Webcrawler

Denna uppgift går ut på att implementera en enkel ”webcrawler” – ett litet program som läser av ett antal sammankopplade webbsidor och skapar en bild av hur de sitter ihop. Den grundläggande algoritmen är enkel:

1. Läs in en webbsida l, extrahera dess länkar i en lista L och spara l i listan B av besökta sidor.
2. För varje länk l i listan L som inte redan är i B, upprepa steg 1.)

Ovanstående algoritm kör tills alla sidor som går att nå från startpunkten har nåtts. Om man t.ex. vill bygga en modell i minnet av alla webbsidor och hur de länkar till varandra kan man skapa en klass `Page` som har en lista med referenser till andra sidor som länkas till av den aktuella sidan.

Uppgiften

1. Webcrawling.

   Skriv ett program som givet en startpunkt och ett ”djup” d söker igenom en webbplats tills hela nätverket av sidor har traverserats eller d steg tagits (vad som nås först). En modell av webbplatsen skall skapas i minnet med sidor som länkar till varandra. Som vi skall se nedan är *texterna i länkarna* också av stort intresse. I följande länk är alltså texten i länken (eller länk-texten) ”Kurswebben för IOOPM”:

   ```
   <a href="http://wrigstad.com/ioopm">Kurswebben för IOOPM</a>
   ```

   Använd regulära uttryck för att tvätta fram data ur HTML-filer. Följande kodsnutt definierar ett regulärt uttryck för att ta fram innehållet i en `<title>`-tagg och spara i strängvariabeln `name`:

   ```
   Pattern TITLE = Pattern.compile("<title>(.*?)</title>",
                                   Pattern.MULTILINE | Pattern.CASE_INSENSITIVE);

   Matcher titleMatcher = TITLE.matcher(pageContents);
   if (titleMatcher.find()) {
     name = titleMatcher.group(1);
   }
   ```

   Använd hjälpklassen `Utility` (bifogad) för att hämta innehållet i en fil via en URL. **OBS!** Använd *alltid lokala filer* när du kör ditt program så du slipper DOS-attakera en webbserver varje gång du får en oändlig loop! **OBS!**

   Skapa ett antal sidor som länkar till varandra och använd dem för att testa ditt system. Naturligtvis kan du också ladda ned HTML-filerna för hela Wikipedia och leka med, men det blir svårare att verifiera att test är korrekta om man inte har bra kontroll över testdatat.

2. Taggmoln.

   Ett taggmoln är en grafisk representation av hur ofta ett nyckelord förekommer på en webbplats. De populäraste nyckelorden är större, i en annan färg eller typsnitt än övriga.

   Utöka programmet med stöd för att generera taggmoln i form av ett frekvens-index. Utdata är en lista av alla ord som förekommer i texten i länkar, och för varje ord, dess frekvens i procent av samtliga ord, max 100 stycken, som finns i länktexter och avrundat nedåt.

   T.ex. om det finns tre länkar på en webbplats, två med texten ”foo” och en med texten ”bar” så skulle utdatat från programmet bli:

   ```
   foo     66
   bar     33
   ```

3. Filtrering.

   Man vill ofta ha möjligheten att filtrera bort vissa ord ur taggmolnsgenereringen. T.ex. ord som ”och”, ”i”, etc. dominerar ofta i frekvens men är innehållslösa. Implementera stöd för att filtrera bort ord med hjälp av en fil `excludes.txt` som är en radorienterad textfil med ett ord per rad som

skall ignoreras vid generering av taggmoln. Om filen finns och är tom skall en varning ges, men om filen inte finns tolkas det som om inga ord skall ignoreras.

4. Stöd för sidrankning (page ranking).

   En sökmotor letar igenom sidor och indexerar dem så att effektiv sökning kan göras i indexet som pekar till de faktiska sidorna. Ett delproblem i sökning är s.k. ”page ranking”, d.v.s. i vilken ordning presenterar jag resultatet om det finns fler än en sida som matchar sökningen?

   Ett sätt att implementera page ranking är att uppfinna ett enkelt ”poängsystem” som rankar sidorna efter ett par enkla regler. T.ex., för rankning av sidan s:

   (a) Varje förekomst av termen t i löptext ger 1p,

   (b) Varje förekomst av termen t i `<title>`-taggen ger 4p,

   (c) Varje förekomst av termen t i texten till en länk *som länkar till* s ger 5p,

   (d) ...

   Underlätta för en tänkt sökmotor att göra page ranking genom att varje sida s sparar en förteckning över vilka ord som förekommer i texterna i länkar till s samt antal förekomster, t.ex.:

   ```
   bar 4
   foo 2
   quux 1
   ```

   Ovanstående visar t.ex. att det finns två länkar till den aktuella sidan som har ordet ”foo” i sin länk-text.

5. Wikirace eller Kevin Bacon-index.

   Ett wikirace är en tävling där alla tävlande startar med en given Wikipedia-artikel och skall med så få klick som möjligt ta sig till en given ”målartikel”.

   Kevin Bacon-index, är en lek uppkallad efter skådespelaren Kevin Bacon som går ut på att länka en annan slumpmässigt vald skådespelare till honom genom så få steg som möjligt. De flesta amerikanska skådespelare kan länkas till Kevin Bacon i sex steg eller mindre. Två skådespelare anses länkade om de spelat i samma film. T.ex. kan Janne Loffe Carlsson länkas till Kevin Bacon i tre steg:

   (a) Janne Loffe Carlsson spelade i samma film som Bo Brundin i ”Jordgubbar med riktig mjölk” (2001),

   (b) Bu Brundin spelade i samma film som Brian Wimmer i ”Late for Dinner” (1991), och

   (c) Brian Wimmer spelade i samma film som Kevin Bacon i ”Footloose” (1984)

   Implementera ett ”orakel” för denna typ av aktivitet. Givet två sidor skall den kortaste möjliga vägen mellan dem skrivas ut. Observera att länkning i Kevin Bacon-index är dubbelriktad medan länkar i ett wikirace inte är det. En frivillig utökning är att implementera stöd för sökning efter den kortaste vägen mellan två sidor som går att nå från varandra.

6. Frivillig utökning.

   Implementera stöd för att rita ut en graf över webbplatsen där noderna är sidor och bågarna är länkar vars etiketter är länktexterna. Ett enkelt sätt är att generera en textfil som kan ges som input till programmet `dot` (`http://en.wikipedia.org/wiki/DOT_(graph_description_language)`).