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 №3

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

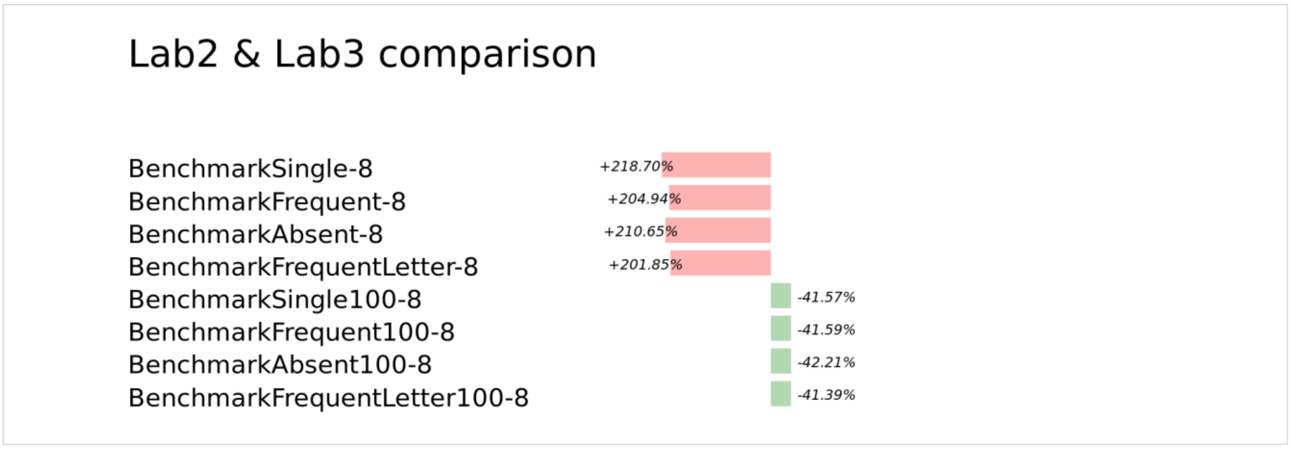
Завдання

Пришвидшення роботи алгоритму більше ніж на 10% порівняно з лабораторною роботою №2 за допомогою паралелізму.

Результати go test –bench=.

```
goos: linux
goarch: amd64
pkg: github.com/sanchaez/kpi-lab-go/lab3/wildcard
BenchmarkSingle-8                      100000         13427 ns/op
BenchmarkFrequent-8                    200000         12896 ns/op
BenchmarkAbsent-8                      100000         13041 ns/op
BenchmarkFrequentLetter-8              100000         12759 ns/op
BenchmarkSingle100-8                 30000000            39.5 ns/op
BenchmarkFrequent100-8               30000000            39.6 ns/op
BenchmarkAbsent100-8                 30000000            39.3 ns/op
BenchmarkFrequentLetter100-8         30000000            39.5 ns/op
PASS
ok      github.com/sanchaez/kpi-lab-go/lab3/wildcard    11.962s
```

Результати benchcmp lab2.bench lab3.bench



```
benchmark                        old ns/op     new ns/op     delta
BenchmarkSingle-8                4213          13427         +218.70%
BenchmarkFrequent-8              4229          12896         +204.94%
BenchmarkAbsent-8                4198          13041         +210.65%
BenchmarkFrequentLetter-8        4227          12759         +201.85%
BenchmarkSingle100-8             67.6          39.5          -41.57%
BenchmarkFrequent100-8           67.8          39.6          -41.59%
BenchmarkAbsent100-8             68.0          39.3          -42.21%
BenchmarkFrequentLetter100-8     67.4          39.5          -41.39%
```

Лістинг:

Пакет Wildcard:

```
package wildcard

import (
      "sort"
)

func hasMatched(sourceStr, wildcardStr string) bool {
      var sourceIndex, wildcardIndex int
      wcardLength, sourceLength := len(wildcardStr), len(sourceStr)

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

            wildcardIndex++
            sourceIndex++

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

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

      // if value found add to results
      if wildcardIndex >= wcardLength {
            return true
      }

      return false
}

func readyWildcardString(str *string) bool {
      length := len(*str)
      begin, end := 0, length
      for i := 0; i < length && (*str)[i] == '*'; i++ {
            begin++
      }
      for i := length - 1; i >= 0 && (*str)[i] == '*'; i-- {
            end--
      }
```

```
	if begin <= end {
		*str = (*str)[begin:end]
	} else {
		*str = ""
	}

	if begin > 0 {
		return true
	}

	return false
}

type matchRoutine func(string, string, int, chan int)

func matchGoroutine(sourceStr, wildcardStr string,
	startIndex int,
	channel chan int) {
	if hasMatched(sourceStr[startIndex:], wildcardStr) {
		channel <- startIndex
	}
}

func matchGoroutineWcard(sourceStr, wildcardStr string,
	startIndex int,
	channel chan int) {
	if hasMatched(sourceStr[startIndex:], wildcardStr) {
		channel <- 0
	}
}

//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) []int {
	hasHeadingWildcard := readyWildcardString(&wildcardStr)

	if wildcardStr == "" {
		return []int{0}
	}

	if len(wildcardStr) > len(sourceStr) {
		return nil
	}

	var fn matchRoutine
	if hasHeadingWildcard {
		fn = matchGoroutineWcard
	} else {
		fn = matchGoroutine
	}

	// make channels
	sourceLength := len(sourceStr)
	const parallelWorkers int = 4
	in := make(chan int, parallelWorkers)
	out := make(chan int, sourceLength)
	done := make(chan bool, parallelWorkers)

	// spawn producer
	go func() {
		for i := 0; i < sourceLength; i++ {
```

```
                if sourceStr[i] == wildcardStr[0] {
                    in <- i
                }
            }

            close(in)
        }()

        // spawn consumers
        for p := 0; p < parallelWorkers; p++ {
            go func() {
                // search the substrings in v
                for v := range in {
                    fn(sourceStr, wildcardStr, v, out)
                }
                // denote completion of this goroutine
                done <- true
            }()
        }

        // spawn waiter
        go func() {
            // wait for workers
            for p := 0; p < parallelWorkers; p++ {
                <-done
            }

            // finish workers
            close(out)
        }()

        // capture values as they come
        var foundValues []int
        for value := range out {
            foundValues = append(foundValues, value)
        }

        sort.Ints(foundValues)
        return foundValues
    }
```

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) }
```