



Resumo III

Bernardo Cavanellas Biondini - 20.1.4112

Nicolle Canuto Nunes - 20.1.4022

Casamento de Cadeias

cadeias de caracteres: sequência de elementos do conjunto alfabeto. em uma cadeia de bits, o alfabeto é $[0,1]$

O problema do casamento de cadeias/casamento de padrão consiste em encontrar todas as ocorrências de um padrão em um texto. Suas aplicações mais comuns são: edição de texto e recuperação de informações.

texto: cadeia $T[0...n-1]$ de tamanho n . está dentro de um determinado arquivo e é transferido para a memória principal

padrão: cadeia $P[0...m-1]$ de tamanho $m \leq n$

os elementos de P e T são escolhidos de um alfabeto finito de tamanho c

casamento de cadeias: dadas as cadeias P e T , deseja-se saber as ocorrências de P em T

```
#define MAXTAMTEXTO 1000
#define MAXTAMPADRAO 10
#define MAXCHAR 256
#define MAXERROS 10 // usado apenas no casamento APROXIMADO

typedef char TTexto[MAXTAMTEXTO];
typedef char TPadrao[MAXTAMPADRAO];
```

Categorias de Algoritmos

1. Força Bruta: P e T não são pré-processados

inicialmente, P e T não são reconhecidos

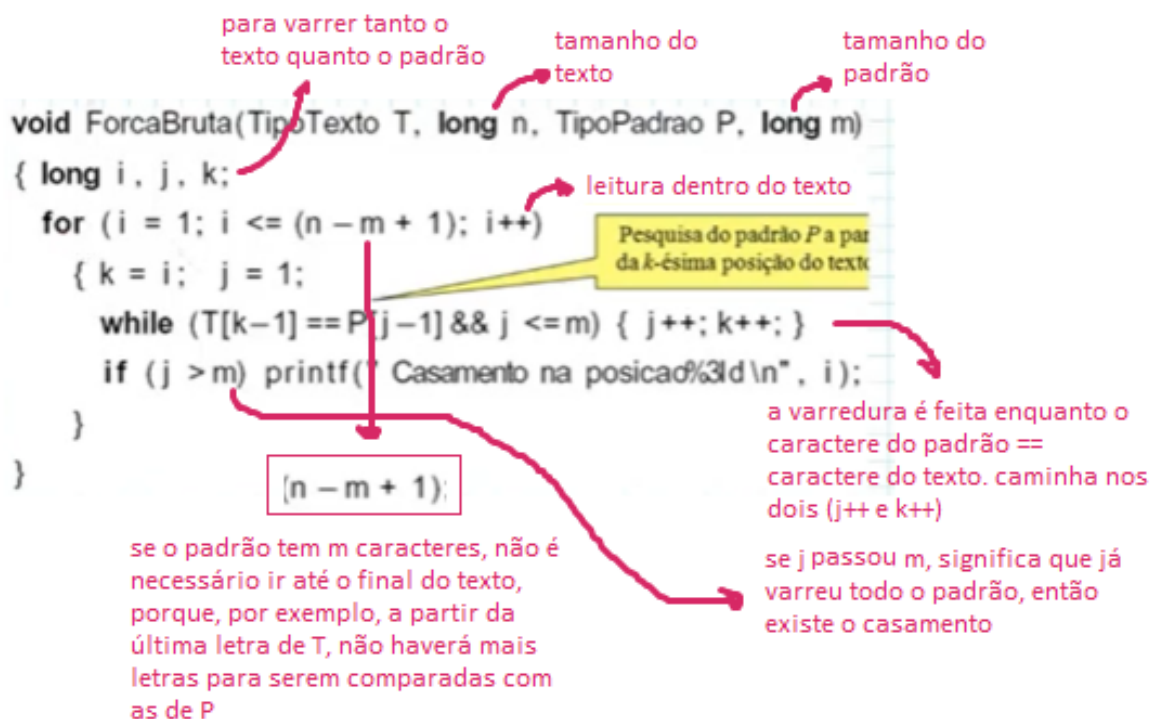
algoritmo sequencial, online, de tempo real

tempo = $O(mn)$

espaço = $O(1)$, pois não há pré-processamento

força bruta

algoritmo mais simples para casamento exato de cadeiras. tenta casar qualquer subcadeia de comprimento m no texto com o padrão buscado



exemplo de busca com sucesso

deseja-se
buscar P
dentro de T

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

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

compara-se o primeiro
caractere de P com o
primeiro de T

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

como são distintos,
passamos para o
próximo caractere de T

```
T (n) = os_testes testam ...  
P (m) = teste
```

como são distintos,
passamos para o
próximo caractere de
T

```
T (n) = os_testes testam ...  
P (m) = teste
```

quando uma igualdade é encontrada,
caminha-se no texto e no padrão,
comparando-se o próximo caractere

⋮

```
T (n) = os_testes testam ...  
P (m) = teste
```

atingimos o final de P, então
encontramos a cadeia
desejada

exemplo de busca sem sucesso



pior caso da força bruta

$T(n) = \text{aaaaaaaaaaaaaaaaaab}$
 $P(m) = \text{aaab}$

quando a diferenciação está apenas no último caractere

pois para cada caractere do T, é necessário refazer a varredura no P inteiro 🤔

complexidade = $O(nm)$

caso esperado da força bruta

depende muito da quantidade de caracteres do alfabeto. se for maior, é melhor. é bem melhor que o pior caso

$$(c/c - 1)(1 - 1/c^m)(n - m + 1) + O(1)$$

2. P é pré-processado

P é reconhecido, permitindo que seja pré-processado

algoritmo sequencial

um exemplo são programas de edição de texto

tempo = $O(n)$

espaço = $O(m+c)$

3. P e T são pré-processados

o algoritmo constrói índice para o texto

é interessante construir um índice quando a base de dados é grande e semi estática (atualizações em intervalos regulares)

o tempo para a geração do índice pode ser grande ($O(n)$ ou $O(n \log n)$), mas as operações de pesquisa são rápidas, então é compensatório

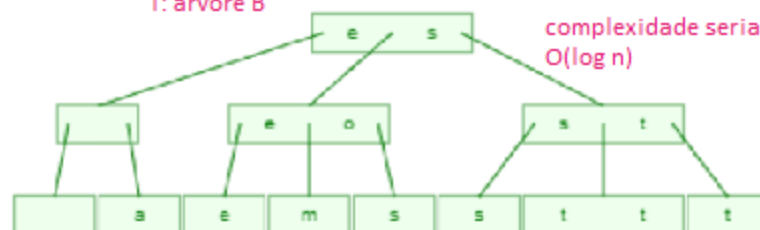
tipos de índices: arquivos invertidos, árvores TRIE, PATRICIA e arranjos de sufixos

tempo = $O(\log n)$

espaço = $O(n)$

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

exemplo de pré-
processamento de
T: árvore B



arquivo invertido

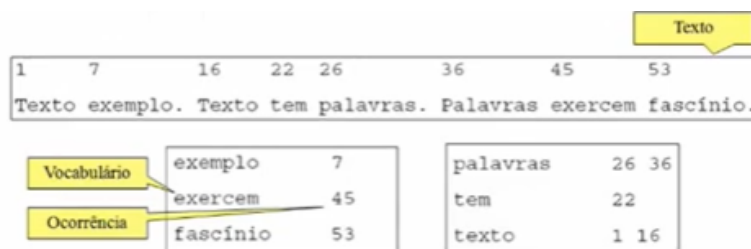
usado em casos de P e T sendo pré-processados

é composto por VOCABULÁRIO e por OCORRÊNCIAS

vocabulário: conjunto de palavras distintas

para cada palavra distinta, uma lista de posição onde ela está no texto é guardada.

esse conjunto de listas são as OCORRÊNCIAS

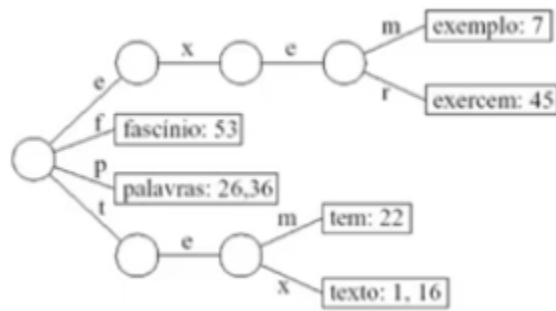


1. extrair o vocabulário. como a pesquisa começa pelo vocabulário, é interessante que ele esteja em um arquivo separado, esse arquivo geralmente cabe em memória principal. quando encontra a palavra, geralmente há um link (ponteiro na estrutura) para as ocorrências, estas, que, possivelmente, não caberão em memória principal.
2. listar ocorrências. geralmente ocupam mais espaço, por isso, são mantidas em um arquivo. geralmente são 30% a 40% do tamanho do texto

para buscar uma palavra:

pode ser realizada usando qualquer estrutura que torne a pesquisa eficiente, como:

- hashing: $O(m)$, sendo m o tamanho da consulta, independentemente do tamanho do texto
- árvore TRIE: $O(m)$, sendo m o tamanho da consulta, independentemente do tamanho do texto



árvore TRIE

- árvore B: $O(\log n)$, guardar as palavras na ordem lexicográfica é barato em termos de espaço e competitivo em desempenho

exemplo: palavra "tem"

1. verificar se existe no vocabulário do arquivo invertido, que pode ter sido implementado de diversas formas, como, por exemplo, em uma árvore B
2. caso exista, busca-se a lista de ocorrências, no caso, o número 22
3. acessa-se a posição 22 dentro do arquivo

para buscar uma frase:

é mais complicado, pois cada palavra da frase será pesquisada separadamente, para obter-se sua lista de ocorrências. a seguir, as listas são percorridas para encontrar as ocorrências das palavras em sequência

exemplo: frase "tem palavras"

1. pesquisa o "tem" e pega a lista de ocorrências
2. pesquisa a "palavras" e pega a lista de ocorrências
3. percorre-se a lista de ocorrências de forma sincronizada para verificar se elas estão seguidas e em ordem $[22 + 3 \text{ caracteres} + 1 \text{ espaço}] = 26$. logo, existe a frase no arquivo

lei de heaps

a medida que vamos preenchendo o arquivo invertido, a probabilidade de que o VOCABULÁRIO aumente é muito baixa, pois já foi preenchido com diversas palavras, podendo ser repetidas. porém, as OCORRÊNCIAS sempre aumentam, mesmo com palavras repetidas.

a lei de heaps gera uma previsão sobre o crescimento do tamanho do vocabulário.

O vocabulário de um texto em linguagem natural contendo n palavras tem tamanho

$$V = Kn^B = O(n^B)$$

onde K e B dependem das características de cada texto.

K = geralmente está entre 10 e 100

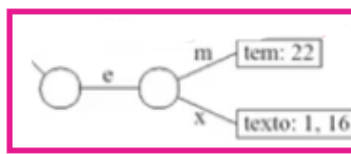
B = constante entre 0 e 1, geralmente entre 0,4 e 0,6

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

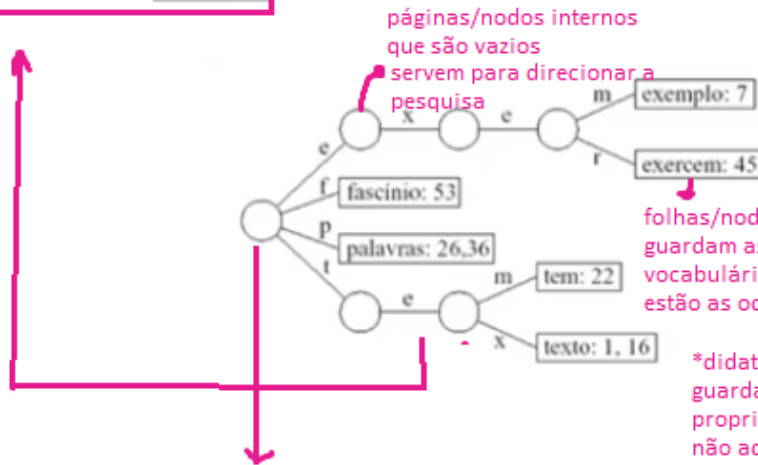
pesquisa em arquivo invertido

- pesquisa no vocabulário: palavras da consulta são isoladas e pesquisadas
- recuperação de ocorrências: as listas de ocorrências das palavras encontradas no vocabulário são recuperadas
- manipulação das ocorrências: as listas de ocorrências são processadas para tratar frases, proximidade e/ou operações lógicas

árvore TRIE



foi formado um galho que se ramifica onde as palavras se diferem:
TEm e TExto



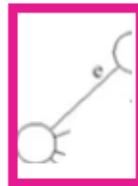
páginas/nodos internos
que são vazios
servem para direcionar a
pesquisa

folhas/nodos externos, que
guardam as palavras do
vocabulário e um link para onde
estão as ocorrências*

*didaticamente, aqui está sendo
guardado também as ocorrências
propriamente ditas, mas na prática isso
não acontece



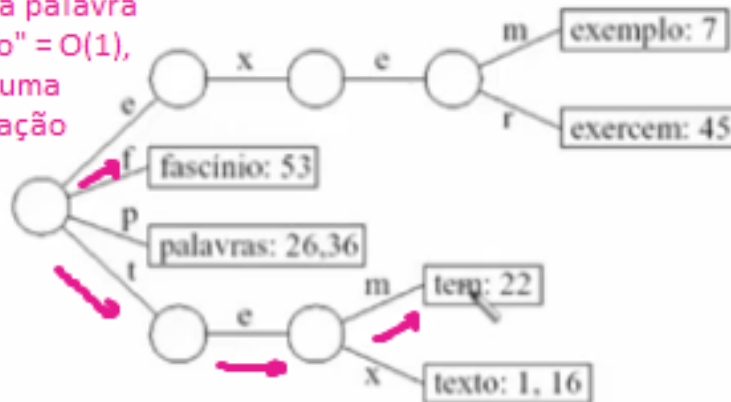
cada nó interno guarda
apontadores, que dizem
respeito a caracteres do
alfabeto



primeiro apontador: foram indexadas as
palavras com a letra E ["exemplo",
"exercem"]

busca em árvore TRIE

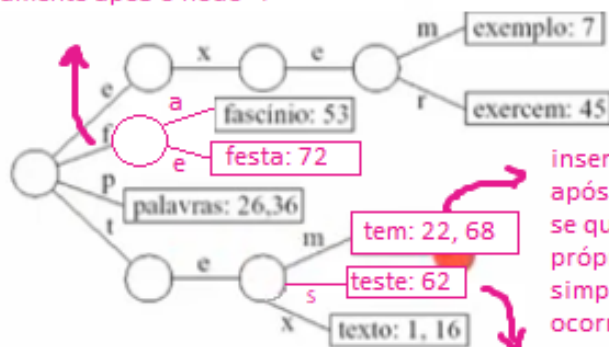
busca da palavra
"fascínio" = $O(1)$,
apenas uma
comparação



busca da palavra "TEM" = pior caso
 $O(m)$ onde m é o tamanho do padrão

inserção em árvore TRIE

inserção de "festa": verificou-se que, seguindo o
nodo "f", existia já uma folha. então, comparou-se
a folha "fascínio" com a folha a ser inserida.
verificou-se que se diferenciam logo na segunda
letra, portanto, criou-se uma ramificação
imediatamente após o nodo "f"



inserção de "tem": entra na ramificação "m"
após percorrer os nodos "t" e "e". verificou-
se que a folha existente corresponde a
própria palavra "tem", portanto,
simplesmente adiciona-se sua nova
ocorrência

inserção de "teste": verifica-se que
não existe a ramificação da letra "s"
após percorrer os nodos "t" e "e"
então adiciona-se a ramificação "s",
a palavra e sua ocorrência

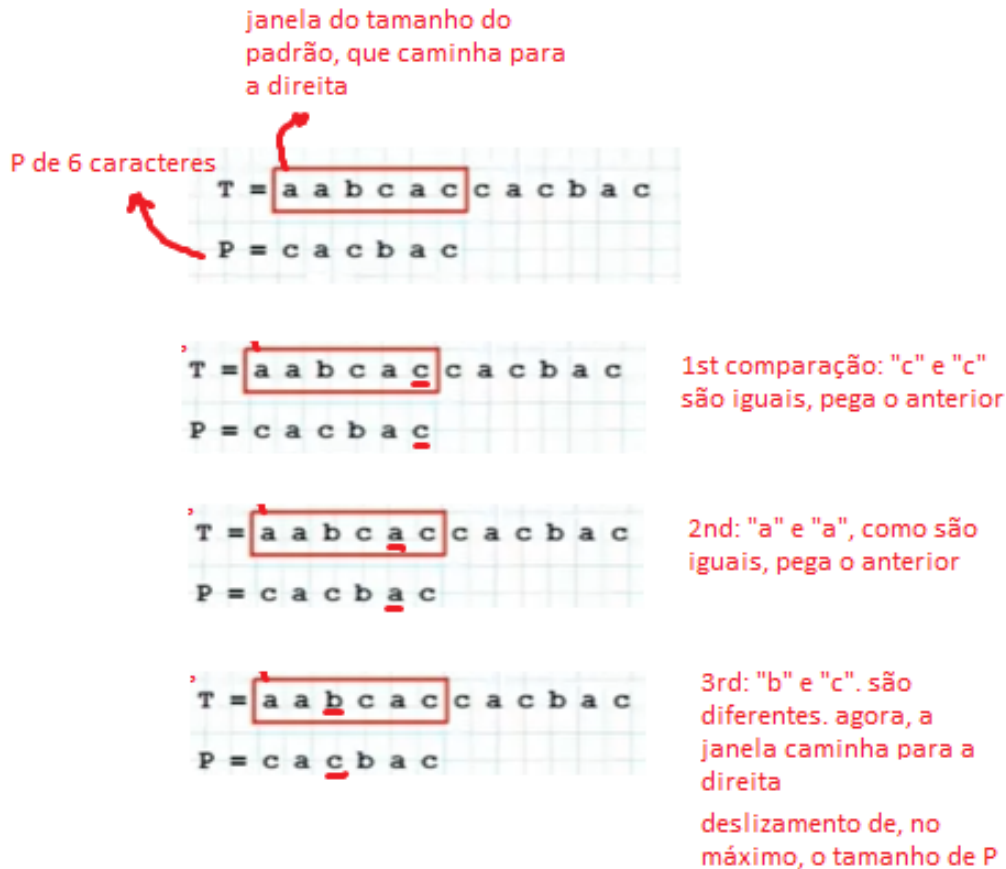
Casamento Exato

consiste em encontrar ocorrências exatas de um padrão em um texto

- leitura dos caracteres de T, um a um, buscando uma ocorrência possível de P - força bruta e shift-and
- pesquisa de P em uma janela que desliza ao longo de T, procurando um sufixo da janela (texto T) que casa com um sufixo de P, comparando da esquerda para a direita - Boyer-moore-horspool : padrão pré-processado

Algoritmo BM

- pesquisa o padrão P em uma janela que desliza ao longo de T
- para cada posição, o algoritmo pesquisa pelo sufixo dela, sufixo este que se case com um sufixo de P (final de P)
- comparações da direita para a esquerda
 - se não teve desigualdade, P foi encontrado
 - se não, calcula-se um deslocamento para a direita antes de tentar novamente



a partir daí, pode-se seguir dois caminhos: usando a heurística de ocorrência ou a heurística de casamento

Heurística Ocorrência: mais indicada

alinha o caractere no texto que causou a colisão com o 1º caractere no padrão, a esquerda do ponto de colisão, que casa com ele

pode ser melhor para alfabetos maiores

se logo no começo houver uma desigualdade, é o melhor caso, pois o deslocamento é do tamanho do padrão

T = aabcaccacbac
P = cacbac

janela do tamanho
de P

T = aabcaccacbac
P = cacbac

compara-se "c" com
"c": são iguais, pega
o caractere anterior

T = aabcaccacbac
P = cacbac

compara-se "a" com
"a": são iguais, pega
o caractere anterior

T = aabcaccacbac
P = cacbac

compara-se "c" com
"b": são diferentes,
houve uma colisão

agora a janela será deslocada para a
direita até encontrar o primeiro
elemento que seja igual ao "c"

busca o primeiro caractere A ESQUERDA no
PADRÃO, que seja igual ao DO TEXTO que
causou a colisão, que foi o "c"

se não existir,
deslização =
tamanho do padrão

T = aabcaccacbac
P = cacbac

pior caso:
deslocamento = 1

"c" está alinhado pois é o 3º
caractere da janela e também de P

T = aabcaccacbac
P = cacbac

compara-se "c" com
"c": são iguais, pega
o caractere anterior

T = aabcaccacbac
P = cacbac

compara-se "c" com "a": houve uma
colisão. agora, busca-se, à esquerda, o
primeiro caractere de P que seja igual
ao "c" (que foi quem causou a colisão)

T = aabcaccacbac
P = cacbac

deslocamento = 2
"c" está alinhado pois é o 3º
caractere da janela e também de P

T = aabcaccacbac
P = cacbac

T = aabcaccacbac
P = cacbac

T = aabcaccacbac
P = cacbac

T = aabcaccacbac
P = cacbac

pior caso:
deslocamento = 1

"c" está alinhado pois é o 3º
caractere da janela e também de P

T = aabcaccacbac
P = cacbac

T = aabcaccacbac
P = cacbac

deslocamento = 2
"b" está alinhado pois é o 4º caractere
da janela e também de P

Heurística Casamento

faz com que, ao mover a janela para a direita, ela deve coincidir com o pedaço do texto anteriormente casado

pode ser melhor pra alfabetos pequenos

se logo no começo houver uma desigualdade, é o pior caso, pois o deslocamento será 1

T = aabcacacbac
P = cacbac

comparou "c" e coincidiu.
comparou "a" e coincidiu.
"c" e "b" conflitaram

logo, a janela será deslocada
para a direita até casar com o
pedaço do texto que tinha dado
certo ("ac")

PIOR CASO:

se logo de cara houver uma colisão, a
janela será deslocada apenas uma
posição, pois não há nada para ser
buscado

T = aabcacacbac
P = cacbac

"c" e "b" conflitaram

logo, a janela será deslocada
para a direita até casar com o
pedaço do texto que tinha dado
certo ("ac")

* vai buscar "ac" dentro
do PADRAO

T = aabcacacbac
P = cacbac

o algoritmo BM decide qual heurística vai seguir, escolhendo a que houver maior deslocamento para o caso atual. ou seja, há comparações a cada colisão, o que aumenta o tempo de processamento

Algoritmo BMH

ideia de melhoria do algoritmo BM. é mais rápido. é simples e eficiente.

qualquer caractere já lido de T (a partir do último deslocamento) pode ser usado para endereçar uma tabela de deslocamentos

- deslocar a janela de acordo com o valor da tabela de deslocamento relativo ao caractere de T que está alinhado ao último caractere de P

como gerar a tabela de deslocamento

o valor inicial do deslocamento para todos os caracteres de T é m (tamanho de P)

em seguida, para os m-1 primeiros caracteres de P (todos, menos o último), os valores de deslocamento são:



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

é o menor valor de j tal que $j = m$ OU

o valor de j estando entre 1 e m e X seja o caractere que esteja na posição m-j do padrão

(para cada caractere de P, qual a distância do último caractere)

para o P "teste", sua tabela é:

$$d["t"] = 1$$

$$d["e"] = 3$$

$$d["s"] = 2$$

$$d["t"] = 1$$

$$d["e"] = 3$$

$d[x] = 5$ (valor de m). para todo caractere de T que não esteja em P

está a 4 de distância, mas é repetido. então pega-se o mínimo entre 4 e 1 = 1

está a 1 de distância do último caractere



esse é o PRE-PROCESSAMENTO do padrão: a criação da tabela de deslocamento

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

PRÉ PROCESSAMENTO: tabela inicializada com o valor de m

d[a] = 6

d[b] = 6

d[c] = 6

d[d] = 6

... para os n-1, calcula-se d

d[a] = min(1,4) = 1

d[b] = min(2) = 2

d[c] = min(3,5) = 3

d[d] = 6

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

tentou fazer o casamento normalmente. houve uma colisão de "c" com "b". pega-se o caractere do texto alinhado com o último do padrão

d[c] = 3, então a janela se locomove 3 posições

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

tentou fazer o casamento normalmente. houve uma colisão de "c" com "b". pega-se o caractere do texto alinhado com o último do padrão

d[c] = 3, então a janela se locomove 3 posições

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

casou

quando tem um caractere que não existe em P

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

$d[d] = 6$, pq d não existe no padrão

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

mais provável de acontecer quando o alfabeto for maior

algoritmo BMH

inicializa a tabela, deixando o deslocamento valendo m para todos os caracteres

qtd maxima de caracteres do alfabeto

tamanho do texto

tamanho do padrao

```
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)
    {
        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

i vai do tamanho do padrão até o tamanho do texto. comparações da direita pra esquerda

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

enquanto os caracteres coincidirem, j e k diminuem, realizando a comparação do final do padrão para o início dele

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).

se houve desigualdade e J não atingiu 0, significa que houve uma colisão. então calcula-se o novo valor de i, ou seja, quanto a janela irá deslocar para continuar o processo

pega todos os caracteres, menos o último e aplica a distância para o valor do deslocamento

já pega o mínimo, pois substitui conforme caminha para mais próximo do último caractere.

Algoritmo BMHS

há uma simplificação, 10 anos depois. é uma variante do BMH. geralmente tem resultados melhores em textos de linguagem natural.

em teoria, a quantidade de deslocamentos necessária é menor que no BHM, pois o tamanho dos deslocamentos no BMHS é maior

propões deslocar a janela de acordo com o valor da tabela de deslocamento relativo ao caractere no texto igual ao caractere após o último caractere do padrão, ao invés do último

agora o caractere causador do deslocamento é o caractere de T após o último caractere de P

valor inicial de deslocamento: não mais m, agora é m+1



para TODOS os caracteres de P, o deslocamento é:

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

padrão TESTE:

$$d["t"] = 2$$

$$d["e"] = 1$$

$$d["s"] = 3$$

$$d["t"] = 2$$

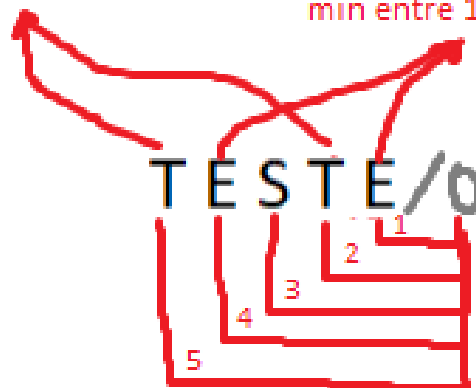
$$d["e"] = 1$$

$$d[x] = 6(m+1) \text{ para todo caractere de T que não exista em P}$$

houve uma melhora de +1 no deslocamento de todos os caracteres, menos o último, porque no BMH $d["e"]$ valia 3, agora vale 1, porque ele é o último

min entre 2 e 5 = 2

min entre 1 e 4 = 1



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

todos os deslocamentos foram
inicializados com 7. depois de calcular
ficou:

$d["a"] = \min(5, 2) = 2$
 $d["b"] = \min(3) = 3$
 $d["c"] = \min(1, 4, 6) = 1$
 $d["d"] = 7$
 ...

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

tentou fazer o casamento normalmente e houve
colisão no "c" e "b"

caractere seguinte ao alinhado = c
deslocamento dele é 1, pior caso :{

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

o causador do deslocamento fica alinhado
tentou casar e houve conflito em "c" com "a"
caractere seguinte ao alinhado = a, deslocamento 2

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

o causador do deslocamento fica alinhado
tentou casar e houve colisão em "c" e "b"
caractere seguinte ao alinhado = c, deslocamento 1

Algoritmo BMHS

a alteração do BMH para virar o BMHS, deve-se alterar:

```

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)
  { 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]];
  }
}

```

$m+1$
 Pré-processamento para se obter a tabela de deslocamentos

$j \leq m$ ou $j < m+1$, pois agora pega todos os caracteres
 $m+1-j$
 Pesquisa por um sufixo do texto (janela) que casa com um sufixo do padrão

$d[T[i-1]]$
 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).
 $d[T[i]]$, pois vai representar a posição seguinte ao último caractere do padrão

Shift-And

mais um dos algoritmos em que o padrão é pré-processado

simula o padrão por meio de um determinado autômato

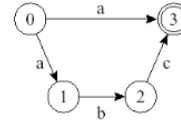
Autômato

modelo de computação muito simples

todo autômato reconhece uma linguagem, a linguagem é determinada pelo conjunto das cadeias que o autômato é capaz de reconhecer

■ 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 ($I \in Q$);
- F é o conjunto de estados finais ($F \subseteq Q$);
- Σ é o alfabeto finito de entrada;
- T é a função que define as transições entre os estados.
 - T associa a cada estado $q \in Q$ um conjunto de estados $\{q_1, q_2, \dots, q_k\} \subseteq Q$, para cada $\alpha \in (\Sigma \cup \{\epsilon\})$, onde ϵ é a transição vazia.



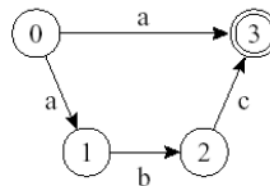
no caso em que será usado, o alfabeto representa o alfabeto de caracteres que dizem respeito aos padrões e textos considerados no casamento de cadeias

a função define para onde "caminhar" no autômato, as transições entre os estados

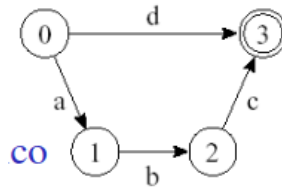
no exemplo acima o padrão representado seriam as cadeias "abc" ou "a"

o padrão desejado dentro de um texto vai ser a sequência de caracteres que serão representados pelo autômato que saem do estado inicial para o estado final

podemos ter autômatos deterministas e não-deterministas. Os autômatos deterministas são aqueles que dado uma determinada entrada, só é possível tomar um "caminho", enquanto que os autômatos não-deterministas são aqueles em que não se pode determinar qual a transição que será tomada, quando a função T permite mais de um "caminho" distinto a partir de uma entrada específica



este é um exemplo de autômato não-determinista, pois dado o caractere a, é possível tomar o caminho para o estado final ou o caminho para o estado 1.

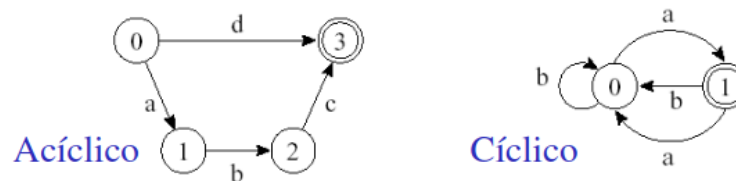


este é um exemplo de autômato determinista, pois só é possível ter um caminho dado o caractere. Caso ele seja a, o caminho seguirá para o estado 1, caso seja d seguirá para o estado final.

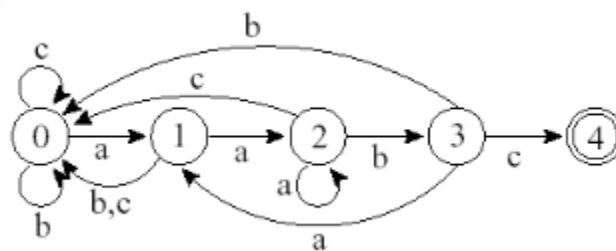
podemos ter transições vazias em que há a transição de estado sem que haja a "leitura" de um caractere, ou seja o caminharmento é feito independentemente do caractere de entrada.

um estado ativo de um autômato é o estado em que se encontra no caminharmento que representa a subcadeia que foi lida até o momento

existem autômatos cíclicos e acíclicos, enquanto que os autômatos acíclicos possuem representações de cadeias determinadas, os cíclicos podem assumir diversas cadeias, dependendo dos caracteres lidos



No shift-and utilizamos autômatos para reconhecer o padrão desejado dessa forma, o autômato deverá ter $m+1$ estados em que m é o tamanho do padrão
Exemplo: caso queiramos reconhecer o padrão $P = \{aabc\}$
o automato deve considerar todas as possíveis entradas dentro de um alfabeto



dado um texto T de alfabeto $\{a, b, c\}$, o autômato deve prever os caracteres que não irão satisfazer o casamento do padrão, ou seja, se o primeiro caractere lido for diferente de a , é necessário que seja lido outra caractere, até que este seja a , para seguir para o próximo estado, caso o segundo seja diferente de a , é preciso retornar ao estado inicial, caso seja a , podemos continuar o caminhamento. e assim por diante, respeitando o padrão desejado.

Complexidade

Tempo: $O(n)$ → cada caractere é lido uma vez

Espaço: $m+1$ para vértice → $m+1$ estados em que m é o tamanho do padrão desejado

$d \times m$ para arestas → a cada estado é preciso cobrir d caracteres em que d é o tamanho do alfabeto do texto

Considerando essa complexidade, a construção de um autômato acaba sendo uma desvantagem, uma vez que para alfabetos grandes a construção desse modelo pode ser custosa e demorada.

nesse sentido o autômato é simulado por uma sequência binária em que cada bit representa um estado do autônomo

aplicando operações binárias em cima dessa sequência é possível simular o caminhamento do autômato

Utiliza o conceito de paralelismo de bit

o padrão será representado por uma sequência de bits que representa um estado o autômato, a cada caractere lido será necessário realizar uma manipulação da sequência binária que possuem complexidade $O(1)$.

- Algumas operações sobre os *bits* de uma palavra são:
 - Repetição de *bits*: exponenciação (ex.: $01^3 = 0111$);
 - " $|$ ": operador lógico or;
 - " $\&$ ": operador lógico and;
 - " $>>$ ": operador que move os *bits* para a direita e entra com zeros à esquerda (ex.: $b_1 b_2 \dots b_{c-1} b_c >> 2 = 00b_1 \dots b_{c-2}$).

Shift-And Exato

Mantém o conjunto de todos os prefixos do padrão P que casam com o texto, o conjunto é representado por uma máscara de bits que representa os estados do autômato a partir do estado inicial, ou seja:

considerando o conjunto $R = (b_1 b_2 \dots b_m)$ b_1 é o estado de número 1

o algoritmo utiliza do paralelismo para atualizar a máscara a cada caractere lido representamos um estado ativo pelo valor 1 e um estado inativo pelo valor 0, portanto quando todos os bits estão inativos significa que estamos no estado inicial, quando o último bit está ativo, significa que encontramos nosso padrão P no texto.

É preciso fazer um pré-processamento sob a máscara para que esteja definido a sequência de bits para cada caractere do padrão

A máscara é inicializada como $R = 0^m$

para cada novo caractere t_{i+1} lido no texto o valor da máscara R' é atualizado pela função

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

** (a fórmula será melhor exemplificada adiante)

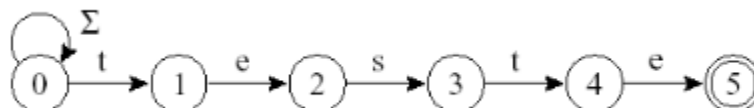
a primeira parte da fórmula $(R \gg 1)$ simula o caminhar do modelo, ela vai fazer os bits "andarem para a direita", colocando um 0 na frente da sequência

a segunda parte $((R \gg 1) \mid 10^{m-1})$ retrata o fato de a cadeia vazia ser marcada com um sufixo do padrão, permitindo o casamento em qualquer posição do texto

a terceira $((R \gg 1) \mid 10^{m-1}) \& M[T[i]]$, vai manter apenas a sequência referente ao estado em que o autômato se encontraria, uma vez que M apresenta a sequência de bits para todos os caracteres do padrão.

Exemplo:

O autônomo simplificado do padrão "teste"



o pré-processamento irá ocorrer para determinar uma máscara para 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

a máscara R no estado inicial teria o seguinte formato → (0 0 0 0 0)

quando o caractere primeiro "t" for lido → (1 0 0 0 0)

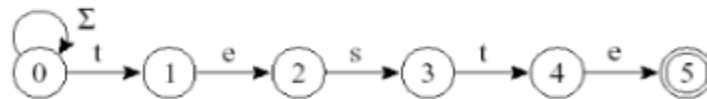
se o primeiro caractere "e" for lido → (0 1 0 0 0)

caso os próximos caracteres lidos estiverem dentro do padrão, chegando ao estado final, lendo o último caractere "e", o último estado estaria ativo, determinando o casamento → (0 1 0 0 1)

Passo a passo:

T = os testes testam...

P = teste



Leitura "o"

R = 00000

R' = 00000 | 10000 & 00000 = 00000

Leitura "s"

R = 00000

R' = 00000 | 10000 & 00100 = 00000

Leitura " "

R = 00000

R' = 00000 | 10000 & 00000 = 00000

Leitura "t"

R = 00000

R' = 00000 | 10000 & 10000 = 10000

Leitura "e"

R = 10000

R' = 01000 | 10000 & 01001 = 01000

Leitura "s"

R = 01000

R' = 00100 | 10000 & 00100 = 00100

A partir daqui o algoritmo começa a encontrar o início de uma outra ocorrência do padrão, por isso não só o estado 4 mas o 1 também fica ativo

Leitura "t"

R = 00100

R' = 00010 | 10000 & 10010 = 10010

Leitura "e"

R = 10010

R' = 01001 | 10000 & 01001 = 01001

Após a leitura do caractere "e" é determinado o casamento do padrão com o texto, uma vez que o último estado fica ativo, ou seja, o último bit da sequência é 1.

Leitura "s"

R = 01001

R' = 00100 | 10000 & 00100 = 00100

Leitura " "

R = 00100

R' = 00010 | 10000 & 00000 = 00000

Somente após a leitura do espaço em branco que é possível descartar a ocorrência de mais um padrão, pois a sequência ficaria (00000)

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

O esquema de implementação desse método

```
Shift-And ( $P = p_1p_2 \dots p_m, T = t_1t_2 \dots t_n$ )
{ /*—Préprocessamento—*/
  for ( $c \in \Sigma$ )  $M[c] = 0^m$ ; /*inicializa a mascara para todo elemento do alfabeto
  for ( $j = 1$ ;  $j \leq m$ ;  $j++$ )  $M[p_j] = M[p_j] | 0^{j-1}10^{m-j}$ ; /* aplica a fórmula para os caracteres
  /*—Pesquisa—*/                                     formados pelo padrão, montando a tabela
   $R = 0^m$ ; /*inicializa o R                               de máscaras
  for ( $i = 1$ ;  $i \leq n$ ;  $i++$ )                               /* para cada caractere do texto,
    {  $R = (R \gg 1 | 10^{m-1}) \& M[T[i]]$ ;                 vamos ler o caractere e aplicar a fórmula para R
      if ( $R \& 0^{m-1}1 \neq 0^m$ ) 'Casamento na posicao  $i - m + 1$ '; /* verifica se o ultimo bit é 1,
    }                                                         informando se houve ou não o
  }                                                         casamento
}
```

Complexidade:

$O(n)$, uma vez que é lido cada caractere do texto e considerando que as operações sobre as sequências de bits sejam realizadas em $O(1)$ e que o padrão caiba em umas poucas palavras do computador.

Shift-And Aproximado

mesma ideia do shif-and exato, porém permite que o casamento seja aproximado

Casamento Aproximado

encontra ocorrências aproximadas de um padrão no texto

deve tratar 3 tipos de operação:

- de Inserção: um caractere a mais entre os caracteres do texto
avança no texto mas não avança no padrão
- de Substituição: uma letra foi substituída por outra
avança no texto e no padrão
- de Retirada: falta de caracteres para fazer o casamento
não avança no texto mas avança no padrão

Distância de edição entre duas cadeias P e P' é o menor número de operações necessárias para converter P em P' ou vice-versa

Exemplo: $ed(\text{teste}, \text{estende}) = 4$ pois para converter "teste" em "estende",

precisamos substituir o "s" por "n" e o "t" pelo "d" e fazer a inserção dos caracteres "e" e "s", portanto 4 operações são feitas sobre a cadeia "teste" para convertê-la em "estende", da mesma forma que precisamos de 4 operações (2 de retirada e 2 de substituição) para converter "estende" em "teste".

O problema de casamento aproximado é encontrar todas as ocorrências de P' no texto tal que $ed(P, P') \leq k$, onde k representa o número máximo de operações de inserção, substituição e retirada para a conversão de P em P' .

O valor de k deve ser bem pensado, uma vez que para grandes quantidades de operações há um distanciamento maior da cadeia desejada, ou seja, se aceitarmos valores de k muito grandes, o casamento será menos assertivo pois estaremos possibilitando grandes quantidades de operações sobre a cadeia → se fizermos 5 operações de substituição em "teste" podemos obter "papel" como resultado, por exemplo.

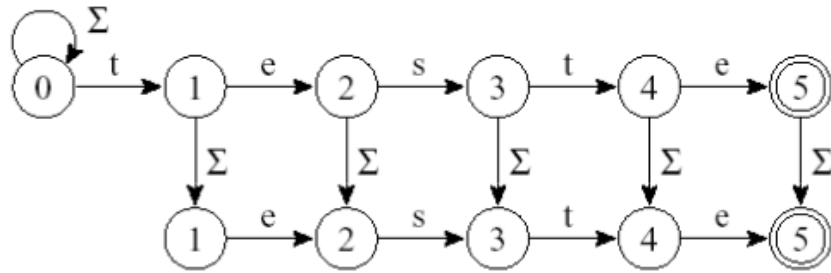
O casamento aproximado só faz sentido para $0 < k < m$ pois para $k = m$, qualquer cadeia de mesmo comprimento pode ser substituída por P e para $k = 0$ o casamento é exato.

Nível de erro adotado $a = k/m$, geralmente $a < 1/2$

A pesquisa com casamento aproximado é modelado por autômatos não-deterministas usando paralelismo de bits.

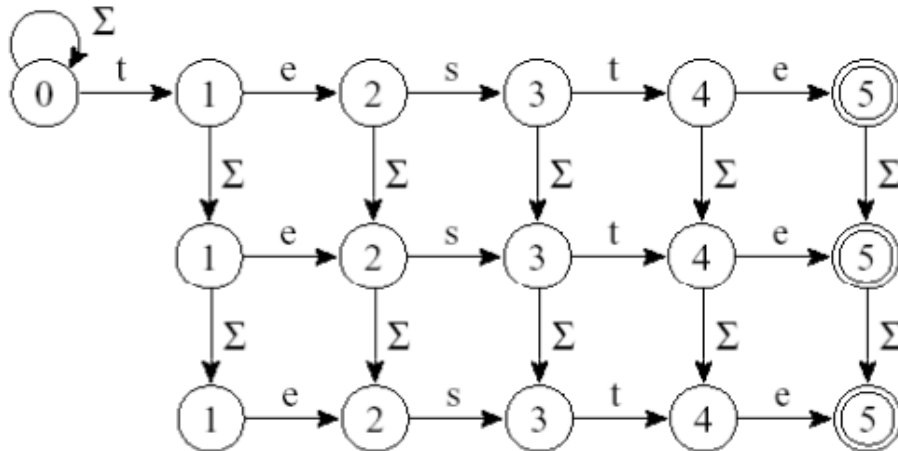
as representações dos autômatos agora precisam considerar as operações de substituição, inserção e retirada sobre a cadeia.

Autômato que reconhece "teste" permitindo uma inserção

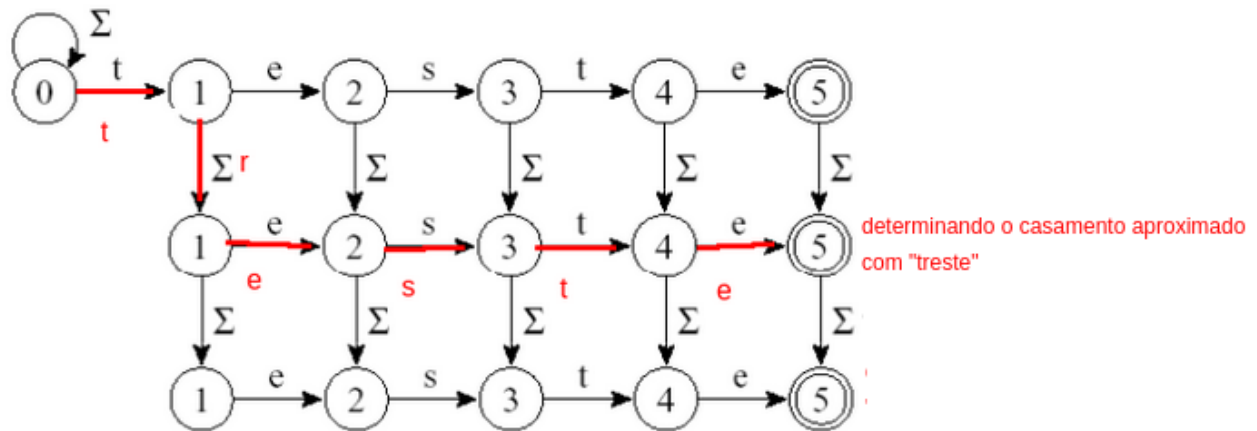


- Uma aresta horizontal representa um casamento de caractere, avançando-se no texto T e no padrão P .
- Uma aresta vertical insere um caractere no padrão P , avançando-se no texto T mas não no padrão P .

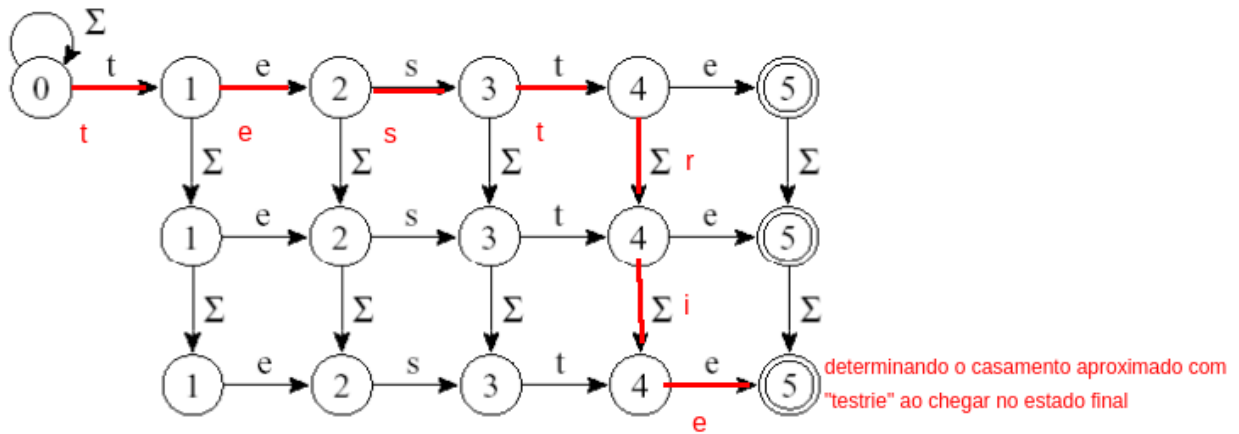
Nesse exemplo, o autômato permite uma inserção sobre o padrão. Caso pudessem ser admitidas outras inserções teríamos mais uma linha do autômato teste em que sairiam outras arestas verticais de cada estado da segunda linha, da seguinte forma:



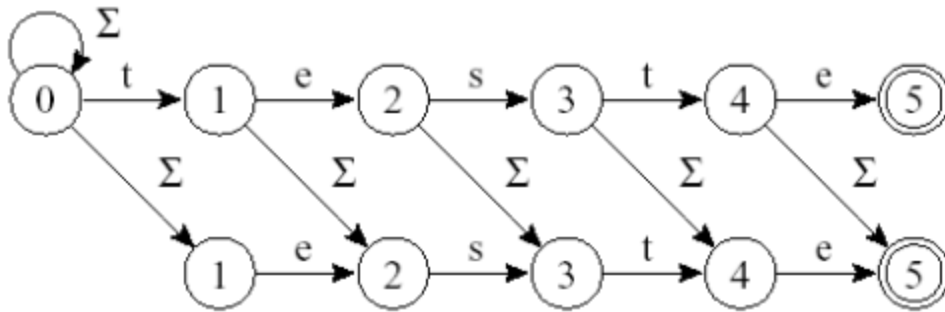
O caminhamento do autômato poderia ser simulado da seguinte forma:
considerando a leitura do texto $T = \text{"treste"}$



ou "testrie"

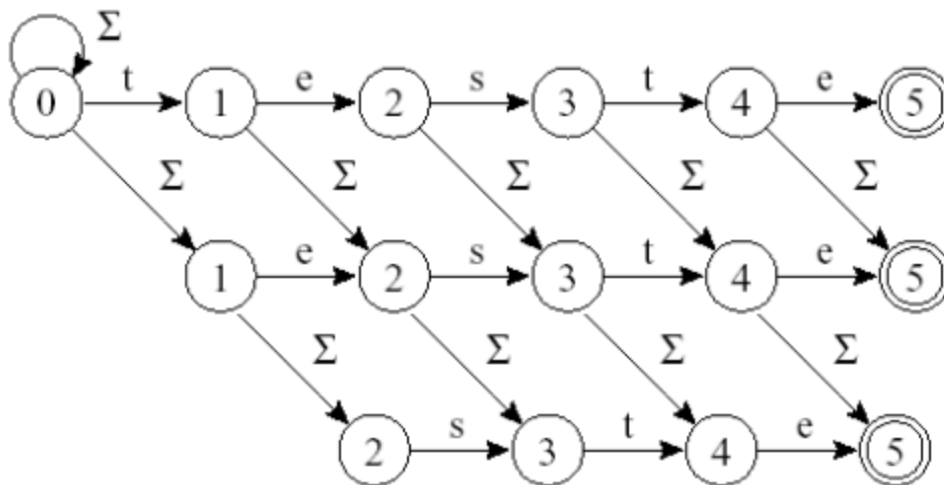


Autômato que reconhece "teste" permitindo uma substituição



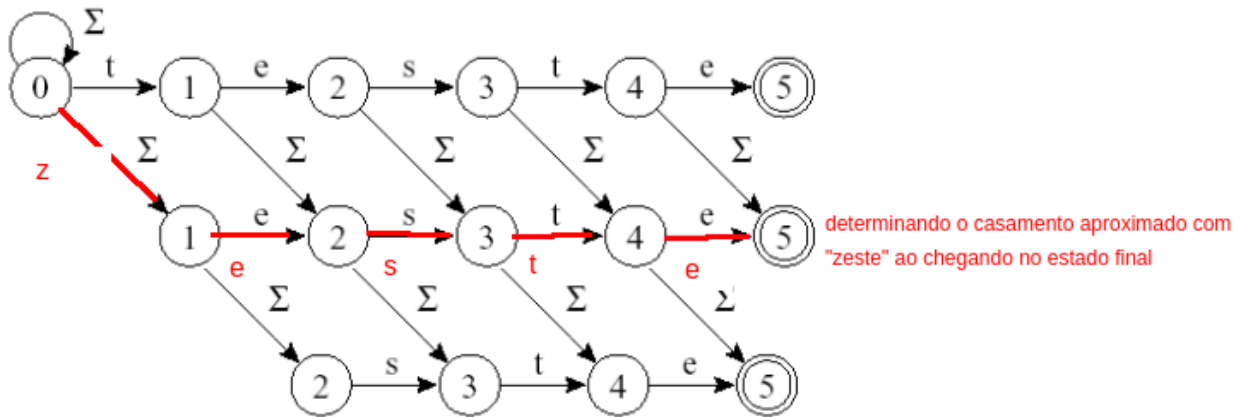
- Uma aresta horizontal representa um casamento de caractere, avançando-se no texto T e no padrão P .
- Uma aresta diagonal substitui um caractere no padrão P , avançando-se no texto T e no padrão P .

Nesse exemplo, o autômato permite uma substituição sobre o padrão. Caso pudessem ser admitidas outras substituições teríamos mais uma linha do autômato teste em que sairiam outras arestas diagonais de cada estado da segunda linha, da seguinte forma:

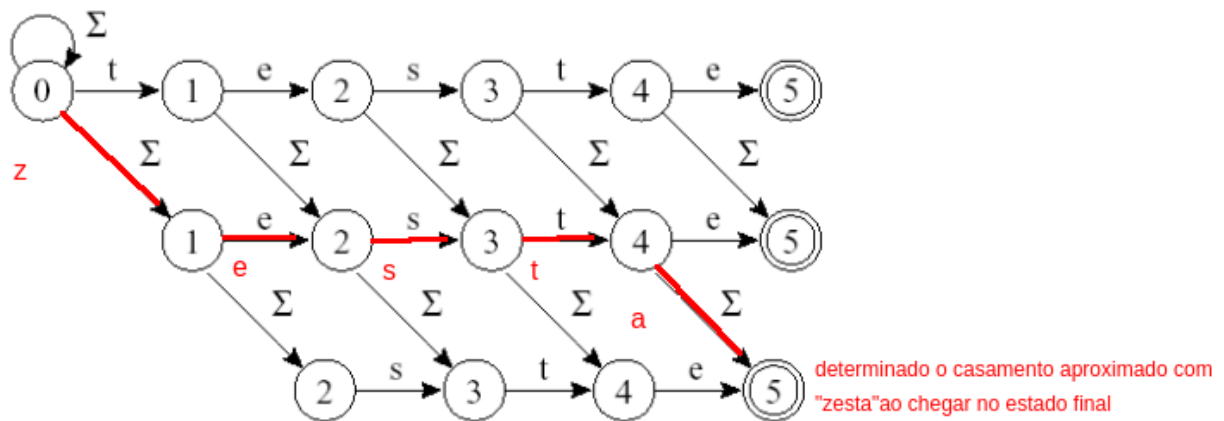


o caminhamento do autômato poderia ser simulado da seguinte forma:

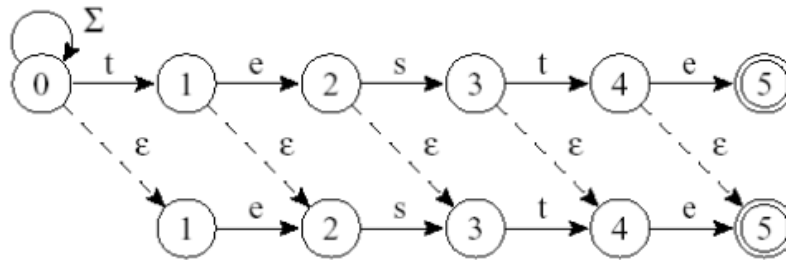
considerando a leitura do texto $T = \text{"zeste"}$



ou "zesta"

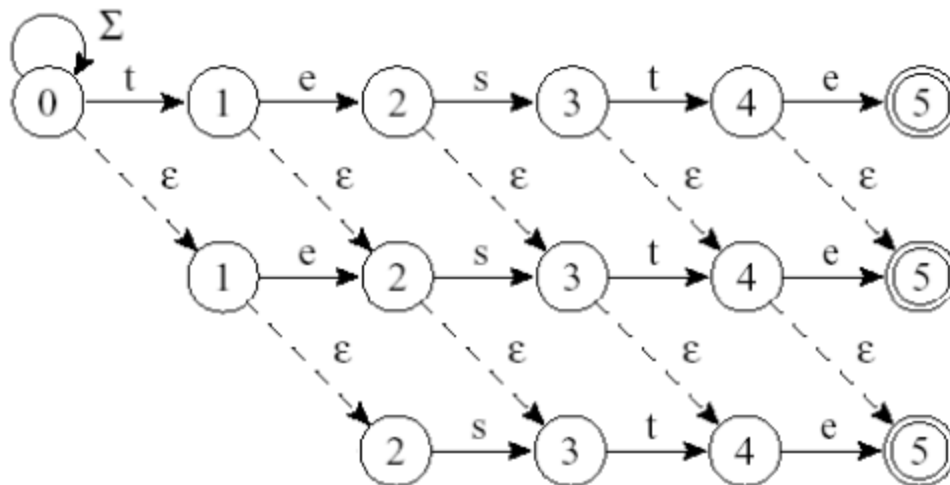


Autômato que reconhece "teste" permitindo uma retirada.



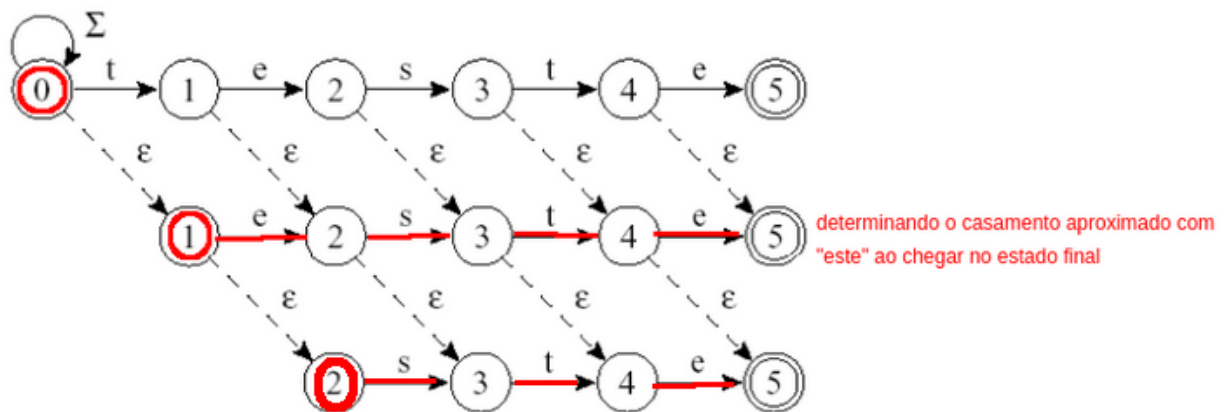
- Uma aresta horizontal representa um casamento de caractere, avançando-se no texto T e no padrão P .
- Uma aresta diagonal tracejada retira um caractere no padrão P , avançando-se no padrão P mas não no texto T (transição vazia).

Nesse exemplo, o autômato permite uma retirada sobre o padrão, o ignorando. Isso ocorre pois existe mais de um estado ativo que vai fazendo o caminhamento a partir da leitura dos caracteres. Caso pudessem ser admitidas outras retiradas teríamos mais uma linha do autômato teste em que sairiam outras arestas diagonais tracejadas de cada estado da segunda linha, da seguinte forma:

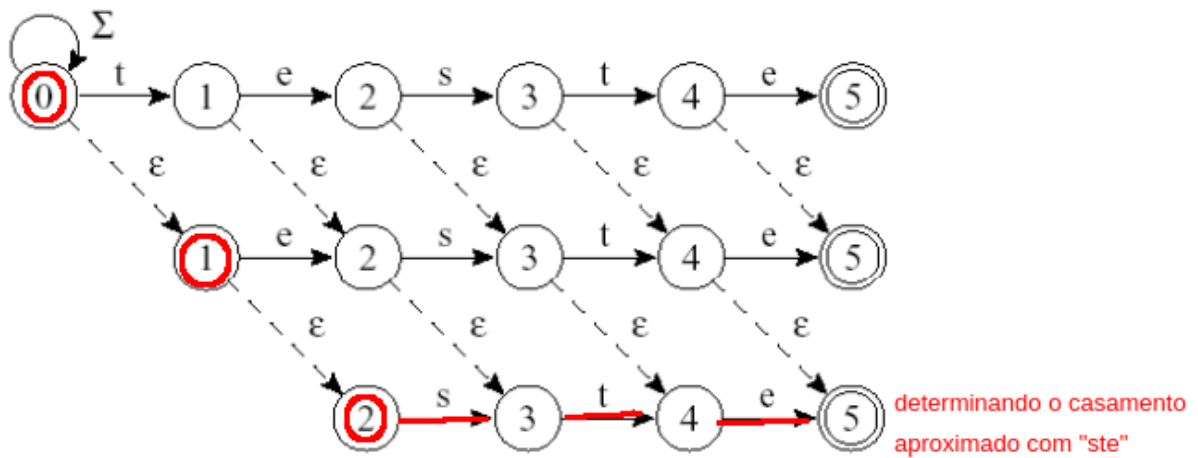


o caminhamento do autômato poderia ser simulado da seguinte forma:

considerando o texto $T = \text{"este"}$

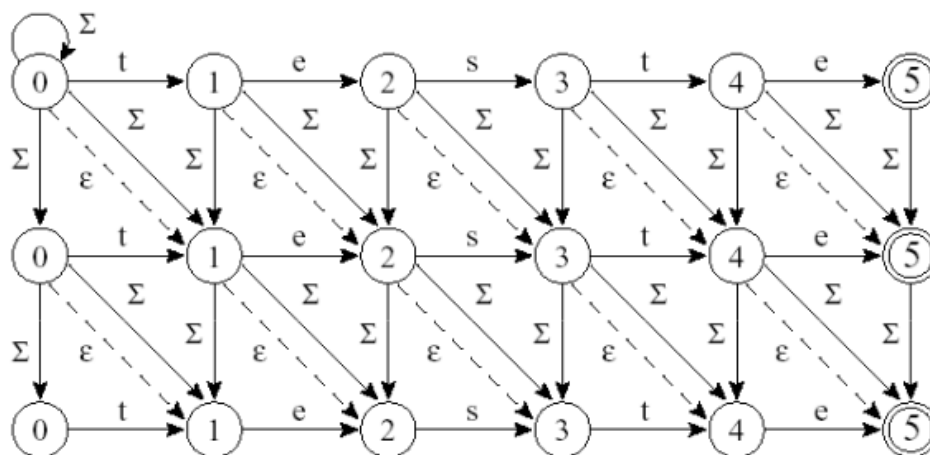


ou "ste"



Autômato que reconhece "teste" permitindo retiradas e/ou substituições e/ou inserções

considerando $k = 2$



■ 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 ($k = 2$).

portanto, o autômato pode determinar o casamento aproximado com textos permitindo até 2 erros, como "peste", "tesla", "testa", "triste" entre vários outros.

Como agora temos mais estados, precisamos conseguir representá-los pelas máscaras. Dessa forma, teremos uma quantidade de máscaras igual a quantidade de $k+1$, para cobrir k possíveis erros mais o casamento exato. No exemplo acima teríamos, por exemplo uma máscara R_0 para representar o casamento exato, uma máscara R_1 para representar o casamento com até 1 erro e uma máscara R_2 para representar o casamento com até 2 erros.

Além disso a inicialização das máscaras deve ser diferente para cada R_n , uma vez que o autômato precisa "prever" erros, precisando "deixar" mais de um estado ativo. A máscara terá seu valor iniciado de acordo com a quantidade k de erros referente a ela. No exemplo acima R_0 será iniciado com todos os estados inativos, enquanto a R_1 será iniciada com o primeiro estado ativo e o resto inativo (já que R_1 admite 1 erro) e R_2 com os estados 1 e 2 ativos (já que R_2 admite 2 erros).

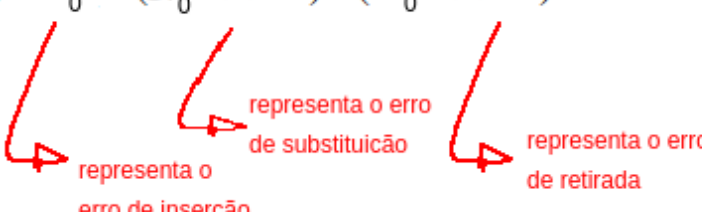
- A máscara R_0 é inicializada como $R_0 = 0^m$
- Para $0 < j \leq k$, R_j é inicializada como $R_j = 1^j 0^{m-j}$
- Considerando M a tabela do algoritmo Shift-And para casamento exato, para cada ovo caractere t_{i+1} lido do texto, as máscaras são atualizadas pelas expressões:

- $R'_0 = ((R_0 \gg 1) \mid 10^{m-1}) \& M[T[i]]$
- Para $0 < j \leq k$,
 $R'_j = ((R_j \gg 1) \& M[T[i]]) \mid R_{j-1} \mid (R_{j-1} \gg 1) \mid (R'_{j-1} \gg 1) \mid 10^{m-1}$

R'_0 é igual a fórmula vista no casamento exato o que muda é para as máscaras que representam casamento com erros

- Considerando o autômato, a fórmula para R' expressa as arestas:
 - horizontais, indicando casamento de um caractere;
 - verticais, indicando inserção (R_{j-1});
 - diagonais cheias, indicando substituição ($R_{j-1} \gg 1$);
 - diagonais tracejadas, indicando retirada ($R'_{j-1} \gg 1$).

considerando valor de $k = 1$, teremos o R_1 e o R'_1 que seria igual a:

$$R'_1 = ((R_1 \gg 1) \& M[T[i]]) \mid R_0 \mid (R_0 \gg 1) \mid (R'_0 \gg 1) \mid 10^{m-1}$$


Na inserção a gente avança no texto e não avança no padrão, ou seja, não executamos um shift, apenas pegamos o valor anterior da máscara (casamento exato antes de ser transformado)

Na substituição a gente avança no padrão e avança no texto, portanto precisamos fazer um shift em relação ao valor da máscara anterior

Na retirada, a gente não avança no texto mas avança no padrão, portanto precisamos avançar no texto transformado, fazendo um shift no R' anterior

Caso algum dos erros não seja desejado é só retirar a fórmula referente àquela operação

Observação: nas fórmulas de inserção e substituição será preciso usar o valor anterior de R, portanto, é necessário que, na implementação, a gente "guarde" esse valor, para conseguirmos utilizá-lo depois, antes de transformá-lo em R'

Verificamos o casamento aproximado quando o último bit for 1, assim como ocorre no casamento exato

Exemplo de Shift-And Aproximado:

considerando o texto T = "os testes testam..." e o padrão P = "teste"

considerando somente erros de inserção

considerando k = 1

	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

R0 = 00000

R1 = 10000

- $R'_0 = ((R_0 \gg 1) \mid 10^{m-1}) \& M[T[i]]$
- $R'_1 = ((R_1 \gg 1) \& M[T[i]]) \mid R_0 \mid 10^{m-1}$

Leitura "o"

R0 = 00000 | 10000 & 00000 = 00000

R1 = 01000 & 00000 | 00000 | 10000 = 10000

Leitura "s"

R0 = 00000 | 10000 & 00100 = 00000

R1 = 01000 & 00100 | 00000 | 10000 = 10000

Leitura " "

R0 = 00000 | 10000 & 00000 = 00000

R1 = 01000 & 00000 | 00000 | 10000 = 10000|

Leitura "t"

R0 = 00000 | 10000 & 10010 = 10000

R1 = 01000 & 10010 | 10000 | 10000 = 10000|

Leitura "e"

R0 = 01000 | 10000 & 01001 = 01000

R1 = 01000 & 01001 | 10000 | 10000 = 11000|

O primeiro 1 da sequência após a leitura do "e" representa o casamento com erro, que sempre começa com 1

Leitura "s"

R0 = 00100 | 10000 & 00100 = 00100

R1 = 01100 & 00100 | 01000 | 10000 = 11100

O segundo 1 da sequência, após a leitura do "s", representa uma tentativa de casamento aproximado com erro de inserção, que será concretizado nas próximas leituras

Leitura "t"

R0 = 00010 | 10000 & 10010 = 10010

R1 = 01110 & 10010 | 00100 | 10000 = 10110

O segundo 1 da sequência após a leitura de "t" representa a tentativa de inserção e o terceiro representa o casamento exato

Leitura "e"

R0 = 01001 | 10000 & 01001 = 01001

R1 = 01011 & 01001 | 10010 | 10000 = 11011|

O penúltimo 1 representa uma tentativa futura de inserção

Após a leitura do "e", o programa identificou a inserção de 1 elemento, no caso, o espaço em branco à frente da palavra "teste", fazendo o casamento com " teste" e, ainda, começa a encontrar uma outra ocorrência do padrão

Leitura "s"

R0 = 00100 | 10000 & 00100 = 00100

R1 = 01101 & 00100 | 01001 | 10000 = 11101

O programa identificou com a leitura do "s", um novo casamento aproximado, com "testes"

O programa está considerando o espaço

Leitura " "

$R_0 = 00010 \mid 10000 \& 00000 = 00000$

$R_1 = 01110 \& 00000 \mid 00100 \mid 10000 = 10100$

em branco como uma inserção pois o padrão está sendo encontrado novamente, uma vez que os últimos caracteres lidos foram "t", "e" e "s"

Leitura "t"

$R_0 = 00000 \mid 10000 \& 10010 = 10000$

$R_1 = 01010 \& 10010 \mid 00000 \mid 10000 = 10010$

Leitura "e"

$R_0 = 01000 \mid 10000 \& 01001 = 01000$

$R_1 = 01001 \& 01001 \mid 10000 \mid 10000 = 11001$

O programa encontrou outro casamento aproximado com "tes te"

E continua ainda para os próximos caracteres do texto

As sequência do exemplo seguem a seguinte forma:

Texto	$(R_0 \gg 1) \mid 10^m - 1$	R'_0	$R_1 \gg 1$	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	

O esquema de implementação do método:

```

void Shift-And-Aproximado ( $P = p_1p_2 \dots p_m, T = t_1t_2 \dots t_n, k$ )
{
    /*--- Préprocessamento---*/
    for ( $c \in \Sigma$ )  $M[c] = 0^m$ ;    ->inicializa todos os caracteres do alfabeto com 0's
    for ( $j = 1; j \leq m; j++$ )  $M[p_j] = M[p_j] | 0^{j-1}10^{m-j}$ ;    -> calcula a máscara dos caracteres do padrão
    /*--- Pesquisa---*/
    for ( $j = 0; j \leq k; j++$ )  $R_j = 1^j0^{m-j}$ ;    -> Inicializa os R's
    for ( $i = 1; i \leq n; i++$ ) -> para cada caractere do texto
    {
        Rant =  $R_0$ ;    -> guarda  $R_0$  antes de ser transformado
        Rnovo =  $((Rant \gg 1) | 10^{m-1}) \& M[T[i]]$ ;    -> calcula o casamento exato
         $R_0 = Rnovo$ ;
        for ( $j = 1; j \leq k; j++$ ) -> para cada valor de k, calcular o R'
        {
            Rnovo =  $((R_j \gg 1 \& M[T[i]]) | Rant | ((Rant | Rnovo) \gg 1))$ ;    -> representa a formula para R'
            Rant =  $R_j$ ;    -> guarda o valor de R antes de ser mudado, para ser utilizado depois
             $R_j = Rnovo | 10^{m-1}$ ;    -> R'
        }
        if  $((Rnovo \& 0^{m-1}1 \neq 0^m)$  'Casamento na posicao  $i$ '; -> verifica se o 1 apareceu no final determinando ou não o casamento
    }
}

```

Em C, implementamos da seguinte forma;


```

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);
  }
}

```