EXPERIMENT 3

AIM: To implement Pattern Matching Algorithm in Jupyter NoteBook.

RESOURCES REQUIRED: Jupyter NoteBook, 4GB RAM and above, i5 Processor and above.

THEORY:

PATTERN MATCHING:

Pattern matching in computer science is the checking and locating of specific sequences of data of some pattern among raw data or a sequence of tokens. Unlike pattern recognition, the match has to be exact in the case of pattern matching. Pattern matching is one of the most fundamental and important paradigms in several programming languages. Many applications make use of pattern matching as a major part of their tasks.

Pattern matching, in its classical form, involves the use of one-dimensional string matching. Patterns are either tree structures or sequences. There are different classes of programming languages and machines which make use of pattern matching. In the case of machines, the major classifications include deterministic finite state automata, deterministic pushdown automata, nondeterministic pushdown automata and Turing machines. Regular programming languages make use of regular expressions for pattern matching. Tree patterns are also used in certain programming languages like Haskell as a tool to process data based on the structure. Compared to regular expressions, tree patterns lack simplicity and efficiency.

There are many applications for pattern matching in computer science. High-level language compilers make use of pattern matching in order to parse source files to determine if they are syntactically correct. In programming languages and applications, pattern matching is used in identifying the matching pattern or substituting the matching pattern with another token sequence.

Steps of Regular Expression Matching:

While there are several steps to using regular expressions in Python, each step is fairly simple.

1. Import the regex module with import re.
2. Create a Regex object with the re.compile() function. (Remember to use a raw string.)
3. Pass the string you want to search into the Regex object's search() method. This returns a Match object.
4. Call the Match object's group() method to return a string of the actual matched text.

Grouping with parentheses

Matching objects: Say you want to separate the area code from the rest of the phone number. Adding parentheses will create groups in the regex: (\d\d\d)-(\d\d\d-\d\d\d\d). Then you can use the group() match object method to grab the matching text from just one group.

Retrieve all the groups at once: If you would like to retrieve all the groups at once, use the groups(), method—note the plural form for the name.

Using mo.groups : mo.groups() will return a tuple of multiple values, you can use the multiple-assignment trick to assign each value to a separate variable, as in the following areaCode, mainNumber = mo.groups() line.

Match a parenthesis : Parentheses have a special meaning in regular expressions, but what do you do if you need to match a parenthesis in your text. For instance, maybe the phone numbers you are trying to match have the area code set in parentheses. In this case, you need to escape the (and) characters with a backslash.

Matching Multiple Groups with the Pipe

The | character is called a pipe. You can use it anywhere you want to match one of many expressions. For example, the regular expression r'Batman|Tina Fey' will match either 'Batman' or 'Tina Fey'. When both Batman and Tina Fey occur in the searched string, the first occurrence of matching text will be returned as the Match object.

Matching Specific Repetitions with Curly Brackets

If you have a group that you want to repeat a specific number of times, follow the group in your regex with a number in curly brackets. For example, the regex (Ha){3} will match the string 'HaHaHa', but it will not match 'HaHa', since the latter has only two repeats of the (Ha) group.

Instead of one number, you can specify a range by writing a minimum, a comma, and a maximum in between the curly brackets. For example, the regex (Ha){3, 5} will match 'HaHaHa', 'HaHaHaHa', and 'HaHaHaHaHa'.

You can also leave out the first or second number in the curly brackets to leave the minimum or maximum unbounded. For example, (Ha){3, } will match three or more instances of the (Ha) group, while (Ha){, 5} will match zero to five instances. Curly brackets can help make your regular expressions shorter.

Optional Matching with the Question Mark

Sometimes there is a pattern that you want to match only optionally. That is, the regex should find a match whether or not that bit of text is there. The **?** character flags the group that precedes it as an optional part of the pattern.

Matching Zero or More with the Star

The * (called the star or asterisk) means "match zero or more"—the group that precedes the star can occur any number of times in the text. It can be completely absent or repeated over and over again.

Matching One or More with the Plus

While * means "match zero or more, " the + (or plus) means **"match one or more."** Unlike the star, which does not require its group to appear in the matched string, the group preceding a plus must appear at least once.

CONCLUSION: Hence we have successfully implemented Pattern matching by Regex Method.