Opgaver uge 4
Tirsdag 19. februar 2008

2008-02-19

Formål med opgaverne

Efter disse øvelser skal du kunne implementere algoritmer for orienterede grafer, for eksempel dybde-først gennemløb, bredde-først gennemløb, finde stærke sammenhængskomponenter, foretage topologisk sortering af en orienteret acyklisk graf (dag).

Implementationen af grafer i fra `krak-kode-faap-1.zip`

Opgave 4.1: Bredde-først gennemløb

Prøvekør metoden `breadthfirst` til bredde-først gennemløb fra forelæsningen på Goodrich & Tamassias eksempel side 314. Bogens eksempel 6.6 på side 304 kan oprettes som en uorienteret graf med metoden `gt6_6` i filen `Exercises4.java` fra kursets hjemmeside.

Tjek at bredde-først gennemløbets resultat er korrekt. Resultatet er næppe identisk med det i bogen, for kanterne er nok oprettet i en anden rækkefølge.

Opgave 4.2: Bredde-først gennemløb med niveauer

Lav en ny metode `breadthfirstlevel` ved at modifiere `breadthfirst` så den udskriver et niveau (eng. "level") sammen med hvert knudes nummer. Du kan for eksempel lave en hjælpeklasse `NodeLevel` som nedenfor, der knytter en `Node` og et niveau sammen:

```
static class NodeLevel {
  public final Node node;
  public final int level;
  public NodeLevel(Node node, int level) {
    this.node = node;
    this.level = level;
  }
}
```

Så skal køen indeholde `NodeLevel`-objekter i stedet for `Node`-objekter. Husk at startknuden har niveau 0 og at en knudes naboer har niveau 1 højere end knuden selv (hvis de ikke allerede er besøgt).

Tjek at de niveauer programmet udskriver er identiske med dem som bogens eksempel viser.

Opgave 4.3: Dybde-først gennemløb

Skriv en metode `depthfirst` som laver dybde-først gennemløb og udskriver knudernes numre i dybde-først rækkefølge. Lav metoden ved at modificere `breadthfirst` til at bruge en stak i stedet for en kø. (Lad være med at lave den efter Goodrich & Tamassias opskrift side 305).

Prøvekør det på Goodrich & Tamassias eksempel side 304 og tjek at resultatet er korrekt. Resultatet er næppe identisk med det i bogen, for kanterne er oprettet i en anden rækkefølge.

Opgave 4.4: Ind-grad

Skriv en metode i klassen `Graph` som for en given knude `node` returnerer antallet af kanter der går ind i knuden, dvs. dens indgrad eller *indegree*:

```
class Graph<E extends Edge<N>, N extends Node> {
  public int indegree(Node node) {
    ...
  }
}
```

Denne metode skal bruges i næste opgave. I denne opgave og den næste kan du antage at grafen kun indeholder orienterede kanter, så `edges.get(node.index)` indeholder alle udgående kanter, og kun dem, mens `reverse_edges.get(node.index)` indeholder alle indgående kanter, og kun dem.

Opgave 4.5: Topologisk sortering

Skriv en metode i klassen `Graph` som for den givne graf enten returnerer en arraylist med grafens knuder i topologisk orden, eller kaster exception `CyclicGraph`:

```
public static ArrayList<Node> toposort(Graph<Edge<Node>,Node> graph)
  throws CyclicGraphException
{
  ...
}
```

Exception-klassen `CyclicGraphException` kan se sådan ud:

```
class CyclicGraphException extends Exception {
  public CyclicGraphException(String message) {
    super(message);
  }
}
```

I denne opgave kan du antage at grafen kun indeholder orienterede kanter, så `edges.get(node.index)` indeholder alle udgående kanter, og kun dem, mens `reverse_edges.get(node.index)` indeholder alle indgående kanter, og kun dem.

Vink 1: Opgaven kan løses ved at implementere Goodrich & Tamassias algoritme TopologicalSort fra side 326.

Vink 2: Opret `incounter` som en `HashMap<N,Integer>` der først initialiseres ved at gennemløbe alle nodes og kalde `indegree(node)` fra opgaven ovenfor.

Tjek at du får et korrekt resultat på Goodrich & Tamassias eksempel 6.21 fra side 328. Denne orienterede graf opbygges af metoden `gt6_21` i filen `Exercises4.java` fra kursets hjemmeside.

Tilføj ekstra kanter til eksempelgrafen (ved at redigere metode `gt6_21`) så grafen bliver cyklisk, og tjek at den topologiske sortering kaster en exception i det tilfælde.