Programmeringsuppgift: Ordkedjor

Jesper Larsson, v. 1.1 (oct. 2017)

Den här uppgiften går ut på att hitta vägar mellan ord med fem bokstäver. Det är ett grafproblem som ges i två varianter: en som kan lösas med bredden-först-sökning, och en som tänkt att lösas med Dijkstras algoritm.

Ordkopplingar

Det går att komma från ord x till ord y i ett steg om de fyra sista bokstäverna i x finns med i ord y. Man kan till exempel gå från ”väska” till ”säkra”, men inte tvärtom eftersom väska inte innehåller något *r*. Däremot kan man gå åt båda hållen mellan ”skola” och ”kloka”. Alla fyra bokstäverna måste bevaras, även om de är upprepningar, så man kan gå från ”salta” till ”altan” (som innehåller båda två *a*) men inte till ”kallt” (bara ett *a*).

Som exempel är här en lång kedja med engelska ord: climb → blimp → limps → pismo → moist → stoic → ioctl → colts → lotsa → stoae → oaten → neath → hated → dated → dater → rater → tread → dared → dread → drear → rarer → reran → arena → earns → snarf → franc → narco → orcas → scare → raced → decaf → fecal → eclat → talcs → clasp → psalm → slams → small → llama → lamas → amass → smash → shame → hames.

In- och utdata

Ett antal exempelfiler bifogas den här uppgiften. Det finns *data*-filer som bara består av fembokstavsord, ett per rad, och *test*-filer som innehåller ett par av order per rad. Sista avsnittet i detta dokument visar kodexempel för att läsa data- och testfiler. Det finns också *output*-filer som innehåller programmets korrekta utdata för testfallen.

Utdata ska bestå av avstånden mellan orden i testfilen, eller `-1` om det inte finns någon väg. Exempelvis så här för exempelfilerna som börjar med `words-14`:

```
1
1
-1
3
0
-1
2
```

Variant: viktade kopplingar

Som frivillig utökning kan man istället ge varje övergång en kostnad, och hitta längden på vägen med den lägsta kostnaden. Kostnaden för en övergång mellan två ord ska

räknas som ett plus avståndet i alfabetet mellan sista bokstäverna i respektive ord. Att gå från ”väska” till ”säkra” har exempelvis kostnad 1, eftersom orden slutar på samma bokstav, medan kostnaden för att gå från ”salta” till ”altan” är 14, eftersom det är 13 steg mellan *a* och *n* i alfabetet. Kostnaden för att gå från ord `x` till `y` (om det går, enligt kriterierna ovan) kan beräknas med `1+Math.abs(x.charAt(4)-y.charAt(4))`.

I denna variant blir korrekt utdata för exemplet `words-14` istället:

```
14
1
-1
15
0
-1
9
```

Krav

Det går bra att använda valfri programkod från boken eller från bokens webbplats, exempelvis för grafrepresentationen, men det är inget krav – det går också bra att specialutforma en grafrepresentation för uppgiften. Javas standardbibliotek går förstås också bra att använda. Vill ni använda andra källor för kod, diskutera med Jesper först.

Minimal lösning

Skriv ett program som bygger en riktad graf som representerar orden och övergångarna mellan dem, och kör BFS på den – eller Dijkstras algoritm för den viktade varianten. Se i synnerhet till att programmet fungerar korrekt för alla testfilerna. Utrymmet ska vara $O(V + E)$, där V och E är antalet noder och bågar i grafen.

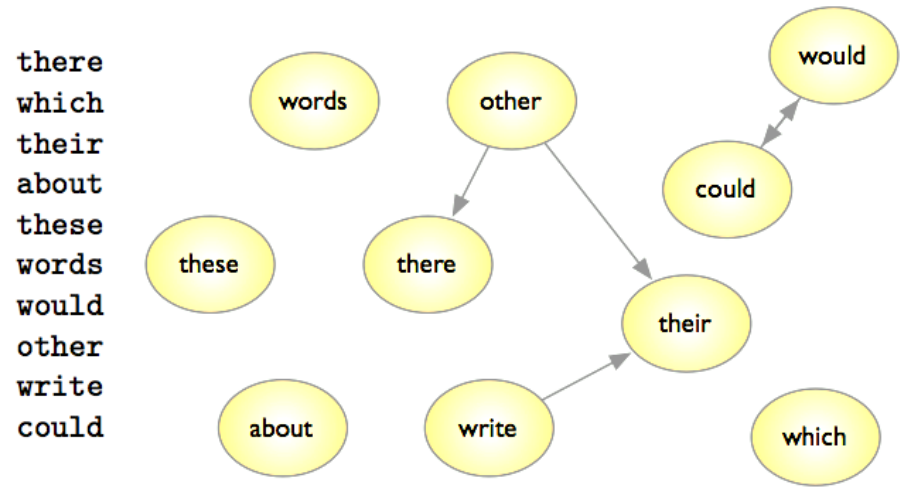
Lämna in programkod, plus en pdf-fil som beskriver lösningen (vilka algoritmer och eventuella källor för programkod ni använt, och annat som kan vara intressant) samt analyserar tids- och utrymmeskomplexitet tillräckligt noggrant för att kunna fungera som bevis. Det går bra med O-uttryck för att ange komplexiteterna. Det går också bra att hänvisa till kända komplexitetsuttryck för komponenter (algoritmer och datastrukturer) som beskrivs i boken. Upprepa alltså inte analyser som finns i boken, utan hänvisa till resultaten och sätt in dem i sammanhanget.

Bra lösning

Som den minimala lösningen, men se till att körtiden är $O(V + E)$ för BFS-varianten, eller $O(V + E \log V)$ för den viktade varianten. Använd alltså exempelvis inte kvadratisk tid för att bygga grafen.

Illustration

Nedanstående figur visar ett exempel på en datafil och motsvarande graf:



Inläsningskod

Här är ett kodexempel för att läsa in en datafil i Java:

```
BufferedReader r =
    new BufferedReader(new InputStreamReader(new FileInputStream(fnam)));
ArrayList<String> words = new ArrayList<String>();
while (true) {
    String word = r.readLine();
    if (word == null) { break; }
    assert word.length() == 5;  // indatakoll, om man kör med assertions på
    words.add(word);
}
```

Här är ett för att läsa och behandla testfall från en testfil:

```
BufferedReader r =
    new BufferedReader(new InputStreamReader(new FileInputStream(fnam)));
while (true) {
    String line = r.readLine();
    if (line == null) { break; }
    assert line.length() == 11; // indatakoll, om man kör med assertions på
    String start = line.substring(0, 5);
    String goal = line.substring(6, 11);
    // ... sök väg från start till goal här
}
```