Arrays & Strings

// **Is Unique** - Implement an algorithm to determine if a string has all unique characters.
// What if you cannot use additional data structures?

```
public class Runner {
	public static void main(String[] args) {
		String[] words = {"apple", "paddle", "kite"};
		for (int i = 0; i < words.size; i++) {
			System.out.println(QuestionA.isUniqueChars(words[i]));
			System.out.println(QuestionB.isUniqueChars(words[i]));
		}
	}
}

public class QuestionA {

	private static final int R = 256;

	public static boolean isUniqueChars(String input) {
		if (input.length() > R) {
			return false;  // input size is bigger than alphabet
		}

		boolean[] marked = new boolean[R];
		for (int i = 0; i < input.length(); i++) {
			char c = input.charAt(i);
			if (marked[c]) {
				return false;
			}
			marked[c] = true;
		}
		return true;
	}
}

public class QuestionB {
	public static boolean isUniqueChars(String input) {
		int checker = 0;
		for (int i = 0; i < input.length(); i++) {
			int val = input.charAt(i) - 'a';
```

```
                        if ((checker & (1 << val)) > 0) {
                                return false;
                        }
                        checker |= (1 << val);
                }
                return true;
        }
}

// Check Permutation - Given two strings, write a method to decide if one is a permutation
// of the other.

// Solution 1 - Sorting complexity O(n * log n) → n = length of string
// Equals takes O(n) time (we have to check every character)

public class QuestionA {

        public static String sort(String input) {
                char[] s = input.toCharArray();
                Arrays.sort(s);
                return new String(s);
        }

        public static boolean isPermutation(String a, String b) {
                if (a.length() != b.length()) {
                        return false;
                }
                return sort(a).equals(sort(b));
        }
}

// Solution 2 - Check that both strings have the same character count

public class QuestionB {

        private static final int R = 256;

        public static boolean isPermutation(String a, String b) {
                if (a.length() != b.length()) {
                        return false;
                }
                int letters[] = new int[R];
                for (int i = 0; i < a.length(); i++) {
```

```
                letters[a.charAt(i)]++;
            }
            for (int i = 0; i < b.length(); i++) {
                letters[b.charAt(i)]--;
                if (letters[b.charAt(i)] < 0) {
                    return false;
                }
            }
            return true;
        }
}

// URLify - Write a method to replace all spaces in a string with '%20'

public class Question {

        public static void urlEncode(char[] str, int trueLength) {
            int spaceCount = 0;

            for (int i = 0; i < trueLength; i++) {
                if (str[i] == ' ') {
                    spaceCount++;
                }
            }

            int index = trueLength + 2 * spaceCount;
            if (trueLength < str.length) str[trueLength] = '\0';

            for (int i = trueLength - 1; i >= 0; i--) {
                if (str[i] == ' ') {
                    str[index - 1] = '0';
                    str[index - 2] = '2';
                    str[index - 3] = '%';
                    index -= 3;
                } else {
                    str[index - 1] = str[i];
                    index--;
                }
            }
        }

        public static String urlEncode(String str) {
            String input = str.trim();
```

```
                StringBuilder encodedString = new StringBuilder();
                for (int i = 0; i < input.length(); i++) {
                        char c = input.charAt(i);
                        if (c == ' ') {
                                encodedString.append("%20");
                        } else {
                                encodedString.append(c);
                        }
                }
                return encodedString.toString();
        }
}
```

// **Palindrom Permutation** - Given a string, write a function to check if it is a permutation of a palindrome.

```
public class Common {

        public static int getCharNumber(Character c) {
                int a = Character.getNumericValue('a');
                int z = Character.getNumericValue('z');
                int val = Character.getNumericValue(c);
                if (a <= val && val <= z) {
                        return val - a;
                }
                return -1;
        }

        public static int[] buildCharFrequencyTable(String input) {
                int[] table = new int['z' - 'a' + 1];
                for (char c : input.toCharArray()) {
                        int x = getCharNumber(c);
                        if (x != -1) {
                                table[x]++;
                        }
                }
                return table;
        }
}
```

// **Solution 1** - using a hash table
// The algorithm takes O(n) time → n is the length of the string

```
public class QuestionA {
	public static boolean isPermutationOfPalindrom(String input) {
		int[] table = Common.buildCharFrequencyTable(input);
		return checkMaxOneOdd(table);
	}

	static boolean checkMaxOneOdd(int[] table) {
		boolean foundOdd = false;
		for (int count : table) {
			if (count % 2 == 1) {
				if (foundOdd) {
					return false;
				}
				foundOdd = true;
			}
		}
		return true;
	}
}

// Solution 2 - check the number of odd counts as we go along

public class QuestionB {
	public static boolean isPermutationOfPalindrome(String input) {
		int countOdd = 0;
		int[] table = new int['z' - 'a' + 1];
		for (char c : input.toCharArray()) {
			int x = Common.getCharNumber(c);
			if (x != -1) {
				table[x]++;
				if (table[x] % 2 == 1) {
					countOdd++;
				} else {
					countOdd--;
				}
			}
		}
		return countOdd <= 1;
	}
}

// Solution 3 - using a bit vector
```

```
public class QuestionC {

	static boolean isPermutationOfPalindrome(String input) {
		int bitVector = createBitVector(input);
		return bitVector == 0 || checkExactlyOneBitSet(bitVector);
	}

	static int createBitVector(String input) {
		// for each letter with value i, toggle the ith bit
		int bitVector = 0;
		for (char c : input.toCharArray()) {
			int x = Common.getCharNumber(c);
			bitVector = toggle(bitVector, x);
		}
		return bitVector;
	}

	static int toggle(int bitVector, int index) {
		if (index < 0) {
			return bitVector;
		}
		int mask = 1 << index;
		if ((bitVector & mask) == 0) {
			bitVector |= mask;
		} else {
			bitVector &= ~mask;
		}
		return bitVector;
	}

	static boolean checkExactlyOneBitSet(int bitVector) {
		return (bitVector & (bitVector - 1)) == 0;
	}
}
```

// **One Away** - Three type of edits on a string → insert character, remove character, replace character. Write a function to check if two strings are one edit away (or zero edits)

// **Solution A** → merge remove with insert and pick based on strings length

```
public class QuestionA {

	public static boolean isOneEditAway(String a, String b) {
		if (a.length() == b.length()) {
			return isOneReplace(a, b);
		} else if (a.length() + 1 == b.length()) {
			return isOneEdit(a, b);
		} else if (a.length() == b.length() + 1) {
			return isOneEdit(b, a);
		}

		return false;
	}

	public static boolean isOneReplace(String a, String b) {
		// Strings are the same length, once character is different
		boolean foundDifference = false;
		for (int i = 0; i < a.length(); i++) {
			if (a.charAt(i) != b.charAt(i)) {
				if (foundDifference) {
					return false;
				}
				foundDifference = true;
			}
		}
		return true;
	}

	public static boolean isOneEdit(String a, String b) {
		int indexA = 0;
		int indexB = 0;

		while (indexA < a.length() && indexB < b.length()) {
			if (a.charAt(indexA) != b.charAt(indexB)) {
				if (indexA != indexB) {
					// there should only be one difference
					return false;
				}
				++indexB;
			} else {
				++indexA;
				++indexB;
			}
```

```
                }
                return true;
        }
}

// Solution B → Do everything in one pass

public class QuestionB {

        public static boolean isOneEditAway(String a, String b) {
                // Check if the difference between strings > 1
                if (Math.abs(a.length() - b.length()) > 1) {
                        return false;
                }

                String s1 = a.length() < b.length() ? a : b;
                String s2 = a.length() < b.length() ? b : a;

                int firstIndex = 0;
                int secondIndex = 0;
                boolean foundDifference = false;

                while (firstIndex < s1.length() && secondIndex < s2.length()) {
                        if (s1.charAt(firstIndex) != s2.charAt(secondIndex)) {
                                if (foundDifference) {
                                        return false;
                                }
                                foundDifference = true;
                                if (s1.length() == s2.length()) {
                                        ++firstIndex;
                                }
                        } else {
                                ++firstIndex;  // if characters are matching
                        }
                        ++secondIndex;  // always move pointer to larger string
                }

                return true;
        }
}

// String Compression → write a method to perform basic string compression using the counts of repeated characters
```

```
public class Question {

	public static String compress(String input) {
		if (input.length() <= 1) {
			return input;
		}
		StringBuilder result = new StringBuilder();
		char c = input.charAt(1);
		int counter = 1;

		for (int i = 1; i < input.length(); i++) {
			if (c == input.charAt(i)) {
				++counter;
			} else {
				result.append(c);
				result.append(Integer.toString(counter));
				c = input.charAt(i);
				counter = 1;
			}
		}

		result.append(c);
		result.append(Integer.toString(counter));

		return result.length() >= input.length() ? input : result.toString();
	}

	public static int countCompress(String input) {
		if (input.length() <= 1) {
			return input.length();
		}

		char c = input.charAt(1);
		int counter = 1;
		int finalLength = 0;

		for (int i = 1; i < input.length(); i++) {
			if (c == input.charAt(i)) {
				++counter;
			} else {
				finalLength += 1 + String.valueOf(counter).length();
				c = input.charAt(i);
				counter = 1;
```

```
                }
            }
            finalLength += 1 + String.valueOf(counter).length();
            return finalLength;
        }
}
```

// **Optimizations** → string concatenation in java has O(n^2) complexity since it requires to create a new string and copy over the existing characters. StringBuilder is efficient, but it would void resizing the backing char array if we could instantiate it with the final length of the compress string

// **Rotate Matrix** → given an image represented by a N x N matrix, where each pixel in the image is 4 bytes, write a method to rotate the image by 90 degrees

```
public class QuestionA {

    public static boolean rotate(int[][] matrix) {
        if (matrix.length == 0 || (matrix.length != matrix[0].length)) {
            return false;
        }

        int n = matrix.length;
        for (int layer = 0; layer < n / 2; layer++) {
            int first = layer;
            int last = n - 1 - layer;

            for (int i = first; i < last; i++) {
                int offset = i - first;
                int top = matrix[first][i]; // save top

                // left -> top
                matrix[first][i] = matrix[last-offset][first];

                // bottom -> left
                matrix[last-offset][first] = matrix[last][last - offset];

                // right -> bottom
                matrix[last][last - offset] = matrix[i][last];

                // top -> right
                matrix[i][last] = top; // right <- saved top
            }
```

```
                }
                return true;
        }
}
```

// **Zero Matrix** → Write an algorithm such that if an element in MxN matrix is 0 its entire row and column are set to 0

```
public class QuestionA {
        public static void nullifyRow(int[][] matrix, int row) {
                for (int j = 0; j < matrix[0].length; j++) {
                        matrix[row][j] = 0;
                }
        }

        public static void nullifyColumn(int[][] matrix, int col) {
                for (int i = 0; i < matrix.length; i++) {
                        matrix[i][col] = 0;
                }
        }

        public static void setZeros(int[][] matrix) {
                boolean[] row = new boolean[matrix.length];
                boolean[] column = new boolean[matrix[0].length];

                // Store the row and column index with value 0
                for (int i = 0; i < matrix.length; i++) {
                        for (int j = 0; j < matrix[0].length;j++) {
                                if (matrix[i][j] == 0) {
                                        row[i] = true;
                                        column[j] = true;
                                }
                        }
                }

                // Nullify rows
                for (int i = 0; i < row.length; i++) {
                        if (row[i]) {
                                nullifyRow(matrix, i);
                        }
                }

                // Nullify columns
```

```
                for (int j = 0; j < column.length; j++) {
                        if (column[j]) {
                                nullifyColumn(matrix, j);
                        }
                }
        }
}
```

// **Optimizations** → do it in place by using the first row and first column. If these need to be nullified you can do it at the and

// **String Roatation** - given two strings s1 and s2, check if s2 is a rotation of s1 with only one call to substring

// xy = s1, yx = s2, s1s1 = xyxy → yx is substring of xyxy

```
public class Question {

        public static boolean isRotation(String s1, String s2) {
                int len = s1.length();

                if (len == s2.length() && len > 0) {
                        String s1s1 = s1 + s1;
                        return s1s1.indexOf(s2) >= 0;
                }
                return false;
        }
}
```