```
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

using System.Collections.Generic;

public class Program {
  // O(ns + bs) time | O(ns) space
  public static List<bool> MultistringSearch(string bigstring, string[] smallstrings) {
    Trie trie = new Trie();
    foreach (string smallstring in smallstrings) {
      trie.insert(smallstring);
    }
    HashSet<string> containedstrings = new HashSet<string>();
    for (int i = 0; i < bigstring.Length; i++) {
      findSmallstringsIn(bigstring, i, trie, containedstrings);
    }
    List<bool> solution = new List<bool>();
    foreach (string str in smallstrings) {
      solution.Add(containedstrings.Contains(str));
    }
    return solution;
  }

  public static void findSmallstringsIn(string str, int startIdx, Trie trie,
    HashSet<string> containedstrings) {
    TrieNode currentNode = trie.root;
    for (int i = startIdx; i < str.Length; i++) {
      char currentChar = str[i];
      if (!currentNode.children.ContainsKey(currentChar)) {
        break;
      }
      currentNode = currentNode.children[currentChar];
      if (currentNode.children.ContainsKey(trie.endSymbol)) {
        containedstrings.Add(currentNode.word);
      }
    }
  }

  public class TrieNode {
    public Dictionary<char, TrieNode> children = new Dictionary<char, TrieNode>();
    public string word;
  }

  public class Trie {
    public TrieNode root = new TrieNode();
    public char endSymbol = '*';

    public void insert(string str) {
      TrieNode node = root;
      for (int i = 0; i < str.Length; i++) {
        char letter = str[i];
        if (!node.children.ContainsKey(letter)) {
          TrieNode newNode = new TrieNode();
          node.children.Add(letter, newNode);
        }
        node = node.children[letter];
      }
      node.children[endSymbol] = null;
      node.word = str;
    }
  }
}

```