Midterm, 4.12.2017 Meno,priezvisko:

Pravidlá:

- **Ļas 100 minút**
- **môžete používať akékoľvek ale len Vami prinesené materialy,**
- **nie je možné zdieľať akékoľvek pomôcky a materialy,**
- **kolektívne riešenia sa netolerujú,**
- **každý príklad riešte na papieri obsahujúcom jeho zadanie,**
- **midterm obsahuje 5 príkladov spolu za 23 bodov.**

1. [4 body ï Fibonacciho slová]

Fibonacciho Ļ ísla vám predstavovať nemusíte. V tejto úlohe sú namiesto Ļ ísel reťazce a definícia je následovná:

```
def fibStr(n : Int, one : String, two : String): String = {
  n match {
    case 1 => one
    case 2 => two
    case _ => fibStr(n-2, one, two) + fibStr(n-1, one, two) // + je zreťazenie reťazcov
  }
}
```

a) Definujte funkciu `def fibLinerane(n : Int, a : String, b : String): String = {...}`, ktorá vygeneruje n-té fibonacciho slovo, ale <u>akceptované budú len efektívne riešenia zložitosti</u> O(n). **[1 bod]**. Príklad:

```
fibLinerane(1, "ab", "cde") == "ab"
fibLinerane(2, "ab", "cde") == "cde"
fibLinerane(3, "ab", "cde") == "abcde"
fibLinerane(4, "ab", "cde") == "cdeabcde"
```

b) Definujte funkciu `def jeFib(kandidat : String, a : String, b : String) : Boolean = {...}`, ktorá zistí, Ļ i `kandidat` môže byť prvkom postupnosti zaĻ ínajúcej reťazcami `a, b.` **[1 bod]**. Príklad:

```
jeFib("abcdeabcdecdeabcdeabcdecdeabcde", "ab", "cde") == true
jeFib("abcdeabcdecdeabcdea", "ab", "cde") == false
```

c) Definujte funkciu `def prveDva(kandidat : String) : (String, String) = {...}`, ktorá nájde ľubovoľnú dvojicu prvých Ļ lenov postupnosti tak, že táto postupnosť obsahuje reťazec `kandidat` a ten obsahuje aspoň 3 reťazce postupnosti. Ak taká postupnosť neexistuje, vráti `("","")`. **[2 body]**. Príklad:

```
prveDva("prvedruheprvedruhedruheprvedruhe") == ("prve", "druhe")
prveDva("prvedruheprvedruhedruhe") == ("","")
prveDva("prvedruheprvedruhe")  == ("p", "rvedruhe")
```

-------- v prípade potreby riešte túto úlohu na zadnú stranu **TOHOTO listu**----------

2. [3 body - priemer]
Tento riadok ilustruje 3-in-1 funkcie `map/filter/reduce` (analogické a vám známe aj z Pythonu).
`DoubleStream.of(1,2,3,4,5).map(x->x*x).filter(y->y<10).reduce(0, (a,b)->a+b);`
`map` prerobí 1,2,3,4,5 na 1,4,9,16,25, `filter` nechá len 1,4,9 a `reduce` ich spoľíta takze 14.0.
Tento riadok vám ilustruje, ako sa `List<Double>` prerobí na `DoubleStream` a aj vytlačí:

```
List.of(1.0,2.0,3.0,4.0,5.0)
       .stream().mapToDouble(Double::doubleValue).forEach(System.out::print);
```

Naprogramujte pomocou uvedených funkcií a **bez použitia cyklov a rekurzie**:
a) aritmetický priemer, teda `Double aritmeticky(List<Double> lst) { ... }` **[1 bod]**
b) geometricky priemer, teda `Double geometricky(List<Double> lst) { ... }` **[1 bod]**
c) harmonický priemer, teda `Double harmonicky(List<Double> lst) { ... }` **[1 bod]**
Príklady (vzorce v poradí, zdroj wiki):

```
aritmeticky(List.of(1.0,2.0,3.0,4.0,5.0)) == 3.0
geometricky(List.of(1.0,2.0,3.0,4.0,5.0)) == 2.605171084697352
harmonicky(List.of(1.0,2.0,3.0,4.0,5.0)) == 2.18978102189781
```

$$A = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{a_1 + a_2 + \cdots + a_n}{n}$$

$$\bar{x} = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = (x_1 x_2 \cdots x_n)^{1/n}$$

$$\bar{x} = n \cdot \left(\sum_{i=1}^{n} \frac{1}{x_i}\right)^{-1}$$

-------- v prípade potreby rieğte túto úlohu na zadnú stranu **TOHOTO listu**----------

3. [4 body ï súļ in]
Súļ in dvoch zoznamov celých ļ ísel dostanete tak, ģe vynásobíte kaģdé z prvého s kaģdým z druhého, a vynecháte duplikáty. Skoro ako kartézsky súļ in. Na poradí prvkov výsledného zoznamu nezáleģí. B**ez pouģitia cyklov** naprogramujte:
a. funkciu `List<Integer> sucin(List<Integer> a, List<Integer> b) { ... }` **[2 body]**

Viete si predstaviŠ ako by vyzerala definicia pre 3, ... vstupné zoznamy ? a * b * c = a * (b * c).
V druhej ļ asti úlohy definujte súļ in zoznamu zoznamov ļ ísel, opäŠ**bez pouģitia cyklov:**
b. funkciu `List<Integer> sucinVela(List<List<Integer>> lst) { ... }` **[2 body]**
Príklady:
sucin(List.of(1,2,3,4,5), List.of(3,4,5,6,7)) == [3, 4, 5, 6, 7, 8, 10, 12, 14, 9, 15, 18, 21, 16, 20, 24, 28, 25, 30, 35]
sucin(List.of(1,2,2,2,2), List.of(3,3,4)) == [3, 4, 6, 8]
sucinVela(List.of(List.of(1,2,3,4,5))) == [1, 2, 3, 4, 5]
sucinVela(List.of(List.of(1,2,3,4,5), List.of(1,2,3,4,5))) == [1, 2, 3, 4, 5, 6, 8, 10, 9, 12, 15, 16, 20, 25]
sucinVela(List.of(List.of(1,2,3,4,5), List.of(1,2,3,4,5), List.of(1,2,3,4,5))) ==
[1, 2, 3, 4, 5, 6, 8, 10, 9, 12, 15, 16, 20, 25, 18, 24, 30, 32, 40, 50, 27, 36, 45, 48, 60, 75, 64, 80, 100, 125]
sucinVela(List.of(List.of(1,2,3), List.of(1,2), List.of(1,3))) == [1, 3, 2, 6, 4, 12, 9, 18]
Pre jednoduchosŠmôģete predpokladaŠ ģe vģetky vstupné zoznamy sú neprázdne.

-------- v prípade potreby rieģte túto úlohu na zadnú stranu **TOHOTO listu**----------

4. [5 bodov ï strom]

Toto je neparametrická definícia binárneho stromu z domácej úlohy aj s funkciou, ktorá nájde maximum v strome s hodnotami Int.

```
case class Tree(
    value: Int,                        // hodnota
    left:  Option[Tree] = None,        // ľavý syn
    right: Option[Tree] = None) {      // pravý syn

  def max(): Int = {
      val a = left match {
        case None => value
        case Some(xx) => xx.max()
      }
      val b = right match {
        case None => value
        case Some(xx) => xx.max()
      }
      val maxab = if (a < b) b else a
      if (maxab < value) value else maxab
  }
```

a) Definujte nerekurzívnu verziu funkcie max, ktorá poľíta to isté **[2 body]**, teda `def max(): Int = {... }`

b) Definujte funkciu maxDiff, ktorá poľíta rozdiel maximálneho a minimálneho prvku v strome. Pozor ale, strom môģete prejsŠlen raz, takģe rieģenie s Ňalġou copy-paste funkciou na min() v tvare max()-min() sa nehodnotí **[3 body]**, teda `def maxDiff(): Int = {... }`

Príklad:

```
object Midterm {
  def main(args: Array[String]) {
    val t =
      Tree(5,
          Some(Tree(2,
              Some(Tree(1, None, None)),
              Some(Tree(3, None, None)))),
          Some(Tree(8,
              Some(Tree(7, None, None)),
              Some(Tree(9, None, None))))))
    t.max() == 9
    t.maxDiff() == 8
```

-------- v prípade potreby rieģte túto úlohu na zadnú stranu **TOHOTO listu**----------

5. [7 bodov ï Go]

Toto je kód z prednáġky ilustrujúci 100 ļ ínskych ġepkárov/agentov, ktorí si posielajú správu typu int, kaġdý Ņalġí k nej pripoļ íta 1. KeĻ dostane správu posledný z nich, vypíġe ju na konzolu.

```
var number = 100
func main() {
    prev := make(chan int)
    first := prev
    go func() { first <- 0 }()
    for i := 0; i < number; i++ {
        next := make(chan int)
        go func(from, to chan int) {
            for {
                to <- 1 + <-from
            }
        }(prev, next)
        prev = next
    }
    fmt.Println(<-prev)
}
```

Do ich správania chceme pridaŠnasledujúce aspekty:

- **[1 bod]** kaġdý ġepkár/agent má svoj vek **v**, ļ o je náhodné ļ íslo **0<=v<90**, a kým sa mu podarí preposlaŠsprávu Ņalej, trvá mu to presne **v** milisekúnd, *hint: rand.Intn(n)*.
- **[1 bod]** novú správu vġdy s nasledujúcim indexom (t.j. 0, 1, 2, 3, 4 é) posielame prvému ġepkárovi/agentovi pravidelne kaġdých 10 sekúnd. Kaġdý ġepkár obdrġanú správu pred jej Ņalġím preposlaním pozmení tak, ġe k jej obsahu pripoļ íta 1000. Preto je vġdy moġné zo správy zistiŠjej pôvodný index (pôvodný obsah) aj poļ et krokov - preposlaní medzi agentami.
- **[1 bod]** keĻ posledný ġepkár/agent s indexom 99 nedostane ġiadnu správu uġ celých 10 sekúnd, protestuje na konzolu (`YOU ARE TOO LATE`).
- **[1 bod]** celá simulácia konļ í po 60 sekundách (`GAME IS OVER`)
- **[3 body]** ġepkár/agent **i neposiela** správu svojmu priamemu následovníkovi **i+1**, ale **náhodne si vyberie** ġepkára/agenta spomedzi vġetkých, a tomu poġle správu. Ten opäŠvyberá náhodného príjemcu. DĜka trasy správy potom nemusí byŠ100 krokov, aj správy nemusia prísŠv poradí, v akom boli odoslané. KeĻ k poslednému ġepkárovi 99 správa dorazí, vypisujte jej index aj poļ et krokov, ktoré ubehla.

Môġete rieġiŠniektoré podúlohy len doplnením kódu. Musí vġak byŠjasné, ļ o platí, ļ o ste doplnili, ļ i vyhodili z pôvodného kódu. V zadaní tieġ jasne oznaļ te, ktoré podúlohy ste podĜa vás vyrieġili. V prípade, ġe rieġite **väļ ġinu podúloh, resp. poslednú**, je rozumnejġie napísaŠvlastný kód (lebo úpravy do pôvodného sú pomerne masívne). Pôvodným kódom sa len inġpirujte... Tu je príklad simulácie, ktorý snáĻ zodpovie vaġe iniciálne otázky.

```
START MESSAGE 0              08:23:57 ← nová správa
STOP MESSAGE 0, STEPS 106    08:24:03
START MESSAGE 1              08:24:07 ← nová správa
YOU ARE TOO LATE !           08:24:13   ←posledný už 10 sekúnd nič nedostal
START MESSAGE 2              08:24:17 ← nová správa
STOP MESSAGE 1, STEPS 186    08:24:18
START MESSAGE 3              08:24:27 ← nová správa
YOU ARE TOO LATE !           08:24:28   ←posledný už 10 sekúnd nič nedostal
STOP MESSAGE 3, STEPS 27     08:24:29
STOP MESSAGE 2, STEPS 219    08:24:29
START MESSAGE 4              08:24:37 ← nová správa
YOU ARE TOO LATE !           08:24:39   ←posledný už 10 sekúnd nič nedostal
STOP MESSAGE 4, STEPS 47     08:24:40
START MESSAGE 5              08:24:47 ← nová správa
STOP MESSAGE 5, STEPS 1      08:24:47
START MESSAGE 6              08:24:57 ← nová správa
GAME IS OVER !               08:24:57       ← simulácia skončila po 60s.
```

-------- v prípade potreby rieġte túto úlohu na zadnú stranu **TOHTO listu** ----------

-------- v prípade potreby rieġte túto úlohu na zadnú stranu **TOHOTO listu** ----------