Universitatea Alexandru Ioan Cuza
Facultatea de Informatică
PMP, Test 08.02.2021

Observaţii:

1. Explicaţiile vor fi scrise sub formă de comentarii.
2. Dacă un răspuns necesită mai multe rulări, includeţi rezultatele rulărilor în comentarii.
3. Faceţi “commit” şi “push” la fiecare 20 minute.

Exerciţiul 1 (15)

A. Să se descrie în Figaro un model probabilist ce modelează calitatea unei săptămâni din februarie. Se ştie că probabilitatea ca să ningă într-o zi este 0.6. Dacă numărul de zile în care ninge este mai mare decât cinci, atunci săptămâna este cu “prea multă ninsoare”, dacă ninge cel mult două zile atunci săptămâna este cu “prea puţină ninsoare”, iar în rest este “normală”. Se va utiliza distribuţia binomială. (5)
B. Să se interogheze modelul pentru a şti probabilitatea ca săptămâna să fie normală. Explicaţi (interpretaţi) rezultatul obţinut. (3 + 2)
C. Generaţi un număr de valori pentru calitatea săptămânii. Numărul de valori să fie suficient de mare ca să acopere toate valorile posibile. Explicaţi de ce unele valori apar mai des. (3+2)

Exerciţiul 2 (15)

A. Să se descrie în Figaro un model probabilist ce modelează temperatura, ştiind că aceasta urmează o distribuţie normală unde:
    a. temperatura medie urmează şi ea o distribuţie normală cu media 7 şi variaţia 5, şi
    b. variaţia poate fi 20 cu probabilitatea 0.5 sau 30 cu aceeaşi probabilitate. (5)
B. Să se interogheze modelul pentru a afla probabilitatea ca temperatura să fie cuprinsă între 20 şi 50. Interpretaţi rezultatul. (3+2)
C. Să se observe că media pentru temperatura medie este 9, apoi să se repete interogarea B. Explicaţi diferenţa dintre cele doua interogări. (3+2)

Exerciţiul 3 (40)

Se consideră un model Markov în care tranziţiile dintre stările ascunse sunt date de următorul tabe (stările ascunse sunt A, B, C, D)l:

	A	B	C	D
A	0.721	0.202	0.067	0.1
B	0	0.581	0.407	0.012
C	0	0	0.75	0.25
D	0	0	0	1.0

Valorile observate sunt:

A	B	C	D
buna	nu prea buna	bolnav	decedat

Universitatea Alexandru Ioan Cuza
Facultatea de Informatică
PMP, Test 08.02.2021

A. Să se scrie o clasă abstractă *State* ce descrie starea sistemului (5)
B. Să se scrie o clasă *InitialState* ce deriveaza din *State* şi creează o stare iniţială, cu starea ascunsă dată ca parametru. (5)
C. Să se scrie o clasă *NextState* ce deriveaza din *State* şi creează o stare următoare, având starea curentă dată ca parametru. (10)
D. Să se creeze o listă cu zece stări ale sistemului, ştiind că starea ascunsă din starea iniţială este A. (5)
E. Să se interogheze modelul pentru a afla starea ascunsă cea mai probabilă în fiecare din cele zece stări. Interpretaţi rezultatul.(3+2)
F. Să se interogheze modelul pentru a afla probabilitatea valorii observate în ultima stare, dacă în starea a şasea s-a observat că era “nu prea buna”. Interpretaţi rezultatul. (3+2)
G. Ştiind în plus că în starea a şaptea valoarea observată a devenit “bolnav”, să se interogheze modelul din nou pentru a afla starea ascunsă cea mai probabilă în fiecare stare. Interpretaţi rezultatul. (3+2)

Prof. dr. Dorel Lucanu

Universitatea Alexandru Ioan Cuza
Facultatea de Informatică
PMP, Test 08.02.2021

```
package test09feb21

 import com.cra.figaro.algorithm.factored.VariableElimination
import com.cra.figaro.language.{Apply}
import com.cra.figaro.library.atomic.discrete.Binomial
import com.cra.figaro.library.compound._

object Ex1 {
   def main(args: Array[String]) {
     val snowyDaysInWeek = Binomial(7, 0.6)
     def getQuality(i: Int): String =
       if (i > 5) "tooSnowy";
       else if (i > 2) "normal";
       else "toLessSnowy"
     val weekQuality = Apply(snowyDaysInWeek, getQuality)
     // step 1
     println(VariableElimination.probability(weekQuality, "normal"))
     // step 2
     weekQuality.generate()
     println("A generated value: " + weekQuality.value)
     // run it several times
   }
}
```

```
package test09feb21

import com.cra.figaro.algorithm.sampling.Importance
import com.cra.figaro.language.{Flip, Select, Dist}
import com.cra.figaro.library.atomic.continuous.Normal
import com.cra.figaro.library.compound._

object Ex2 {
   def main(args: Array[String]) {
     val tempMean = Normal(25, 8)
     val tempVariance = Select(0.5 -> 20.0, 0.5 -> 30.0)
     val temperature = Normal(tempMean, tempVariance)
     println("Probab. of temperature = " +
        Importance.probability(temperature, (d: Double) => 20 <= d && d <= 50))
     tempMean.observe(9)
     println("Probab. of temperature when mean is 9 = " +
        Importance.probability(temperature, (d: Double) => 20 <= d && d <= 50))
   }
}
```

```
package test09feb21

import com.cra.figaro.language._
import com.cra.figaro.library.compound._
import com.cra.figaro.algorithm.OneTimeMPE
import com.cra.figaro.algorithm.factored.{VariableElimination,
MPEVariableElimination}


object Ex3 {
 abstract class State
 {
   val hidState: Element[String]
   def obsState: Element[String] = Apply(hidState, (hs:String) =>
                                        if(hs == "A") "good";
                                        else if(hs == "B") "no so good";
                                        else if (hs == "C")  "ill";
                                        else "death")
 }

 class InitialState(initHidState: Element[String]) extends State
 {
   val hidState = initHidState
 }

 class NextState(current: State) extends State
 {
   val hidState = CPD(current.hidState,
                      "A" -> Select(0.721 -> "A", 0.202 -> "B", 0.067 -> "C",
0.1 -> "D"),
                      "B" -> Select(0.581 -> "B", 0.407 -> "C", 0.012 -> "D"),
                      "C" -> Select(0.75 -> "C", 0.025 -> "D"),
                      "D" -> Select(1.0 -> "D"))
 }
 def stateSequence(n: Int): List[State] =
 {
   if (n == 0)
     List(new InitialState(Constant("A")))
   else
   {
     val last :: rest = stateSequence(n - 1)
     new NextState(last) :: last :: rest
```

```
    }
  }

  def main(args: Array[String])
  {
    var steps = 10;
    val stateSeq = stateSequence(steps)

    val algorithm = MPEVariableElimination()

    algorithm.start()
    for { i <- 0 until steps}
    {
        println(algorithm.mostLikelyValue(stateSeq(steps - 1 - i).hidState))
    }

    algorithm.kill()

    stateSeq(steps-1-6).obsState.observe("no so good")
    print(VariableElimination.probability(stateSeq(0).obsState, "ill"))

    stateSeq(steps-1-6).obsState.observe("ill")
    algorithm.start()
    for { i <- 0 until steps}
    {
        println(algorithm.mostLikelyValue(stateSeq(steps - 1 - i).hidState))
    }

    algorithm.kill()
  }
}
```