Resumo 3 - EDII

Aluno: *Nathann Zini dos Reis*

Matrícula: *19.2.4007*

Aula 11 - Introdução, arquivo invertido, força bruta

Definição:

> pesquisa de cadeia de caracteres dentro de algum texto qualquer. A ideia é localizar um "padrão" (uma string) dentro de um volume de texto. Ou seja, encontrar alguma frase/palavras (padrões) dentro de algum textos. Além de encontrar o padrão correto, também pode-se fazer casamento aproximado de cadeias, que localiza padrões próximos ao desejado (com mudança, por exemplo, de apenas o último caractere do padrão).

Conceitos:

> Cadeia de caracteres: uma string/sequencia de caracteres. Estes são sempre escolhidos a partir de um alfabeto. Exemplo, cadeia de bits o alfabeto é {0,1}.

Objetivo:

> Encontrar todas as ocorrências de um determinado padrão em um texto.

Introdução

Primeiro passo é a formalização do problema que consiste em definir:

- Texto: cadeia T[0..n-1], em que n é o tamanho total de caracteres do texto;
- Padrão: cadeia P[0..m-1], em que m é o tamanho total de caracteres do padrão;

- Os elementos de T e P devem fazer parte de um alfabeto finito E de tamanho c.

  como dito lá em cima, os alfabetos E podem ser bits de tamanho 2 que representa {0,1} ou E = {a, b, ..., z} ...

- Enfim, descobrir as ocorrências de P em T, desde que m ≤ n;

Categorias de Algoritmos:

1º - *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).

2 º - *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.

Exemplo da categoria 1, em que tanto P quanto T não são pré-processados:

Nessa categoria, deverá apenas encontrar uma estratégia de tentar encontrar o P dentro de T do jeito em que ambos são. Sem fazer qualquer tipo de alteração.

Um exemplo de fazer essa comparação é utilizando o algoritmo de força bruta que consiste em procurar pelo primeiro elemento de P em T, quando encontrar procura pelo segundo elemento em seguida, caso encontre, faz a procura pelo terceiro elemento de P. Caso não tenha sido encontrado deverá voltar a procurar o padrão desde o inicio novamente para verificar se há uma nova ocorrência.

Como mencionado anteriormente, o pior caso tem complexidade O(mn) e acontecerá quando se encontra todos menos o ultimo elemento do padrão dentro de T até a última verificação.

Exemplo de P: aaaaaaaaaaaab; e T: aaab;

Exemplo da categoria 2, em que o P é pré-processado:

A ideia de pré processar é aplicar, previamente, alguma estrutura de dados à cadeia em questão, nesse caso, no padrão. A vantagem é diminuir a complexidade de acesso ao padrão.

3º - *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).

Exemplo da categoria 13 em que tanto P quanto T são pré-processados:

Como no exemplo anterior, transforma o padrão P e cria o padrão P' em que foi aplicado algum tipo de processamento/estrutura de dados àquele padrão. O mesmo

é feio com a cadeia T.

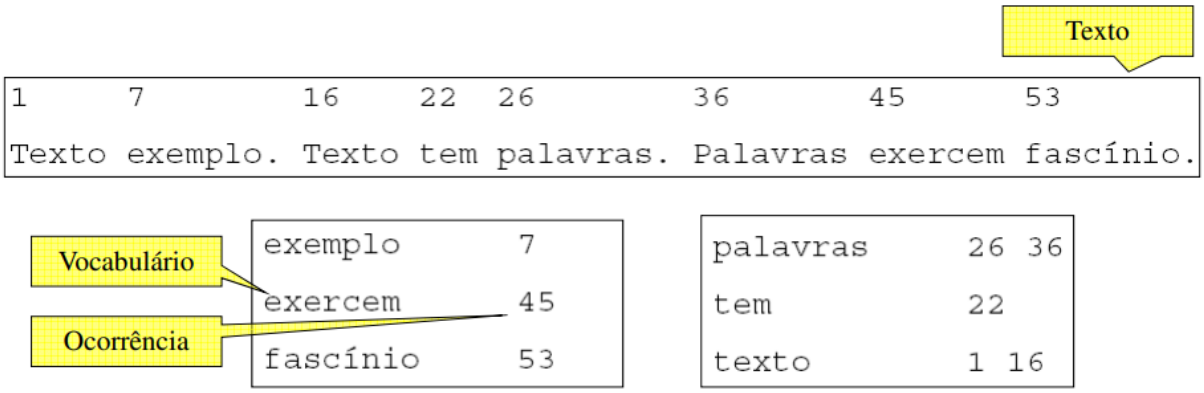
Um exemplo de processamento do Texto T é criar o T' em que foi aplicado a estrutura árvore B em que os índices são as próprias palavras e então realizar uma busca da outra cadeia padrão P' em que foi aplicada alguma estrutura a partir de P.

Arquivo Invertido

o arquivo é composto por vocabulário e ocorrências, em que:

os vocabulários é o conjunto de palavras distintas no texto;

as ocorrências é uma lista que guarda em quais posições do texto cada palavra do vocabulário foi encontrada no texto;



Em termos de memória, diferentemente desse exemplo, o que acontece é que as ocorrências ocupam muito mais espaço que o vocabulário. Exemplo disso quando acontece é quando o número de ocorrências de uma palavra ocupam mais espaço que a própria palavra. Por isso, é comum guardar as ocorrências em arquivos, enquanto o vocabulário é guardado em memória principal.

O crescimento das ocorrências é linear (O(n)), pois cada palavra gera uma nova ocorrência. Já o crescimento do vocabulário é um crescimento de raiz quadrada, definida pela lei de Heaps.

- 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.

A pesquisa em um arquivo invertido é realizada em três passos:

Primeiramente, busca cada palavra separadamente do padrão P dentro do vocabulário, para saber se todas estão ou não no texto T;

Depois, é buscado e recuperado as ocorrências de cada uma dessas palavras de P;

Por fim, é feito uma manipulação das ocorrências para saber se elas se dão de maneira que formem a frase que deseja ser buscada por P;

A pesquisa feita por palavra simples é feita de modo mais simples e depende da estrutura utilizada

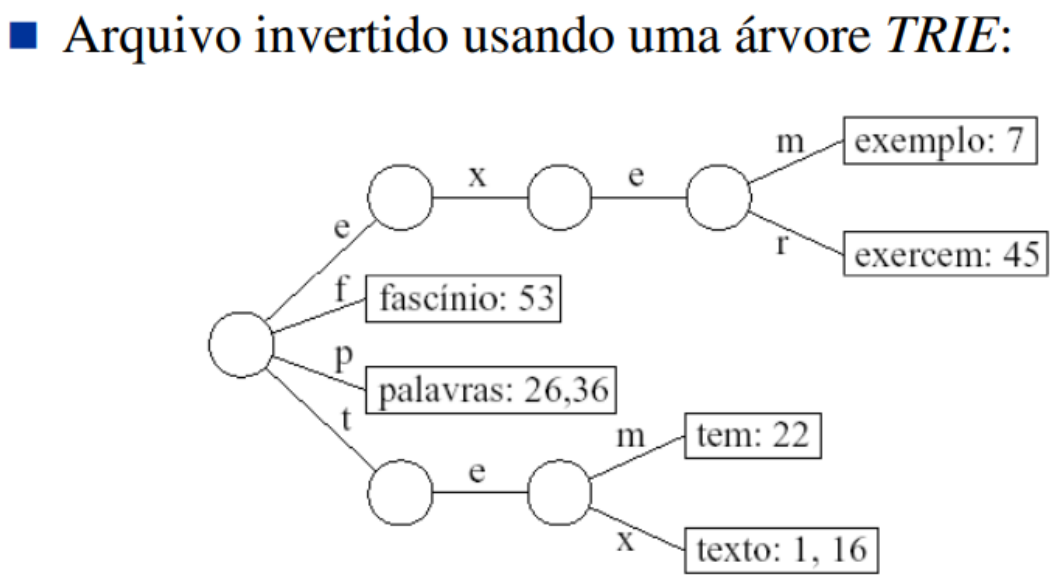
para ser dada a complexidade. Generalizando, é buscado a palavra em questão no vocabulário e então suas ocorrências.

Exemplo de estrutura para arquivo invertido: ÁRVORE **TRIE**

complexidade de pesquisa em árvore trie

- 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 árvore trie:



Consiste em uma árvore em que cada nó interno representa uma letra e os nós folhas representam as palavras do vocabulário.

O nó raiz tem no máximo 30 apontadores, no caso do alfabeto representado pela linguagem natural, como mostrado no exemplo. Caso de bits teriam 2 {0,1}.

Para realizar a pesquisa, precisa, no pior caso, de uma complexidade de ordem O(m) em que m é o padrão. Exemplo na imagem acima acontece para procurar a palavra "tem" em que seria preciso fazer 3 comparações para achar a palavra.

Casamento Exato

Como o próprio nome sugere, o casamento exato procura por ocorrências exatas de algum padrão dentro de um texto.

Pode ser resolvido com algoritmos que fazem a leitura um a um dos caracteres do padrão, como o de força bruta e Shift-And. Ou pode ser resolvido com pesquisa do padrão P em uma janela que desliza ao longo do rexto T, procurando por um sufixo da janela (texto T) que casa com um sufixo de P, mediante a comparações

realizadas da direita para a esquerda, como nos algoritmos de Boyer-Moore, Boyer-Moore-Horspool e Boyer-Moore-Horspool-Sunday.

Algoritmo de 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) { 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*

O pior caso é é O(mn), porém quando o tamanho do alfabeto é grande, é muito improvável que o pior caso aconteça, ao contrário de quando é alfabeto, por exemplo, binário, que possui tamanho 2.

Aula 12 - Algoritmos BM, BMH e BMHS

Os algoritmos em questão são da categoria em que o padrão é pré processado.

Boyer-Moore

Consiste numa sistemática que "desliza" uma janela contendo tamanho do padrão P ao longo do texto T.

Essa janela caminha da esquerda pra direita dentro do texto, entretanto, o padrão é comparado com a janela no sentido da direita para a esquerda. Ou seja, compara o ultimo elemento do padrão com o ultimo elemento da janela. Caso haja alguma

diferença, a janela é deslocada uma casa pra direita no texto e é refeita a comparação até que ou acabe o texto ou ache o casamento de padrão.

Esse método é similar à força bruta por fazer comparação de todos os elementos do padrão com os elementos do texto. A diferença é feita no deslizamento da janela. Ela não necessariamente deve deslizar-se uma casa apenas, mas pode deslizar n casas, dependendo da heurística utilizada (o máximo do tamanho do deslocamento é o tamanho do padrão).

Heurística Ocorrência

Consiste numa heurística que alinha o caractere no texto que causou a colisão com o 1º caractere no padrão, à esquerda do ponto de colisão, que casa com ele. Ou seja, quando o caractere x do texto não for igual ao terceiro, por exemplo, do padrão de tamanho m = 6, vou procurar o caractere x do texto dentro do padrão à partir da posição seguinte à posição em que ocorreu essa diferença de caracteres. Nesse caso, irei comparar o caractere x com a posição 4, 5 e 6 no padrão. Caso haja alguma correspondência eu alinho essa posição do padrão com a posição do caractere x. Se o casamento tiver sido na posição 5 do padrão, terei que deslizar a janela duas posições para que o item x deixe de ser comparado com a posição 3 do padrão e passe a ser comparada com a posição 5.

Ou seja, a janela vai ser deslizada "posição de ocorrência - posição de colisão" vezes. Caso não ache uma correspondência, é movido m (tamanho do padrão) vezes.

É interessante em casos em que o alfabeto haja um grande número de elementos, que é muito maior a chance de encontrar discrepância ou então um caractere no texto que não esteja no padrão e então seria movido a janela no tamanho do padrão.

Heurística Casamento

Parecido com a heurística ocorrência, ao invés de procurar o caractere x no restante do padrão, é procurado os caracteres que haviam formado o casamento antes da discrepância no restante do padrão. Caso haja uma correspondência, a janela é deslizada de forma que o casamento fique alinhado com a nova correspondência encontrada.

Ou seja, a janela vai ser deslizada "posição do inicio da nova correspondência - tamanho do casamento" vezes. Caso haja discrepância logo no inicio, a janela é movida apenas uma única vez. Fazendo com que essa heurística seja interessante para textos cujo o alfabeto contenha um pequeno número de elementos.

Boyer-Moore-Horpool

Similarmente ao funcionamento do BM normal, a diferencia encontra-se no deslocamento que funciona da seguinte maneira:

Foi proposto a criação de uma tabela de deslocamento que dirá a quantidade de casas que a janela deverá ser "deslizada"/deslocada para a direita no texto.

Inicialmente cria-se uma tabela contendo todos os caracteres presentes no alfabeto utilizado no texto. O deslocamento referente a cada um desses caracteres é iniciado com o valor m (tamanho do padrão).

Para cada caractere do padrão, menos o ultimo, ou seja, para os m-1 primeiros caracteres serão feita uma análise da distância desse caractere até o último do padrão e será atualizado o valor dessa distância nessa tabela. **Nota:** É sempre mantido o menor valor de deslocamento para um mesmo caractere. Exemplo, se no padrão haja 2 letras "k", será mantido na tabela a distância do caractere "k" que seja a menor. No caso do padrão "KAKA", o primeiro K tem distancia 3 e o segundo tem distancia 1, logo será mantida o tamanho de deslocamento 1.

O deslizamento vai sempre ser feito quando houver alguma discrepância entre o padrão e a parte do texto que está sendo analisada. Para saber qual caractere analisar na tabela pra pegar o tamanho do deslizamento, SEMPRE se pega o elemento do texto que é comparado com o ultimo elemento 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).

primeiro for inicia cada elemento do alfabeto com o tamanho do padrao;

segundo for atualiza o deslocamento para cada elemento do padrão referente à distancia dele do fim, pegando sempre o menor valor.

o while faz a analise da tentativa do casamento e realiza o deslizamento da janela até a janela chegar ao fim do texto;

Nota: apesar de achar ou não o casamento, o código continua rodando até o fim do texto.

Boyer-Moore-Horpool-Sunday

Similarmente ao BMH, o BMHS (alterou apenas a fórmula de deslocamento, a de casamento é identica) faz o mesmo processo, porém, ao invés de analisar o caractere do texto que corresponde ao elemento do texto que esteja comparando com o último item do padrão, é analisado o elemento do texto seguinte ao elemento

que é comparado com o último item do padrão. Agora, não só os m-1 itens são analisados em relação ao ultimo item do padrão, agora os m itens são analisados em relação ao item m+1 (que seria o '\0').

Além disso, a vetor que contem todos os elementos do alfabeto são iniciado para cada item do alfabeto uma distancia de m (tamanho do padrão) +1 casas.

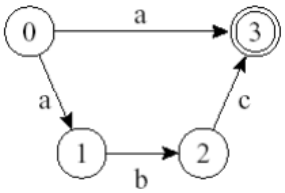
Aula 13 - Algoritmo Shift-And exato

Como o método apresentado anteriormente, o shift-and trabalha com o pré-processamento do padrão a ser procurado dentro do texto.

O algoritmo shift-and se baseia na utilização de autômatos

Autômatos

- 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.



Cada nó/vértice representa um estado, que são definidos como ativos ou não. Um vértice ser ativo significa que a leitura está naquele ponto do autômato. As arestas representa a possibilidade de caractere a ser lido do texto; são os caracteres do alfabeto do texto;

Se chegar ao vértice final, quer dizer que aquele padrão foi encontrado;

autômato se divide em dois tipos

- *autômato finito não-determinista:* em que um caractere lido poder seguir pra mais de 1 nó/vértice distinto. Nesse caso, em determinado ponto, poderá haver vários vértices ativos;
- *autômato finito determinista:* em que um caractere lido pode seguir apenas para 1 vértice. Nesse caso, sempre haverá apenas 1 nó ativo.

A quantidade de vértices é sempre o tamanho do padrão m + 1, e a quantidade de arestas é sempre a quantidade de caracteres no alfabeto para cada um dos vértices menos o último, ou seja E x m;

Shitf-And

O shift-and é baseado em autômato, porém não é criado o autômato em si, mas uma simulação usando o conceito de paralelismo de bit.

é utilizado as seguintes operações binárias:

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

Os nó/vértices do autômato, no shift-And, é representado por uma variável R que contém uma máscara de bits. Cada bit representa um nó. o primeiro bit representa o nó 1, o segundo o nó 2, e assim sucessivamente. Importante notar que o primeiro nó (0) não é representado por R, pois entende-se que inicialmente ele sempre será ativo. Á medida que vai lendo os caracteres e encontrando ou não dentro do padrão, o bit que representa aquele estado no autômato é então mudado para ativo (representado pelo bit 1), caso esteja desativo é representado por 0;

Será dado como encontrado o padrão quando o M ésimo bit de R estiver ativo (1);

O pré-processamento do Padrão P se dá pela construção de uma tabela M que irá armazenar, para cada caractere contido no padrão, uma máscara de bit que será representada por 1 na posição em que aquele caractere é encontrado no padrão ou 0 na posição em que não é encontrado pelo padrão.

- 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*.

Para os demais caracteres do alfabeto, a mascara de bits é 0, pois não se encontram no padrão.

- 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 quantidade de bits de R vai ser iniciada com m 0's.

Primeiro passo da fórmula é fazer o shift de R uma casa. ou seja, move todos os bits pra direita 1 casa e adiciona 1 0 a esquerda; faz a operação OR com a mascara 1 m0's (será analisada bit com bit e onde tiver 1 ficará 1) e então é feito a operação AND com a máscara da tabela referente ao caractere lido do texto (analisa bit por bit e onde tiver 0 ficará 0). A máscara resultado dessa operação será o R' e ela dirá

onde está o vértice ativo do autômato. Caso o último bit seja 1, então encontrou uma correspondência.

Nota: se haver mais de um bit 1 na máscara, quer dizer que há chance de outra nova ocorrência do padrão no texto à medida que continuar sendo comparado os caracteres do texto com o padrão;

```
Shift-And (P = p1p2···pm, T = t1t2···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';
    }
}
```

código referente ao funcionamento do shift-and;

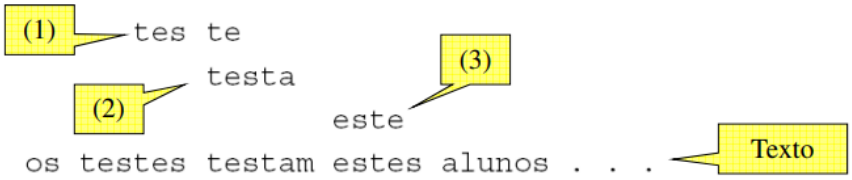
A complexidade de shift-And é O(n) pois é feito uma operação sobre bits para cada caractere e cada uma dessas operações é O(1).

Casamento Aproximado

Casamento aproximado corresponde à encontrar as ocorrências aproximadas ao padrão dentro do texto, dado alguma lógica de aproximação.

exemplos de tipos de ocorrências aproximadas:

- 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.



Distância de edição: é o menor número de operações necessárias para converter P em P' ou vice-versa. Representado por uma constante K o limite máximo de edições permitidas de fazer, sempre sendo tamanho máximo variando de 1 ≤ k ≤ m-1. As edições são dadas pela comparação de P (padrão) por uma outra palavra do texto P'. Exemplo:

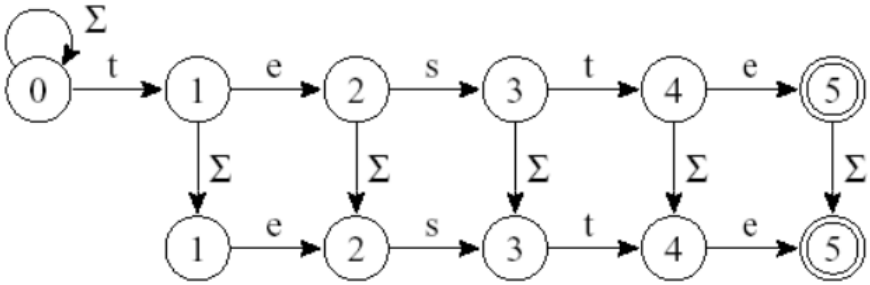
- Distância de edição entre duas cadeias *P* e *P'*, denotada por *ed*(*P*, *P'*), é o menor número de operações necessárias para converter *P* em *P'* ou vice-versa.
  - Por exemplo, *ed*(teste, estende) = 4: valor obtido por meio da retirada do primeiro **t** de *P* e a inserção dos caracteres **nde** ao final de *P*.

Se k assumir valor 0, obtém-se o casamento exato de cadeias.

A simulação do autômato é similar ao do shift-and exato, a diferencia é que terá mais de um nível de cadeias, sendo que cada nível a mais corresponde a um erro permitido na comparação do casamento; Caso haja um erro, se permitido, desce o nível e continua a comparação do casamento até o fim ou que encontre um erro e não seja permitido um novo erro, ai volta no inicio;

Representação do erro de inserção:

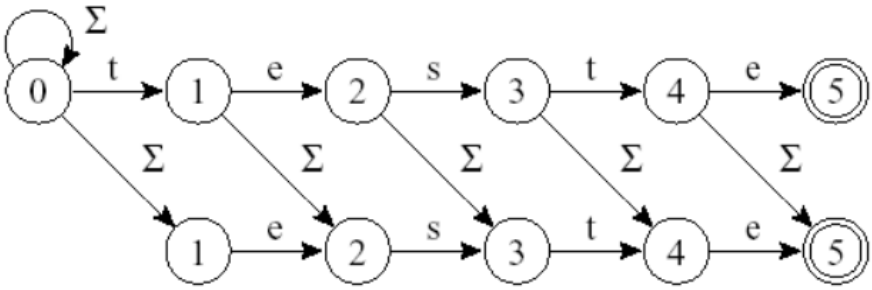
- Autômato que reconhece $P = \{teste\}$, permitindo uma inserção:



  - 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.

Representação do erro de substituição:

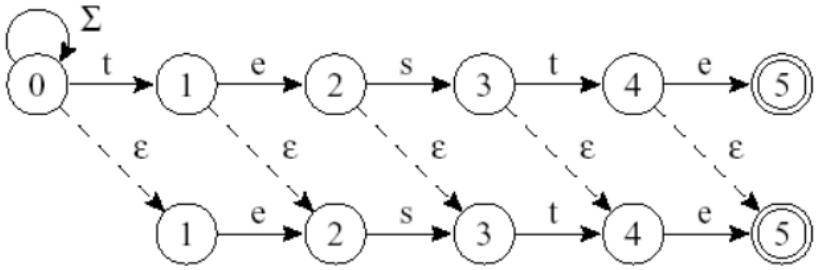
- Autômato que reconhece $P = \{teste\}$, permitindo uma substituição:



  - 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.

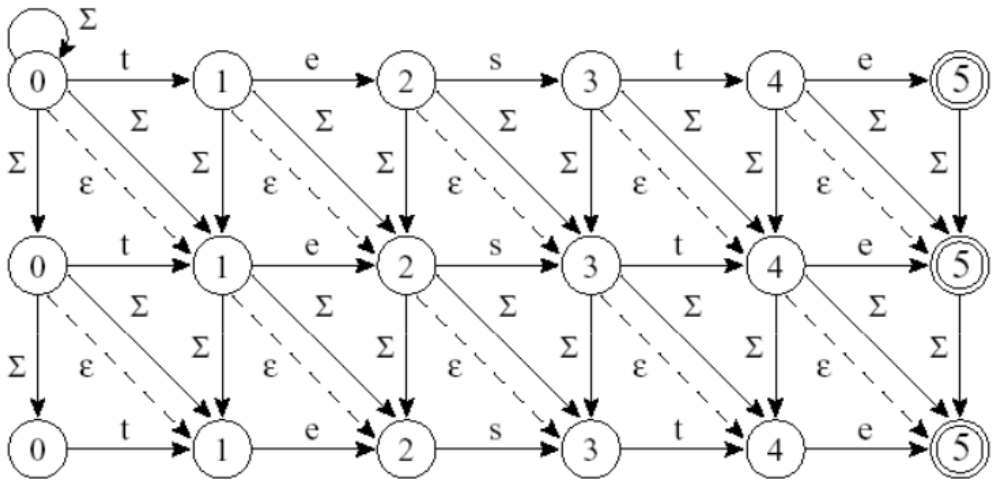
Representação do erro de retirada:

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

Exemplo de autômato que permite 2 erro de qualquer um dos tipos apresentado anteriormente:



- 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 ($k = 2$).

Neste caso, pode-se ter vários passos ativos ao mesmo tempo. Também faz-se necessário de uma máscara de bit R para cada um dos níveis do autômato.

Para cada nível representado a quantidade k de erros, a iniciação é dada por

- A máscara R_0 (casamento exato) é inicializada como $R_0 = 0^m$.
- Para $0 < j \leq k$, Rj é inicializada como R$j = 1^j 0^{m-j}$.

Exemplo, para m = 5, R0 = 00000, R1 = 10000, R2 = 11000 ...

Toda a fórmula para quanto não se tem nenhum erro, é idêntico ao casamento exato. Para 0 < j ≤ k muda alguns passos a mais de comparação que é adicionado

- 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 ($R'_{j-1} >> 1$).

A fórmula muda conforme qual a possibilidade de erro. Se apenas é possível o erro de inserção, é considerado o passo "| R j-1" e os outros dois passos adicionais, referente aos outros dois tipos de erros são ignorados.

Exemplo:

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

exemplo do passo a passo da aplicação da fórmula:

```
R0 = 00000 =>{o} 00000 =>{s} 00000 =>{ } 00000 =>{t} 10000 =>{e} 01000 =>{s}
     00100 =>{t} 10010 =>{e} 01001 =>{s} 00100 =>{ } 00000 =>{t} 10000 =>{e}
     01000

R1 = 10000 =>{o} 10000 =>{s} 10000 =>{ } 10000 =>{t} 10000 =>{e} 11000 =>{s}
     11100 =>{t} 10110 =>{e} 11011 =>{s} 11101 =>{ } 10100 =>{t} 10010 =>{e}
     11001
```

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);
    }
}
```

Primeiro passo é calcular a mascara de cada elemento do alfabeto, iniciando todos com 0; Depois atualiza a mascara de todos os elementos do padrão;

Depois inicia as mascaras de R, conforme cada nível. depois aplica a formula conforme estudado anteriormente em cada um dos elementos lidos e fazendo a análise de casamento ao final;

Como cada operação é feita sobre bits, tem-se a complexidade linear O(n), para obter tanto casamento exatos e aproximados.