Regular Expressions

Pattern matching

Substring Search : find a single string in text

Pattern Matching : find one of a specified set of strings in text

Regular Expression - notation to specify a set of strings

Operations

1) concatenation
2) or |
3) closure * (0 or more occurrences)
4) parenthesis ( )
5) wildcard .
6) character class [A-Z a-z] [a-z]
7) at least 1 occurrences +

8) exactly k $\{k\}$

* <u>Regular expressions</u> are surprisingly expressive

- substring search

  .* SPB. *

- social security numbers

  $[0-9]\underbrace{\{3\}}_{k \text{ digits}}-[0-9]\{2\}-[0-9]\{4\}$

- email

  $[a-z]+@([a-z]+\backslash.)+(edu|com)$

* RE important in the theory of computation

<u>RE / FA</u>

- <u>RE</u> (concise way to describe a string)
- <u>DFA</u> (Discrete Finite Automaton) - machine to recognize wheter a given string is in a given set

Kleene's theorem

- for any DFA, there exists a RE that describes the same set of strings
- for any RE, there exists a DFA that recognizes the same set of strings

Pattern matching implementation

Basic Plan

- build a DFA from RE
- simulate DFA with text as input

Bad news

- basic plan is infeasible; DFA may have exponential # of states

Solution – we use NFA instead

NFA = Non-deterministic finite automaton

Regular Expression matching NFA

- RE enclosed in parentheses

- one state per RE character (start = 0, accept = M)

epsilon

- Red ε-transition (change state, but don't scan text)
- Black match transition (change state and scan to next text char)
- accept if any sequence of transitions ends in accept state

after scanning all text characters

* systematically consider all possible transition sequences

NFA Simulation

Representation

- state names - integers from 0 to M
- match transitions - keep regular expression in array re[]

- ε transitions => store in digraph G

* maintain set of all possible states that NFA could be in after reading in the first i text characters

<u>How to perform reachability</u> ?

- Read next input character
  - find states reachable by match transitions
  - find states reachable by ε-transitions
- When no more input characters:
  - accept if any state reachable is an accept state
  - reject otherwise

<u>Digraph reachability</u>

- find all vertices reachable from a given source or set of vertices

<u>Solution</u> : run DFS from each source

without unmarking vertices

* runs in time proportional to $E+V$
* worst case it takes time proportional to MN

NFA Construction

- concatenation ⟶ match transitions
- parentheses ⟶ ε - transition edge from parentheses to next state
- closure ⟶ add 3 ε- transition edges for each * operator

or 2 ε- transition edges for each | operator

<u>challenges</u> - remember left parentheses to implement closure and or; remember | to implement or.

<u>Solution</u> $\boxed{\text{STACK}}$

(push

| push

) pop

Regular Expression Applications

- $\boxed{\text{grep}}$ => build an N/FA
- use grep to solve crossword puzzles
- RegExp built in programming languages
- Java string library [input.matches(re)]
- Java Util → matcher, → pattern
- evolution —> writing a compiler