Solution 1

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

package main

import "strings"

type intervals []*interval
type interval struct {
  left  int
  right int
}

// O(n*m) | O(n) space
func UnderscorifySubstring(str string, substring string) string {
  locations := getLocations(str, substring)
  locations = locations.collapse()
  return underscorify(str, locations)
}

func getLocations(str, substring string) intervals {
  result := intervals{}
  for start := 0; start < len(str); {
    nextIndex := strings.Index(str[start:], substring)
    if nextIndex == -1 {
      break
    }
    nextIndex += start
    result = append(result, &interval{nextIndex, nextIndex + len(substring)})
    start = nextIndex + 1
  }
  return result
}

func (array intervals) collapse() intervals {
  // If the array is empty, nothing to do
  if len(array) == 0 {
    return array
  }

  result := intervals{array[0]}
  previous := array[0]
  for i := 1; i < len(array); i++ {
    current := array[i]
    if current.left <= previous.right {
      // Collapse the two intervals
      previous.right = current.right
    } else {
      result = append(result, current)
      previous = current
    }
  }
  return result
}

func underscorify(str string, locations intervals) string {
  if len(locations) == 0 {
    return str
  }

  // We know the resulting string will have an additional 2*len(intervals)
  // characters
  result := make([]rune, len(str)+2*len(locations))
  resultIndex := 0
  locationIndex := 0
  for i, r := range str {
    location := locations[locationIndex]
    if i == location.left {
      result[resultIndex] = '_'
      resultIndex += 1
    } else if i == location.right {
      result[resultIndex] = '_'
      resultIndex += 1
      if locationIndex+1 < len(locations) {
        locationIndex += 1
      }
    }
    result[resultIndex] = r
    resultIndex += 1
  }

  if locations[locationIndex].right == len(str) {
    result[len(result)-1] = '_'
  }
  return string(result)
}
```