

МІНІСТЕРСТВО ОСВІТИ І НАУКИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО»  
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ  
КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І СПЕЦІАЛІЗОВАНИХ  
КОМП'ЮТЕРНИХ СИСТЕМ

**ЛАБОРАТОРНА РОБОТА №1**

з дисципліни «Архітектура комп'ютерів»  
Тема: «Реалізація пошуку всіх послідовностей в рядку»

Виконали  
студенти 4 курсу  
гр.КВ-43  
*Шапошніков Олександр*  
*Комарянський Леонід*  
*Маяков Дмитро*

Перевірив:

---

Київ  
2017

## Завдання

Реалізація пошуку всіх послідовностей в рядку, які відповідають заданому шаблону, що складається з набору довільних символів та символу “\*”. Символ “\*” відповідає довільному набору символів та може зустрічатися в шаблоні декілька разів.

## Лістинг:

### Пакет Wildcard:

```
package wildcard

import "strings"

//Match does string matching with `*` wildcard support.
//Returns a slice with N positions of all matched wildcardStr substrings in
sourceStr.
//Trailing wildcards `str***` are ignored.
//With heading wildcards `***str` position 1 will be returned N times.
func Match(sourceStr, wildcardStr string) (foundValues []int) {
    hasHeadingWildcard := false
    foundValues = nil

    //remove trailing *
    wildcardTrimmed := strings.TrimRight(wildcardStr, "*")

    // if nothing left pattern always matches all the string
    if wildcardTrimmed == "" {
        foundValues = []int{0}
        return
    }

    if len(wildcardTrimmed) > len(sourceStr) {
        return
    }

    if strings.HasPrefix(wildcardTrimmed, "*") {
        wildcardTrimmed = strings.TrimLeft(wildcardTrimmed, "*")
        hasHeadingWildcard = true
    }

    startIndex, sourceIndex, wildcardIndex, wasWildcard, isSearchReset := 0,
0, 0, false, false
    wcardLength, sourceLength := len(wildcardTrimmed), len(sourceStr)
    // naïve algorithm
    // Example pattern: ex*p*e (example)
    for startIndex < sourceLength {

        // skip wildcards until the last one
        // guaranteed to have at least one non-wildcard at the end
        if wildcardTrimmed[wildcardIndex] == '*' {
            for wildcardTrimmed[wildcardIndex] == '*' {
                wildcardIndex++
            }
            wasWildcard = true
        }

        // update source index if it mathces the non-wildcard symbol
        // advance wildcard counter
```

```

    if sourceStr[sourceIndex] == wildcardTrimmed[wildcardIndex] {
        sourceIndex++
        wildcardIndex++
        wasWildcard = false
    } else if wasWildcard { // wildcard always matches
        sourceIndex++
    } else { // search complete, start anew
        isSearchReset = true
    }

    // if value found add to result
    if wildcardIndex >= wcardLength && !isSearchReset {
        if hasHeadingWildcard {
            foundValues = append(foundValues, 0)
        } else {
            foundValues = append(foundValues, startIndex)
        }
        isSearchReset = true
    }

    // reset (advance) search
    if isSearchReset || sourceIndex == sourceLength {
        wildcardIndex = 0
        startIndex++
        sourceIndex = startIndex
        isSearchReset = false
    }
}

return
}

```

## Тести:

```
package wildcard

import (
    "reflect"
    "testing"
)

type testpair struct {
    source   string
    pattern  string
    expected []int
}

type testCases []testpair

var deadCases = testCases{
    {"", "", []int{0}},
    {"no test", "", []int{0}},
    {"", "no origin", nil},
    {"small", "bigger than original", nil},
}

var singleLetter = testCases{
    {"test", "e", []int{1}},
    {"test", "t", []int{0, 3}},
    {"test", "x", nil},
    {"abracadabra", "a", []int{0, 3, 5, 7, 10}},
    {"abracadabra", "x", nil},
    {"She called a storm upon this town.", "s", []int{13, 27}},
}

var multipleLetters = testCases{
    {"test", "*", []int{0}},
    {"test", "es", []int{1}},
    {"test", "te", []int{0}},
    {"test", "test", []int{0}},
    {"abracadabra", "acad", []int{3}},
    {"She called a storm upon this town.", " t", []int{23, 28}},
    {"She called a storm upon this town.", "called", []int{4}},
}

var wildcardSingle = testCases{
    {"test", "*es", []int{0}},
    {"test", "te*", []int{0}},
    {"test", "t*t", []int{0}},
    {"abracadabra", "ac**ad", []int{3}},
    {"abracadabra", "a*a", []int{0, 3, 5, 7}},
    {"She called a storm upon this town.", "st*m", []int{13}},
    {"She called a storm upon this town.", "***st*m", []int{0}},
    {"test", "*ak", nil},
    {"test", "dd*", nil},
    {"test", "t*dt", nil},
}

var wildcardMultiple = testCases{
    {"abracadabra", "ac*o*ad", nil},
    {"abracadabra", "a*b*a", []int{0, 3, 5, 7}},
    {"She called a storm upon this town.", "a*st*m", []int{5, 11}},
}
```

```

func auxTestLoop(t *testing.T, tests *testCases) {
    for _, x := range *tests {
        result := Match(x.source, x.pattern)
        if !reflect.DeepEqual(result, x.expected) {
            t.Errorf(
                "\ninput:      %#v\n"+
                "pattern:    %#v\n"+
                "expected:   %#v\n"+
                "result:     %#v\n", x.source, x.pattern,
x.expected, result,
            )
        }
    }
}

func TestSingleLetter(t *testing.T) {
    auxTestLoop(t, &singleLetter)
}

func TestMultipleLetters(t *testing.T) {
    auxTestLoop(t, &multipleLetters)
}

func TestDeadCases(t *testing.T) {
    auxTestLoop(t, &deadCases)
}

func TestWildcardSingle(t *testing.T) {
    auxTestLoop(t, &wildcardSingle)
}

func TestWildcardMultiple(t *testing.T) {
    auxTestLoop(t, &wildcardMultiple)
}

```

## Бенчмарк

```
package wildcard

import (
    "strings"
    "testing"
)

var sourceString = `Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Aenean commodo
ligula eget dolor. Aenean massa. Cum sociis
natoque penatibus et magnis dis parturient montes,
nascetur ridiculus mus. Donec quam felis,
ultricies nec, pellentesque eu, pretium quis, sem.
Nulla consequat massa quis enim. Donec pede justo,
fringilla vel, aliquet nec, vulputate eget, arcu.
In enim justo, rhoncus ut, imperdiet a, venenatis vitae,
justo. Nullam dictum felis eu pede mollis pretium.
Integer tincidunt. Cras dapibus. Vivamus elementum
semper nisi. Aenean vulputate eleifend tellus. Aenean
leo ligula, porttitor eu, consequat vitae, eleifend ac,
enim. Aliquam lorem ante, dapibus in, viverra quis,
feugiat a, tellus. Phasellus viverra nulla ut metus varius
laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies
nisi vel augue. Curabitur ullamcorper ultricies nisi.
Nam eget dui. Etiam rhoncus. Maecenas tempus,
tellus eget condimentum rhoncus, sem quam semper libero,
sit amet adipiscing sem neque sed ipsum. Nam quam nunc,
blandit vel, luctus pulvinar, hendrerit id, lorem.
Maecenas nec odio et ante tincidunt tempus. Donec
vitae sapien ut libero venenatis faucibus. Nullam quis
ante. Etiam sit amet orci eget eros faucibus tincidunt.
Duis leo. Sed fringilla mauris sit amet nibh. Donec sodales
sagittis magna. Sed consequat, leo eget bibendum sodales,
augue velit cursus nunc.`

var simplePattern = `ma*sa`
var frequentPattern = `a*a`
var absentPattern = `abrakad*abra`
var frequentLetter = `a`

func benchmarkMatch(b *testing.B, pattern string, srcMultiplier int) {
    strA := []string{sourceString}
    for i := 0; i < srcMultiplier; i++ {
        strA = append(strA, sourceString)
    }
    str := strings.Join(strA, "")
    for n := 0; n < b.N; n++ {
        // run the Match function b.N times
        Match(sourceString, str)
    }
}

func BenchmarkSingle(b *testing.B) { benchmarkMatch(b, simplePattern, 0) }
func BenchmarkFrequent(b *testing.B) { benchmarkMatch(b, frequentPattern, 0) }
func BenchmarkAbsent(b *testing.B) { benchmarkMatch(b, absentPattern, 0) }
func BenchmarkFrequentLetter(b *testing.B) { benchmarkMatch(b, frequentLetter, 0) }
```

```

func BenchmarkSingle100(b *testing.B)      { benchmarkMatch(b, simplePattern,
100) }
func BenchmarkFrequent100(b *testing.B)    { benchmarkMatch(b,
frequentPattern, 100) }
func BenchmarkAbsent100(b *testing.B)      { benchmarkMatch(b, absentPattern,
100) }
func BenchmarkFrequentLetter100(b *testing.B) { benchmarkMatch(b,
frequentLetter, 100) }

```

## Результати go test -bench=.

```

goos: linux
goarch: amd64
pkg: github.com/sanchaez/kpi-lab-go/lab1/wildcard
BenchmarkSingle-8          300000      5451 ns/op
BenchmarkFrequent-8        300000      5453 ns/op
BenchmarkAbsent-8          300000      5434 ns/op
BenchmarkFrequentLetter-8  300000      5476 ns/op
BenchmarkSingle100-8       2000000      65.8 ns/op
BenchmarkFrequent100-8     2000000      65.5 ns/op
BenchmarkAbsent100-8       2000000      65.1 ns/op
BenchmarkFrequentLetter100-8 2000000      66.9 ns/op
PASS
ok      github.com/sanchaez/kpi-lab-go/lab1/wildcard 12.342s

```