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;
	}
}
```