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 №2

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

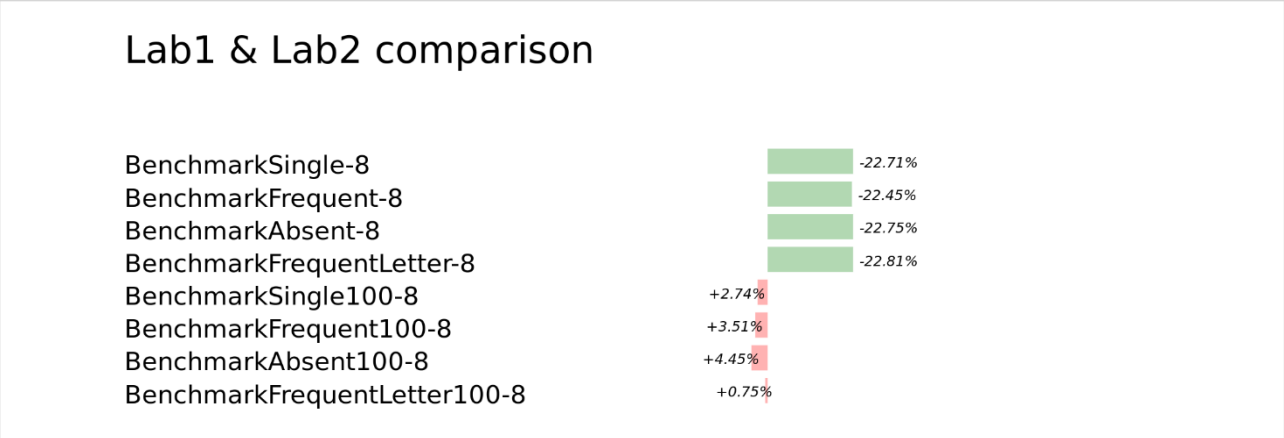
## Завдання

Пришвидшення роботи алгоритму більше ніж на 10% порівняно з лабораторною роботою №1.

## Результати go test –bench=.

```
goos: linux
goarch: amd64
pkg: github.com/sanchaez/kpi-lab-go/lab2/wildcard
BenchmarkSingle-8                    300000          4213 ns/op
BenchmarkFrequent-8                  300000          4229 ns/op
BenchmarkAbsent-8                    300000          4198 ns/op
BenchmarkFrequentLetter-8            300000          4227 ns/op
BenchmarkSingle100-8               20000000            67.6 ns/op
BenchmarkFrequent100-8             20000000            67.8 ns/op
BenchmarkAbsent100-8               20000000            68.0 ns/op
BenchmarkFrequentLetter100-8       20000000            67.4 ns/op
PASS
ok    github.com/sanchaez/kpi-lab-go/lab2/wildcard   10.949s
```

## Результати benchcmp lab1.bench lab2.bench




```
benchmark                       old ns/op     new ns/op     delta
BenchmarkSingle-8               5451          4213          -22.71%
BenchmarkFrequent-8             5453          4229          -22.45%
BenchmarkAbsent-8               5434          4198          -22.75%
BenchmarkFrequentLetter-8       5476          4227          -22.81%
BenchmarkSingle100-8            65.8          67.6          +2.74%
BenchmarkFrequent100-8          65.5          67.8          +3.51%
BenchmarkAbsent100-8            65.1          68.0          +4.45%
BenchmarkFrequentLetter100-8    66.9          67.4          +0.75%
```

# 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 := 0, 0, 0, false
	wcardLength, sourceLength := len(wildcardTrimmed), len(sourceStr)

	// main search loop
	for startIndex < sourceLength {
		wildcardIndex = 0
		sourceIndex = startIndex

		// loop to find an entry starting from startIndex
		for wildcardIndex < wcardLength &&
			sourceIndex < sourceLength &&
			wildcardTrimmed[wildcardIndex] == sourceStr[sourceIndex] {

			wildcardIndex++
			sourceIndex++

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

```
                  // wildcardIndex points to non-wildcard character in
wildcardTrimmed
                  // loop the string until it is found in sourceStr
                  if wasWildcard && wildcardIndex < wcardLength {
                        for sourceIndex < sourceLength &&
                              wildcardTrimmed[wildcardIndex] !=
sourceStr[sourceIndex] {
                              sourceIndex++
                        }
                  }
            }

            // if value found add to results
            if wildcardIndex >= wcardLength {
                  if hasHeadingWildcard {
                        foundValues = append(foundValues, 0)
                  } else {
                        foundValues = append(foundValues, startIndex)
                  }
            }

            // advance start index
            startIndex++
      }

      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,
consectetuer 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) }
```