MATA54 - Estruturas de Dados e Algoritmos II

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
UFBA

Prof. George Lima, Ph.D.

Busca de padrões em texto

Motivação: busca de palavras

Aplicações: busca de uma sub-sequência de caracteres

- Texto: sequência de caracteres t de tamanho n
- Padrão: sequência de caracteres p de tamanho $m \leq n$
- Qual é a posição s em t onde p inicia?

Abordagens a serem estudadas

- Força-bruta
- KMP (Knuth-Morris-Pratt)
- BM (Boyer-Moore)
- RK (Rabin-Karp)

Força bruta: ilustração

Ex: t= “abababababaababb” e p = “ababb”

a	b	a	b	a	b	a	b	a	a	b	a	b	b
a	b	a	b	b									
$\rightarrow$	a	b	a	b	b								
	$\rightarrow$	a	b	a	b	b							
		$\rightarrow$	a	b	a	b	b						
			$\rightarrow$	a	b	a	b	b					
				$\rightarrow$	a	b	a	b	b				
					$\rightarrow$	a	b	a	b	b			
						$\rightarrow$	a	b	a	b	b		
							$\rightarrow$	a	b	a	b	b	
								$\rightarrow$	a	b	a	b	b

Força bruta

Busca de $p[0, \ldots, m-1]$ em $t[0, \ldots, n-1]$, $m \leq n$.

```c
int strMacher(char t[], char p[]) {
  int i, j, m, n;

  n = strlen(t);
  m = strlen(p);
  for (i = 0; i < n-m; i++) {
    for (j = 0; j < m; j++) {
      if (p[j] != t[i+j]) break;
    }
    if (j == m) return (i);
  }
  return (-1);
}
```

Complexidade $\Theta(nm)$ – exemplo?

É possível melhorar pré-processando o padrão p

Algoritmo KMP: ideia básica

Objetivo: evitar fazer retrocesso sobre o texto

O que se sabe quando j elementos do padrão foram comparados com o texto e o elemento $p[j+1]$ difere do texto na posição i?

- Que $t[i-j \dots i] = p[0 \dots j]$ (obs: sequência iniciando de 0)
  - Se há prefixo de $p[0 \dots j]$ com tamanho k igual a seu sufixo, então ao avançar p sobre t, pode-se mover $j-k$ posições para que o prefixo emparelhe às posições do texto já comparadas ao seu sufixo.
- Que pode-se evitar retrocessos sobre o texto
- Que o deslocamento do padrão sobre o texto é função do conhecimento sobre o padrão

t:	a	b	a	b	a	b	a	b	a	a	b	a	b	b
p:					a	b	a	b	b					
						$\rightarrow$	a	b	a	b	b			

KMP: ilustração

Ex: $t =$“abababababaababb” e $p =$ “ababb”

a	b	a	b	a	b	a	b	a	a	b	a	b	b
a	b	a	b	b									
	$\to$	a	b	a	b	b							
			$\to$	a	b	a	b	b					
					$\to$	a	b	a	b	b			
							$\to$	a	b	a	b	b	
								$\to$	a	b	a	b	b

KMP: pré-processamento do padrão

Objetivo

Para cada posição j de p, encontrar o tamanho do maior sufixo que é igual a um prexixo de $p[0 \dots j-1]$

$$\pi[j] = \begin{cases} \text{máximo } k > 0 & \text{tal que } p[j-k \dots j-1] = p[0 \dots k-1] \\ 0 & \text{se tal } k \text{ não existe e } j > 0 \\ -1 & \text{se } j = 0 \end{cases}$$

Ex: $p =$ "ababbababa"

p	a	b	a	b	b	a	b	a	b	a
j	0	1	2	3	4	a	b	a	b	a
π	-1	0	0	1	2	0	1	2	3	4

KMP: pré-processamento do padrão

```
void getPi(char p[], int pi[])  {

  int i, j, m;

  m = strlen(p);
  if (m > 0) pi[0] = -1;
  if (m > 1) pi[1] = 0;

  for(i = 2; i < m; i++) { //  m - 2 iteracoes
    j = pi[i-1];
    while (j >= 0 && p[i-1] != p[j]) // < m iteracoes
      j = pi[j];

    pi[i] = j+1;

  }
  return;
}
```

Complexidade: $\Theta(m)$

KMP: busca

```
int kmpMatcher(char t[], char p[], int pi[]) {

int i = 0, j = 0, m, n;

m = strlen(p);
n = strlen(t);
while(j < m && i < n) { // no max. 2n iteracoes
  if (t[i] == p[j]) {
    i++; j++;
  } else {
    j = pi[j];
    if (j <= 0) {
      j = 0; i++;
    }
  }
  }
  if (j == m) return(i - m);
  else return (-1);
}
```

KMP: Complexidade

Pior caso

- Para cada bloco de m caracteres do texto, há não mais que $2m$ comparações
- Menos para o último bloco, para o qual há até m comparações
  - Ex.: t:"$aa \cdots a$" e p:"$aa \cdots b$". A cada comparação com $p[m-1] = b$, avança-se o padrão sobre o texto de apenas uma posição.

  Então, o número total de comparações é limitado a

$$\left\lfloor \frac{n-m}{m} \right\rfloor 2m + m \leq 2(n-m) + m < 2n$$

- O pré-processamento do padrão requer $\Theta(m)$ operações
- Complexidade do pré-processamento e busca é $\Theta(n+m)$

Algoritmo Boyer-Moore: ideia básica

Objetivo: minimizar retrocessos sobre o texto e maximizar deslocamentos do padrão

- Comparações do padrão com o texto da direita para a esquerda
- Deslocamentos para a direita

Se $p[m-j-1] \neq t[i-j]$? (deslocamento do caractere ruim)

- Pode-se avançar $m-k$ posições sobre o texto; k ou é a posição de $t[i]$ mais à direita em p ou é m se $t[i] \neq p$

t:	a	b	a	b	a	b	a	b	a	a	b	a	b	b
p:	a	b	a	a										
		$\rightarrow$	a	b	a	a								
				$\rightarrow$	a	b	a	a						
						$\rightarrow$	a	b	a	a				

Algoritmo Boyer-Moore: ideia básica

Objetivo: minimizar retrocessos sobre o texto e maximizar deslocamentos do padrão

- Comparações do padrão com o texto da direita para a esquerda
- Deslocamentos para a direita

Se $p[j] \neq t[i]$ e $p[m-j+1 \ldots m-1] = t[i+1 \ldots i+m-j-1]$? (deslocamento do bom sufixo)

- Seja $s = p[m-j+1 \ldots m-1]$. Pode-se avançar o p sobre t de forma a alinhar alguma ocorrência mais à esquerda de s com t[i+1 ... i+m-j-1]; Caso s não exista em $p[0 \ldots j]$, o deslocamento do padrão é máximo.

t:	a	b	a	b	a	b	b	a	b	a	a	b	a
p:	a	a	b	a	a	b							
			$\rightarrow$	a	a	b	a	a	b				
							$\rightarrow$	a	a	b	a	a	b

Pré-processamento: Deslocamento do caractere ruim

```c

#define ASIZE 256 // para alfabeto = asc II

void badC(char p[], int badCShift[]) {

  int i, m;

 m = strlen(p);
  for (i = 0; i < ASIZE; i++)
    badCShift[i] = m;
  for (i = 0; i < m - 1; i++)
    badCShift[(int) p[i]] = m - i - 1;
}
```

- Complexidade: $\Theta(m + |\Sigma|)$; $|\Sigma|$: tamanho do alfabeto
- O algoritmo pode ser melhorado para $\Theta(m)$ com uso de hashing

Boyer-Moore: busca

Usando apenas o deslocamento do caractere ruim: Boyer-Moore-Horspool

```
int bmMatcher(char t[], char p[]) {

  int i, j, m, n, badCShift[ASIZE];

  m = strlen(p);
  n = strlen(t);
  badC(p,badCShift);

  i = m - 1;
  while(i < n) {
    j = 0;
    while (j < m && t[i-j] == p[m-j-1]) j++;
    if (j == m) return (i - m + 1);
    i = i + badCShift[(int) t[i]];
  }
  return (-1);
}
```

Boyer-Moore (busca): Complexidade

Pior caso

Há retorcessos, o que leva a $\Theta(nm)$ comparações: igual a força-bruta Ex.: t:"$aa \cdots a$" e p:"$ba \cdots a$".

Melhor caso: considerando padrão não encontrado

Avanços de padrão sobre o texto ocorre com deslocamento máximo, levando a $\Theta(\frac{n}{m})$ comparações
Ex.: t:"$aa \cdots a$" e p:"$bb \cdots b$".

Caso médio*: depende do tamanho do alfabeto Σ

Sob a hipótese de uniformidade da ocorrência dos símbolos no texto e no padrão, o número médio de comparações é

$$O\left(\frac{n}{|\Sigma|}\right)$$

* R. A. Baeza-Yates, M. Regnier. "Average running time of the Boyer-Moore-Horspool algorithm". Theoretical Computer Science 92(992):19-3I, 1992.

Algoritmo Rabin-Karp: ideia básica

Objetivo: comparar padrão com o texto apenas quando necessário

- Uso de função uma hashing $h(.)$ sobre o padrão e sobre cada bloco de m caracteres do texto

Se $h(p) = h(t[i \ldots i + m - 1])$, verifica-se se $t[i \ldots i + m - 1]$ é igual ao padrão. Caso contrário, avança-se uma posição no texto.

$h(p) \neq h(t)$

t:	a	b	a	b	a	b	a	b	a	a	b	a	b	b
p:	a	b	a	a										
	$\rightarrow$	a	b	a	a									
		$\rightarrow$	a	b	a	a								
			$\rightarrow$	a	b	a	a							
				$\rightarrow$	a	b	a	a						
					$\rightarrow$	a	b	a	a					
						$\rightarrow$	a	b	a	a				

$h(p) = h(t)$

Construção da função hashing

Símbolos do alfabeto Σ são codificados numericamente

- Existem $|\Sigma|$ valores numéricos no alfabeto (ex.: ASC II)
- Padrão é então constituído de m dígitos numa base $\mathbf{b}$, $p = d_0 d_{m-2} \cdots d_{m-1}$. Exemplo: $\Sigma = \{0, 1, \ldots, 9\}, \mathbf{b} = 10$
- Com q escolhido de forma apropriada, durante o pré-processamento do padrão,

$$h(p) = \overbrace{\left(\sum_{i=0}^{m-1} p[i] \times \mathbf{b}^{m-i-1}\right)}^{\text{pode ser muito grande}} \bmod q$$

$$h(t[0 \cdots m-1]) = \underbrace{\left(\sum_{i=0}^{m-1} t[i] \times \mathbf{b}^{m-i-1}\right)}_{\text{pode ser muito grande}} \bmod q$$

Controlando valores numéricos

Observações

- Módulo da soma é igual ao módulo da soma dos módulos:

$$h(x) = \left(\sum_{i=0}^{m-1} x[i] \times \mathbf{b}^{m-i-1} \right) \bmod q = \left(\sum_{i=0}^{m-1} \left(x[i] \times \mathbf{b}^{m-i-1} \bmod q \right) \right) \bmod q$$

- q deve ser tal que o valor $q \times \mathbf{b}$ caiba numa palavra de computador (ex.: 64 bits)
- q deve ser preferencialmente primo para diminuir o número de colisões
- Valor de $h(x)$ é computado em $\Theta(m)$

Construindo a solução de Rabin-Karp

Primeira tentativa:

1. $v_p \leftarrow h(p)$ $\Theta(m)$
2. $v_t \leftarrow h(t[0, \ldots, m-1])$ $\Theta(m)$
3. $s \leftarrow 0$
4. Enquanto $s \leq m - n$ $\Theta(n - m)$
   4.1 Se $v_p = v_t$
      4.1.1 Se $p = t[s, \ldots, s + m - 1]$ $\Theta(m)$ somente se $v_p = v_t$
      Retorna s
   4.2 $s \leftarrow s + 1$
   4.3 $v_t \leftarrow h(t[s, \ldots, s + m - 1])$ $\Theta(m)$
5. Retorna -1

Pior caso: $\Theta(m(m - n))$ independentemente de v_p e v_t!!!

Solução: calcular v_t no passo 4.3 em $\Theta(1)$!!!

Construindo a solução de Rabin-Karp

Melhorando os cálculos para $h(t[s, \ldots, s+m-1])$

1. $v_p \leftarrow h(p)$ $\Theta(m)$
2. $v_t \leftarrow h(t[0, \ldots, m-1])$ $\Theta(m)$
3. $w \leftarrow b^{m-1} \bmod q$
4. $s \leftarrow 0$
5. Enquanto $s \leq m - n$ $\Theta(n-m)$
   5.1 Se $v_p = v_t$
      5.1.1 Se $p = t[s, \ldots, s+m-1]$
      Retorna s
   5.2 $s \leftarrow s + 1$
   5.3 Se $s \leq m - n$
      5.3.1 $v_t \leftarrow (v_t - t[s-1] \times w) \times b + t[s+m-1]) \bmod q$ $\Theta(1)$
6. Retorna -1

Passo 5.3.1 calcula v_t usando seu valor da iteração anterior

Rabin-Karp

```
int rkMatcher(char t[], char p[]) {
  long m , n , w =  1, vp = 0, vt = 0, s = 0;
  const long q = 7919; // q primo

   m = strlen(p); n = strlen(t);
  for(int i = 1; i < m; i++)  // equiv B^{m-1}
     w = (w * B) % q;
  for (int i = 0; i < m; i++) { // hash inicial
     vp = (vp * B + p[i]) % q;
     vt = (vt * B + t[i]) % q;
  }
  while(s <= n - m) {
     if (vp == vt && strncmp(p,t+s,m) == 0)
        return s;
  s++;
  if (s <= n - m) {
          vt = (B*(vt - t[s-1] * w) + t[s + m - 1])%q;
          if (vt < 0) vt = vt + q;
  }
 }
  return -1;
}
```

Rabin-Karp: Complexidade

- Apenas pré-processamento: $\Theta(m)$
- Pior caso: colisão com probabilidade 1. Em todas as iterações, $h(p) = h(t[s, \ldots, s+m-1])$: $\Theta(m) + \Theta((n-m+1)m) = O(nm)$
- Melhor caso: colisão com probabilidade 0. Com padrão não encontrado, em todas as iterações $h(p) \neq h(t[s, \ldots, s+m-1])$: $\Theta(m+n-m+1) = \Theta(n)$
- Caso médio*: Considere que $\Pr\{Colisao\} \approx \frac{1}{m}$. Para cada valor de $s = 0, 1, \ldots, m-n$, a verificação custa $\Theta(m)$, então, em média,

$$\Theta(m) + \sum_{s=0}^{n-m} \frac{1}{m}\Theta(m) = \Theta(m) + \Theta(n-m+1) = \Theta(m+n)$$

* R. M. Karp e M. O. Rabin. "Efficient randomized pattern-matching algorithms". IBM Journal of Research and Development. 31 (2): 249?260.

Resumo

- Casamento de padrões em texto, um problema recorrente em várias aplicações
- Força-bruta: solução simples, mas com custo maior que o necessário
- KMP: evita retrocessos no texto com base no conhecimento sobre o padrão
- Boyer-Moore: Considera alfabeto e compara o padrão da direita para esquerda. Pode haver retrocessos sobre o texto, mas com baixa probabilidade para alfabetos grandes
- Rabin-Karp: Uso de hashing. Desempenho depende da função hashing escolhida. Pode ser estendido para casamento de padrões em duas dimensões.