Casamento de Cadeias

Estrutura de Dados II (BCC203)

Prof. Guilherme Tavares de Assis

Universidade Federal de Ouro Preto – UFOP

Instituto de Ciências Exatas e Biológicas – ICEB

Departamento de Computação – DECOM

Introdução

- Cadeia de caracteres: sequência de elementos denominados caracteres.
  - Os caracteres são escolhidos de um conjunto denominado alfabeto. Por exemplo, em uma cadeia de *bits*, o alfabeto é {0, 1}.
- O problema de casamento de cadeias (casamento de padrão) consiste em encontrar todas as ocorrências de um determinado padrão em um texto.
- Alguns exemplos de aplicação são:
  - edição de texto;
  - recuperação de informação;
  - estudo de sequências de DNA em biologia computacional.

Introdução

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

Estrutura de Dados

```
#define MAXTAMTEXTO 1000

#define MAXTAMPADRAO 10

#define MAXCHAR 256

#define NUMMAXERROS 10

typedef char TipoTexto[MAXTAMTEXTO];

typedef char TipoPadrao[MAXTAMPADRAO];
```

Categorias de Algoritmos

- *P* e *T* não são pré-processados:
  - Padrão e texto não são conhecidos a priori.
  - Algoritmo sequencial, on-line e de tempo-real.
  - Complexidade de tempo: $O(mn)$.
  - Complexidade de espaço: $O(1)$.

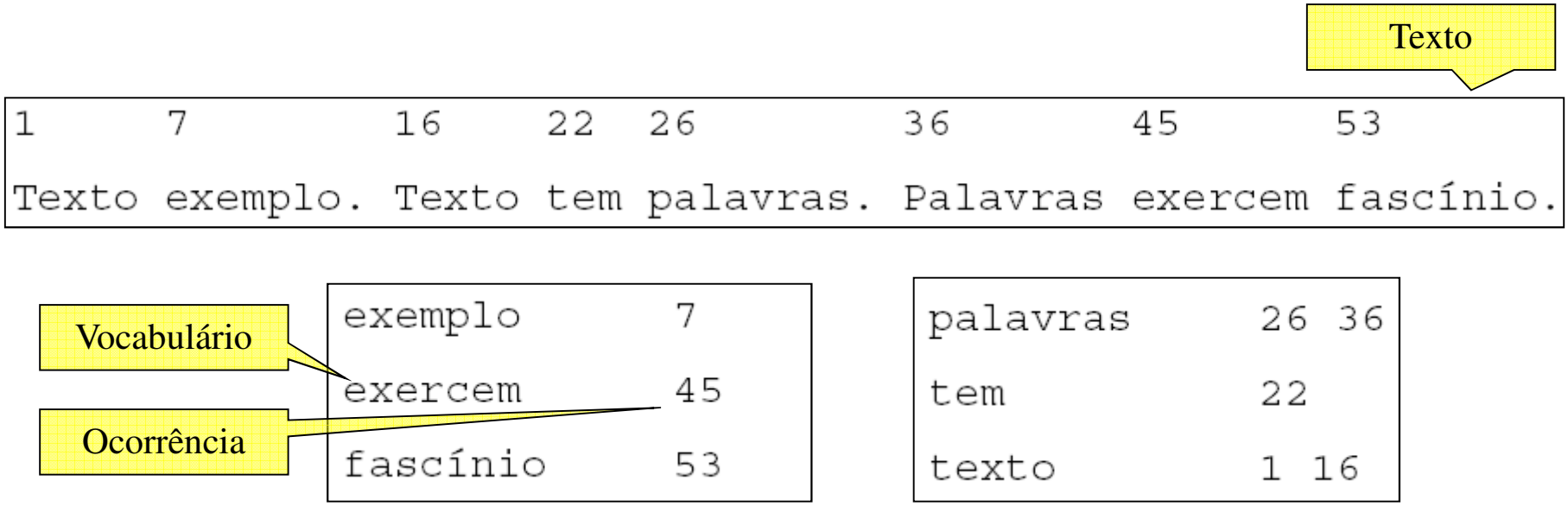
- *P* pré-processado:
  - Padrão conhecido a priori, permitindo seu pré-processamento.
  - Algoritmo sequencial.
  - Complexidade de tempo: $O(n)$.
  - Complexidade de espaço: $O(m + c)$.
  - Exemplo de aplicação: programas para edição de textos.

Categorias de Algoritmos

- *P* e *T* são pré-processados:
  - Padrão e texto são conhecidos a priori.
  - 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).
    - 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 tipos de índices são:
    - arquivos invertidos;
    - árvores TRIE e árvores PATRICIA;
    - arranjos de sufixos.
  - Complexidade de tempo: $O(\log n)$.
  - Complexidade de espaço: $O(n)$.

Arquivo Invertido

- Um arquivo invertido é composto por vocabulário e ocorrências.
  - O vocabulário é o conjunto das palavras distintas no texto.
  - Para cada palavra distinta, uma lista de posições onde ela ocorre no texto é armazenada; o conjunto das listas é chamado de ocorrências.



Arquivo Invertido

- O vocabulário ocupa pouco espaço em relação às ocorrências.
- A previsão sobre o crescimento do tamanho do vocabulário é definida pela lei de *Heaps*.
  - O vocabulário de um texto em linguagem natural contendo n palavras tem tamanho $V = Kn^{\beta} = O(n^{\beta})$, onde K e β dependem das características de cada texto.
  - K geralmente assume valores entre 10 e 100, e β é uma constante entre 0 e 1 (na prática entre 0,4 e 0,6).
  - Na prática, o vocabulário cresce com o tamanho do texto, em uma proporção perto de sua raiz quadrada.
- As ocorrências ocupam bem mais espaço.
  - O espaço necessário é $O(n)$ já que cada palavra é referenciada uma vez na lista de ocorrências.
  - Na prática, o espaço fica entre 30% e 40% do tamanho do texto.

Arquivo Invertido

- A pesquisa é realizada em três passos:
  - Pesquisa no vocabulário: palavras da consulta são isoladas e pesquisadas no vocabulário.
  - Recuperação das 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.

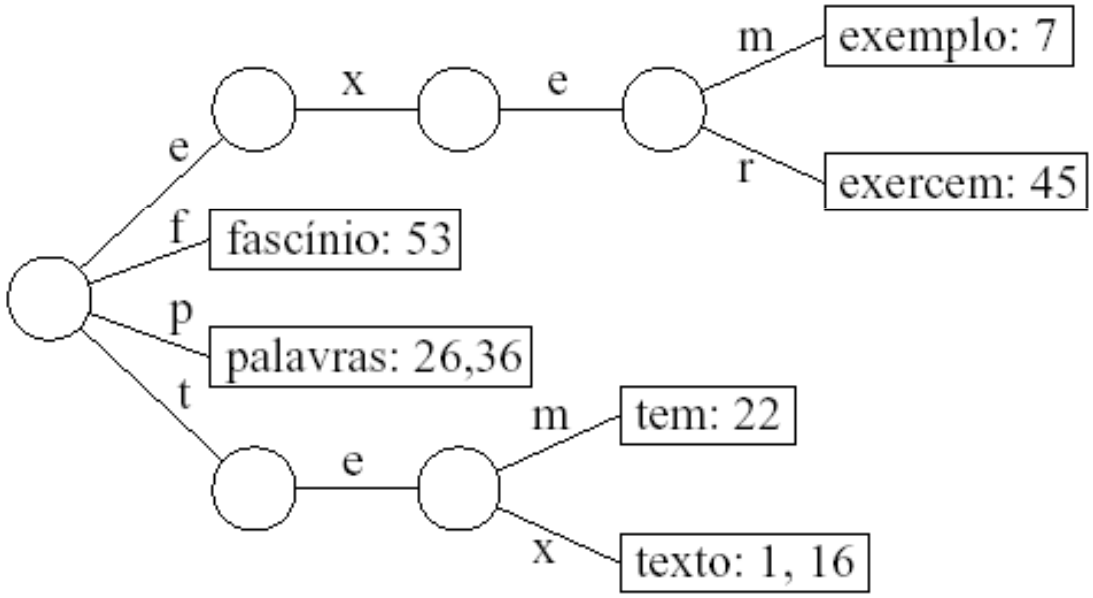
- Como a pesquisa em um arquivo invertido sempre começa pelo vocabulário, é interessante mantê-lo em um arquivo separado.
  - Geralmente, esse arquivo cabe na memória principal.

Arquivo Invertido

- 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*.
  - As duas primeiras têm custo $O(m)$, onde m é o tamanho da consulta (independentemente do tamanho do texto).
  - A última possui 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.
  - Cada palavra da frase deve ser pesquisada separadamente, no intuito de recuperar suas listas de ocorrências.
  - A seguir, as listas devem ser percorridas de forma sincronizada no intuito de encontrar as ocorrências nas quais todas as palavras aparecem em sequência.

Arquivo Invertido: Exemplo

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



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

Arquivo Invertido: Exemplo

- Cada nova palavra lida do texto é pesquisada na *TRIE*.
  - Se a pesquisa não tiver sucesso, a palavra é inserida na árvore e uma lista de ocorrências é inicializada com a posição de tal nova palavra no texto.
  - Caso contrário, uma vez que a palavra já se encontra na árvore, a nova posição é inserida ao final da lista de ocorrências da mesma.

Casamento Exato

- O problema de casamento exato de cadeias consiste em encontrar as ocorrências exatas de um padrão em um texto.
- Os algoritmos podem ser categorizados em relação à forma como o padrão é pesquisado no texto:
  - leitura dos caracteres do texto *T* um a um, no intuito de identificar uma ocorrência possível do padrão *P*.
    - Exs.: algoritmos força bruta e Shift-And.
  - pesquisa do padrão *P* em uma janela que desliza ao longo do texto *T*, procurando por um sufixo da janela (texto *T*) que casa com um sufixo de *P*, mediante comparações realizadas da direita para a esquerda.
    - Exs.: algoritmos Boyer-Moore, Boyer-Moore-Horspool e Boyer-Moore-Horspool-Sunday.

Algoritmo Força Bruta

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

```
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) { j++; k++; }
      if (j > m) printf(" Casamento na posicao%3ld\n", i);
    }
}
```

Pesquisa do padrão *P* a partir da *k*-ésima posição do texto *T*

Algoritmo Força Bruta

- Pior caso: $C_n = m \times n$.
  - Ex.: P = aab e T =aaaaaaaaaa.
- Caso esperado: $\overline{C_n} = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right)(n - m + 1) + O(1)$
  - O caso esperado é bem melhor do que o pior caso.
- Para conhecimento, em experimento realizado por Baeza-Yates (1992) com texto randômico e alfabeto de tamanho $c = 4$, o número esperado de comparações por caractere do texto foi aproximadamente 1,3.

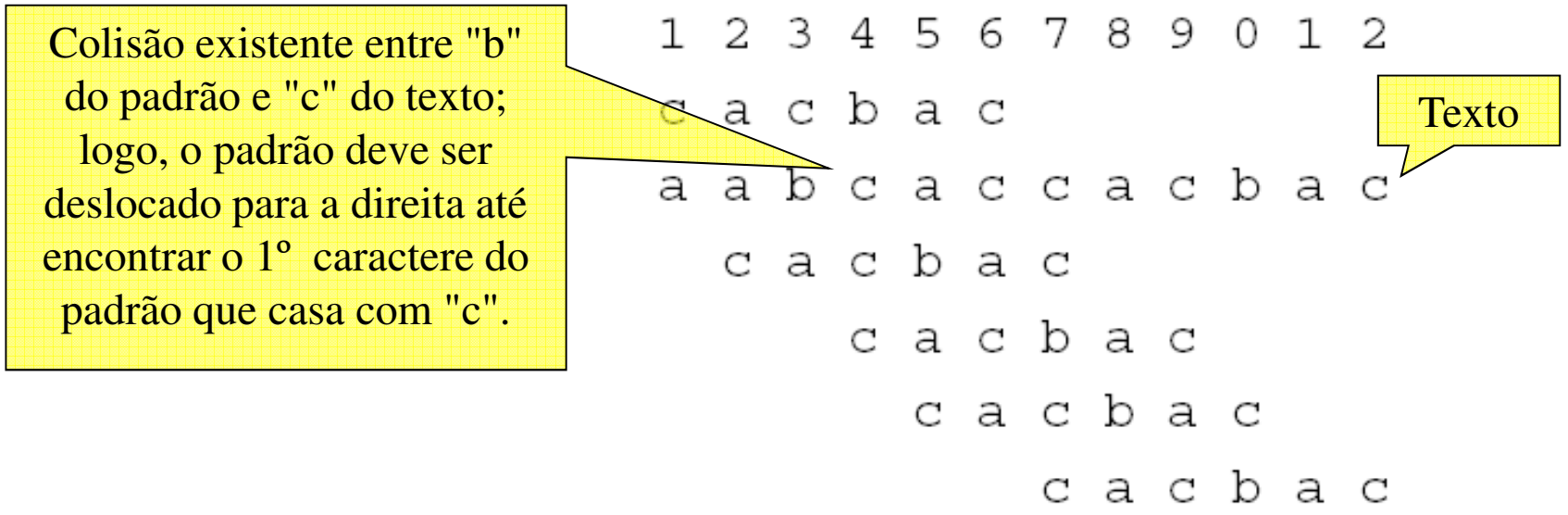
Algoritmo Boyer-Moore

- O algoritmo clássico Boyer-Moore (BM) surgiu em 1977.
- Funcionamento:
  - O BM pesquisa o padrão *P* em uma janela que desliza ao longo do texto *T*.
  - Para cada posição desta janela, o algoritmo pesquisa por um sufixo da mesma que casa com um sufixo 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 BM propõe duas heurísticas para calcular o deslocamento: ocorrência e casamento.

Algoritmo Boyer-Moore

- A heurística ocorrência 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.

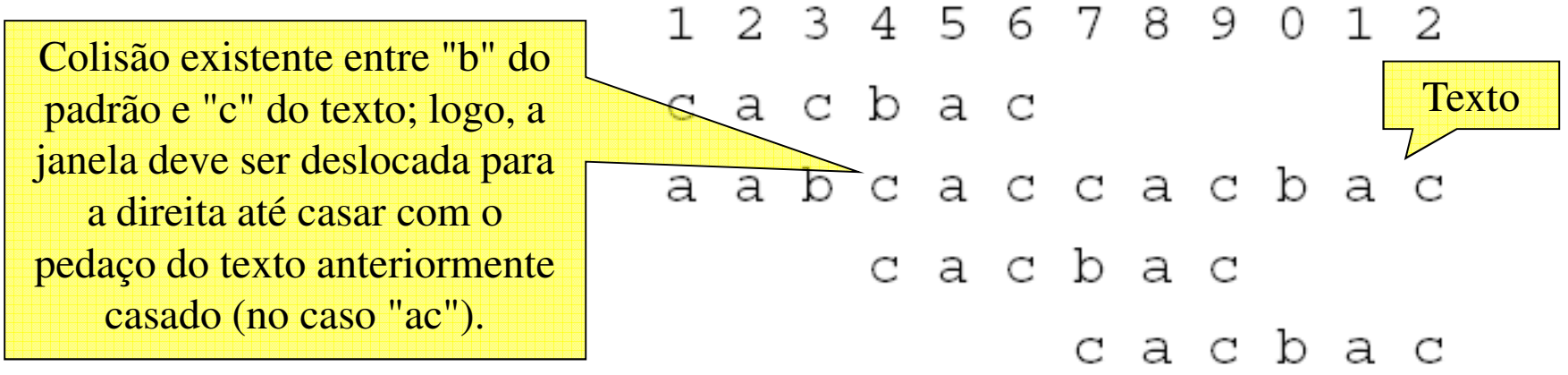
- Ex.: P ={cacbac}, T ={aabcaccacbac}.



Algoritmo Boyer-Moore

- 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.

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



Algoritmo Boyer-Moore

- O algoritmo BM decide qual das duas heurísticas deve seguir, escolhendo aquela que provoca o maior deslocamento do padrão.
  - Esta escolha implica em realizar comparações para cada colisão que ocorrer, penalizando o desempenho do algoritmo com relação a tempo de processamento.
  - Ao longo dos anos, várias propostas de simplificação surgiram, sendo que os melhores resultados foram obtidos por aquelas que consideraram apenas a heurística ocorrência.

Algoritmo Boyer-Moore-Horspool

- Em 1980, Horspool apresentou uma simplificação no algoritmo BM, que o tornou mais rápido, ficando conhecido como algoritmo Boyer-Moore-Horspool (BMH).
  - Pela extrema simplicidade de implementação e comprovada eficiência, o BMH deve ser escolhido em aplicações de uso geral que necessitam realizar casamento exato de cadeias.

- Simplificação:
  - Parte da observação de que qualquer caractere já lido do texto a partir do último deslocamento pode ser usado para endereçar uma tabela de deslocamentos.
  - 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.

Algoritmo Boyer-Moore-Horspool

- Para definir a tabela de deslocamentos, faz-se:
  - O valor inicial do deslocamento para todos os caracteres do texto é igual a m.
  - Em seguida, para os m-1 primeiros caracteres do padrão P, os valores do deslocamento são calculados pela regra:

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

- Ex.: Para o padrão P ={teste}, os valores da tabela são:
  - 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.

Algoritmo Boyer-Moore-Horspool

```
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).

Algoritmo Boyer-Moore-Horspool-Sunday

- Em 1990, Sunday apresentou uma simplificação importante para o algoritmo BMH, ficando conhecido como algoritmo Boyer-Moore-Horspool-Sunday (BMH).

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

Algoritmo Boyer-Moore-Horspool-Sunday

- Para definir a tabela de deslocamentos, faz-se:
  - O valor inicial do deslocamento para todos os caracteres do texto é igual a $m+1$.
  - Em seguida, para os m primeiros caracteres do padrão P, 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}, os valores da tabela são:
  - 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.

Algoritmo Boyer-Moore-Horspool-Sunday

```
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]];
    }
}
```

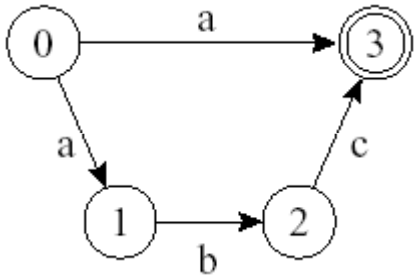
Pré-processamento para se obter a tabela de deslocamentos

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).

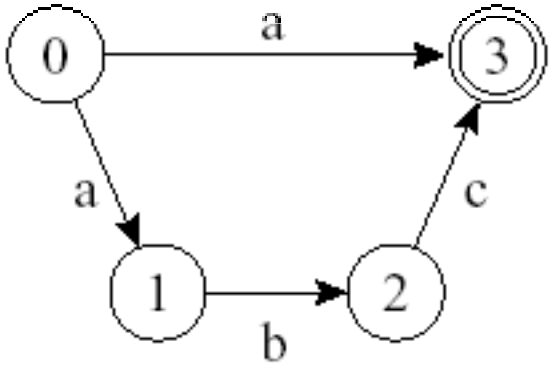
Autômatos

- 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 ($I \in Q$);
  - 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 $\{q_1, q_2, \ldots, q_k\} \subseteq Q$, para cada $\alpha \in (\Sigma \cup \{\varepsilon\})$, onde ε é a transição vazia.



Autômatos

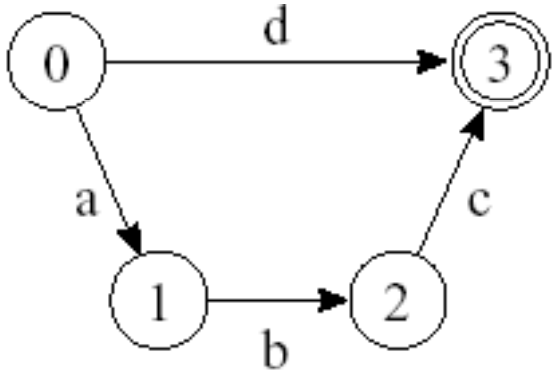
- Autômato finito não-determinista:
  - Ocorre quando a função T possibilita a associação de um estado q e um caractere α para mais de um estado do autômato (ou seja, $T(q, \alpha) = \{q_1, q_2, \ldots, q_k\}$ para $k > 1$), ou quando existe alguma transição rotulada por ε.



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

Autômatos

- 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 (ou seja, $T(q, \alpha) = \{q_1\}$).

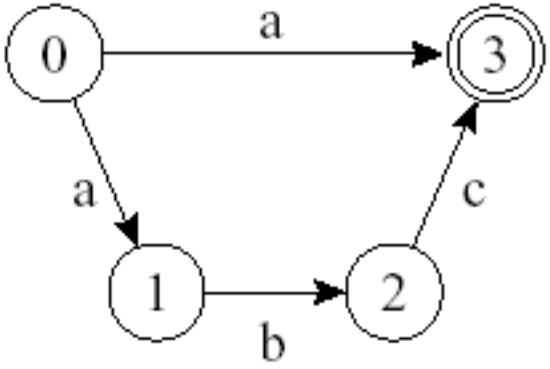


Para cada caractere de transição, todos os estados levam a um único estado.

Autômatos

- Uma cadeia é reconhecida pelo autômato (Q, I, F, Σ, T) se o mesmo rotula um caminho, que vai do estado inicial até um estado final, compreendendo a cadeia em questão.

- A linguagem reconhecida por um autômato é o conjunto de cadeias que o autômato é capaz de reconhecer.
  - Por exemplo, a linguagem reconhecida pelo autômato abaixo é o conjunto formado pelas cadeias {a} e {abc}.



Autômatos

- Transições vazias são transições rotuladas com a cadeia vazia ε, chamadas de transições-ε em autômatos não-deterministas.
  - Não há necessidade de se ler um caractere do alfabeto para se caminhar por meio de uma transição vazia.
  - As transições vazias simplificam a construção do autômato.
  - Sempre existe um autômato equivalente que reconhece a mesma linguagem sem transições vazias.

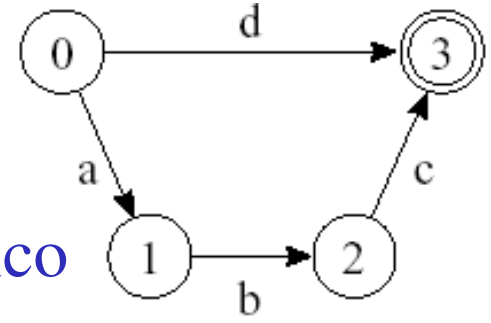
Autômatos

- Se uma cadeia **x** rotula um caminho de I até um estado q, então o estado q é considerado ativo depois de ler **x**.
  - Um autômato finito determinista possui, no máximo, um estado ativo em um determinado instante.
  - Um autômato finito não-determinista pode ter vários estados ativos em um determinado instante.
    - Casamento aproximado de cadeias pode ser resolvido por meio de autômatos finitos não-deterministas.

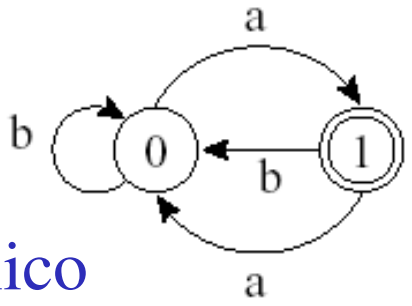
Autômatos

- Autômatos acíclicos são aqueles cujas transições não formam ciclos.

- Autômatos cíclicos são aqueles que formam ciclos.
  - Autômatos finitos cíclicos, deterministas ou não-deterministas, são úteis para casamento de expressões regulares.
  - A linguagem reconhecida por um autômato cíclico pode ser infinita. Por exemplo, o autômato a direita reconhece {ba}, {bba}, {bbba}, {bbbba}, ...

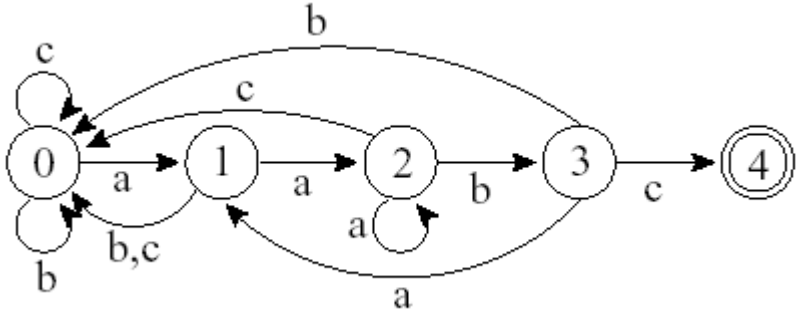


Acíclico



Cíclico

Autômatos



- O autômato reconhece $P = \{aabc\}$.
- A pesquisa de P sobre um texto T com alfabeto $\Sigma = \{a, b, c\}$ pode ser vista como a simulação do autômato na pesquisa de P sobre T.
  - No início, o estado ativo é o estado inicial 0.
  - Para cada caractere lido do texto, a aresta correspondente a partir do estado ativo é seguida, ativando o estado destino.
  - Se o estado 3 estiver ativo e um caractere **c** é lido, o estado final se torna ativo, resultando em um casamento de {aabc} com o texto.
- Como cada caractere do texto é lido uma vez, a complexidade de tempo é $O(n)$. A complexidade de espaço é $(m+1)$ para vértices e $(|\Sigma| \times m)$ para arestas.

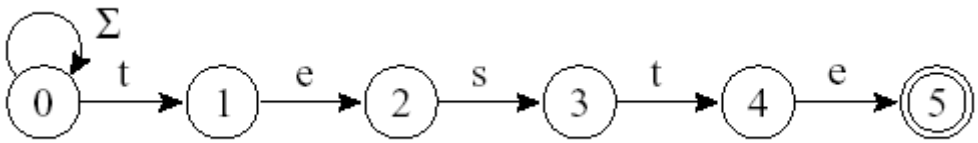
Algoritmo *Shift-And* Exato

- 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* (b_1 ... b_c) 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*: 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.: $b_1\ b_2 \ldots b_{c-1}\ b_c >> 2 = 00b_1 \ldots b_{c-2}$).

Algoritmo *Shift-And* Exato

- O algoritmo *Shift-And* mantém o conjunto de todos os prefixos do padrão *P* que casam com o texto já lido.
  - Este conjunto é representado por uma máscara de bits R = (b_1 b_2 ... b_m).
  - O algoritmo utiliza o paralelismo de *bit* para atualizar tal máscara de *bits* a cada caractere lido do texto.

- O algoritmo *Shift-And* 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}



Algoritmo *Shift-And* Exato

- O valor 1 é colocado na j-ésima posição de R = (b_1 b_2 ... b_m) se e somente se (p_1 ... p_j) é um sufixo de (t_1 ... t_i), onde i corresponde à posição corrente no texto.
  - Nesse caso, a j-ésima posição de R é dita estar ativa.
  - b_m ativo significa um casamento exato do padrão.

- 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, (p_1 ... p_j) é sufixo de (t_1 ... t_i), e t_{i+1} casa com p_{j+1}.
  - Com o uso de paralelismo de *bit*, é possível computar a nova máscara com custo $O(1)$.

Algoritmo *Shift-And* Exato

- Pré-processamento do algoritmo:
  - Construção de uma tabela M para armazenar uma máscara de *bits* (b_1 b_2 ... b_m) para cada caractere do padrão *P*.
  - Por exemplo, as máscaras de *bits* para os caracteres presentes em P ={teste} são:

	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*.

Algoritmo *Shift-And* Exato

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

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

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

Algoritmo *Shift-And* Exato

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

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

Casamento exato

Algoritmo *Shift-And* Exato

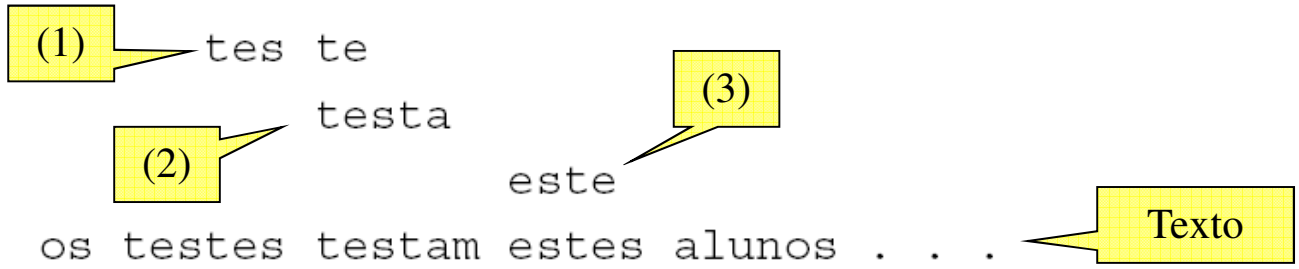
```
Shift-And (P = p1 p2 ··· pm, T = t1 t2 ··· tn)
{ /*—Préprocessamento—*/
  for (c∈ Σ) M[c] = 0^m;
  for (j = 1; j <= m; j++)  M[pj] = M[pj] | 0^(j-1) 10^(m-j);
  /*— Pesquisa —*/
  R = 0^m;
  for (i = 1; i <= n; i++)
    { R = ((R >> 1 | 10^(m-1)) & M[T[i]]);
      if ( R & 0^(m-1) 1 ≠ 0^m) 'Casamento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

- **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 problema de casamento aproximado de cadeias consiste em encontrar as ocorrências aproximadas de um padrão no texto.
  - Logo, deve tratar operações de inserção, substituição e retirada de caracteres do padrão.

- No exemplo abaixo, aparecem três ocorrências aproximadas do padrão {teste}:
  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.



Casamento Aproximado

- 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.

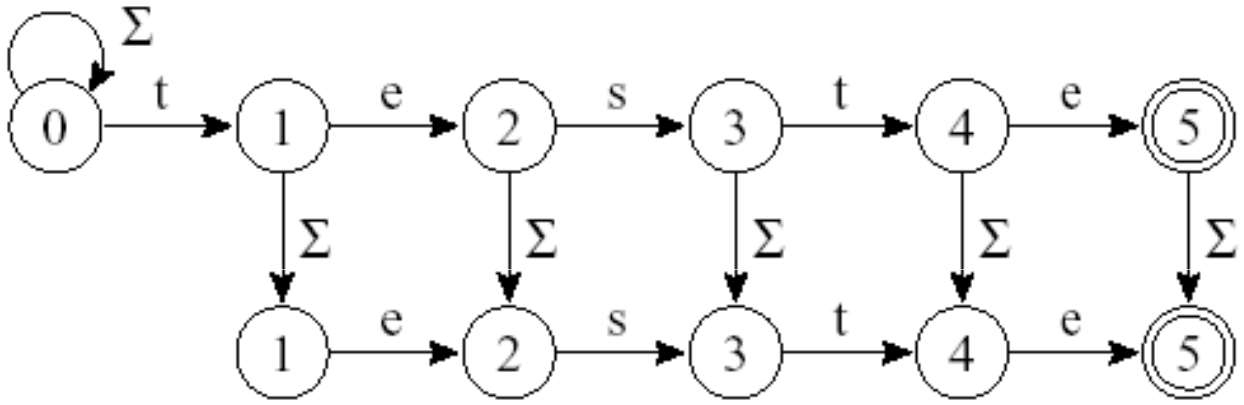
- Formalmente, o problema do casamento aproximado de cadeias é o de encontrar todas as ocorrências de P' no texto T tal que $ed(P, P') \leq k$, onde k representa o número limite de operações de inserção, substituição e retirada de caracteres necessárias para transformar o padrão P em uma cadeia P'.

Casamento Aproximado

- 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 de cadeias.
- Uma medida da fração do padrão que pode ser alterada é dada pelo nível de erro $\alpha = k/m$.
  - Em geral, $\alpha < 1/2$ para a maioria dos casos.
- A pesquisa com casamento aproximado é modelado por autômatos não-deterministas.
  - Os algoritmos usam paralelismo de bit.

Casamento Aproximado

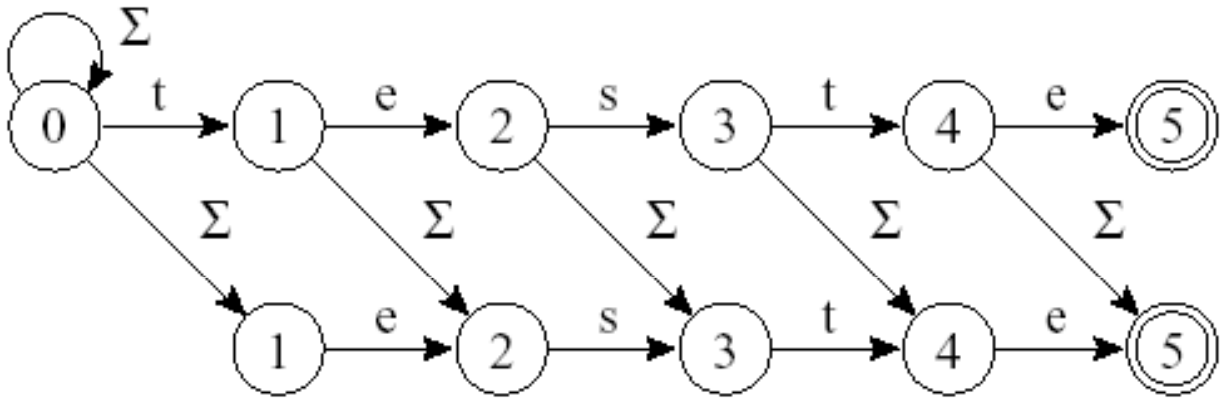
- Autômato que reconhece $P = \{\text{teste}\}$, permitindo uma inserção:



  - 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.

Casamento Aproximado

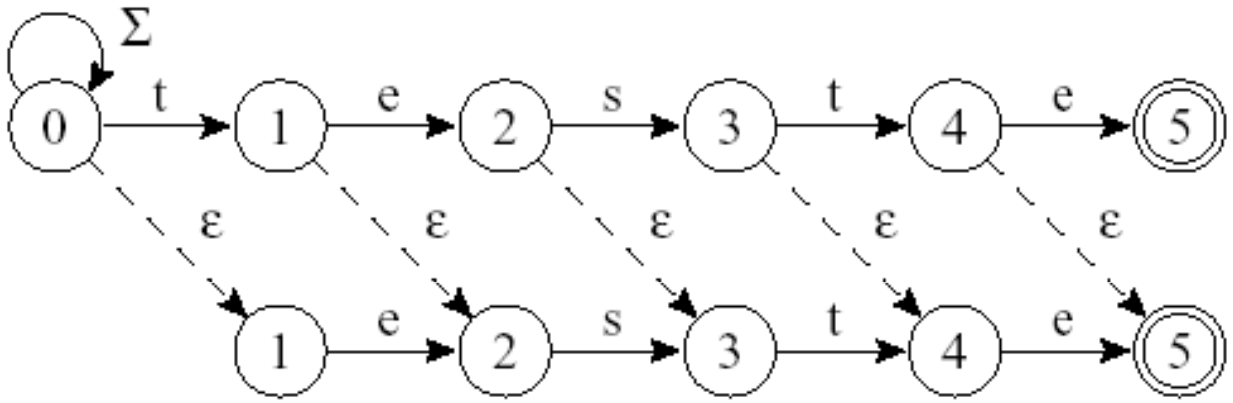
- Autômato que reconhece $P = \{\text{teste}\}$, permitindo uma substituição:



  - 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.

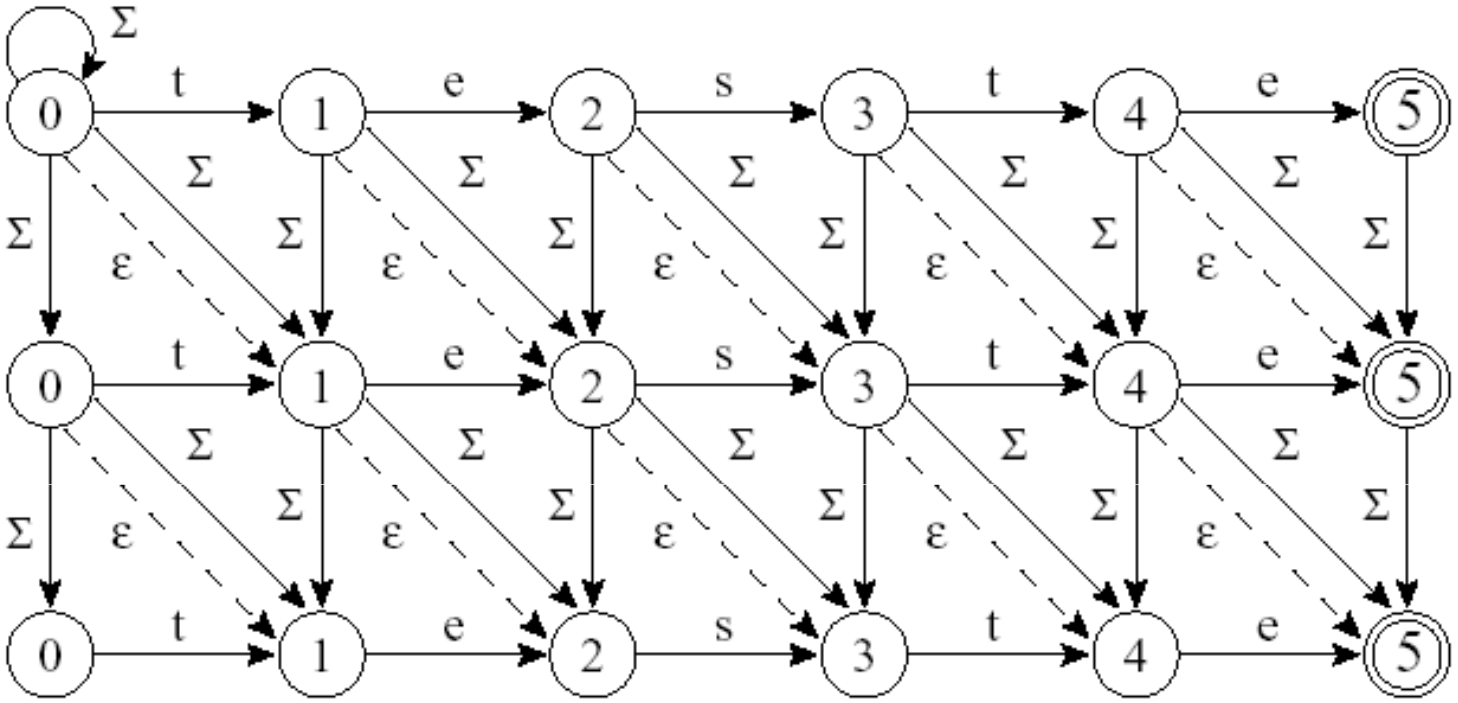
Casamento Aproximado

- Autômato que reconhece $P = \{\text{teste}\}$, 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).

Casamento Aproximado



- 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 ($k = 2$).

Algoritmo *Shift-And* Aproximado

- O algoritmo *Shift-And* aproximado simula um autômato não-determinista, utilizando paralelismo de *bit*.

- O algoritmo empacota cada linha j ($0 < j \leq k$) do autômato não-determinista em uma palavra Rj diferente do computador.
  - Para cada novo caractere lido do texto, todas as transições do autômato são simuladas usando operações entre as k+1 máscaras de *bits*: R_0 (casamento exato), R_1 (um erro), R_2 (dois erros), ..., R_k (k erros).

Algoritmo *Shift-And* Aproximado

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

      $R'_j = ((R_j >> 1) \ \& \ M[T[i]]) \mid R_{j-1} \mid (R_{j-1} >> 1) \mid (R'_{j-1} >> 1) \mid 10^{m-1}$
  - Considerando o autômato, a fórmula para R' expressa as arestas:
    - horizontais, indicando casamento de um caractere;
    - verticais, indicando inserção (R_{j-1});
    - diagonais cheias, indicando substituição ($R_{j-1} >> 1$);
    - diagonais tracejadas, indicando retirada ($R'_{j-1} >> 1$).

Algoritmo *Shift-And* Aproximado

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

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

Algoritmo *Shift-And* Aproximado

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

Algoritmo *Shift-And* Aproximado

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

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

Algoritmo *Shift-And* Aproximado

Texto	$(R_0 >> 1)\|10^{m-1}$	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
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***
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***
	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
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
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***
m	1 0 0 0 0	0 0 0 0 0	0 1 1 0 1	1 0 0 0 0

Casamento aproximado R

Casamento exato

Casamento aproximado I

Casamento aproximado I

Casamento aproximado R

Casamento aproximado S

Algoritmo *Shift-And* Aproximado

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

Algoritmo *Shift-And* Aproximado

```
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);
    }
}
```