Valfri tentamen
EDAA45 Programmering, grundkurs

2022-01-08, 08:00-13:00

Hjälpmedel: Snabbreferens för Scala & Java.

Instruktioner

- Skriv din anonymkod + personlig identifierare här: _____
  Om du skriver icke-anonymt ange personnummer + namn i stället.
- Tillåtet hjälpmedel: Snabbreferens för Scala & Java.
- **Del A** (uppgift 1) ska besvaras genom att fylla i en tabell i *detta häfte*.
- **Del B** innehåller uppgift 2, 3, ... med svar i form av programkod som du ska skriva på *separata vita papper*. Skriv bara på *ena* sidan av varje inlämnat blad. Skriv din anonymkod + personlig identifierare (eller personnummer + namn om du skriver icke-anonymt) *överst* på *varje* inlämnat blad. Det ska tydligt framgå vilken (del)uppgift du löser.
- Detta häftet ska lämnas in tillsammans med ifyllt omslag och svaren på uppgifterna i del B.
- Innan du går måste du lämna in detta häfte och ifyllt omslag, även om du inte löst några uppgifter. Du får inte lämna salen innan du lämnat in. Du får tidigast lämna in 1 timme efter start.

Upplysningar

- Tentamen är valfri och kan, om du är tentamensberättigad, resultera i överbetygen 4 eller 5 om du uppfyller kraven för överbetyg, enligt nedan.
- För att vara tentamensberättigad ska du före tentamens start vara godkänd på alla obligatoriska moment (laborationer, projekt och muntligt prov), samt ha genomfört diagnostisk kontrollskrivning. Om du uppfyller dessa krav får du minst betyg 3 oavsett resultat på denna tentamen.
- Tentamen kan maximalt ge 100p, varav del A omfattar 20p och del B omfattar 80p.
- Preliminära krav för överbetyg:
  - För överbetyg krävs minst 10p på del A, samt totalt minst 67p för betyg 4 och 83p för betyg 5.
  - Om du erhåller färre än 10p på del A så bedöms inte del B och överbetyg medges ej.
  - Poäng och delpoäng som anges ovan och i uppgifterna är preliminära och kan komma att justeras när den slutliga bedömningen fastställs.
- Du ska *inte* tentera om du ej är tentamensberättigad. Om du ändå tenterar utan att vara tentamensberättigad annulleras din skrivning utan att bedömas. För att undvika att någon skrivning annulleras av misstag kommer alla som, enligt institutionens noteringar, tenterat utan att vara tentamensberättigade att kontaktas via epost. Felaktigheter i institutionens noteringar kan därefter påtalas inom en månad.
- Lösningar läggs ut på kursens hemsida efter tentamen.
- Resultatet läggs in i Ladok när bedömningen är klar.

Del A. 20p. Uppgift 1.

Följande kod finns kompilerad utan kompileringsfel och tillgänglig på din *classpath*:

```
sealed abstract class Betyg(val toInt: Int)
case object A extends Betyg(9)
case object B extends Betyg(-3)
case object Saknas  extends Betyg(0)

trait Djur:
  def läte: String
  def namn: String
  var vikt: Int = 1000
  infix def ät(d: Djur, tugga: Int = 1): Unit = d.vikt -= tugga
case class Katt(namn: String) extends Djur { val läte = "mjau"; vikt = 10 }
case class Hund(namn: String) extends Djur { val läte = "vov";  vikt = 20 }
class Lejon(namn: String = "") extends Katt(namn) { override val läte = "grrrr"}
class Mutant extends Djur { def läte = "!#%&#!"; def namn = "Tjernobyl" }

object kattutställning:
  private val db = scala.collection.mutable.ArrayBuffer.empty[(Djur, Betyg)]
  def anmäl(d: Djur): Unit = if d.namn != "" then db.append(d -> Saknas)
  def bedöm(d: Djur, b: Betyg): Unit = db.append(d -> b)
  def betyg(d: Djur): Vector[Betyg] = (d match {
    case k: Katt => db.filter(_._1 == k).map(_._2).toVector
    case _ => Vector(Saknas)
  }).filterNot(_ == Saknas)
  def sitterBredvidVarandra(d1: Djur, d2: Djur): Unit =
    if d1.vikt > d2.vikt then d1 ät d2 else if d1.vikt < d2.vikt then d2 ät d1
```

Du ska fylla i tabellen på nästa sida enligt följande. Antag att du skriver in nedan kod i Scala REPL rad för rad. För varje variabel med namn `u1` ... `u5`, ange statisk **typ** (alltså den typ kompilatorn härleder), samt det **värde** variabeln får efter initialisering, *eller* sätt i stället kryss i rätt kolumn om det blir ett **kompileringsfel** respektive **exekveringsfel**. Vid frånvaro av fel, svara på samma sätt som Scala REPL skriver ut typ respektive värde, enligt exempel `u0` i tabellen.

```
import kattutställning.*
val Gustav = Katt("Gustav")
val Laban = Hund("Laban")
val u0 = 41.0 + 1
val u1 = (new Djur).vikt
val u2 = Mutant() == Mutant() && Katt("Gustav") == Katt("Gustav")
val u3 = (Lejon().läte + Lejon().namn).length + Mutant().vikt
val u4 = { anmäl(Gustav); anmäl(Laban); anmäl(new Lejon); bedöm(Gustav, A) }
val u5 = db.size
val u6 = { sitterBredvidVarandra(Gustav, Laban); bedöm(Laban, B); Gustav.vikt }
val u7 = { bedöm(Gustav, B); (betyg(Gustav) ++ betyg(Laban)).map(_.toInt).min }
val u8 = Mutant().läte.apply(0)
val u9 = { u8 = 'A'; u8 + 1 }
val u10 = "K" + Gustav.namn.toUpperCase.reverse.drop(1).take(2)(2) + "T"
```

	Vid kompi-leringsfel sätt kryss.	Vid exekve-ringsfel sätt kryss.	Ange statisk **typ** som kompilatorn härleder om ej kompilerings- eller exekveringsfel.	Ange det **värde** som tilldelas vid exekvering, med samma format som vid utskrift av värdets `toString`, om ej kompilerings- eller exekveringsfel.
`u0`			`Double`	`42.0`
`u1`				
`u2`				
`u3`				
`u4`				
`u5`				
`u6`				
`u7`				
`u8`				
`u9`				
`u10`				

Del B. 80 p. Uppg. 2–6. System för hantering av den vetenskapliga granskningsprocessen.

Du ska färdigställa ett påbörjat program som hanterar granskningar av inskickade forskningsartiklar som utgör kandidater att väljas ut för presentation och publikation vid vetenskapliga konferenser.

Domänbeskrivning och övergripande krav

Inför akademiska forskningskonferenser kan forskare skicka in artiklar (eng. *papers*) med forskningsresultat för granskning (eng. *review*) av de meriterade forskare som är medlemmar i konferensens programkommitté. Granskningsprocessen har två steg: (1) allokering (uppg. 2, 3, 4) och (2) selektion (uppg. 5, 6).

Allokering av granskare. *Indata*: artiklar och granskare med eventuella intressekonflikter. *Utdata*: allokering av artiklar till granskare. Till varje artikel allokeras normalt 3 granskare. En forskare ska aldrig granska sina egna artiklar. En granskare av en artikel får ej heller ha någon intressekonflikt (eng. *conflict of interest*) med någon av dess författare. (Exempel på intressekonflikt kan vara att granskaren tidigare varit handledare för en forskare under hens doktorandtid, eller om granskaren nyligen deltagit i samma forskningsprojekt som någon av författarna.)

Selektion av publikationer. *Indata*: granskningbetyg. *Utdata*: lista med accepterade artiklar grupperade enligt specifika betygskriterier. De artiklar som får tillräckligt bra omdömen av granskarna accepteras för presentation vid konferensen, enligt granskningskommitténs beslut, och blir därmed erkända vetenskapliga publikationer.

Indataformat och -exempel

Till den prestigefyllda internationella forskningskonferensen i kravhantering vid namn REFSQ[1] skickades flera forskningsartiklar in för granskning som visas i fig. 1. Varje rad innehåller en kommaseparerad lista med författare, ett semikolon och artikelns titel. Fig. 2 visar granskare och deras deklarerade intressekonflikter. Fig. 3 visar granskningar.

Fig. 1: Filen `papers.txt` med en rad per inskickad forskningsartikel med respektive författare och titel.

```
0 Thomas Olsson,Krzysztof Wnuk;Quality Requirements Management
1 Soren Lauesen;Problem-Oriented Requirements in Practice
2 Wasim Alsaqaf,Maya Daneva,Roel Wieringa;Quality Requirements Challenges
3 Nelly Condori,Patricia Lago;Strategies in Requirements Prioritization
4 Johanna Bergman,Thomas Olsson,Isabelle Johansson,Kirsten Gröhn;User Experience
```

Fig. 2: Filen `conflicts.txt` med granskare och ev. artikelindex för vilka granskaren har deklarerat intressekonflikt.

```
Krzysztof Wnuk;2,4
Nelly Condori;3
Soren Lauesen;
Maya Daneva;2
Tao Yue;3,4
Didar Zowghi;
```

Fig. 3: Filen `reviews.txt` med alla granskares granskningsbetyg i en kommaseparerad lista med par `i=g` där artikelindex `i` har givits betyg `g`. Det finns 4 olika betyg, varav två är positiva (`StrongAccept` och `Accept`) och de andra två är negativa.

```
Didar Zowghi;0=Accept,1=Reject,4=Accept
Maya Daneva;0=StrongAccept,3=StrongReject,4=Accept
Nelly Condori;0=Accept,2=StrongAccept,4=Reject
Tao Yue;1=Reject,2=Accept
Krzysztof Wnuk;1=Reject,3=StrongAccept
Soren Lauesen;2=Accept,3=Reject
```

[1] REFSQ betyder *Requirements Engineering: Foundation for Software Quality*. Exemplet är en förenkling av den verkliga konferensen från ett tidigare år, som egentligen har fler artiklar med längre titlar, etc. All granskningsdata och intressekonflikter är fiktiva. Författarnamnen tillhör dock verkliga kravhanteringsforskare.

Filerna `papers.txt` och `conflicts.txt` utgör indata till allokeringen. Exempel på utdata från allokeringen visas i körningen av huvudprogrammet nedan. Filen `reviews.txt` är indata till urvalssteget och granskarna granskar enligt allokeringen i föregående steg. Du kan anta att indata uppfyller dessa krav:

- Artikelindex numreras från noll enligt ordningen på artiklarna i filen `papers.txt`.
- Om en granskare ej har angivit någon intressekonflikt i filen `conflicts.txt` så avslutas raden med ett semikolon.
- Inga indatafiler är tomma.

Allokering av granskare

Du ska göra klart de påbörjade programdelarna `FromFile`, `Multimap`, och `Allocator` som används av följande huvudprogram. Singelobjektet `FromFile` och case-klassen `Multimap` finns på nästa sida. Singelobjektet `allocation` med klassen `Allocator` finns på sidan 7.

```scala
import allocation.*

@main def Allocate =
  val papers = FromFile.loadPapers("papers.txt")
  val conflicts = FromFile.loadConflicts("conflicts.txt")
  val alloc = Allocator(papers, Multimap(conflicts))
  alloc.allocate(n = 3) // allokera 3 granskare per artikel

  println("\n*** För varje granskare, mängden av allokerade artiklar:")
  for (r, pis) <- alloc.currentAllocation do println(s"$r -> $pis")

  println("\n*** För varje artikel, mängden av allokerade granskare:")
  for pi <- papers.indices do println(s"$pi -> ${alloc.reviewersOfPaper(pi)}")
```

Med indata från fig. 1 och 2 ges en utskrift som liknar följande:

```
> scala Allocate

*** För varje granskare, mängden av allokerade artiklar:
Didar Zowghi -> Set(0, 1, 4)
Maya Daneva -> Set(0, 3, 4)
Nelly Condori -> Set(0, 2, 4)
Tao Yue -> Set(1, 2)
Krzysztof Wnuk -> Set(1, 3)
Soren Lauesen -> Set(2, 3)

*** För varje artikel, mängden av allokerade granskare:
0 -> Set(Didar Zowghi, Maya Daneva, Nelly Condori)
1 -> Set(Didar Zowghi, Tao Yue, Krzysztof Wnuk)
2 -> Set(Nelly Condori, Tao Yue, Soren Lauesen)
3 -> Set(Maya Daneva, Krzysztof Wnuk, Soren Lauesen)
4 -> Set(Didar Zowghi, Maya Daneva, Nelly Condori)
```

Programmet fungerar enligt följande:

- En vektor med artiklar skapas genom läsning från textfilen `papers.txt`.
- En nyckel-värde-tabell med granskare som nycklar och intressekonfliktmängder som värden skapas genom läsning från textfilen `conflicts.txt`.
- Klassen `Allocator` används för att skapa en allokering `alloc` där granskare tilldelas artiklar så att inga intressekonflikter förekommer och inga granskare granskar sina egna artiklar.
- Allokeringen visas genom utskrift av artiklar per granskare och granskare per artikel.

Uppgift 2 (8p). Programmet på sidan 5 läser indata med hjälp av singelobjektet `FromFile` nedan. Implementera metoden `loadPapers` med hjälp av metoden `getLines` och case-klassen `Paper` som har attributen artikeltitel och artikelförfattare enligt deklaration i singelobjektet `allocation` på nästa sida.

```
object FromFile:
  import allocation.*

  def getLines(file: String): Vector[String] =
    scala.io.Source.fromFile(file).getLines.toVector

  def loadPapers(file: String): Vector[Paper] = ??? // 8p

  def loadConflicts(file: String): Map[Reviewer, Set[Int]] =
    getLines(file).map { line =>
      val xs = line.split(';').toVector
      val reviewer = xs(0)
      val conflicts =
        if xs.length > 1 then xs(1).split(',').map(_.toInt).toSet
        else Set[Int]()
      reviewer -> conflicts
    }.toMap
```

Uppgift 3 (15p). Programmet på sidan 5 drar nytta av datastrukturen *multimap* som är en nyckel-värdetabell där värdena utgörs av en mängd. Du ska implementera case-klassen `Multimap` nedan som hanterar nycklar av strängtyp och värden som är heltalsmängder. Implementera metoderna `get`, `add` och `count`:

```
case class Multimap(toMap: Map[String, Set[Int]] = Map()):
  def get(key: String): Set[Int]       = ??? // 4p
  def add(kv: (String, Int)): Multimap = ??? // 6p
  def count(value: Int): Int           = ??? // 5p
```

Din implementation av `Multimap` ska uppfylla följande krav:

- Metoden `get` ska ge det värde som är associerat med nyckeln `key`. Om `key` saknas ska en tom mängd ges.
- Metoden `add` ska ge en ny `Multimap` där artikelindex `kv._2` är tillagt i den heltalsmängd som nyckeln `kv._1` är associerad med. Om nyckeln saknas ska en ny mappning läggas till som associerar nyckeln till en mängd det givna artikelindexet som enda element.
- Metoden `count` ska ge summan av antalet nycklar som är associerade med en mängd som innehåller heltalet `value`.
- `Multimap` ska fungera enligt följande exempel:

```
scala> var mm = Multimap().add("hej" -> 1)
mm: Multimap = Multimap(Map(hej -> Set(1)))

scala> mm = mm.add("hej" -> 2).add("svejs" -> 1).add("svejs" -> 1)
mm: Multimap = Multimap(Map(hej -> Set(1, 2), svejs -> Set(1)))

scala> mm.get("hello")
res0: Set[Int] = Set()

scala> mm.count(1)
res1: Int = 2
```

Uppgift 4 (38p). Gör klart de saknade delarna i `allocation` nedan enligt efterföljande krav och tips.

```
object allocation:
  case class Paper(title: String, authors: Vector[String])

  type Reviewer = String
  type PaperIndex = Int

  class Allocator(val papers: Vector[Paper], val conflicts: Multimap):
    private var allocations = Multimap()

    def currentAllocation: Map[Reviewer, Set[PaperIndex]] = allocations.toMap

    def reviewers: Set[Reviewer] = conflicts.toMap.keySet

    def papersOfReviewer(r: Reviewer): Set[PaperIndex] = allocations.get(r)

    def nbrOfReviewers(pi: PaperIndex): Int = ??? // 3p

    def reviewersOfPaper(pi: PaperIndex): Set[Reviewer] = ??? // 7p

    def isBadAllocation(reviewer: Reviewer, pi: PaperIndex): Boolean = ??? // 8p

    def allocReviewerToPaper(r: Reviewer, pi: PaperIndex): Unit =
      allocations = allocations.add(r, pi)

    def indexOfNextPaper(n: Int): PaperIndex = ??? // 5p

    def findNextReviewer(pi: PaperIndex): Option[Reviewer] = ??? // 15p

    def allocate(n: Int): Unit =
      val pi: PaperIndex = indexOfNextPaper(n)
      val rOpt: Option[Reviewer] = findNextReviewer(pi)
      if pi >= 0 && rOpt.nonEmpty then
        allocReviewerToPaper(rOpt.get, pi)
        allocate(n)
```

- Klassparametern `papers` innehåller alla artiklar i indexordning.
- Klassparametern `conflicts` innehåller alla granskare med tillhörande eventuella intressekonflikter.
- Attributet `allocations` lagrar gjorda allokeringar som mappar namn på en granskare till en artikelindexmängd i en `Multimap` som från början är tom.
- Metoden `nbrOfReviewers` ska räkna antalet granskare av artikelindex `pi`.
- Metoden `reviewersOfPaper` ska ge mängden av alla granskare av artikelindex `pi`.
- Metoden `isBadAllocation` ska ge **`true`** om `reviewer` finns bland författarna till artikeln med index `pi` eller om artikeln med index `pi` redan är allokerad till `reviewer` eller om `reviewer` har någon intressekonflikt med artikeln med index `pi`, annars **`false`**.
- Metoden `indexOfNextPaper` ska ge index för den artikel med lägst index som har mindre än `n` granskare. Om ingen sådan artikel finns ska `-1` returneras.
- Metoden `findNextReviewer` ska ge en `Option` med namnet på en granskare med minst antal allokerade artiklar som dessutom ej är dålig att allokeras till `pi` enligt `isBadAllocation`. Om flera sådana granskare finns kan vilken som helst av dessa granskare returneras. Om ingen sådan granskare finns eller om `pi` inte är ett giltigt artikelindex ska `None` returneras. *Tips:* Du kan ha nytta av `sortBy` och `dropWhile`.
- Metoden `allocate` allokerar en artikel till en granskare så länge det finns artiklar som har färre än `n` granskare enligt `indexOfNextPaper` och det finns granskare att allokera enligt `findNextReviewer`.

Selektering av publikationer

Uppgift 5 (12p). Vid urval av vilka forskningsartiklar som ska accepteras behövs sammanfattande information om granskningarna för varje artikel. Speciellt vill man veta medelbetyget för en artikel och om en artikel ej har några negativa betyg och därmed är en s.k. *clear accept*.

Klassen `ReviewMatrix` nedan lagrar alla granskningsbetyg i en matris som indexeras med artikelindex och index för granskare. `None` i matrisen på position (i, j) innebär att det inte finns någon granskning av artikel i som är gjord av granskare j.

```
enum Grade:
  case StrongReject, Reject, Accept, StrongAccept

  def isNegative: Boolean = Set(StrongReject, Reject).contains(this)

  def score: Int = Map(0 -> -2, 1 -> -1, 2 -> 1, 3 -> 2).apply(ordinal)

class ReviewMatrix(val nbrOfPapers: Int, val nbrOfReviewers: Int):
  private val matrix: Array[Array[Option[Grade]]] =
    Array.fill(nbrOfPapers, nbrOfReviewers)(None)

  def apply(paperIndex: Int, reviewIndex: Int): Option[Grade] =
    matrix(paperIndex)(reviewIndex)

  def update(paperIndex: Int, reviewIndex: Int, grade: Grade): Unit =
    matrix(paperIndex)(reviewIndex) = Some(grade)

  def averageGrade(paperIndex: Int): Option[Double] = ??? // 6p

  def isClearAccept(paperIndex: Int): Boolean = ??? // 6p
```

Du ska implementera de saknade delarna enligt nedan krav och tips:

- `ReviewMatrix` lagrar granskningsbetygen i matrisattributet `matrix`.
- När instanser av `ReviewMatrix` konstrueras ges storleken på matrisen av parametrarna `nbrOfPapers` och `nbrOfReviewers`.
- Metoden `averageGrade` ska beräkna ett numerisk medelvärde av alla betyg för en viss artikel med index `paperIndex`, där varje betygs motsvarande numeriska värde ges av `score`. Medelvärdesberäkningen ska baseras på motsvarande betygs `score` för de `reviewIndex` där betyg finns. Om en artikel helt saknar betyg så ska `None` returneras.
- Metoden `isClearAccept` ska returnera **`false`** för en viss artikel med index `paperIndex` om minst ett negativt betyg finns för denna artikel eller om det inte finns några betyg alls för denna artikel, annars ska **`true`** returneras.

Uppgift 6 (7p). Huvudprogrammet `Select` nedan använder sig av `ReviewMatrix` för att dela upp alla artiklar i kategorierna *clear accept*, *borderline* och *clear reject*. Gränsfallen *borderline* ska diskuteras vidare på ett granskningskommittémöte.

Om programmet körs med indata enligt fig. 1 och fig. 3 ska följande utskrift ske:

```
> scala Select
Clear accept: 0,2
Borderline:   3,4
Clear reject: 1
```

I huvudprogrammet nedan ska du implementera `selectPaperIndices` och `bordeline`.

```scala
@main def Select =
    val reviewerIndex: Map[String, Int] =
      FromFile.loadConflicts("conflicts.txt").keySet.toVector.zipWithIndex.toMap

    val papers = FromFile.loadPapers("papers.txt")

    val rm = ReviewMatrix(papers.length, reviewerIndex.size)

    for line <- FromFile.getLines("reviews.txt") do
      val xs = line.split(';')
      if xs(0).nonEmpty && xs(1).nonEmpty then
        val ri = reviewerIndex(xs(0))
        val reviews = xs(1).split(',').map(_.split('='))
        for Array(pi, g) <- reviews do rm(pi.toInt, ri) = Grade.valueOf(g)

    def selectPaperIndices(p: Int => Boolean): Seq[Int] = ??? // 2p

    val clearAccepts: Seq[Int] = selectPaperIndices(pi => rm.isClearAccept(pi))

    println("Clear accept: " + clearAccepts.mkString(","))

    val borderline: Seq[Int] = ??? // 5p

    println("Borderline:   " + borderline.mkString(","))

    val clearReject: Seq[Int] = papers.indices diff clearAccepts diff borderline

    println("Clear reject: " + clearReject.mkString(","))
```

Dina implementationer ska beakta dessa krav:

- `selectPaperIndices` ska ge en sekvens av växande index för de artiklar i `papers` för vilka predikatet `p` är sant.
- Sekvensen *borderline* ska skapas genom att selektera de artiklar som ej är *clear accept* och har ett medelbetyg som är större än `-0.4`.
- Implementationen av `borderline` ska använda sig av `selectPaperIndices`.