Algoritmos de busca em bancos de dados constituídos de texto[1]

Nivio Ziviani

[1] Conjunto de transparências elaborado por Nivio Ziviani e Goedson Teixeira Paixão

Bancos de dados costituídos de texto

- Categoria crescente
  - Indústria de publicação
  - Baixo custo de memória secundária
  - Internet
- Estático
- Não estruturado
- Consultas não previsíveis
- Exemplos: Livros, dicionários, artigos técnicos e científicos, dados históricos, casos legais, jornalismo etc.

Casamento de padrões

Notação:

- n: tamanho do texto
- m: tamanho do padrão
- c: tamanho do alfabeto Σ
- C_n: pior caso para nº de comparações
- $\overline{C_n}$: nº esperado de comparações

Dados 2 strings

P: Padrão $|P| = m$
T: Texto $|T| = n \quad n \gg m$

Deseja-se saber as ocorrências de P em T.

Busca de string: Definição

- Problema básico: ocorrências exatas
- Variações:
  - Permitir trocas
  - Permitir trocas, inserções e remoções
- Exemplos:

```
  text
      text
A text example of a . . .
     t ext
       ex
```

Ponto de vista algoritmico: Dados de entrada

- Padrão e texto não processados
  - seqüencial, on-line, algoritmos de tempo-real
- Padrões preprocessados
  - algoritmos seqüenciais
- Padrões preprocessados e texto → índice
  - Arquivos invertidos
  - Tries
  - Arranjos de sufixo
  - Arquivos de assinatura
  - Algoritmos com tempo logaritmico ou melhor

Algoritmos de pesquisa: Força bruta (FBY 92)

```
bfsearch(text, n, pat, m)

/* Search pat[1..m] in text[1..n] */

char text[], pat[];
int n, m;
{
  int i, j, k, lim;

  lim = n - m + 1;
  for( i = 1; i <= lim; i++) /* Search */
  {
    k = 1;
    for(j=1; j<=m && text[k]==pat[j]; j++)
          k++;
    if( j>m ) printf(">> match at %d\n", i);
  }
}
```


Algoritmos de pesquisa: Força bruta - complexidade

- Pior caso: $C_n = m \times n$
- Caso esperado: $\overline{C_n} = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right)(n - m + 1) + O(1)$

Gráfico

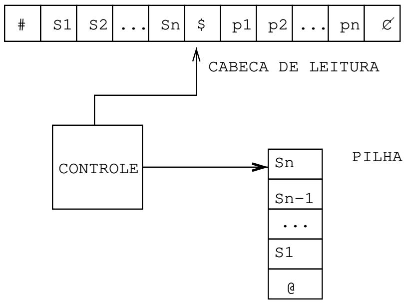
Teorema de Cook

- Até 1971 acreditava-se que o limite inferior para busca exata de padrões seria $O(mn)$
- Cook 71:

  Qualquer problema que puder ser resolvido por um *Two-way Deterministic Pushdown Store Automaton (2DPDA)* pode ser resolvido em tempo linear por uma máquina RAM
- 2DPDA:

  Fita apenas para leitura
  Pilha de dados (memória temporária)
  Controle de estado
  - pode mover para esquerda ou direita
  - Empilhar ou desempilhar símbolos
  - mudar de estado

Casamento de padrões no 2DPDA



Entrada: $\#S_1S_2\cdots S_n\$p_1p_2\cdots p_m\Phi$

1. A partir de $\#$ empilha todos os caracteres até $\$$
2. Cotinua lendo até Φ
3. Volta 1 a esquerda e compara p_m com S_n da pilha
4. Se chegar até $\$$ $\Rightarrow$ Sucesso

Algoritmo baseado na idéia de Cook

- Knuth & Pratt - trabalharam na simulação linear que Cook obteve para o 2DPDA para obterem a 1ª versão do algoritmo
- Morris - Desenvolveu um algoritmo semelhante

$$\Downarrow$$

Algoritmo KMP

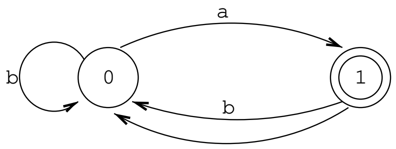
KMP - Knuth - Morris - Pratt (1977)
1º Algoritmo $O(n)$

- Mismatch $\Rightarrow$ “Falso Inicio” consiste de caracteres que já examinamos!
- pré-processamento de p permite
  1. Apontador para texto nunca é decrementado
  2. Não reexaminar caracteres

Autômato finito

- É um modelo de computação muito simples
- É definido por uma tupla $(Q, q_0, A, \Sigma, \delta)$:
  - Q - conjunto finito de estados
  - $q_0 \in Q$ - estado inicial
  - $A \subseteq Q$ - conjunto de estados finais (accepting states)
  - Σ - alfabeto finito de entrada
  - δ - função de transição de $Q \times \Sigma$ para Q

Exemplo: $A = 1$
$Q = 0, 1$
$q_0 = 0$
$\Sigma = a, b$

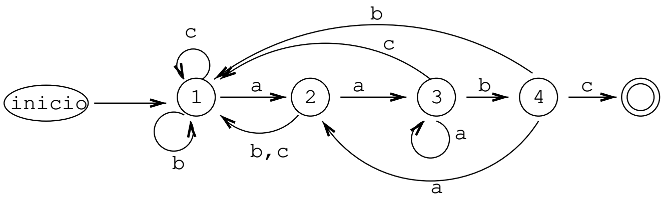


δ :

	a	b
0	1	0
1	0	0

Autômato finito

- Reconhece `AABC`:



- $\Sigma = A, B, C$
- Tempo $O(n)$
- Espaço:
  - Nós $2 + m$
  - Arcos $|\Sigma| m$

BoyerMoore

Heurísticas para o deslocamento:

1. Ocurrence

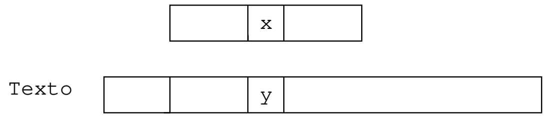
   Alinha Posição no texto que causou "mismatch" com o 1º caractere no padrão

2. Match

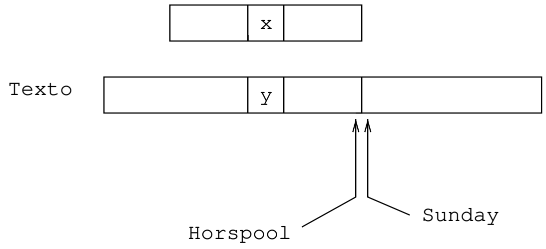
   Ao mover o padrão para a direita, faça-o casar com o pedaço do texto anteriormente casado

Agoritmos clássicos

- Knutt-Morris-Pratt



- Boyer-Moore



Boyer-Moore(Pré-processamento) (FBY92)

```
initd ( pat, m, d )
char pat[];
int m, d[];
{
    int k;
    for(k=0; k<=MAX_ALPHABET_SIZE; k++ ) d[k] = m;
    for( k=1; k<=m; k++ ) d[pat[k]] = m-k;
}
initdd ( pat, m, dd )
char pat[];
int m, dd[];
{
    int j, k, t, f[MAX_PATTERN_SIZE+1];

    for( k=1; k<=m; k++ ) dd[k] = 2*m-k;
    for( j=m, t=m+1; j>0; j--, t-- )
    {
        f[j] = t;
        while( t <= m && pat[j] != pat[t] )
        {
            if ( dd[t] > m-j ) dd[t] = m-j;
            t = f[t];
        }
    }
}
```


Boyer-Moore(FBY92)

```
bmsearch( text, n, pat, m )
/* Search pat[1..m] in text[1..n] */
char text[], pat[];
int n, m;
{
    int k, j, skip;
    int d[MAX_ALPHABET_SIZE], dd[MAX_PATTERN_SIZE];
    initd( pat, m, d ); /* Preprocess pattern */
    initdd( pat, m, dd ); /* Preprocess pattern */

    k = m;  skip = dd[1] + 1;
    while( k <= n )
    {
        j = m;
        while( j>0 && text[k] == pat[j] )
        { j--; k--; }
        if ( j == 0 )
        {
            printf(">> match at %d\n", k+1 );
            printpat( text, n, pat, m, k+1 );
            k += skip;
        }
        if ( d[text[k]] < dd[j] ) k += dd[j];
        else k += d[text[k]];
    }
}
```


Boyewr-Moore - Complexidade

- Tempo
  - Pior caso: $C_n = O(n + rm)$ onde r = nº total de casamentos
  - Melhor caso: $\frac{n}{m}$
  - Caso esperado: $\overline{C_n} = \frac{n}{m}$ se c não é pequeno e m não é muito grande
  - Custo de pré-processamento do padrão: $O(m + c)$
- Espaço: $m + c + O(1)$

Boyer-Moore simplicado (FBY92)

```
bmsearch( text, n, pat, m )
/* Search pat[1..m] in text[1..n] */
char text[], pat[];
int n, m;
{
    int k, j;
    int d[MAX_ALPHABET_SIZE];
    /* Preprocess pattern */
    for(k=0; k<=MAX_ALPHABET_SIZE; k++ ) d[k] = m;
    for( k=1; k<=m; k++ ) d[pat[k]] = m-k;

    k = m;
    while( k <= n )
    {
        j = m;
        while( j>0 && text[k] == pat[j] )
        { j--; k--; }
        if ( j == 0 ) printf( ">>␣match␣at␣%d\n", k+1 );
        k += d[text[k]];
    }
}
```

Boyer-Moore-Horspool

Pré-processamento do pdrão

- Qualquer caractere do texto (a partir do último deslocamento) pode ser usado para endereçar a tabela de ocorrências.
- Horspool: endereça a tabela com o caractere no texto correspondente ao último caractere do padrão.
  $d[x]$=min$\{s | s = m$ ou $(1 \leq s < m$ e p$[m - s] = x)\}$

Boyer-Moore-Horspool

```
bmhsearch( text, n, pat, m )
/* Search pat[1..m] in text[1..n] */
char text[], pat[];
int n, m;
{
    int d[MAX_ALPHABET_SIZE], i, j, k;

    /* Preprocess the pattern */
    for( j=0; k<MAX_ALPHABET_SIZE; j++ ) d[j] = m;
    for( j=1; j<m; j++ ) d[pat[j]] = m-j;

    for( i=m; i <= n; i += d[text[i]] )  /* Search */
    {
        k = i;
        for( j=m; j > 0 && text[k] == pat[j]; j-- ) k--;
        if( j == 0 ) printf( ">> match at %d\n", k+1 );
    }
}
```


Boyer-Moore-Horspool Complexidade

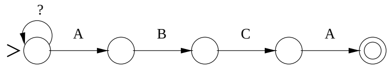
- Tempo
  - Pior caso: $C_n = O(nm)$
  - Melhor caso: $\frac{n}{m}$
  - Caso esperado: $\overline{C_n} = \frac{n}{m}$
    se c não é pequeno e m não é muito grande
  - Custo de pré-processamento do padrão: $O(c)$
- Espaço: $c + O(1)$

Busca aproximada em textos

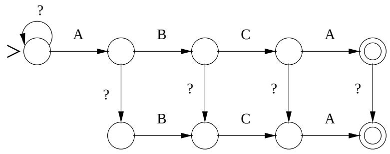
Representação de Busca em Texto Através de Autômatos

- A busca em texto pode ser modelada por autômatos.
- Os caracteres do texto são usados como entradas
- Os casamentos ocorrem quando ativamos um dos estados finais do autômato

Busca exata:

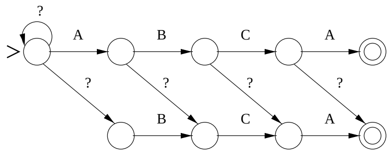


Permitindo 1 Inserção:

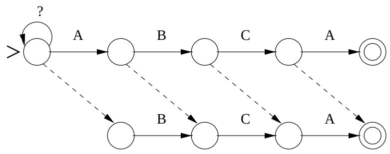


Representação de Busca em Texto Através de Autômatos

Permitindo 1 Substituição:



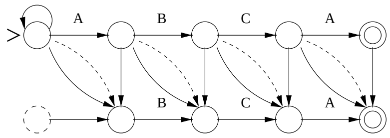
Permitindo 1 Remoção:



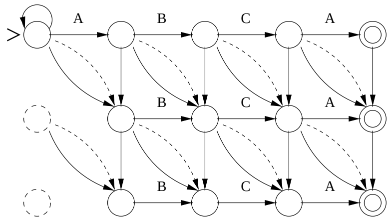
Linhas tracejadas representam transições expontâneas (transições λ).

Representação de Busca em Texto Através de Autômatos

Permitindo 1 Erro:



Permitindo 2 Erros:



Linhas tracejadas representam transições expontâneas (transições λ).

Representação de Autômatos usando bits

Proposta por Baeza-Yates e Gonnet em 1992 (shift-or);

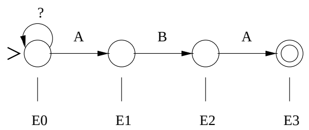
por Wu e Manber 1992 (shift-and).

- Cada estado é representado por 1 bit.
- Estados de uma mesma linha são representados por bits de uma mesma palavra de máquina.

Exemplo:

Padrão: aba

Autômato:



Representação dos estados em bits usando um registrador(R_0):

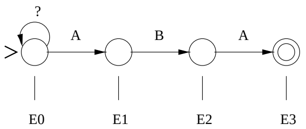
	E 1	E 2	E 3
R_0	0	0	0

Estado Inicial (E0) não precisa ser representado.

Representação de Autômatos usando bits

- Criamos uma máscara para cada caractere do alfabeto.
- Máscara contém 1 nas posições correspondentes aos estados que podem ser ativados pela ocorrência do caractere.

Padrão: aba

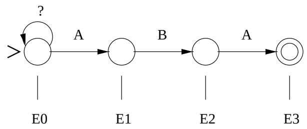


Máscaras:

M[A]	1	0	1
M[B]	0	1	0
M[C]	0	0	0

Representação de Autômatos usando bits

Autômato:



- Pode-se verificar que E_i será ativado se E_{i-1} estiver ativado e M[caractere ocorrido] tiver 1 no i-ésimo bit.

  Ex: A ocorrência de um A no autômato acima sempre ativa o estado E1.

  (A máscara de A possui 1 na posição correspondente ao E1 e E0 está sempre ativado)

- Cálculo da ativação dos estados em R_0:

  $R_0^{atual} = (R_0^{anterior} >> 1)\&M[texto[i]]$

  onde: $>>$ é um shift que preeche o espaços com 1s:

  $0001000 >> 1 = 1000100$

  texto[i] é o caractere que está sendo processado.

Como os valores dos registradores seriam calculados nos demais autômatos ?

Exemplo de uso do shift-and

Padrão: ABAB

Texto: ACCBABABA

1 - Cálculo das máscaras:

M[A]	1	0	1	0
M[B]	0	1	0	1
M[C]	0	0	0	0

Texto	$R_0^{ant} >> 1$					R_0^{atual}			
A	1	0	0	0		1	0	0	0
C	1	1	0	0		0	0	0	0
C	1	0	0	0		0	0	0	0
B	1	0	0	0		0	0	0	0
A	1	0	0	0		1	0	0	0
B	1	1	0	0		0	1	0	0
A	1	0	1	0		1	0	1	0
B	1	1	0	1		0	1	0	*1*
A	1	0	1	0		1	0	1	0
B	1	1	0	1		0	1	0	*1*

Fórmula para calculo de R_0:

$R_0^{atual} = (R_0^{anterior} >> 1)\&M[texto[i]]$