Datastrukturer og algoritmer
Henning Christiansen
22/9-2002

Besvarelse af opgave til formuleret til øvelserne 10. september 2002

Følgende er ikke en eksakt besvarelse af opgaven, idet den benytter hashtabeller og ikke den banale array-repræsentation, som er foreslået i opgaven (hashtabeller først omtalt på kurset efter opgaven blev stillet). Opgaveformuleringen forudsættes kendt og forstået. Kildetekst til besvarelse: `Word.java`, `Segment.java`, `SegmentCount.java`

Spørgsmål 1

Word-klassen har kun et ikke-statisk medlem som er en heltalsvariabel kaldet `wordNo`. Konstruktoren kaldes med en tekststreng som parameter, og klassen opretholder en en-til-en relation mellem streng og `wordNo` for de strenge som er forekommet (dvs. sendt med konstruktoren). En hashtabel som er fælles for alle objekter i klassen afbilder strenge over i wordNo, så den kan bruges af konstruktoren. Første forekommende streng tildeles nr. 0, næste nr. 1, og så fremdeles. For at kunne implementere en `toString` metode på effektiv måde opretholdes afbildningen fra nummer til faktisk string i et array, så `Word` objekt med nr. n har sin streng i dette arrays position n. Endelig indeholder klassen en equals-metode som sammenligner på `wordNo`.

Kildeteksten `Word.java` minus faciliteter til de efterfølgende spørgsmål kan skitseres som følger:

```
public class Word {
public Word(String s)
   {Object foundWordNo;
    if( (foundWordNo = table.get(s)) != null )
       wordNo = ((Integer)foundWordNo).intValue();
    else { wordNo = wordCount++;
           table.put(s, new Integer(wordNo));
           stringTable[wordNo]=s;};};

public String toString() {return stringTable[wordNo];};

// the following very inefficient way to compare two ints... sic, Java :(
public boolean equals(Object rhs){
  if(rhs==null || getClass() != rhs.getClass()) return false;
  return wordNo == ((Word)rhs).wordNo;};

private int wordNo;
public int wordNo() {return wordNo;};
private static int wordCount = 0;
private static HashMap table = new HashMap(10000);
private static String [] stringTable = new String [10000];
}
```

Spørgsmål 2

Sekvenser af længde mellem 1 og 4 ord repræsenteres ved en klasse `Segment`, som uden grundlæggende ser således ud:

```
public class Segment {

  public static final int maxSegmentLength = 4; // hardwired in code :(

  Word [] words;

  public Segment(Word w1)
    { words = new Word [1]; words[0]=w1;};
  public Segment(Word w1,Word w2)
    { words = new Word [2];words[0]=w1;words[1]=w2;};
  public Segment(Word w1,Word w2,Word w3)
    { words = new Word[3];words[0]=w1;words[1]=w2;words[2]=w3;};
  public Segment(Word w1,Word w2,Word w3,Word w4)
    { words = new Word[4];words[0]=w1;words[1]=w2;words[2]=w3;
              words[3]=w4;};

public String toString()
     {String s ="";
      for(int i=0; i<words.length; i++) s= s + " " + words[i];
      return s;}

  public boolean equals(Object rhs){
    if(rhs==null || getClass() != rhs.getClass()) return false;
    if(words.length != ((Segment)rhs).words.length) return false;
    for(int i=0; i<words.length; i++)
      if(!words[i].equals( ((Segment)rhs).words[i])) return false;
    return true;};
}
```

Vi har valgt at repræsentere sekvensen ved et array af `Word` objekter, hvor længden af arrayet svarer til antallet af ord. Der er fire forskellige konstruktører svarende til de forskellige længder. Dette er ikke helt hensigtsmæssigt, da der skal skrives nye konstruktører såfremt nogen kunne finde på at sætte den mulige maksimale sekvenslængde op. Det havde været mere hensigtsmæssigt i det lys med kun en konstruktor (der så skulle tage et array eller lignende af `Word` objekter som argument).

Optælling af strenge sker ved en separat klasse `SegmentCount` der indeholder en hashtabel fra segment til antal gange det er forekommet, og så en main-metode som indlæser ord fra en fil.

Opgaveformuleringen beskriver, at inputfilen skal slutte med ordet "`endoffile`" så vi kan bruge det til at teste om vi er færdige. For at få det til at virke, tilføjer vi følgende til `Word` klassen:

```
private static int endoffileWordCount;
public boolean isEndoffileWord() {return wordNo == endoffileWordCount;};
public static Word readWord(FileReader f) { ... se kildetekst ... };
```

Konstanten endoffile defineres i en statisk initialiseringdel i `Word` ved at skrive

```
new Word("endoffile"); endoffileWordCount=wordCount-1;
public boolean isEndoffileWord(){return wordNo==endoffileWordCount;};
```

Klassen `SegmentCount` kan skitseres som følger; det bliver noget forgræmmet på grund af indlæsning fra fil:

```
public class SegmentCount {

public static void main(String [] args) throws IOException{
        FileReader dataFile = new FileReader("duckling");
        Word w1 = Word.readWord(dataFile);
        Word w2 = Word.readWord(dataFile);
        Word w3 = Word.readWord(dataFile);
        Word w4 = Word.readWord(dataFile);
        while(!w4.isEndoffileWord())
          {count(new Segment(w1));
               count(new Segment(w1,w2));
               count(new Segment(w1,w2,w3));
               count(new Segment(w1,w2,w3,w4));};
            w1=w2;w2=w3;w3=w4;w4=Word.readWord(dataFile);
          };
        //the end
        count(new Segment(w1));
        count(new Segment(w1,w2));
        count(new Segment(w1,w2,w3));
        count(new Segment(w2));
        count(new Segment(w2,w3));
        count(new Segment(w3));

        //print out those occurring 3 or more times
        slettet her, da det bliver meget klodset pga. designbøffer i
Java; se kildetekst
};


  private static HashMap segmentCounts = new HashMap(10000) ;

  private static class IntegerVar //to be use in segmentCounts only
      {public int value;
       public IntegerVar(int i){value=i;};
       public void inc(){value++;};
       public String toString(){return ""+value;};};

  public static void count(Segment s)
   {Object foundSegCount;
    if( (foundSegCount = segmentCounts.get(s)) != null )
       ((IntegerVar)foundSegCount).inc();
    else
      segmentCounts.put(s, new IntegerVar(1));
   };
```

Strukturen omkring hashtabellen skyldes at vi ikke kan opbare heltal i en hashtabel, men det skal være noget som er underklasse af `Object`. Vi kunne i princippet benytte objekter af `java.lang`'s `Integer` klasse, men de har den egenskab, at deres værdi ikke kan tælles op (de er "immutable"). Det ville betyde, at for at tælle én op for en sekvens, så skulle vi slette en indgang i hashtabellen (med det gamle `Integer` objekt), oprette en ny Integer objekt med en højere værdi og endelig lægge det eksplicit ind i tabelle.
For at undgå alt det, har vi indført en lokal klasse `IntegerVar` som kan tælles op med med en metode "`inc`".

Bemærk at koden som foretager optællingen er afhængig af at den valgte max-længde for sekvenser. Det ville være bedre at formulere denne del af koden i form af en løkkekonstruktion, som aflæste konstanten `maxSegmentLength` i klassen `Segment`.

Spørgsmål 3

Der beskrives en løsning, som tager hensyn til de såkaldte stopwords med det samme. Desuden tages der et særligt hensyn til ordet "the" som ikke fremgår af opgaven; forklares nedenfor.

Vi har indført en metode i klassen `Segment` som afgør om det givne segment er interessant:

```
public boolean isInteresting() {....};
```

For at kunne implementere den må vi kunne klassificere de enkelt `Word` objekter i underarter. En mulighed kunne være at lave forskellige underklasser af Word, men det ville medføre at Java-Compileren må tilføje ekstra dynamisk typeinformation til den faktiske objektrepræsentation. I stedet har vi valgt at koble bool'ske metoder på Word klassen, som spørger til arten af `Word` objektet. Stopord skal indlæses fra en fil når klassen startes, så derfor har Word en statisk initialiseringsdel, som gør det. Implementationen er så visseligt indrettet, at den tildeler ordnr. 0 til "`end-of-file`". Derefter indlæses all stopordene og "oprettes" med `new`... dvs. de får de efterfølgende numre, og vi kan registrere ordtællerens aktuelle værdi i en skjult variabel `maxStopwordCount`; dvs. ord med lavere nummer er stopord, med højere, ikke et stopord. Som et eksperiment er artiklen "the" ikke i dette program opfattet som storord, men som et særligt ord (det kommer vi til senere).

Den statiske initialiseringsdel i word bliver nu sådan her (bemærk, at stopordsfilen også forventes at slutte med "`endoffile`"):

```
new Word("endoffile"); endoffileWordCount = wordCount-1;
FileReader stopWordFile = new FileReader("stopwords");
Word w = readWord(stopWordFile);
while(!w.isEndoffileWord()) w=readWord(stopWordFile);
maxStopwordCount = wordCount-1;
new Word("the"); theWordCount = wordCount-1;
```

Metoderne i klasse `Word` til klassifikation bliver nu således:

```
public boolean isStopword() {return wordNo <= maxStopwordCount;};
public boolean isTheWord() {return wordNo == theWordCount;};
public boolean isEndoffileWord() {return wordNo == endoffileWordCount;};
public boolean isContentWord() {return !isStopword() && !isTheWord();};
```

Som det fremgår indfører vi en ny kategori "`ContentWord`" som er alt som ikke er stopord eller "the"; bemærk at "`end-of-file`" på denne måde også kategoriseres som stopord, men det skulle helst være lige meget, da den ikke er tænkt at skulle proppes ind i segmenter.

Vi kan nu udfylde metoden i `Segment` som afgør hvorvidt en sekvens er interessant (dvs. om vi gider at tælle den med):

```
public boolean isInteresting()
   { if(words[0].isStopword()) return false;
     if(!words[words.length-1].isContentWord()) return false;
```

```
      return true;};
```

Det er lige for at modificere main metoden i `SegmentCount` så den kun tæller de interessante med. Vi har i udskriften af analysen af den grimme ælling på engelsk begrænset os til at vise sekvenser på længde 3 og derover som forekommer mindst 4 gange:

```
among the rushes: 3
the tom cat: 4
said the hen: 3
the poor duckling: 3
the poor little: 4
said the duckling: 3
said the old duck: 3
said the duck: 3
said the old: 4
said the mother: 4
the old duck: 3
```