Landero, Christian Jay V

BSCS 2A

**B. EXPLORE & EXPLAIN**

1. Pattern Searching

* Pattern searching is the process of finding occurrences or matches of a specified pattern (a sequence of characters) within a given text or string. It's a fundamental problem in computer science and is used in various applications, including text processing, data mining, information retrieval, and more. The primary goal of pattern searching algorithms is to efficiently locate all occurrences of a pattern within a larger text or document.

1. Pattern Matching

* Pattern matching is a process in computer science and programming where a system or program identifies and locates specific patterns within data. These patterns can be sequences of characters, structured data, or more complex structures, and the matching process involves searching for instances of these patterns within a given dataset or input.

1. Alphabetical Sorting

* Alphabetical sorting, also known as lexicographic sorting or alphabetical order, is a method of arranging items in a list, such as words or strings, based on their alphabetical or lexicographic order. In this order, items are arranged according to the order of characters in the alphabet, typically from A to Z (or from a to z) for English letters.

1. Anagram

* An anagram is a word or phrase formed by rearranging the letters of another word or phrase, typically using all the original letters exactly once. In other words, anagrams are created by shuffling or permuting the characters of a word or phrase to produce a new word or phrase that has a different spelling but uses the same set of characters.

1. Palindrome

* A palindrome is a word, phrase, number, or other sequence of characters that reads the same forward and backward (ignoring spaces, punctuation, and capitalization, in most cases). In simpler terms, a palindrome is a sequence that remains unchanged when its characters are reversed.

1. The Knuth-Morris-Pratt (KMP)

* The Knuth-Morris-Pratt (KMP) algorithm is a string pattern matching algorithm that efficiently searches for occurrences of a pattern within a text. It is known for its linear time complexity and is particularly useful when dealing with large texts. The key insight behind the KMP algorithm is the use of a "partial match table" (also known as the "failure function" or "prefix function") to avoid unnecessary character comparisons during the search.

1. The Rabin-Karp algorithm

* The Rabin-Karp algorithm is a string pattern matching algorithm that uses hashing to search for occurrences of a pattern within a text. It's particularly efficient for finding multiple occurrences of the pattern in the text. The algorithm works by creating a hash value for the pattern and then sliding a window of the same length over the text while maintaining a rolling hash value. When the hash value of the current window matches the hash value of the pattern, a character-by-character comparison is performed to confirm the match.

1. Longest Common Subsequence Algorithm

* The Longest Common Subsequence (LCS) algorithm is a dynamic programming technique used to find the longest subsequence that two or more strings have in common. A subsequence is a sequence of characters that appear in the same order as in the original strings but may not be consecutive. In other words, the LCS algorithm finds the longest sequence of characters that are shared among the given strings, with the constraint that the characters must appear in the same order.

1. Manacher's algorithm

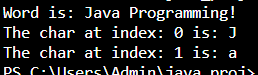
* Manacher's algorithm is a linear-time algorithm used to find the longest palindromic substring within a given string. The algorithm is based on the concept of palindrome symmetry and employs a combination of techniques, including dynamic programming and clever data structures, to avoid unnecessary character comparisons.

1. Finite Automata

* A finite automaton, often referred to simply as a finite automaton or FA, is a mathematical model used in computer science and formal language theory to describe and recognize patterns and languages. Finite automata are used in various applications, including lexical analysis in compilers, regular expression matching, and natural language processing.

II. Programming Exercises:

1. Write a Java program to get the character at the given index within the String.



Code:

class Find\_The\_Char {

    public void find\_char() {

        String word = "Java Programming!";

        int idx1 = 0;

        int idx2 = 1;

        System.out.println("Word is: " + word);

        System.out.println("The char at index: " + idx1 + " is: " + word.charAt(idx1));

        System.out.println("The char at index: " + idx2 + " is: " + word.charAt(idx2));

    }

}

public class Find\_Index {

    public static void main(String[] args) {

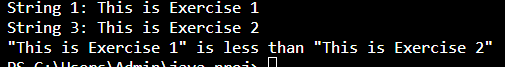
        Find\_The\_Char test = new Find\_The\_Char();

        test.find\_char();

    }

}

1. Write a java program to compare two strings lexicographically.



Code:

class Lexicographically\_Comparator {

    private String string1;

    private String string2;

    public Lexicographically\_Comparator(String string1, String string2) {

        this.string1 = string1;

        this.string2 = string2;

    }

    public void compareStrings() {

        System.out.println("String 1: " + this.string1);

        System.out.println("String 3: " + this.string2);

        int result = string1.compareTo(string2);

        if (result < 0) {

            System.out.println("\"" + string1 + "\" is less than \"" + string2 + "\"");

        } else if (result > 0) {

            System.out.println("\"" + string1 + "\" is greater than \"" + string2 + "\"");

        } else {

            System.out.println("\"" + string1 + "\" is equal to \"" + string2 + "\"");

        }

    }

}

public class Lexicographically {

    public static void main(String[] args) {

        String string1 = "This is Exercise 1";

        String string2 = "This is Exercise 2";

        Lexicographically\_Comparator comparator = new Lexicographically\_Comparator(string1, string2);

        comparator.compareStrings();

    }

}

1. Write a Java program to compare a given string to the specified character sequence.



public class Compare\_String {

    public static void main(String[] args) {

        String str1 = "example.com";

        String str2 = "Example.com";

        String str3 = "example.com";

        boolean result1 = str1.contentEquals(str2);

        boolean result2 = str1.contentEquals(str3);

        System.out.println("Comparing " + str1 + " and " + str2 + ": " + result1);

        System.out.println("Comparing " + str1 + " and " + str3 + ": " + result2);

    }

}

1. Write a Java program to create a new String object with the contents of a character array. char[] arr num = new char[] { '1', '2', '3', '4' };



public class Create\_Char\_String {

    public static void main(String[] args) {

        char[] num = { '1', '2', '3', '4' };

        String numString = new String(num);

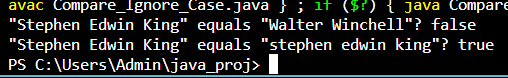
        String sentence = "The book contains " + numString + " pages.";

        System.out.println(sentence);

    }

}

1. Write a Java program to compare a given string to another string, ignoring case considerations.



public class Compare\_Ignore\_Case {

    public static void main(String[] args) {

        String str1 = "Stephen Edwin King";

        String str2 = "Walter Winchell";

        String str3 = "stephen edwin king";

        boolean result1 = str1.equalsIgnoreCase(str2);

        System.out.println("\"" + str1 + "\" equals \"" + str2 + "\"? " + result1);

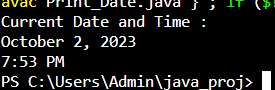
        boolean result2 = str1.equalsIgnoreCase(str3);

        System.out.println("\"" + str1 + "\" equals \"" + str3 + "\"? " + result2);

    }

}

1. Write a java program to print current date and time in the specified format.



import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class Print\_Date {

    public static void main(String[] args) {

        LocalDateTime currentDateTime = LocalDateTime.now();

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MMMM d, yyyy\nh:mm a");

        String formattedDateTime = currentDateTime.format(formatter);

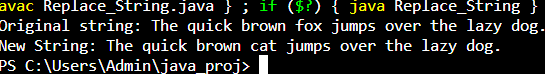
        System.out.println("Current Date and Time :");

        System.out.println(formattedDateTime);

    }

}

1. Write a Java program to replace each substring of a given string that matches the given regular expression with the given replacement using string operations.



public class Replace\_String {

    private String originalString;

    public Replace\_String(String originalString) {

        this.originalString = originalString;

    }

    public String replaceSubstring(String searchString, String replacementString) {

        String newString = originalString.replace(searchString, replacementString);

        return newString;

    }

    public static void main(String[] args) {

        String originalString = "The quick brown fox jumps over the lazy dog.";

        String searchString = "fox";

        String replacementString = "cat";

        Replace\_String replace\_String = new Replace\_String(originalString);

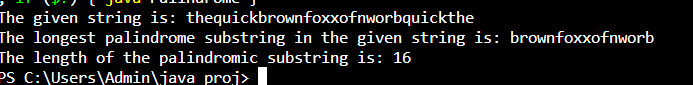
        String newString = replace\_String.replaceSubstring(searchString, replacementString);

        System.out.println("Original string: " + originalString);

        System.out.println("New String: " + newString);

    }

}

1. Write a Java program to find longest Palindromic Substring within a string. 

public class Palindrome {

    private String inputString;

    public Palindrome(String inputString) {

        this.inputString = inputString;

    }

    public String find\_longest\_palindrome() {

        int maxLength = 0;

        String longestPalindrome = "";

        for (int i = 0; i < inputString.length(); i++) {

            for (int j = i + 1; j <= inputString.length(); j++) {

                String substring = inputString.substring(i, j);

                if (is\_palindrome(substring) && substring.length() > maxLength) {

                    maxLength = substring.length();

                    longestPalindrome = substring;

                }

            }

        }

        return longestPalindrome;

    }

    private boolean is\_palindrome(String s) {

        int left = 0;

        int right = s.length() - 1;

        while (left < right) {

            if (s.charAt(left) != s.charAt(right)) {

                return false;

            }

            left++;

            right--;

        }

        return true;

    }

    public static void main(String[] args) {

        String inputString = "thequickbrownfoxxofnworbquickthe";

        Palindrome finder = new Palindrome(inputString);

        String longestPalindrome = finder.find\_longest\_palindrome();

        System.out.println("The given string is: " + inputString);

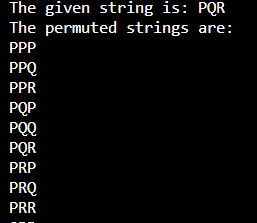
        System.out.println("The longest palindrome substring in the given string is: " + longestPalindrome);

        System.out.println("The length of the palindromic substring is: " + longestPalindrome.length());

    }

}

1. Write a Java program to print all permutations of a given string with repetition.



import java.util.ArrayList;

import java.util.List;

public class Permutation {

    private String inputString;

    public Permutation(String inputString) {

        this.inputString = inputString;

    }

    public List<String> generate\_permutation() {

        List<String> permutations = new ArrayList<>();

        generate\_it(inputString, "", permutations);

        return permutations;

    }

    private void generate\_it(String remaining, String current, List<String> permutations) {

        if (current.length() == inputString.length()) {

            permutations.add(current);

        } else {

            for (int i = 0; i < remaining.length(); i++) {

                char currentChar = remaining.charAt(i);

                generate\_it(remaining, current + currentChar, permutations);

            }

        }

    }

    public static void main(String[] args) {

        String inputString = "PQR";

        Permutation permutationsGenerator = new Permutation(inputString);

        List<String> permutations = permutationsGenerator.generate\_permutation();

        System.out.println("The given string is: " + inputString);

        System.out.println("The permuted strings are:");

        for (String permutation : permutations) {

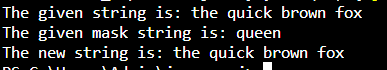
            System.out.println(permutation);

        }

    }

}

1. Write a Java program to remove duplicate characters from a given string presents in another given string.



public class Remove\_Duplicate {

    public static String remove\_duplicates(String originalString, String maskString) {

        String result = "";

        for (char c : originalString.toCharArray()) {

            if (!maskString.contains(String.valueOf(c)) || result.indexOf(c) == -1) {

                result += c;

            }

        }

        return result;

    }

    public static void main(String[] args) {

        String originalString = "the quick brown fox";

        String maskString = "queen";

        String resultString = remove\_duplicates(originalString, maskString);

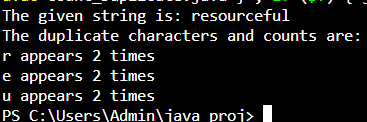
        System.out.println("The given string is: " + originalString);

        System.out.println("The given mask string is: " + maskString);

        System.out.println("The new string is: " + resultString);

    }

}

1. Write a Java program to count and print all the duplicates in the input string.

import java.util.HashMap;

import java.util.Map;

public class Count\_Duplicate {

    private String inputString;

    public Count\_Duplicate(String inputString) {

        this.inputString = inputString;

    }

    public void count\_duplicate() {

        Map<Character, Integer> charCountMap = new HashMap<>();

        for (char c : inputString.toCharArray()) {

            if (Character.isWhitespace(c)) {

                continue; // Skip whitespace characters

            }

            // Update character count in the map

            charCountMap.put(c, charCountMap.getOrDefault(c, 0) + 1);

        }

        System.out.println("The given string is: " + inputString);

        System.out.println("The duplicate characters and counts are:");

        for (Map.Entry<Character, Integer> entry : charCountMap.entrySet()) {

            if (entry.getValue() > 1) {

                System.out.println(entry.getKey() + " appears " + entry.getValue() + " times");

            }

        }

    }

    public static void main(String[] args) {

        String inputString = "resourceful";

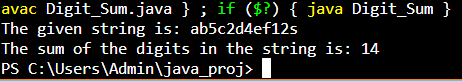
        Count\_Duplicate count\_Duplicate = new Count\_Duplicate(inputString);

        count\_Duplicate.count\_duplicate();

    }

}

1. Write a Java program to return the sum of the digits present in the given string. If there is no digits the sum return is 0.



public class Digit\_Sum {

    private String inputString;

    public Digit\_Sum(String inputString) {

        this.inputString = inputString;

    }

    public int calculate\_digit\_sum() {

        int sum = 0;

        for (char c : inputString.toCharArray()) {

            if (Character.isDigit(c)) {

                int digit = Character.getNumericValue(c);

                sum += digit;

            }

        }

        return sum;

    }

    public static void main(String[] args) {

        String inputString = "ab5c2d4ef12s";

        Digit\_Sum calculator = new Digit\_Sum(inputString);

        int sumOfDigits = calculator.calculate\_digit\_sum();

        System.out.println("The given string is: " + inputString);

        System.out.println("The sum of the digits in the string is: " + sumOfDigits);

    }

}

**C. ELABORATE & EVALUATION**