DCC207 – Algoritmos 2

Aula 01 – Algoritmos para manipulação de sequências

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

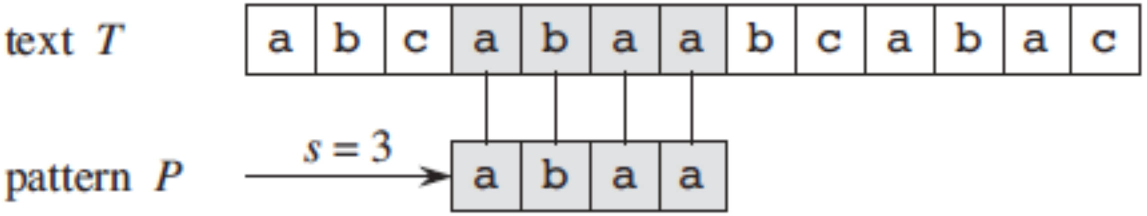
- Nesta aula veremos algoritmos para manipulação de sequências
- Esse problema é recorrente em programas de edição de texto para, por exemplo, encontrar ocorrências de um padrão
- Contudo, o problema também ocorre em outras áreas como:
  - Bioinformática para a busca por padrões em cadeias de DNA ou proteínas
  - Compiladores, interpretadores e processadores de linguagens de programação em geral
  - Compressão de texto
  - Construção de antivírus
- Para simplificar as discussões, a menos que seja dito o contrário, trabalharemos com strings (sequências de caracteres)

String matching

- O primeiro problema que consideraremos é o da busca de padrões em textos
- Um conjunto finito de símbolos (caracteres) Σ é chamado de alfabeto
- Definimos a sequência $T[1..n]$ de caracteres como texto
- Analogamente, um padrão é uma sequência $P[1..m]$ de caracteres
- O problema que queremos tratar então é encontrar um inteiro s ($0 \leq s \leq n-m$) tal que $T[s+1..s+m] = P[1..m]$

String matching

- A figura abaixo ilustra um exemplo de casamento de padrões



String matching

- Antes de iniciarmos a discussão sobre os algoritmos para solucionar o problema, é necessária a definição de alguns conceitos
- O conjunto de todas as strings formadas com os símbolos de um alfabeto é denotado por Σ^*
  - Strings são sequências finitas de símbolos do alfabeto
- O tamanho de uma string w, $|w|$, é o número de símbolos que ela contém
- A string com zero símbolos (tamanho vazio) é denotada por λ
- A concatenação de duas strings x e y é a justaposição dos símbolos de y após os de x
  - x = ab, y = cd, xy = abcd

String matching

- Seja w, x, y strings tais que w=xy
- x é chamada de **prefixo** de w; |x| ≤ |w| e denotamos por $x \sqsubset w$
- y é chamada de **sufixo** de w; |y| ≤ |w| e denotamos por $y \sqsupset w$
- Exemplos:
  - $ab \sqsubset abcca$
  - $cca \sqsupset abcca$
  - $\lambda \sqsupset w$ e $\lambda \sqsubset w$ para $w \in \Sigma^*$
- Note que $\sqsubset$ e $\sqsupset$ são relações transitivas, e $xa \sqsupset ya$ sse $x \sqsupset y$, para $a \in \Sigma$

String matching

- Lema 1 (sobreposição de sufixos): Sejam x, y, z strings quaisquer. Suponha que $x \sqsupset z$ e $y \sqsupset z$. Podemos concluir o seguinte:
  - Se $|x| \leq |y|$, então $x \sqsupset y$
  - Se $|x| \geq |y|$, então $y \sqsupset x$
  - Se $|x| = |y|$, então $x = y$
- A validade do lema pode ser facilmente percebida (visualmente)
- A utilidade do lema será revelada mais à frente

String matching

- Para simplificar a notação, vamos denotar o prefixo P[1..k] do padrão P por P_k
  - Logo, $P_0 = \lambda$ e $P_m = P$
- Utilizaremos a mesma notação para denotar um prefixo do texto
- Logo, o problema do casamento de padrões pode ser formalizado por:
  - Encontrar s tal que $0 \leq s \leq n\text{-}m$, e $P \sqsupset T_{s+m}$

Solução ingênua (força-bruta)

```
NAIVE-STRING-MATCHER(T, P)
n = T.length
m = P.length
for s = 0 to n − m
    if P[1..m] == T[s + 1..s + m]
        print "Pattern occurs with shift" s
```

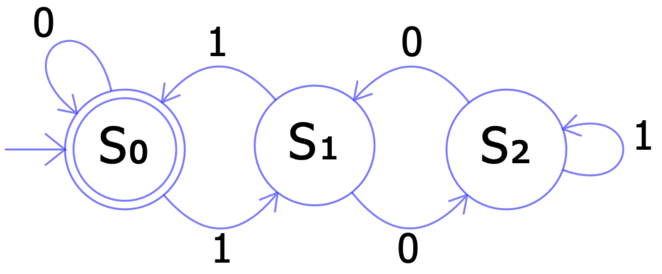
Solução ingênua (força-bruta)

- Essa solução cria uma janela deslizante sobre o texto, testando a cada passo se $P \sqsupset T_{s+m}$
- O custo do algoritmo no pior caso é $O((n-m+1)m)$
- Cada teste $P \sqsupset T_{s+m}$ tem custo $O(m)$
- Note que, para $m \geq n/2$, o algoritmo terá custo $\Theta(n^2)$
- Esse desempenho pode ser melhorado se os casamentos parciais não forem negligenciados

Casamento usando autômatos finitos

- Muitos dos algoritmos mais elaborados para casamento de padrões utilizam autômatos finitos para auxiliar na busca pelos padrões
- Um autômato finito é uma máquina (teórica) para o processamento de dados
- Formalmente, um autômato finito determinístico (AFD) é $M = (Q,\Sigma,\delta,i,F)$ tal que:
  - Q é um conjunto finito de estados (a memória da máquina)
  - $i \in Q$ é o estado inicial da máquina (estado em que ela começa o processamento)
  - $F \subseteq Q$ é um conjunto de estados finais (onde termina computações bem-sucedidas)
  - Σ é o alfabeto de entrada (através do qual as instruções são escritas)
  - $\delta: Q \times \Sigma \rightarrow Q$ é a função de transição (descreve o que a máquina deve fazer ao ler um símbolo qualquer)

Casamento usando autômatos finitos



δ	0	1
S0	S0	S1
S1	S2	S0
S2	S1	S2

Casamento usando autômatos finitos

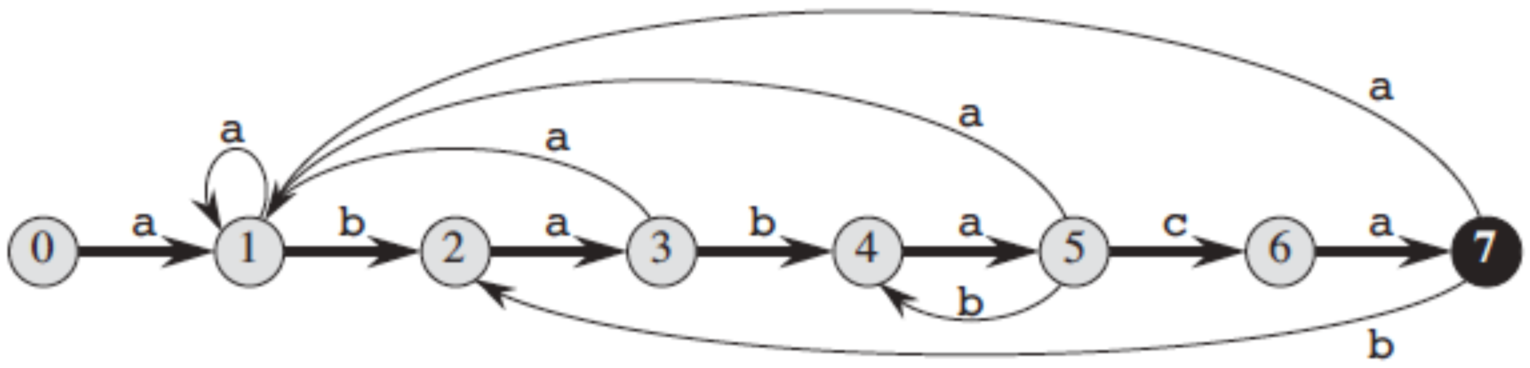
- Podemos estender a função de transição para aceitar strings ao invés de somente símbolos
- A função de transição estendida φ: $Q \times \Sigma^* \rightarrow Q$ é definida recursivamente como:
  - $\varphi(q,\lambda) = q$
  - $\varphi(q,wa) = \delta(\varphi(q,w),a)$
- Para simplificar, caso estejamos aplicando a função sobre o estado inicial, o omitiremos da chamada
  - Ou seja, $\varphi(w) = \varphi(i,w)$

Casamento usando autômatos finitos

- Como dito anteriormente, AFDs são usados como suporte para o casamento de padrões
- Eles auxiliam o processamento indicando o reconhecimento do padrão
- Como veremos mais à frente, eles também funcionam como uma ‘memória’ para os algoritmos, guardando casamentos parciais
- Os AFDs devem ser construídos para cada padrão específico

Casamento usando autômatos finitos

P=ababaca



i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

Casamento usando autômatos finitos

- Definimos uma função sufixo $\sigma: \Sigma^* \rightarrow \mathbb{N}$:
  - $\sigma(w) = \max \{k \mid P_k \sqsupset w\}$
- A função retorna o tamanho do maior prefixo de P que é sufixo de w
- Exemplos (suponha P=ab):
  - $\sigma(\lambda) = 0$
  - $\sigma(ccaca) = 1$
  - $\sigma(ccab) = 2$
- Por definição $x \sqsupset w \rightarrow \sigma(x) \leq \sigma(w)$

Casamento usando autômatos finitos

- Os estados do AFD $Q = \{0, 1, .., m\}$ indicam o número de símbolos casados
  - O estado inicial é 0
  - $F = \{m\}$
- As transições do autômato são definidas por:
  - $\delta(q,a) = \sigma(P_q a)$
- Tal definição objetiva preservar o maior prefixo do padrão casado no texto até o momento
- Assim, supondo que $\varphi(T_i) = q$, sabemos que $P_q \sqsupset T_i$ e $\sigma(T_i) = q$
- Logo, estando no estado q e supondo $T[i+1]=a$, devemos ir para o estado $\sigma(T_i a)=\sigma(P_q a)$

Casamento usando autômatos finitos

```
COMPUTE-TRANSITION-FUNCTION(P, Σ)
m = P.length
for q = 0 to m
    for each character a ∈ Σ
        k = min(m + 1, q + 2)
        repeat
            k = k − 1
        until P_k ⊐ P_q a
        δ(q, a) = k
return δ
```

Casamento usando autômatos finitos

- O algoritmo constrói a função de transição usando sua definição diretamente
- Ele começa tentando avançar para o próximo estado (q+1) e, caso o prefixo atual não possa ser expandido, diminui seu tamanho incrementalmente até que ele se case com $P_q a$
- O algoritmo possui custo $O(m^3|\Sigma|)$
  - O laço mais interno pode requerer m+1 iterações
  - Em cada iteração, o teste de $P_k \sqsupset P_q a$ tem custo O(m)
- Veremos algoritmos mais eficientes para a construção do autômato

Casamento usando autômatos finitos

FINITE-AUTOMATON-MATCHER (T, δ, m)

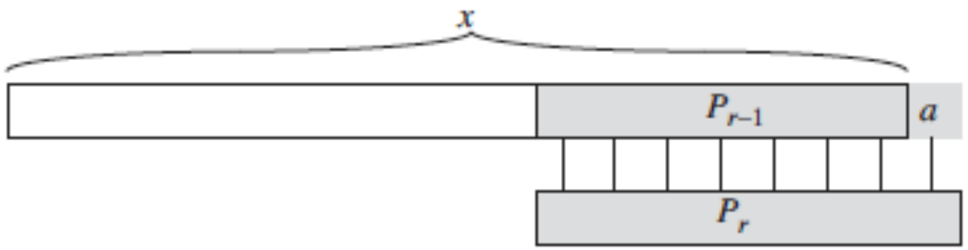
```
n = T.length
q = 0
for i = 1 to n
    q = δ(q, T[i])
    if q == m
        print "Pattern occurs with shift" i − m
```

Casamento usando autômatos finitos

- Para demonstrar a correção do algoritmo, devemos demonstrar que o autômato se encontra no estado $\sigma(T_i)$ após a leitura do caractere T[i]
- Como $\sigma(T_i)=m$ sse $P \sqsupset T_i$, o autômato atingirá o estado final se, e somente se ele tiver lido o padrão P
  - Portanto, a prova da propriedade acima demonstrará também a correção do algoritmo
- Para demonstrar a propriedade, precisamos demonstrar outros dois lemas

Casamento usando autômatos finitos

- Lema 2: Sejam x e a, respectivamente, uma string e um caractere arbitrários. Então, σ(xa) ≤ σ(x) + 1



Casamento usando autômatos finitos

- Lema 3: Seja $x \in \Sigma^*$ e $a \in \Sigma$. Se $q = \sigma(x)$, então $\sigma(xa) = \sigma(P_q a)$
- Prova: Suponha que $\sigma(x) = q$. Temos dois casos:
- $\sigma(P_q a) \leq \sigma(xa)$: $\sigma(x) = q \rightarrow P_q \sqsupset x$. Logo, $P_q a \sqsupset xa$, e $\sigma(P_q a) \leq \sigma(xa)$.
- $\sigma(xa) \leq \sigma(P_q a)$: Suponha que $\sigma(xa) = r$. Logo, $P_r \sqsupset xa$. Pelo lema 2, $|P_r| = r \leq q+1 = |P_q a|$. Como $P_r \sqsupset xa$, $P_q a \sqsupset xa$ e $|P_r| \leq |P_q a|$, segue, pelo lema 1, que $P_r \sqsupset P_q a$. Portanto, $\sigma(xa) \leq \sigma(P_q a)$.

Casamento usando autômatos finitos

- Teorema: Sejam φ a função de transição estendida para o autômato de um padrão P, e T[1..n] um texto. Para todo i = 0..n, $\varphi(T_i) = \sigma(T_i)$.
- Prova: Por indução no número de caracteres lidos.
- Passo base: $\varphi(T_0) = 0 = \sigma(T_0)$
- Passo indutivo. Suponha que $\varphi(T_i) = \sigma(T_i) = q$. Seja T[i+1] = a, um símbolo arbitrário.
  - $$\begin{aligned} \varphi(T_{i+1}) &= \varphi(T_i a) \\ &= \delta(\varphi(T_i),a) \\ &= \delta(q,a) \\ &= \sigma(P_q a) \\ &= \sigma(T_i a) && \text{(lema 3)} \\ &= \sigma(T_{i+1}) \end{aligned}$$

Casamento usando autômatos finitos

- O teorema mostra que, após a leitura de i símbolos, o autômato armazenará o maior prefixo do padrão encontrado como sufixo da sequência lida. Em outras palavras, o autômato está no estado final após a leitura do i-ésimo símbolo sse o padrão foi encontrado no texto.
- Pelo laço principal do algoritmo, percebemos que seu custo é $\Theta(n)$
- Esse custo é muito menor que do algoritmo ingênuo, porém ele não inclui o custo de construir o autômato

Leitura

- Cormen seções 32.1, 32.3 (de onde as figuras foram retiradas).