Methods of Analysis of Textual Data (MATD)

Jiří Dvorský
October 11, 2019

Department of Computer Science
VŠB – TU Ostrava
1. Pattern Matching

   Exact pattern matching

     Searching for finite set of patterns
     Searching for (Regular) Infinite Set of Patterns in Text

   Approximate pattern matching
Pattern Matching

Jiří Dvorský

Department of Computer Science
VŠB – TU Ostrava
Pattern Matching

Exact pattern matching
1. How to describe infinite set of pattern i.e. string?  
   Regular Expressions

2. What shall we use to perform matching?  
   Finite Automata
## Regular Expressions and Languages

<table>
<thead>
<tr>
<th>Regular expression $R$</th>
<th>Value of expression $h(R)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Atomic expressions</strong></td>
<td></td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>${\varepsilon}$</td>
</tr>
<tr>
<td>$a, a \in \Sigma$</td>
<td>${a}$</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td></td>
</tr>
<tr>
<td>$U \cdot V$</td>
<td>${uv</td>
</tr>
<tr>
<td>$U + V$</td>
<td>$h(U) \cup h(V)$</td>
</tr>
<tr>
<td>$V^k = V \cdot V \cdot \ldots \cdot V$</td>
<td>(k\ times)</td>
</tr>
<tr>
<td>$V^+ = V^0 + V^1 + V^2 + \ldots$</td>
<td></td>
</tr>
<tr>
<td>$V^* = V^0 + V^1 + V^2 + \ldots$</td>
<td></td>
</tr>
</tbody>
</table>
Regular Expression Features

\[ U + (V + W) = (U + V) + W \]
\[ U \cdot (V \cdot W) = (U \cdot V) \cdot W \]
\[ U + V = V + U \]
\[ (U + V) \cdot W = (U \cdot W) + (V \cdot W) \]
\[ U \cdot (V + W) = (U \cdot V) + (U \cdot W) \]
\[ U + U = U \]
\[ \varepsilon \cdot U = U \]
\[ \emptyset \cdot U = \emptyset \]
\[ U + \emptyset = U \]
\[ U^* = \varepsilon + U^* \]
Deterministic Finite Automaton

Definition

Deterministic Finite Automaton (DFA) is a quintuple \( A = (Q, \Sigma, q_0, \delta, F) \), where

- \( Q \) is a finite set of states
- \( \Sigma \) is an alphabet
- \( q_0 \in Q \) is an initial state
- \( \delta : Q \times \Sigma \rightarrow Q \) is a transition function
- \( F \subseteq Q \) is a set of final states
Configuration of Finite Automaton

\[(q, w) \in Q \times \Sigma^*\]

Transition of Finite Automaton is a relation

\[\mapsto : (Q \times \Sigma^*) \times (Q \times \Sigma^*)\]

such as

\[(q, aw) \mapsto (q', w) \iff \delta(q, a) = q'\]

Automaton accepts word \(w\) if

\[(q_0, w) \mapsto^* (q, \varepsilon), \; q \in F\]
### Nondeterministic Finite Automaton

#### Definition

Nondeterministic Finite Automaton (NFA) is a quintuple $A = (Q, \Sigma, q_0, \delta, F)$, where

- $Q$ is a finite set of states
- $\Sigma$ is an alphabet
- $q_0 \in Q$ is an initial state
- $\delta : Q \times \Sigma \rightarrow P(Q)$ is a transition function
- $F \subseteq Q$ is a set of final states

- Alternatively NFA can be defined as $A = (Q, \Sigma, S, \delta, F)$, where $S \subseteq Q$ is a set of initial states.
- For each NFA, there is a DFA such that it recognizes the same formal language.
Set of patterns $P = \{\text{he}, \text{her}, \text{she}\}$
The DFA can be constructed using the powerset construction. NFA $A = (Q, \Sigma, S, \delta, F) \rightarrow$ DFA $A' = (Q', \Sigma', q'_0, \delta', F')$

- $Q' \subseteq P(Q)$
- $\Sigma' = \Sigma$
- $q'_0 = S$
- $\delta'(q', x) = \cup \delta(q, x)$ for all $q \in q'$
- $F' = \{q' \in Q' | q' \cap F \neq \emptyset\}$
NFA $\rightarrow$ DFA Conversion I

<table>
<thead>
<tr>
<th>State</th>
<th>Label</th>
<th>$e$</th>
<th>$h$</th>
<th>$r$</th>
<th>$s$</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1, 4, 8}$</td>
<td>$q'_1$</td>
<td>${1, 4, 8}$</td>
<td>${1, 2, 4, 5, 8}$</td>
<td>${1, 4, 8}$</td>
<td>${1, 4, 8, 9}$</td>
<td>${1, 4, 8}$</td>
</tr>
<tr>
<td>${1, 2, 4, 5, 8}$</td>
<td>$q'_2$</td>
<td>${1, 3, 4, 6, 8}$</td>
<td>${1, 2, 4, 5, 8}$</td>
<td>${1, 4, 8}$</td>
<td>${1, 4, 8, 9}$</td>
<td>${1, 4, 8}$</td>
</tr>
<tr>
<td>${1, 4, 8, 9}$</td>
<td>$q'_3$</td>
<td>${1, 4, 8}$</td>
<td>${1, 2, 4, 5, 8, 10}$</td>
<td>${1, 4, 8}$</td>
<td>${1, 4, 8, 9}$</td>
<td>${1, 4, 8}$</td>
</tr>
<tr>
<td>${1, 3, 4, 6, 8}$</td>
<td>$q'_4$</td>
<td>${1, 4, 8}$</td>
<td>${1, 2, 4, 5, 8}$</td>
<td>${1, 4, 7, 8}$</td>
<td>${1, 4, 8, 9}$</td>
<td>${1, 4, 8}$</td>
</tr>
<tr>
<td>${1, 2, 4, 5, 8, 10}$</td>
<td>$q'_5$</td>
<td>${1, 3, 4, 6, 8, 11}$</td>
<td>${1, 2, 4, 5, 8}$</td>
<td>${1, 4, 8}$</td>
<td>${1, 4, 8, 9}$</td>
<td>${1, 4, 8}$</td>
</tr>
<tr>
<td>${1, 4, 7, 8}$</td>
<td>$q'_6$</td>
<td>${1, 4, 8}$</td>
<td>${1, 2, 4, 5, 8}$</td>
<td>${1, 4, 8}$</td>
<td>${1, 4, 8, 9}$</td>
<td>${1, 4, 8}$</td>
</tr>
<tr>
<td>${1, 3, 4, 6, 8, 11}$</td>
<td>$q'_7$</td>
<td>${1, 4, 8}$</td>
<td>${1, 2, 4, 5, 8}$</td>
<td>${1, 4, 7, 8}$</td>
<td>${1, 4, 8, 9}$</td>
<td>${1, 4, 8}$</td>
</tr>
</tbody>
</table>

Only reachable states, transitions to state $q'_1$ are not shown.
### NFA → DFA Conversion II

<table>
<thead>
<tr>
<th>State</th>
<th>Label</th>
<th>( e )</th>
<th>( h )</th>
<th>( r )</th>
<th>( s )</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>( q'_1 )</td>
<td>{1}</td>
<td>{1,2}</td>
<td>{1}</td>
<td>{1,5}</td>
<td>{1}</td>
</tr>
<tr>
<td>{1,2}</td>
<td>( q'_2 )</td>
<td>{1,3}</td>
<td>{1,2}</td>
<td>{1}</td>
<td>{1,5}</td>
<td>{1}</td>
</tr>
<tr>
<td>{1,5}</td>
<td>( q'_3 )</td>
<td>{1}</td>
<td>{1,2,6}</td>
<td>{1}</td>
<td>{1,5}</td>
<td>{1}</td>
</tr>
<tr>
<td>{1,3}</td>
<td>( q'_4 )</td>
<td>{1}</td>
<td>{1,2}</td>
<td>{1,4}</td>
<td>{1,5}</td>
<td>{1}</td>
</tr>
<tr>
<td>{1,2,6}</td>
<td>( q'_5 )</td>
<td>{1,3,7}</td>
<td>{1,2}</td>
<td>{1}</td>
<td>{1,5}</td>
<td>{1}</td>
</tr>
<tr>
<td>{1,4}</td>
<td>( q'_6 )</td>
<td>{1}</td>
<td>{1,2}</td>
<td>{1}</td>
<td>{1,5}</td>
<td>{1}</td>
</tr>
<tr>
<td>{1,3,7}</td>
<td>( q'_7 )</td>
<td>{1}</td>
<td>{1,2}</td>
<td>{1,4}</td>
<td>{1,5}</td>
<td>{1}</td>
</tr>
</tbody>
</table>

![NFA Diagram](image1)

![DFA Diagram](image2)
Derivation of Regular Expression

For given regular expression $R$, derivation is defined as

$$h\left(\frac{dR}{dx}\right) = \{y | xy \in h(R)\}$$

Example

For $R = a + shell + stop + plot$ and its value $h(R) = \{a, shell, stop, plot\}$ derivations are

$$h\left(\frac{dR}{da}\right) = \{\varepsilon\}$$
$$h\left(\frac{dR}{ds}\right) = \{hell, top\}$$
$$h\left(\frac{dR}{dt}\right) = \emptyset$$
Derivation of Regular Expression – properties

\[
\frac{d\emptyset}{da} = \emptyset, \forall a \in \Sigma \\
\frac{d\varepsilon}{da} = \emptyset, \forall a \in \Sigma \\
\frac{da}{da} = \varepsilon, \forall a \in \Sigma \\
\frac{db}{da} = \emptyset, \forall b \neq a \\
\]

\[
\frac{d(U + V)}{da} = \frac{dU}{da} + \frac{dV}{da} \\
\frac{d(U \cdot V)}{da} = \frac{dU}{da} \cdot V, \varepsilon \notin U \\
\frac{d(U \cdot V)}{da} = \frac{dU}{da} \cdot V + \frac{dV}{da}, \varepsilon \in U \\
\frac{dV^*}{da} = \frac{dV}{da} \cdot V^* \\
\]

\[
\frac{dV}{dx} = \frac{d}{da_n} \left( \frac{d}{da_{n-1}} \left( \ldots \frac{d}{da_2} \left( \frac{dV}{da_1} \right) \right) \right), \text{ for } x = a_1 a_2 \ldots a_n
\]
Construction of DFA Derivations of RE

• Derivation of regular expressions allows directly and algorithmically build DFA for any regular expression.

• Let $V$ is given regular expression in alphabet $\Sigma$.

• Each state of DFA defines a set of words, that move the DFA from this state to any of final states.

So, every state can be associated with regular expression, defining this set of words

\[
q_0 = V \\
\delta(q, x) = \frac{\text{dq}}{\text{dx}} \\
F = \{q \in Q | \varepsilon \in h(q)\}
\]
Lest’s have $V = (0 + 1)^* \cdot 01$ over alphabet $\Sigma\{0, 1\}$.

Then $q_0 = (0 + 1)^* \cdot 01$

Example of derivations:

$$\frac{d((0 + 1)^* \cdot 01)}{d0} = \frac{d((0 + 1)^*)}{d0} \cdot 01 + \frac{d01}{d0}$$

$$= \frac{d(0 + 1)}{d0} \cdot (0 + 1)^* \cdot 01 + 1$$

$$= \left(\frac{d0}{d0} + \frac{d1}{d0}\right) \cdot (0 + 1)^* \cdot 01 + 1$$

$$= (\varepsilon + \emptyset) \cdot (0 + 1)^* \cdot 01 + 1$$

$$= (0 + 1)^* \cdot 01 + 1$$
\[ \begin{align*}
&\frac{d((0 + 1)^* \cdot 01)}{d1} = \frac{d((0 + 1)^*)}{d1} \cdot 01 + \frac{d01}{d1} \\
&= \frac{d(0 + 1)}{d1} \cdot (0 + 1)^* \cdot 01 + \emptyset \\
&= \left(\frac{d0}{d1} + \frac{d1}{d1}\right) \cdot (0 + 1)^* \cdot 01 \\
&= (\emptyset + \varepsilon) \cdot (0 + 1)^* \cdot 01 \\
&= (0 + 1)^* \cdot 01
\end{align*} \]
Construction of DFA Derivations of RE – example (cont.)

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(0 + 1)^* \cdot 01$</td>
<td>$q_0$</td>
<td>$(0 + 1)^* \cdot 01 + 1$</td>
<td>$(0 + 1)^* \cdot 01$</td>
</tr>
<tr>
<td>$(0 + 1)^* \cdot 01 + 1$</td>
<td>$q_1$</td>
<td>$(0 + 1)^* \cdot 01 + 1$</td>
<td>$(0 + 1)^* \cdot 01 + \varepsilon$</td>
</tr>
<tr>
<td>$(0 + 1)^* \cdot 01 + \varepsilon$</td>
<td>$q_2$</td>
<td>$(0 + 1)^* \cdot 01 + 1$</td>
<td>$(0 + 1)^* \cdot 01$</td>
</tr>
</tbody>
</table>
Pattern Matching
Approximate pattern matching
Approximate pattern matching

- **String metric** (string distance function) is a metric that measures distance between two text strings for approximate string matching.
- String metric can be considered as “inverse similarity” – how two strings are dissimilar.
- There are two classic metrics
  1. Hamming distance
  2. Levenshtein distance
- Yes, string dissimilarity, distance can be measured. Both distances are metrics from mathematical point of view – non-negativity, identity, symmetry, and triangle inequality.
**Hamming distance**

<table>
<thead>
<tr>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.</td>
</tr>
</tbody>
</table>

In other words, it measures the minimum number of substitutions required to change one string into the other.

<table>
<thead>
<tr>
<th><strong>Example</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming distance of “karolin” and “kathrin” is 3.</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccccc}
\text{karolin} & 1 & 1 & 1 & 1 & 0 & 0 \\
\text{kathrin} & 0 & 0 & 1 & 1 & 1 & 0 \\
\end{array}
\]
Levenshtein distance

Definition

Levenshtein distance (1965) between two strings is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one string into the other.
Example
Levenshtein distance between “kitten” and “sitting” is 3:

1. kitten → sitten (substitution of “s” for “k”)
2. sitten → sittin (substitution of “i” for “e”)
3. sittin → sitting (insertion of “g” at the end).

There is no way to do it with fewer than three edits.
Upper and lower bounds

The Levenshtein distance has several simple upper and lower bounds:

- It is at least the difference of the sizes of the two strings.
- It is at most the length of the longer string.
- It is zero if and only if the strings are equal.
- If the strings are the same size, the Hamming distance is an upper bound on the Levenshtein distance.
- The Levenshtein distance between two strings is no greater than the sum of their Levenshtein distances from a third string (triangle inequality).
Levenshtein distance (cont.)

\[ d(i, j) = \begin{cases} 
  i, & \text{if } j = 0 \\
  j, & \text{if } i = 0 \\
  \min \left( \begin{array}{c} 
    d(i - 1, j) + 1, \\
    d(i, j - 1) + 1, \\
    d(i - 1, j - 1) + c(i, j) 
  \end{array} \right) 
\end{cases} \]

where

\[ c(i, j) = \begin{cases} 
  0 & \text{if } a_i = b_j \\
  1 & \text{otherwise} 
\end{cases} \]

First element in the minimum corresponds to deletion (from \( a \) to \( b \)), the second to insertion and the third to match or mismatch.
Levenshtein distance (cont.)

```c
int LevenshteinDistance(const char *s, int len_s, const char *t, int len_t)
{
    int cost;

    /* base case: empty strings */
    if (len_s == 0) return len_t;
    if (len_t == 0) return len_s;

    /* test if last characters of the strings match */
    if (s[len_s-1] == t[len_t-1])
        cost = 0;
    else
        cost = 1;

    return cost;
}
```
Levenshtein distance (cont.)

    /* return minimum of delete char from s, delete char from t, and delete char from both */
    return minimum
    (LevenshteinDistance(s, len_s-1, t, len_t) + 1,
     LevenshteinDistance(s, len_s, t, len_t-1) + 1,
     LevenshteinDistance(s, len_s-1, t, len_t-1) + cost
    );
Levenshtein distance (cont.)

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>i</th>
<th>t</th>
<th>t</th>
<th>e</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>2a</td>
<td>2b</td>
<td>2c</td>
<td>3d</td>
<td>3e</td>
</tr>
<tr>
<td>i</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>t</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>n</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>g</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

- subst. of k for s
- i is equal i
- t is equal t
- subst. of e for i
- n is equal n
- delete g
Approximate pattern matching using finite automata

NFA for the exact string matching ($m = 4$)
Approximate pattern matching using finite automata (cont.)

NFA for the approximate string matching using the Hamming distance \((m = 4, k = 3)\)
Approximate pattern matching using finite automata (cont.)

NFA for the approximate string matching using the Levenshtein distance \((m = 4, k = 3)\)