Fábio Henrique Alves Fernandes - 19.1.4128

Resumo: Casamento de Cadeia

Casamento de Cadeia

Casamento de Cadeia (ou Casamento de Padrão) é uma forma de encontrar todas as ocorrências de um determinado padrão em uma cadeia de caracteres (texto). Algumas das principais aplicações são em adição de texto, recuperação de informação e estudo de sequências de DNA em biologia computacional.

Para a solução, temos que levar em conta um texto ‘T’ de tamanho ‘n’ e um padrão ‘P’ de tamanho ‘m’, onde m <= n, e todos os elementos de ‘P’ e ‘T’ são escolhidos em um alfabeto finito de tamanho ‘c’.

Temos três tipos de algoritmos para casamento de cadeias: Quando ‘P’ e ‘T’ não são pré-processados, com complexidade de tempo O(mn) e a complexidade de espaço é de O(1); quando somente ‘P’ é pré-processado, a complexidade de tempo é de O(n) e a complexidade de espaço é O(m+c); e quando ‘P’ e ‘T’ são pré-processados, a complexidade de tempo é de O(log n) e a complexidade de espaço é de O(n). Dizemos que uma cadeia é pré processada quando os elementos já foram lidos e armazenados.

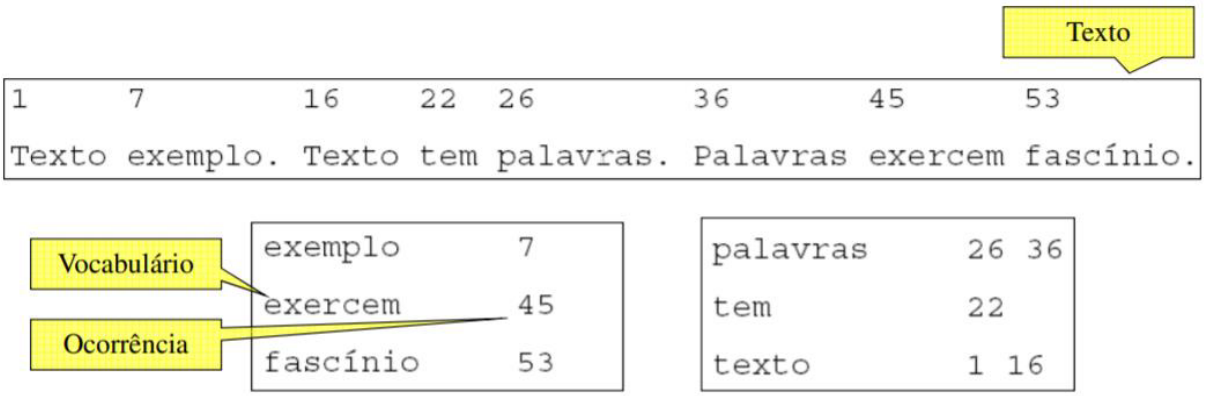
Dos tipos mencionados, o mais utilizado é o último. O próprio algoritmo constrói índices para o texto, principalmente quando a base de dados é grande e com atualizações em intervalos regulares. O tempo para geração do índice pode ser tão grande quanto O(n) ou O(n log n), mas é compensado por muitas operações de pesquisa no texto. Alguns dos tipos de índices mais conhecidos são Arquivo Invertido, Árvores TRIE e Árvores PATRICIA (Árvores de prefixos), e arranjo de sufixos.

Arquivo Invertido

Arquivo Invertido é uma estratégia de indexação que permite a realização de buscas precisas e rápidas, em troca de maior dificuldade no ato de inserção e atualização de documentos. É composto por uma tabela de vocabulário e ocorrências, onde o vocabulário é

um conjunto de palavras distintas na base de dados e a ocorrência é uma lista onde é armazenada as posições onde cada palavra aparece.

Exemplo: Usando a expressão “Texto exemplo. Texto tem palavras. Palavras exercem fascínio”, vamos criar o arquivo invertido.



Levando em consideração uma base de dados maior, o vocabulário ocupa bem menos espaço que as ocorrências, pois quanto maior o texto, maior a possibilidade de ter repetições, aumentando a ocorrência da palavra a cada leitura.

A previsão sobre o crescimento do tamanho do vocabulário é definida pela lei de Heaps (ou lei de Herdan), uma lei empírica que descreve o número de palavras diferentes em um documento (ou conjunto de documentos) em função do comprimento do documento (os chamados relação tipo token). Pode ser formulado como:

$$V_R(n) = Kn^{\beta}$$

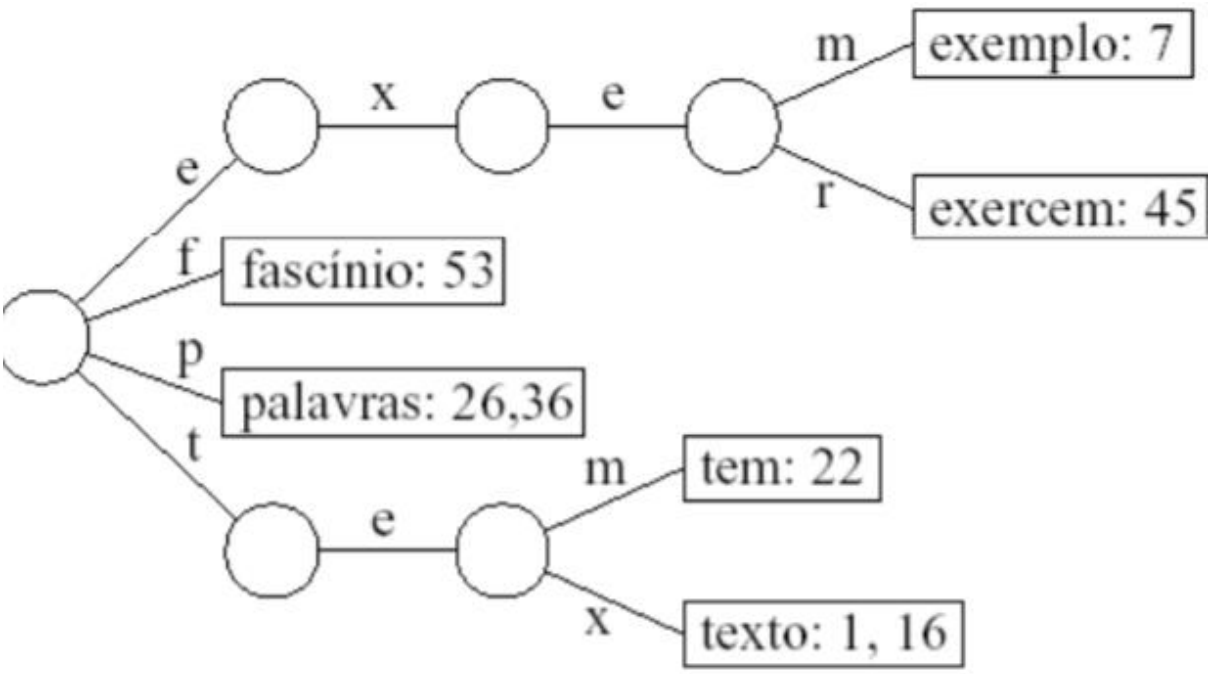
onde K e β dependem das características de cada texto, K geralmente assume valores entre 10 e 100, e β é uma constante entre 0 e 1 (na prática entre 0,4 e 0,6).

A pesquisa é realizada em 3 passos: pesquisa no vocabulário, recuperação das ocorrências e manipulação das ocorrências. Como a pesquisa de um arquivo invertido sempre começa pelo vocabulário, é interessante mantê-lo em um arquivo separado, geralmente esse arquivo cabe na memória principal.

A pesquisa por palavras simples pode ser realizada usando qualquer estrutura de dados que torne a pesquisa eficiente, como hashing, árvore TRIE ou árvore B, onde as duas primeiras têm custo O(m), onde m é o tamanho da consulta (independentemente do tamanho do texto) e a última possui o custo O (log n); guardar as palavras na ordem lexicográfica é

barato em termos de espaço e competitivo em desempenho. A pesquisa por frases usando índices é mais difícil, já que cada palavra da frase deve ser pesquisada separadamente, no intuito de recuperar listas de ocorrência. Depois, as listas devem ser percorridas de forma sincronizada para encontrar as ocorrências nas quais as palavras aparecem em ordem.

Exemplo: Usando Árvore TRIE



O vocabulário do texto é adicionado na árvore, armazenando a lista de ocorrências de cada palavra. A cada nova palavra lida no texto, é feita uma pesquisa na árvore. Caso ela já exista, uma nova ocorrência é adicionada, caso contrário, ela é inserida na árvore e uma nova lista de ocorrências é feita para ela.

Para o casamento de cadeias, temos dois tipos de problemas: o casamento exato, e o casamento aproximado.

Casamento Exato

No casamento exato, temos que encontrar a ocorrência exata de um determinado padrão dentro do texto. Os algoritmos podem ser caracterizados da forma como o padrão é pesquisado no texto: quando a leitura é feita caractere a caractere, buscando o padrão no texto, como exemplo os algoritmos Força Bruta e o Shift-And; e quando a leitura é feita

como uma janela deslizando pelo texto ‘T’, buscando pelo padrão, como os algoritmos Boyer-Moore e suas variantes.

Algoritmo Força Bruta

O algoritmo Força Bruta é o mais simples de todos, que consiste em tentar casar qualquer subcadeia de tamanho ‘m’ no texto com o padrão desejado.

Analisando o pior caso: C = m x n, ex: P=aab e T=aaaaaaaaaa

Também temos como analisar o caso esperado, que é bem melhor que o pior caso:

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

Implementação do algoritmo Força Bruta:

```
void ForcaBruta (TipoTexto T, long n, TipoPadrao P, long m){
      long i, j, k;
      for (i = 1; i <= (n - m + 1); i++){
            k = i;
            j = 1;
            while (T[k-1] == P[j-1] && j <= m){
            //Pesquisa do padrão P a partir da k-ésima posição do
            //texto T
                  j++;
                  k++;
            }
      if (j < m)
            printf ("Casamento na posição %3ld\n", i);
      }
}
```

Algoritmo Boyer-Moore

Diferente do Força Bruta, o algoritmo Boyer-Moore pesquisa o padrão em uma janela que desliza ao longo do texto. Para cada posição dessa janela, ele pesquisa por um 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. O algoritmo propõe duas heurísticas para calcular o deslocamento: ocorrência e casamento.

Heurística de Ocorrência:

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

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

1 2 3 4 5 6 7 8 9 0 1 2
a a b c a c c a c b a c
c a c b a c – colisão em 'b' do padrão com 'c' do texto
  c a c b a c – colisão em 'a' do padrão com 'c' do texto
    c a c b a c – colisão em 'c' do padrão com 'c' do texto
      c a c b a c – colisão em 'c' do padrão com 'a' do texto
        c a c b a c – Casamento exato

Heurística de Casamento:

A Heurística Casamento faz com que, ao mover o padrão à direita, a janela em questão com o pedaço do texto anteriormente casado.

Usaremos o mesmo exemplo feito para a Heurística Ocorrência

1 2 3 4 5 6 7 8 9 0 1 2
a a b c a c c a c b a c
c a c b a c - Colisão do 'b' com o 'c'
    c a c b a c - Colisão do 'b' com o 'a'
      c a c b a c - Casamento exato

O algoritmo Boyer-Moore consegue decidir qual dessas heurísticas usar, escolhendo qual provoca o maior deslocamento do padrão, fazendo comparações a cada colisão.

Algoritmo Boyer-Moore-Horspool

Nigel Horspool foi responsável por apresentar uma simplificação no algoritmo BM, tornando-o mais rápido, de implementação simples e eficiente. Boyer-Moore-Horspool, como ficou conhecido, é feito para uso geral para casamento exato de cadeias. Pensando que qualquer caractere já lido do texto a partir do último deslocamento pode ser usado para endereçar uma tabela de deslocamentos, a janela será deslocada de acordo com o valor da tabela relativo ao caractere no texto correspondente ao último caractere do padrão.

Para criar a tabela, o valor inicial do deslocamento para todos os caracteres do texto recebem o valor de 'm'. Em seguida, para os m-1 primeiros caracteres do padrão, os valores

do deslocamentos são calculados pela regra:

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

Ex: Usando o padrão P = “teste”, a sua tabela de deslocamento será

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.

Implementação:

```
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-1;
   i = m;
   //pré-processamento para se
   //obter a tabela de deslocamentos

   while (i <= n){
      //Pesquisa
      k = i;
      j = m;
      while (T[k-1] == P[j-1] && j > 0){
         //Pesquisa por um sufixo do texto
         //que casa com um sufixo do padrão
         k--;
         j--;
      }
      if (j == 0)
         printf ("Casamento na posição: %3ld\n", k+1);

      i += d[T[i-1]]; // 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
   }
```

```
}
```

Algoritmo Boyer-Moore-Horspool-Sunday

Já o Algoritmo Boyer-Moore-Horspool-Sunday, feito por Sunday, é uma variante do BMH, com mais uma simplificação. Dessa vez, Sunday propôs endereçar a tabela com o caractere no texto correspondente ao próximo caractere do padrão, em vez de deslocar o padrão usando o último caractere como no BMH.

Para criar a tabela, o valor inicial do deslocamento para todos os caracteres do texto recebem o valor de m+1. Em seguida, para os m primeiros caracteres do padrão, os valores do deslocamento são calculados pela regra:

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

Ex.: Para o padrão P = “teste”, a sua tabela de deslocamento será:

d[“t”] = 2, 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.

Implementação

```
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;
   //pré-processamento para se
   //obter a tabela de deslocamentos

   while (i <= n){
      //Pesquisa
      k = i;
      j = m;
      while (T[k-1] == P[j-1] && j > 0){
         //Pesquisa por um sufixo do texto
         //que casa com um sufixo do padrão
         k--;
```

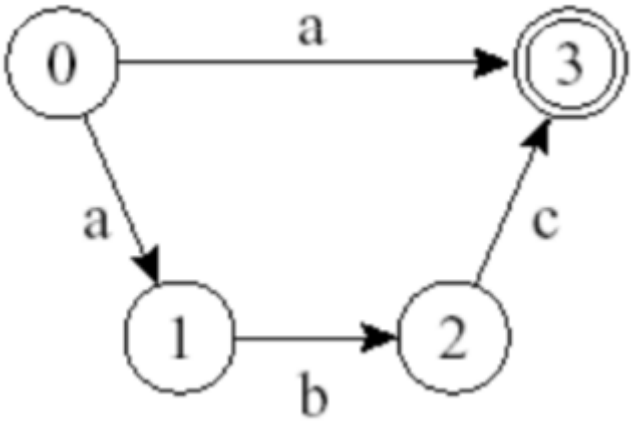
```
            j--;
        }
        if (j == 0)
            printf ("Casamento na posição: %3ld\n", k+1);

        i += d[T[i]]; // 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
    }
}
```

Autômatos

Um autômato é um modelo de computação muito simples. 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 ϵ 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 e associa a cada estado q ϵ Q um conjunto de estados (q1, q2, ..., qk} ⊆ Q, para cada α ϵ {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 (ou seja, T (q, α) = {q1, q2, ..., qk} para k>1), ou quando existe alguma transição rotulada por ε.

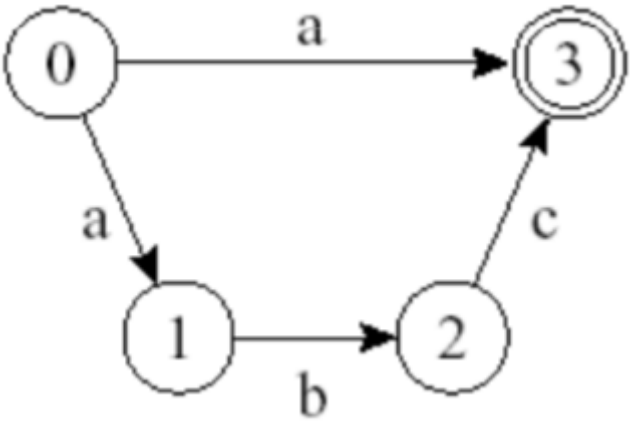


No exemplo acima, a partir do estado 0, por meio do caractere de transição a, é

possível atingir os estados 1 e 3.

Autômato finito deterministas

Ocorre quando a função T permite a associação de um estado q e um caractere α para apenas um estado do autômato (ou seja, T (q, α) = {q1}



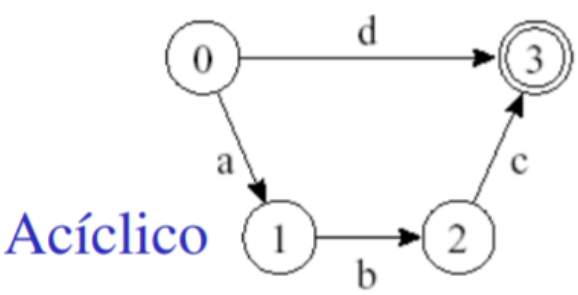
Para cada caractere de transição, todos os estados levam a um único estado.

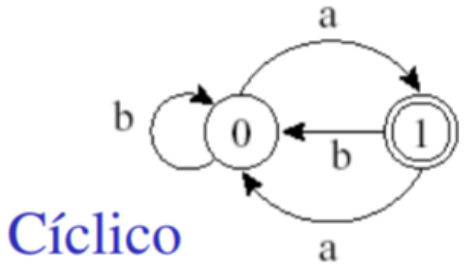
Transições vazias são transições rotuladas com a cadeia vazia ε, chamadas de transições ε em autômatos não deterministas, onde não há necessidade de se ler um caractere do alfabeto para se caminhar por meio de uma transição vazia. As transições vazias simplificam a construção do autômato. Sempre existe 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. Um autômato finito não-determinista pode ter vários estados ativos em um determinado instante. O casamento aproximado de cadeias pode ser resolvido por meio de autômatos finitos não-deterministas.

Autômato cíclicos

São aqueles que formam ciclos, isso é, se repetem várias vezes dada certa expressão. A linguagem reconhecida por um autômato cíclico pode ser infinita. Por exemplo, o autômato a direita reconhece {ba}, {bba}, {bbba}, {bbbba}, …





Shif-And Exato

O algoritmo Shift-And usa o conceito de paralelismo de bit, técnica que tira proveito do paralelismo das operações sobre bits dentro de uma palavra de computador, sendo possível empacotar muitos valores em uma única palavra e atualizar todos eles em uma única operação. Uma sequência de bits (b1 ... bc) é chamada de máscara de bits de comprimento c, e é armazenada em alguma posição de uma palavra w do computador.

Algumas operações sobre os bits de uma palavra são:

Repetição de bits como exponenciação (ex.: $01^3 = 0111$);

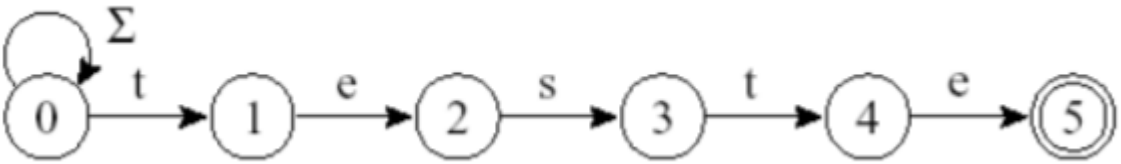
"|": operador lógico or;

"&": operador lógico and;

">>": operador que move os bits para a direita e entra zeros à esquerda (ex.:b1 b2 ... bc-1 bc >> 2 = 00b1... bc-2).

O algoritmo Shift-And mantém o conjunto de todos os prefixos do padrão P que casam o texto já lido, esse conjunto é representado por uma máscara de bits R = (b1 b2 ... bm); o algoritmo utiliza o paralelismo de bits a cada caractere lido do texto. Ele também corresponde à simulação de um autômato não-determinista que pesquisa pelo padrão P no texto T.

Ex.: Autômato que reconhece os prefixos de P = "teste"



O valor 1 é colocado na j-ésima posição de R (b1 b2 ... bn) se e somente se (p1 ... pj) é um sufixo de (t1 ... ti), onde i corresponde à posição corrente no texto, nesse caso, a j-ésima posição de R é dita estar ativo e o bm átimo significa um casamento exato do padrão.

R' (novo valor da máscara R) é calculado na leitura do próximo caractere t_{i+1} do texto. A posição 'j+1' em R' ficará ativa se e somente se a posição 'j' estava ativa em R, ou seja, (p1 ... pj) é sufixo de (t1 ... ti), e t_{i+1} casa com pj+1. Com o uso de paralelismo de bit, é possível computar a nova máscara com o custo O (1).

Para o pré-processamento do algoritmo, construímos uma tabela M para armazenar

uma máscara de bits (b1 b2 ... bm) para cada caractere do padrão P.

Ex: Tabela das máscaras de bits para os caracteres de 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.

Para o 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. Já a operação ((R >> 1) | 10^m−1) retrata o fato de que a cadeia vazia ε é também marcada como um sufixo do padrão, permitindo um casamento em qualquer 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) | 10^m−1) e a máscara M relativa ao caractere lido do texto.

Ex: Tabela que simula o funcionamento do algoritmo:

Texto	$(R >> 1)\mid 10^{m-1}$					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

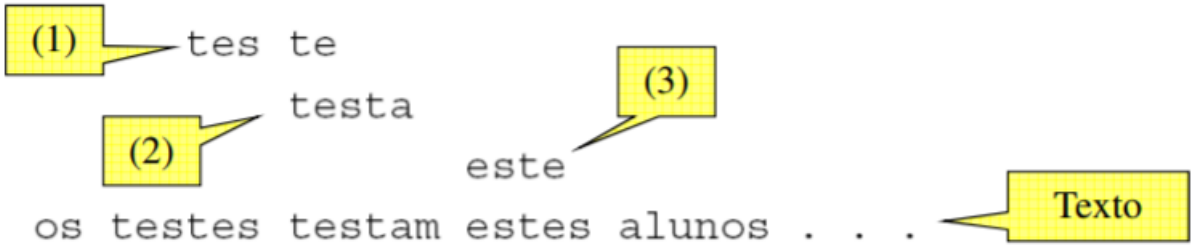
Casamento exato

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

Casamento Aproximado de cadeias é basicamente encontrar as ocorrências aproximadas de um padrão no texto, logo deve tratar operações de inserção, substituição e retirada de caracteres do padrão.

Vamos usar o padrão P = "teste", onde aparecem três ocorrências aproximadas do padrão citado.



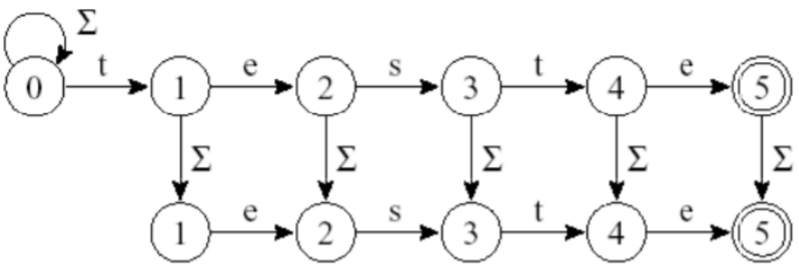
Onde:

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

A distância de edição entre duas cadeias P e P', denotada por ed(P, P'), é o menor número de operações necessárias para converter P em P' ou vice-versa. 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. Formalmente, o problema do casamento aproximado de cadeias é o de encontrar todas as ocorrências de P' no texto T tal que ed(P, P') ≤ k, onde k representa o número limite de operações de inserção, substituição e retirada de caracteres necessárias para transformar o padrão P em uma cadeia P'.

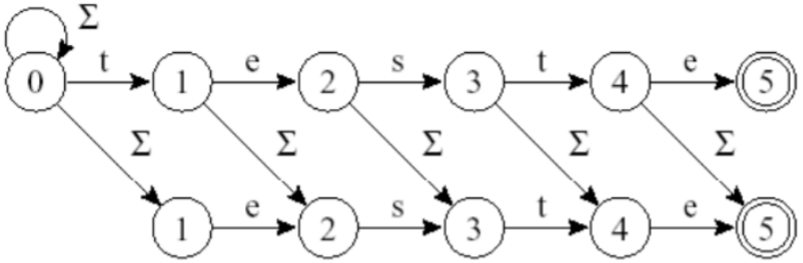
O casamento aproximado só faz sentido para 0 < k < m pois, para k = m, toda subcadeia de comprimento m pode ser convertida em P por meio da substituição de m caracteres, k = 0 corresponde ao casamento exato. Uma medida da fração do padrão que pode ser alterada é dada pelo nível de erro α = k/m. Em geral, α < ½ para a maior parte dos casos. A pesquisa com casamento aproximado é modelada por autômatos não-deterministas.

Ex: Autômato que reconhece o padrão P = "teste" e permite inserção



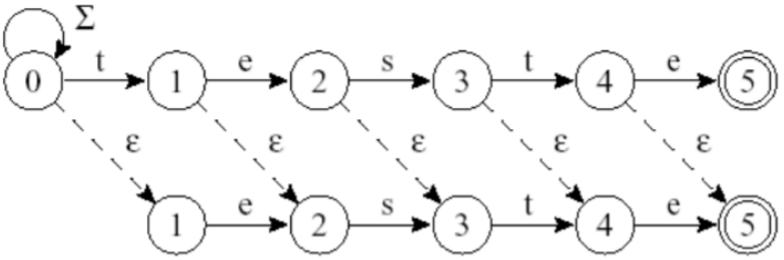
Duas arestas, uma Horizontal, representando o casamento de caractere, avançando-se no texto T e no padrão P, e uma vertical, que insere um caractere no padrão P, avançando-se no texto T mas não no padrão P.

Ex: Autômato que reconhece o padrão P = “teste” e permite substituição



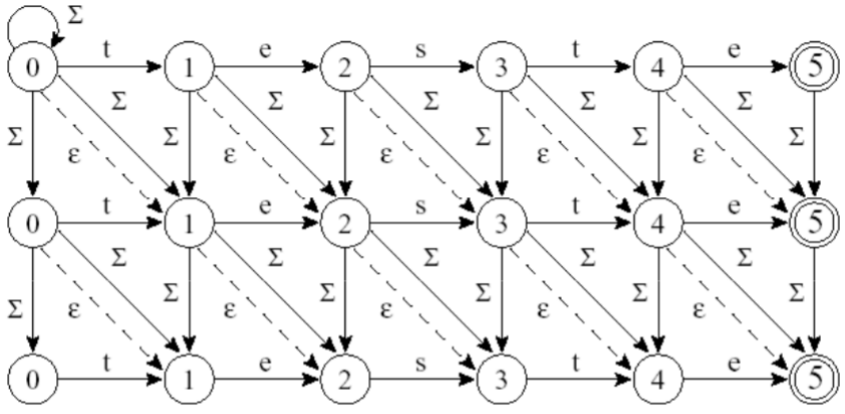
Duas arestas, uma Horizontal, representando o casamento de caracteres, avançando-se no texto T e no padrão P, e uma diagonal, que substitui um caractere no padrão P, avançando-se no texto T e no padrão P.

Ex: Autômato que reconhece o padrão P = “teste” e permite retirada



Duas arestas, uma Horizontal, representando o casamento de caractere, avançando-se no texto T e no padrão P, e uma tracejada, que retira um caractere no padrão P, avançando-se no padrão P mas não no texto T (transição vazia).

Ex: Autômato que reconhece o padrão 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).

O algoritmo Shift-End aproximado simula um autômato não-determinista, usando de paralelismo de bit, empacotando cada linha j (0 < j <= k) do autômato não-determinista. Para cada caractere lido, todas as transações do autômato são simulados usando operações entre as k+1 máscaras de bits.

Para o algoritmo, consideramos uma máscara R_0 que é inicializada com $R_0 = 0^m$. Para $(0 < j \leq k)$, R_j é inicializada como $R_j = 1^j 0^{m-j}$. Além disso, consideramos M a tabela do Shift-End exato a ser criada para cada novo caractere t_{i+1} lido do texto. As máscaras são atualizadas pelas seguintes expressões:
$R'0 = ((R0 >> 1) \mid 10^{-1}) \& M[T[i]]$;
Para $(0 < j \leq k)$;
$R'j = ((R_j >> 1) \& M[T[i]]) \mid R_j\text{-}1 \mid (R_j\text{-}1 >> 1) \mid (R'_j\text{-}1 >> 1) \mid 10^m{-}1$

Exemplo de execução do algoritmo:

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***	Casamento exato
s	1 0 1 0 0	0 0 1 0 0	0 1 1 0 1	1 1 1 0 ***1***	Casamento aproximado
	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***	Casamento aproximado
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	

Implementação do algoritmo

```
void ShiftEndAproximado ( 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-1);
   }

   R[0] = 0;
   Ri = 1 << (m - 1);

   for (j = 1; j <= k; j++) R[j] = (1 << (m - 1)) | 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 posição %12ld\n", i+1);
   }
}
```