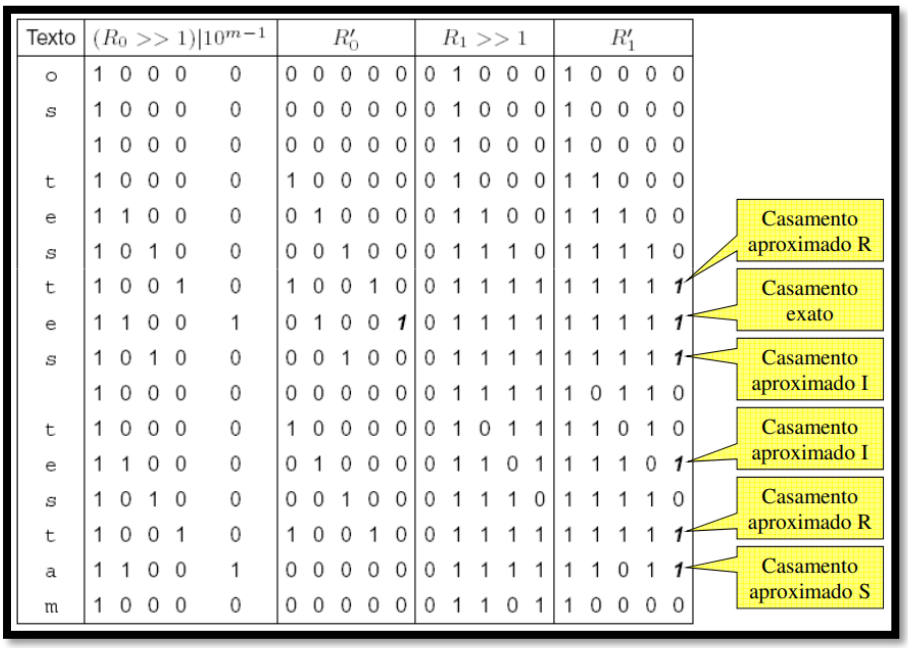

Figura 14 - exemplo de execução do algoritmo shift-and aproximado considerando o padrão 'teste' e uma possibilidade de cada tipo de erro (k = 1)