AlgoExpert | Quad Layout | C# | 12px | Sublime | Monokai | 00:00:00

Prompt | Scratchpad | Our Solution(s) | Video Explanation | Run Code

Solution 1

```
using System.Collections.Generic;

public class Program {
  // O(n^2 + m) time | O(n + m) space
  public static string[] PatternMatcher(string pattern, string str) {
    if (pattern.Length > str.Length) {
      return new string[] {};
    }
    char[] newPattern = getNewPattern(pattern);
    bool didSwitch = newPattern[0] != pattern[0];
    Dictionary<char, int> counts = new Dictionary<char, int>();
    counts['x'] = 0;
    counts['y'] = 0;
    int firstYPos = getCountsAndFirstYPos(newPattern, counts);
    if (counts['y'] != 0) {
      for (int lenOfX = 1; lenOfX < str.Length; lenOfX++) {
        double lenOfY =
          ((double)str.Length - (double)lenOfX *
          (double)counts['x']) /
          (double)counts['y'];
        if (lenOfY <= 0 || lenOfY % 1 != 0) {
          continue;
        }
        int yIdx = firstYPos * lenOfX;
        string x = str.Substring(0, lenOfX);
        string y = str.Substring(yIdx, (int)lenOfY);
        string potentialMatch = buildPotentialMatch(newPattern, x, y);
        if (str.Equals(potentialMatch)) {
          return didSwitch ? new string[] {y, x} : new string[] {x,
                                                                 y};
        }
      }
    } else {
      double lenOfX = str.Length / counts['x'];
      if (lenOfX % 1 == 0) {
        string x = str.Substring(0, (int)lenOfX);
        string potentialMatch = buildPotentialMatch(newPattern, x, "");
        if (str.Equals(potentialMatch)) {
          return didSwitch ? new string[] {"", x} : new string[] {x,
                                                                  ""};
        }
      }
    }
    return new string[] {};
  }

  public static char[] getNewPattern(string pattern) {
    char[] patternLetters = pattern.ToCharArray();
    if (pattern[0] == 'x') {
      return patternLetters;
    }
    for (int i = 0; i < patternLetters.Length; i++) {
      if (patternLetters[i] == 'x') {
        patternLetters[i] = 'y';
      } else {
        patternLetters[i] = 'x';
      }
    }
    return patternLetters;
  }

  public static int getCountsAndFirstYPos(char[] pattern, Dictionary<char, int> counts) {
    int firstYPos = -1;
    for (int i = 0; i < pattern.Length; i++) {
      char c = pattern[i];
      counts[c] = counts[c] + 1;
      if (c == 'y' && firstYPos == -1) {
        firstYPos = i;
      }
    }
    return firstYPos;
  }

  public static string buildPotentialMatch(char[] pattern, string x, string y) {
    StringBuilder potentialMatch = new StringBuilder();
    foreach (char c in pattern) {
      if (c == 'x') {
        potentialMatch.Append(x);
      } else {
        potentialMatch.Append(y);
      }
    }
    return potentialMatch.ToString();
  }
}
```