• String matching
  – Task: Search for text within a file or collection of files
  – Pervasive: Text editors, Web crawlers, Unix grep command, ...
  – Simple Web example: search for e-mail addresses

• Brute force technique
  – $j = \text{sub.length}()$
  – if $\text{str.substring}(i, i + j).\text{equals}(\text{sub})$ for some $i$, return $i$
  – If no such $i$ can be found, return -1
  – Complexity??

• More efficient technique
  – At each position $i < \text{str.length}()$, maintain a set of indices $j$ such that $\text{str.substring}(i - j, i).\text{equals}(\text{sub.substring}(0, j))$
  – If $\text{sub.length}() \in j$ then return $i$
  – If $i == \text{str.length}()$ return -1
  – Complexity??

• String matching with patterns
  – Often, we wish to find a substring which matches a pattern
  – E-mail addresses:
    1. Any number of alphanumeric characters and/or dots (not a dot at beginning or end)
    2. @
    3. Any number of alphanumeric characters and/or dots (not a dot at beginning or end); must be at least one dot
  – E-mails: lytinen@cs.depaul.edu, steven.lytinen@gmail.com, steve23@yahoo.com
  – Not e-mails: .lytinen@cs.depaul.edu, lytinen@depaul steve.@depaul.edu.
• **Regular expressions: An algebra for defining patterns**
  
  – Atomic Operands:
    1. a single character
    2. \( \epsilon \) (empty string)
    3. a variable
  
  – Operators:

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Description</th>
<th>Symbol</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenation</td>
<td>What it sounds like</td>
<td>(no symbol)</td>
<td>abc</td>
</tr>
<tr>
<td>Union</td>
<td>OR</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>Closure</td>
<td>0 or more</td>
<td>*</td>
<td>a*</td>
</tr>
</tbody>
</table>

  – Parentheses can be used as necessary; e.g., \((ab)^*\)

• **Regular expressions and languages**
  
  – A **language** is simply a set of strings over an alphabet

    Examples: Alphabet: \( a, b, c \)

    \[
    \{a, ab, abb, abc\} \\
    \{a, aa, aaa, aaaa, \ldots\} \\
    \{\epsilon, a\} \\
    \{\} \\
    \]

  – A regular expression denotes a language

  – For atomic operands:

    \[
    L(x) = \{x\} \\
    L(\epsilon) = \{\epsilon\} \\
    \]

  – For operators:

    \[
    L(R|S) = L(R) \cup L(S) \\
    L(RS) = \{rs|r \in L(R), s \in L(S)\} \\
    L(R^*) = \{\epsilon\} \cup L(RR^*) \\
    \]

  – Precedence: Closure, Concatenation, Union
Examples

\[ L(a) = L(\epsilon) = L(abc) = L(a^*b^*) = L(ab)^* = L(((a|b|c)d)^*) = L(a|b|cd^*) = \]

- **Regular expression matching**
  
  - Task: Does a string \( S \) match a regular expression \( E \)?
  - Equivalent task? Is \( S \in L(E) \)
  - Example: Does \( aaaaaabbb \) match \( a^*b^* \)? (yes)

- **Java regular expression matching**
  
  - \texttt{java.regexp.Pattern} is a class which can be used to match strings against regular expressions
  - If a pattern is to be used frequently, \texttt{Pattern.compile} is a static method which compiles a regular expression (makes matching more efficient)
    
    * Example of typical usage
      
      ```java
      Pattern p = Pattern.compile("a*b");
      Matcher m = p.matcher("aaaaab");
      boolean b = m.matches(); // returns true
      ```
    * Or, if a pattern will not be used frequently:
      
      ```java
      Pattern.matches("a*b", "aaaaab"); // returns true
      ```

- **Java regular expressions**
  
  - \url{http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html}
  - Examples (assume only lower case letters)
    
    * No a’s
    * At least one a
    * Odd number of a’s
    * All a’s before all b’s
    * Vowels consecutive and in order
- Vowels in order, but not necessarily consecutive
- Only a’s and b’s; even number of each
- Only a’s and b’s; even number of both or odd number of both

- **Example application**: Find e-mail addresses

- **Compiling regular expressions**
  - Regular expressions can be matched in $\Theta(n)$ time ($n$ is the length of the text to be matched)
  - However, first they must be compiled into an equivalent *deterministic finite state automaton* (DFA)
  - 2-step compilation
    1. Regular expression $\rightarrow$ **Non-deterministic** finite state automaton (NFA)
    2. NFA $\rightarrow$ DFA

- **Finite State Automata**
  - $A = (I, \Sigma, r, A, \sigma)$
  - $I$: set of possible **input symbols**
  - $\Sigma$: set of **states**
  - $r$: a **next-state** relation from $\Sigma \times I$ into $P(\Sigma)$ (power set = set of all possible subsets)
  - $A \subseteq \Sigma$: set of **accepting states**
  - $\sigma$: the **initial state** ($\sigma \in \Sigma$)

- **FSA and Graphs**
  - **Directed graph**
  - Vertices are the members of $\Sigma$
  - Members of $A$ are marked (drawn with double circle)
  - Initial state $\sigma$ is designated by an arrow
  - Directed edge $\sigma_a, \sigma_b$ exists for each $i \in I$ such that $(\sigma_a, i, \sigma_b) \in r$, labeled $i$ (there may be parallel edges and loops)
• FSAs and accepting strings

- \( A = (I, \Sigma, r, A, \sigma) \)
- Input \( \alpha = x_1...x_n \)
- \( \alpha \) is accepted by \( A \) if there exist states \( \sigma_0...\sigma_n \) such that:
  \[
  \begin{align*}
  \sigma_0 &= \sigma \\
  \sigma_i &\in r(\sigma_{i-1}, x_i) \\
  \sigma_n &\in A
  \end{align*}
  \]

- Example: draw a graph which represents an FSA that accepts strings over \{a,b\} which contain an even number of a’s

• Nondeterministic finite state automata

- Consider an FSA which contains some \( \sigma_v \in \Sigma \) and the set of edges \( (\sigma_v, \sigma_w) \) for any \( w \).
  If more than one such edge has the same label \( i \), then the automaton is a nondeterministic finite state automaton (NFA)
- Example:

  \[
  \begin{array}{ccc}
  a, b \\
  \rightarrow & \rightarrow & \\
  \downarrow & / & \\
  \downarrow & / & \\
  --- & a & --- \\
  \rightarrow & | 0 | & --- & | 1 | \\
  --- & --- & \\
  \end{array}
  \]

  State 1 is an accepting state

  This NFA accepts strings over \{a,b\} which end with ‘a’

• Deterministic Finite State Automata (DFA)

- If an FSA contains no pair of edges \( (\sigma_v, \sigma_w) \) with the same label, then it is said to be deterministic.
- Restating, \( r \) in earlier definition is a function from \( \Sigma \times I \) into \( P(\Sigma) \)
- A DFA can accept (or not accept) a string of length \( n \) in \( \Theta(n) \) time
• Examples
  – Accept strings over $a, b$ that contain an odd number of $a$’s
  – Accept strings over $a, b, c$ of the form $a^*b^*c^*$
  – Accept strings over $a, b$ in which every $a$ is followed by $a \ b$

• Constructing an equivalent DFSA from an NFA
  – Technique: subset construction
  – States in DFSA correspond to sets of states in the NFA
  – $N$: NFA. $D$: DFSA.
  – Start state of $D = $ start state of $N$
  – To construct other states:
    Consider each possible input character for an already constructed state $S$ of DFSA.
    Compute $T$:
    $T = \{ t \in \Sigma_N | \text{for some } s \in S, \{ t, \cdot, s \} \in R \}$

• Subset construction examples
  1. $a^+b^+c^+$ (or $a^*ab^*bc$)
     
     \[
     \begin{array}{c}
     \text{a} & \text{b} & \text{c} \\
     \text{-<-} & \text{-<-} & \text{-<-} \\
     \text{\textbackslash /} & \text{\textbackslash /} & \text{\textbackslash /} \\
     \text{--\textgreater} 0 \text{ ------\textgreater} 1 \text{ ------\textgreater} 2 \text{ ------\textgreater} 3 \\
     \text{a} & \text{b} & \text{c}
     \end{array}
     \]
    (3 is accepting)

  2. $((a\mid b)^*b(a\mid b)^*b(a\mid b)^*b(a\mid b)^*$
     
     \[
     \begin{array}{c}
     \text{a,b} & \text{a,b} & \text{a,b} & \text{a,b} \\
     \text{-<-} & \text{-<-} & \text{-<-} & \text{-<-} \\
     \text{\textbackslash /} & \text{\textbackslash /} & \text{\textbackslash /} & \text{\textbackslash /} \\
     \text{--\textgreater} 0 \text{ ------\textgreater} 1 \text{ ------\textgreater} 2 \text{ ------\textgreater} 3 \\
     \text{b} & \text{b} & \text{b}
     \end{array}
     \]
    (3 is accepting)
• Constructing an FSA from a regular expression

  – An \( \epsilon \) transition is an edge in an NFA which may be followed to new state without consuming any input symbols. Example: \( a^*b^* \)

  \[
  \begin{array}{ccc}
  a & b \\
  \text{-<-} & \text{-<-} \\
  \text{	extbackslash /} & \text{	extbackslash /} \\
  \text{---> 0 -----> 1} \\
  \text{eps}
  \end{array}
  \]

  – Automata for regular expression operations:

  1. Concatenation: \( ab \)

  \[
  \begin{array}{ccc}
  a & b \\
  \text{0 -----> 1 -----> 2}
  \end{array}
  \]

  2. Union: \( a\mid b \): requires \( \epsilon \) transitions

  \[
  \begin{array}{ccc}
  e & a & e \\
  \text{------> 1 -----> 2 -->>--} \\
  \text{/} \\
  \text{0} \\
  \text{\textbackslash /} \\
  \text{------> 3 -----> 4 -->>--} \\
  \text{e} & b & e
  \end{array}
  \]

  3. Closure: \( a^* \)

  \[
  \begin{array}{ccc}
  e \\
  \text{-<-----} \\
  \text{\textbackslash | \textbackslash} \\
  \text{e} & a & e \\
  \text{0 -----> 1 -----> 2 -----> 3} \\
  \text{\textbackslash /} \\
  \text{\textbackslash /} \\
  \text{---------->-----------}
  \end{array}
  \]

• Examples

\[
\begin{align*}
\text{a\mid ab} \\
\text{(ab)*c} \\
\text{ab\mid(ab*c)*}
\end{align*}
\]

• Eliminating \( \epsilon \) transitions

  – Add to \( \mathcal{A} \) all states \( \sigma_v \) for which a path from \( \sigma_v \) to \( \sigma_w \) consists of only \( \epsilon \) transitions, and \( \sigma_w \in \mathcal{A} \)
– Eliminate those states (except start state) which only $\epsilon$ transitions lead to
– Replace each path whose edges are all labeled $\epsilon, \ldots, \epsilon, i$ with an edge labelled $i$
– Example: $a^*|b^*$

\[
\begin{array}{cccc}
e & a & e \\
-----> 1 -----> 2 -->-- \\
/ & \quad/ & e & / \\
/ & \quad--<>-- & \quad/ \\
---> 0 \quad\quad\quad 5 \\
\quad/ & \quad\quad/ & e \quad\quad/ \\
\quad\quad/ & \quad\quad/ & e \quad\quad/ \\
-----> 3 -----> 4 -->-- \\
e \quad b \quad e
\end{array}
\]
also an epsilon edge from 0 to 5

5 is accepting
  * Add 0, 2 and 4 to $A$, since epsilons lead to 5
  * Eliminate 1, 3, and 5 since only epsilons lead to them
  * Edges: (0,a,2), (0,b,4), (2,a,2), (4,b,4)

\[
\begin{array}{c}
a \\
-<-
\end{array}
\begin{array}{c}
a \quad/ \\
-----> 2 \quad/ \\
/ \\
---> 0 \\
\quad/ \\
-----> 4 \quad/ \\
\quad/ \quad/<-
\end{array}
\begin{array}{c}
b
\end{array}
\]
– Example: $(a|c|abc)^*$