Inlämningsuppgift 5

Uppgift 1

Rekursivitet

Många iterativa algoritmer kan också skrivas rekursivt. Testa följande algoritm:

```
public static void print(int[] a, int n)
{
        if(n>=0)
      {
              System.out.println(a[n]); return
                print(a,n-1);
        }
}
```

Uppgift:

1. Skriv en rekursiv metod som skrivet ut alla talen från 0-n.
2. En palindrom är ett ord som är lika oavsett från vilket ände du läser den. Dvs RADAR och KAJAK är palindrom. Skriv en rekursiv metod som undersöker om ett ord är palindrom eller inte.
3. Skriv en rekursiv metod som tar som argument en bas och en exponent och beräknar bas^exponet.

Uppgift 2

a)Använd klassen TreeSet från java biblioteket, http://docs.oracle.com/javase/6/docs/api/java/util/TreeSet.html,iett program som läser in alla orden från en textfil till ett TreeSet. (Använd vilken fil du vill men den skall innehålla min 500 ord). Därefter med hjälp av trädets iterator skriv ut alla orden till System.out eller till en fil.

Fråga: I vilken ordning skrivs orden ut? Varför?

b) För att du skall kunna veta både hur många unika ord som finns i en fil men också antalet förekomst av varje ord skall du utveckla ett speciellt träd datastruktur. Kalla klassen TreeSetCounter. En nod i ett sådant träd består av ett ord samt en räknare. Följande metoder skall implementeras:

addWord- lägger till ordl i träder om orden inte redan finns. Om ordet finns skall räknaren uppdateras
makeEmpty()- gör trädet tom
getMaxFrek() –returnera ordet som har förekommint mest antal gånger . Hur kan du göra det mest effektivt?
iterator()- returnera en iterator till trädet. Den skall användas för att skriva ut trädet i alfabetisk ordning.

Använd som utgångspunkt någon balanserat trädimplementation.
Skriv om programmet från punkt **a** så att du läser en fil och skriver ut både orden och antalet förekomst av varje ord.

Uppgift 3

Binära träd används i många applikationer bland annat i komprimerings algoritmer.
Läs mer om Huffman komprimering:
http://www.cs.duke.edu/csed/poop/huff/info/

Några exempel på hur mycket man kan komprimera filerna med denna metod:
en fil innehållande svensk text på 13032 bits komprimeras till 7364 bits.
en fil innehållande java källkod på 16912 bits komprimeras till 10395
en fil innehållande java bytecode på 28576 bits komprimeras till 20727 bits
en postscript fil innehållande 959632 komprimeras till 307385 bits.

Läs, förstå och förklara med dina egna ord algoritmen för Huffman komprimering. Varför spelar filens typ roll i hur mycket filen komprimeras.

Ni skall komplettera följande program som beräknar koderna för de tecken som förekommer i en fil enligt Huffmans metod. Programmet är strukturerat på följande sätt:

En nod i en Huffaman träd innehåller data (i detta fallet ett tecken) och ett numerisk värde (i detta sammanhang kallat vikt).

Vi har därför definierat en klass **HuffmanTree implements Comparable** som innehåller en binär nod och ett heltal (för vikten). Klassen erbjuder två konstruerare, en metod för att jämföra Huffman träd och en metod för att beräkna koderna. Själva algoritmen för att bygga det trädet som innehåller koderna skall du implementera i en klass Huffman där den enda metod som erbjuds är main().

Följande filer skall du ladda ner för uppgiften. De finns i blackboard.

1. BinaryNode.java
2. HuffmanTree.java
3. HuffmanKomprimering.java

Obs! Ni behöver ange ett filnamn som komandoradens argument! Det är filen som skall analyseras av Huffman. Som du ser i skeletten för klassen Huffman (enligt min design av klassen) skall följande metoder implementeras:

1. En metod som läser tecken från filen beräknar frekvenser för varje tecken(hur många gånger ett tecken förekommer i en text). Ett sätt att göra det är genom att använda en array med platser för 256 tecken. Så många olika standardiserade tecken finns i Java. Se instansvariablerna som jag deklarerat i klassen Huffman. Använd t.ex klassen BufferedReder för att läsa från fil http://docs.oracle.com/javase/1.5.0/docs/api/java/io/BufferedReader.html.

Då kan du läsa ett tecken i taget från filen med metoden *read().*
OBS! Metoden read() returnerar -1 när filen är slut dvs. EOF.

2. En metod som bygger ett Huffman träd med hjälp av de tecken som har lästs in och sina frekvenser (se kurslitteraturen).

Tips! Använd en lista (klassen ArrayList från biblioteket) för att lagra dina Huffman träd i sorterad ordning. Sätt ihop träden enligt beskrivningen i kursen tills du får ett enda träd.

3. En metod som använder det resulterande Huffman trädet för att beräkna koden för varje tecken (se kurslitteraturen).

4. En metod showResult () som visar (skriver ut) : frekvenser för tecken i filen ,själva tecknet och koden som tecknet har fått.

5. En metod makeFile() som utifrån de koder ni fått fram översätter och bygger upp en komprimerad fil. I klassen Huffman hittar du metoden stringTobyte(String s) som du bör använda för att omvandla strängar av ettor och nollor till byte som kan sparas till fil.

Testa ditt program med filen: testbild1.txt Då skall din komprimerade fil som du får fram i metoden **makeFile()** ha följande innehåll (OBS! Innehållet i testcompress.txt kan du inte se om du inte sparar först filen på hårddisken) testcompress.txt (textfil, 444 bytes) och utskriften från metoden **showResult()** result.txt (textfil, 521 bytes).

6. Testa ditt program med några andra typer av filer till exempel: en fil som innehåller java program och en som innehåller samma program men kompilerad (bytekodfilen)

Frågor att besvara:

2) Beskriv med dina egna ord en algoritm för att avkomprimering av Huffman komprimerade filer.