EDAA45 Programmering, grundkurs

Läsvecka 5: Sekvensalgoritmer

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2016

5 Sekvensalgoritmer

- Vad är en sekvensalgoritm?
- SEQ-COPY
- SEQ-INSERT
- StringBuilder
- Registrering
- Variablet antal argument
- Att välja sekvenssamling

Vad är en sekvensalgoritm?

Vad är en sekvensalgoritm?

- En algoritm är en stegvis beskrivning av hur man löser ett problem.
- En sekvensalgoritm är en algoritm där dataelement i sekvens utgör en viktig del av problembeskrivningen och/eller lösningen.
- Exempel: sortera en sekvens av personer efter deras ålder.
- Två olika principer:
  - Skapa **ny sekvens** utan att förändra indatasekvensen
  - Ändra **på plats** (eng. *in place*) i den **förändringsbara** indatasekvensen

SEQ-COPY

Algoritm: SEQ-COPY

Pseudokod för algoritmen SEQ-COPY som kopierar en sekvens, här en Array med heltal:

Indata : Heltalsarray *xs*
Resultat: En ny heltalsarray som är en kopia av *xs*.

result $\leftarrow$ en ny array med plats för *xs.length* element
$i \leftarrow 0$
while $i < xs.length$ **do**
  $result(i) \leftarrow xs(i)$
  $i \leftarrow i + 1$
end
return *result*

Implementation av SEQ-COPY med `while`

```
1  object seqCopy {
2
3    def arrayCopy(xs: Array[Int]): Array[Int] = {
4      val result = new Array[Int](xs.length)
5      var i = 0
6      while (i < xs.length) {
7        result(i) = xs(i)
8        i += 1
9      }
10     result
11   }
12
13   def test: String = {
14     val xs = Array(1,2,3,4,42)
15     val ys = arrayCopy(xs)
16     if (xs sameElements ys) "OK!" else "ERROR!"
17   }
18
19   def main(args: Array[String]): Unit = println(test)
20 }
```

Implementation av SEQ-COPY med `for`

```
1  object seqCopyFor {
2
3    def arrayCopy(xs: Array[Int]): Array[Int] = {
4      val result = new Array[Int](xs.length)
5      for (i <- xs.indices) {
6        result(i) = xs(i)
7      }
8      result
9    }
10
11   def test: String = {
12     val xs = Array(1,2,3,4,42)
13     val ys = arrayCopy(xs)
14     if (xs sameElements ys) "OK!" else "ERROR!"
15   }
16
17   def main(args: Array[String]): Unit = println(test)
18 }
```

Implementation av SEQ-COPY med `for-yield`

```
1  object seqCopyForYield {
2
3    def arrayCopy(xs: Array[Int]): Array[Int] = {
4      val result = for (i <- xs.indices) yield xs(i)
5      result.toArray
6    }
7
8    def test: String = {
9      val xs = Array(1,2,3,4,42)
10     val ys = arrayCopy(xs)
11     if (xs sameElements ys) "OK!" else "ERROR!"
12   }
13
14   def main(args: Array[String]): Unit = println(test)
15 }
```

Implementation av SEQ-COPY i Java med `for`-sats

```
1  public class seqCopyForJava {
2
3      public static int[] arrayCopy(int[] xs){
4          int[] result = new int[xs.length];
5          for (int i = 0; i < xs.length; i++){
6              result[i] = xs[i];
7          }
8          return result;
9      }
10
11     public static String test(){
12         int[] xs = new int[]{1, 2, 3, 4, 42};
13         int[] ys = arrayCopy(xs);
14         for (int i = 0; i < xs.length; i++){
15             if (xs[i] != ys[i]) {
16                 return "FAILED!";
17             }
18         }
19         return "OK!";
20     }
21
22     public static void main(String[] args) {
23         System.out.println(test());
24     }
25 }
```

SEQ-INSERT-COPY

SEQ-INSERT-IN-PLACE

StringBuilder

Registrering

Variablet antal argument

Parameter med variabelt antal argument, ”variargs”

Med en asterisk efter parametertypen kan antalet argument variera:

```
def sumSizes(xs: String*): Int = xs.map(_.size).sum
```

```
scala> sumSizes("Zaphod")
res0: Int = 6

scala> sumSizes("Zaphod","Beeblebrox")
res1: Int = 16

scala> sumSizes("Zaphod","Beeblebrox","Ford","Prefect")
res3: Int = 27

scala> sumSizes()
res4: Int = 0
```

Typen på `xs` blir en `Seq[String]`, egentligen en `WrappedArray[String]` som kapslar in en array så den beter sig mer som en ”vanlig” Scala-samling.

Sekvenssamling som argument till varargs-parameter

```
def sumSizes(xs: String*): Int = xs.map(_.size).sum

val veg = Vector("gurka","tomat")
```

Om du *redan har* en sekvenssamling så kan du applicera den på en parameter som accepterar variabelt antal argument med typannoteringen

`: _*`

direkt **efter** sekvenssamlingen.

```
scala> sumSizes(veg: _*)
res5: Int = 10
```

Denna veckas övning: `sequences`

- Kunna implementera funktioner som tar argumentsekvenser av godtycklig längd.
- Kunna tolka enkla sekvensalgoritmer i pseudokod och implementera dem i programkod, t.ex. tillägg i slutet, insättning, borttagning, omvändning, etc., både genom kopiering till ny sekvens och genom förändring på plats i befintlig sekvens.
- Kunna använda föränderliga och oföränderliga sekvenser.
- Förstå skillnaden mellan om sekvenser är föränderliga och om innehållet i sekvenser är föränderligt.
- Kunna välja när det är lämpligt att använda `Vector`, `Array` och `ArrayBuffer`.
- Känna till att klassen `Array` har färdiga metoder för kopiering.
- Kunna implementera algoritmer som registrerar antalet förekomster av objekt i en sekvens som indexeras med antalet förekomster.
- Kunna generera sekvenser av pseudoslumptal med specificerat slumptalsfrö.
- Kunna implementera sekvensalgoritmer i Java med **`for`**-sats och primitiva arrayer.
- Kunna beskriva skillnaden i syntax mellan arrayer i Scala och Java.
- Kunna använda klassen `java.util.Scanner` i Scala och Java för att läsa in heltalssekvenser från `System.in`.

Denna veckas laboration: `shuffle`

- Kunna skapa och använda sekvenssamlingar.
- Kunna använda sekvensalgoritmen SHUFFLE för blandning på plats av innehållet i en array.
- Kunna registrera antalet förekomster av olika värden i en sekvens.

Att välja sekvenssamling

Oföränderlig eller förändringsbar?

- **Oföränderliga**: Kan ej ändra elementreferenserna, men effektiva på att skapa kopia som är (delvis) förändrad
  (vanliga i Scala men inte i Java): **Vector** eller **List**
- **Förändringsbara**: kan ändra elemententreferenserna
  - Kan **ej ändra storlek** efter allokering:
    Scala+Java: **Array**: indexera och uppdatera varsomhelst
  - Kan ändra storlek efter allokering:
    Scala: **ArrayBuffer** eller **ListBuffer**
    Java: **ArrayList** eller **LinkedList**

Egenskaper hos några sekvenssamlingar

- `Vector`
  - **Oföränderlig**. Snabb på att skapa kopior med små förändringar.
  - Allsidig prestanda: **bra till det mesta**.
- `List`
  - **Oföränderlig**. Snabb på att skapa kopior med små förändringar.
  - Snabb vid bearbetning **i början**.
  - Smidig & snabb vid **rekursiva** algoritmer.
  - Långsam vid upprepad **indexering** på godtyckliga ställen.
- `Array`
  - **Föränderlig**: **snabb indexering & uppdatering**.
  - Kan **ej ändra storlek**; storlek anges vid allokering.
  - Har särställning i JVM: ger snabbaste minnesaccessen.
- `ArrayBuffer`
  - **Föränderlig**: **snabb indexering & uppdatering**.
  - Kan **ändra storlek** efter allokering. Snabb att indexera överallt.
- `ListBuffer`
  - **Föränderlig**: snabb indexering & uppdatering **i början**.
  - Snabb om du bygger upp sekvens genom många tillägg i början.

Vilken sekvenssamling ska jag välja?

- `Vector`
  - Om du vill ha oföränderlighet: `val xs = Vector[Int](1,2,3)`
  - Om du behöver ändra (men ej prestandakritiskt):
    `var xs = Vector.empty[Int]`
  - Om du ännu inte vet vilken sekvenssamling som är bäst; du kan alltid ändra efter att du mätt prestanda och kollat flaskhalsar.
- `List`
  - Om du har en rekursiv sekvensalgoritm och/eller bara lägger till i början.
- `Array`
  - Om det behövs av prestandaskäl och du **vet** storlek vid allokering:
    `val xs = Array.fill(initSize)(initValue)`
- `ArrayBuffer`
  - Om det behövs av prestandaskäl och du **inte** vet storlek vid allokering:
    `val xs = scala.collection.mutable.empty[Int]`
- `ListBuffer`
  - om det behövs av prestandaskäl och du bara behöver lägga till i början:
    `val xs = scala.collection.mutable.ListBuffer.empty[Int]`

Lämna det öppet: använd Seq[T]

```
def varannanBaklänges[T](xs: Seq[T]): Seq[T] =
  for (i <- xs.indices.reverse by -2) yield xs(i)
```

Fungerar med alla sekvenssamlingar:

```
scala> varannanBaklänges(Vector(1,2,3,4,5))
res0: Seq[Int] = Vector(5, 3, 1)

scala> varannanBaklänges(List(1,2,3,4,5))
res1: Seq[Int] = List(5, 3, 1)

scala> varannanBaklänges(collection.mutable.ListBuffer(1,2))
res2: Seq[Int] = Vector(2)
```

Scalas standardbibliotek returnerar ofta lämpligaste specifika sekvenssamlingen som är subtyp till `Seq[T]`.