Solution 1

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

package main

import "strings"

type counts struct {
  x int
  y int
}

// O(n^2 + m) time | O(n + m) space
func PatternMatcher(pattern string, str string) []string {
  if len(pattern) > len(str) {
    return []string{}
  }
  pattern, switched := getNewPattern(pattern)
  count, firstY := getCountsAndFirstYPos(pattern)
  if count.y != 0 {
    for lenx := 1; lenx < len(str); lenx++ {
      totalLeny := len(str) - lenx*count.x
      if len(str) <= lenx*count.x || totalLeny%count.y != 0 {
        continue
      }
      leny := totalLeny / count.y
      yindex := firstY * lenx
      x, y := str[:lenx], str[yindex:yindex+leny]
      potentialMatch := doReplace(pattern, x, y, count)
      if str == potentialMatch {
        if !switched {
          return []string{x, y}
        }
        return []string{y, x}
      }
    }
  } else {
    if len(str)%count.x == 0 {
      lenx := len(str) / count.x
      x := str[:lenx]
      potentialMatch := strings.Repeat(x, len(pattern))
      if str == potentialMatch {
        if !switched {
          return []string{x, ""}
        }
        return []string{"", x}
      }
    }
  }

  return []string{}
}

func doReplace(pattern, x, y string, count counts) string {
  result := make([]byte, 0)
  for _, r := range pattern {
    if r == 'x' {
      result = append(result, []byte(x)...)
    } else {
      result = append(result, []byte(y)...)
    }
  }
  return string(result)
}

func getNewPattern(pattern string) (string, bool) {
  if pattern[0] == 'x' {
    return pattern, false
  }
  runes := make([]rune, len(pattern))
  for i := range pattern {
    if pattern[i] == 'x' {
      runes[i] = 'y'
    } else {
      runes[i] = 'x'
    }
  }
  return string(runes), true
}

func getCountsAndFirstYPos(pattern string) (counts, int) {
  firstY := strings.Index(pattern, "y")
  count := counts{}
  for _, r := range pattern {
    if r == 'x' {
      count.x += 1
    } else {
```