String Matching: Regular Expressions

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 10

Searching for patterns

- So far, we have done string matching for a fixed pattern `p` in a string `t`

Searching for patterns

- So far, we have done string matching for a fixed pattern `p` in a string `t`
- What is we want to look for a pattern?

Searching for patterns

- So far, we have done string matching for a fixed pattern `p` in a string `t`
- What is we want to look for a pattern?
- Unsure of spelling — look for `Srivatsan` or `Srivathsan`

Searching for patterns

- So far, we have done string matching for a fixed pattern `p` in a string `t`
- What is we want to look for a pattern?
- Unsure of spelling — look for `Srivatsan` or `Srivathsan`
- Check for a sequence — look for a word that has `sub` followed by `tion`
  - `substitution`, `subtraction`

Searching for patterns

- So far, we have done string matching for a fixed pattern `p` in a string `t`
- What is we want to look for a pattern?
- Unsure of spelling — look for `Srivatsan` or `Srivathsan`
- Check for a sequence — look for a word that has `sub` followed by `tion`
  - `substitution`, `subtraction`
- Repetition — one or more copy of `na`
  - `pennant`, `banana`

Describing patterns

- Finite alphabet $\Sigma = \{a, b, c, \ldots\}$

Describing patterns

- Finite alphabet $\Sigma = \{a, b, c, \ldots\}$
- A pattern matches a set of words over Σ

Describing patterns

- Finite alphabet $\Sigma = \{a, b, c, \ldots\}$
- A pattern matches a set of words over Σ
- Building patterns

Describing patterns

- Finite alphabet $\Sigma = \{a, b, c, \ldots\}$
- A pattern matches a set of words over Σ
- Building patterns
- Each $a \in \Sigma$ is a pattern — matches $\{a\}$

Describing patterns

- Finite alphabet $\Sigma = \{a, b, c, \ldots\}$
- A pattern matches a set of words over Σ
- Building patterns
- Each $a \in \Sigma$ is a pattern — matches $\{a\}$
- If p is a pattern matching S_p and q is a pattern matching S_q, then $p + q$ matches $S_p \cup S_q$

Describing patterns

- Finite alphabet $\Sigma = \{a, b, c, \ldots\}$
- A pattern matches a set of words over Σ
- Building patterns
- Each $a \in \Sigma$ is a pattern — matches $\{a\}$
- If p is a pattern matching S_p and q is a pattern matching S_q, then $p + q$ matches $S_p \cup S_q$
- If p is a pattern matching S_p and q is a pattern matching S_q, then pq matches $S_p \cdot S_q = \{uv \mid u \in S_p, v \in S_q\}$

Describing patterns

- Finite alphabet $\Sigma = \{a, b, c, \ldots\}$
- A pattern matches a set of words over Σ
- Building patterns
- Each $a \in \Sigma$ is a pattern — matches $\{a\}$
- If p is a pattern matching S_p and q is a pattern matching S_q, then $p + q$ matches $S_p \cup S_q$
- If p is a pattern matching S_p and q is a pattern matching S_q, then pq matches $S_p \cdot S_q = \{uv \mid u \in S_p, v \in S_q\}$
- If p is a pattern matching S_p, then p^+ matches any word w that can be decomposed as $w_1 w_2 \ldots w_k$ where each $w_j \in S_p$
  - p^+ is 1 or more repetitions of p, p^* is 0 or more repetitions

Examples

- Pattern $a + b$ matches $\{a, b\}$
  - a matches $\{a\}$, b matches $\{b\}$, take the union

Examples

- Pattern $a + b$ matches $\{a, b\}$
  - a matches $\{a\}$, b matches $\{b\}$, take the union
- Likewise, pattern $c + d$ matches $\{c, d\}$

Examples

- Pattern $a + b$ matches $\{a, b\}$
  - a matches $\{a\}$, b matches $\{b\}$, take the union
- Likewise, pattern $c + d$ matches $\{c, d\}$
- Pattern $(a + b)(c + d)$ matches $\{ac, bc, ad, bd\}$
  - $a + b$ matches $\{a, b\}$, $c + d$ matches $\{c, d\}$, construct all words with first part from $\{a, b\}$ followed by second part from $\{c, d\}$

Examples

- Pattern $a + b$ matches $\{a, b\}$
  - a matches $\{a\}$, b matches $\{b\}$, take the union
- Likewise, pattern $c + d$ matches $\{c, d\}$
- Pattern $(a + b)(c + d)$ matches $\{ac, bc, ad, bd\}$
  - $a + b$ matches $\{a, b\}$, $c + d$ matches $\{c, d\}$, construct all words with first part from $\{a, b\}$ followed by second part from $\{c, d\}$
- Pattern $[(a + b)(c + d)]^+$ matches $\{acac, acbc, acad, acbd, bcac, acacac, \ldots\}$
  - Match any word that can be decomposed into words from $\{ac, bc, ad, bd\}$

Shortcuts

- Want to match a followed by b
  - Any symbol from Σ can come in between
  - $\Sigma = \{a_1, a_2, \ldots, a_k\}$
  - $a_1 + a_2 + \cdots + a_k$ matches any symbol in Σ
  - Use Σ itself as an abbreviation for this special pattern
  - The pattern we want is $a\Sigma^+b$
  - If we allow no gap between a and b, $a\Sigma^*b$

Shortcuts

- Want to match a followed by b
  - Any symbol from Σ can come in between
  - $\Sigma = \{a_1, a_2, \ldots, a_k\}$
  - $a_1 + a_2 + \cdots + a_k$ matches any symbol in Σ
  - Use Σ itself as an abbreviation for this special pattern
  - The pattern we want is $a\Sigma^+b$
  - If we allow no gap between a and b, $a\Sigma^*b$
- Looking for *Srivatsan* or *Srivathsan*
  - Pattern is $(\Sigma^* Srivatsan \Sigma^*) + (\Sigma^* Srivathsan \Sigma^*)$
  - By convention, can drop initial and final Σ^* — $(Srivatsan + Srivathsan)$ matches anywhere in the text
  - More compactly $Srivat(h + \varepsilon)san$
  - ε matches the empty string, with no characters

Anchoring patterns

- Our convention is now that p matches anywhere in a string

Anchoring patterns

- Our convention is now that p matches anywhere in a string
- To match the start of the string, write $\hat{}p$

Anchoring patterns

- Our convention is now that p matches anywhere in a string
- To match the start of the string, write $\hat{}p$
- To match the start of the string, write $p\$$

Anchoring patterns

- Our convention is now that p matches anywhere in a string
- To match the start of the string, write $\hat{}p$
- To match the start of the string, write $p\$$
- $\hat{}ba$ and $na\$$ both match *banana*

Anchoring patterns

- Our convention is now that p matches anywhere in a string
- To match the start of the string, write $\hat{}p$
- To match the start of the string, write $p\$$
- $\hat{}ba$ and $na\$$ both match *banana*
- $\hat{}p\$$ will match if entire word matches p

Anchoring patterns

- Our convention is now that p matches anywhere in a string
- To match the start of the string, write $\hat{}p$
- To match the start of the string, write $p\$$
- $\hat{}ba$ and $na\$$ both match *banana*
- $\hat{}p\$$ will match if entire word matches p
- $\hat{}bana\$$ does not match *banana*, but $\hat{}ba(na)^{+}\$$ does

Regular expressions

- Recall that the Knuth-Morris-Pratt algorithm built a finite state automaton for the longest prefix matched by a pattern

Regular expressions

- Recall that the Knuth-Morris-Pratt algorithm built a finite state automaton for the longest prefix matched by a pattern
- The automaton we built had a linear structure, with a single path from start to finish

Regular expressions

- Recall that the Knuth-Morris-Pratt algorithm built a finite state automaton for the longest prefix matched by a pattern
- The automaton we built had a linear structure, with a single path from start to finish
- In general, automata can follow multiple paths, accept multiple words

Regular expressions

- Recall that the Knuth-Morris-Pratt algorithm built a finite state automaton for the longest prefix matched by a pattern
- The automaton we built had a linear structure, with a single path from start to finish
- In general, automata can follow multiple paths, accept multiple words
- The set of words that automata can accept are called regular sets

Regular expressions

- Recall that the Knuth-Morris-Pratt algorithm built a finite state automaton for the longest prefix matched by a pattern
- The automaton we built had a linear structure, with a single path from start to finish
- In general, automata can follow multiple paths, accept multiple words
- The set of words that automata can accept are called regular sets
- Each pattern p describes a set of words, those that it matches

Regular expressions

- Recall that the Knuth-Morris-Pratt algorithm built a finite state automaton for the longest prefix matched by a pattern
- The automaton we built had a linear structure, with a single path from start to finish
- In general, automata can follow multiple paths, accept multiple words
- The set of words that automata can accept are called regular sets
- Each pattern p describes a set of words, those that it matches
- The sets we can describe using patterns are exactly the same as those that can be accepted by automata

Regular expressions

- Recall that the Knuth-Morris-Pratt algorithm built a finite state automaton for the longest prefix matched by a pattern
- The automaton we built had a linear structure, with a single path from start to finish
- In general, automata can follow multiple paths, accept multiple words
- The set of words that automata can accept are called regular sets
- Each pattern p describes a set of words, those that it matches
- The sets we can describe using patterns are exactly the same as those that can be accepted by automata
- Our patterns are called regular expressions

Regular expressions

- For every automaton, we can construct a pattern p that matches exactly the words that the automaton accepts

Regular expressions

- For every automaton, we can construct a pattern p that matches exactly the words that the automaton accepts
- For every pattern p, we can construct an automaton that accepts all words that match p

Regular expressions

- For every automaton, we can construct a pattern p that matches exactly the words that the automaton accepts
- For every pattern p, we can construct an automaton that accepts all words that match p
- We can extend string matching to pattern matching by building an automaton for a pattern p and processing the text through this automaton, like Knuth-Morris-Pratt for fixed strings

Regular expressions

- For every automaton, we can construct a pattern p that matches exactly the words that the automaton accepts
- For every pattern p, we can construct an automaton that accepts all words that match p
- We can extend string matching to pattern matching by building an automaton for a pattern p and processing the text through this automaton, like Knuth-Morris-Pratt for fixed strings
- Python provides a library for matching regular expressions