Algoritmiek, Graph Exercise

Mathijs de Weerdt

Delft University of Technology

Introduction For this exercise you first need to study Chapter 3 from *Algorithm Design.* Do that before you start working on this exercise. If you have studied and understood the material well, this exercise should take you about six to eight hours.

Learning objectives After working on this exercise you should be able to

- recognize a graph structure in a problem description,
- implement the data structure for a graph,
- produce an efficient algorithm to derive a simple graph property,
- analyze its running time, and
- understand and use graph related terms and their definitions.

Expected results For this exercise we expect two products from you: Java code that is **completely written by yourself** and an explanation of your answers in **your own words**. Make sure that the Java code

- works with JDK version 1.6,
- writes all results to the standard output stream in *exactly the format described in the exercise* (for example by using the `System.out.println()` method; don't use console input or graphical in/output, because your code will be screened automatically),
- meets the requirements for neat programming from earlier courses,
- accepts a *filename* as the first argument of the `main()` method, and uses this filename to read in the input data,
- has comments and explanations at the appropriate places, and
- is collected in a .jar file including both the source and the class files with the correct Main-Class attribute set.[1]

Your report on the way you implemented the exercise should meet the following constraints:

- It should be at most four sides of an A4 (excluding an optional front page, space used by diagrams, and an answer to the bonus question (if any)), and in PDF.
- It should be included in a `doc/` folder within the .jar file.
- It should be written in clear, readable, and correct Dutch or English. (Too many errors or unreadable sentences may lead to fewer points.)
- It should contain for each question a clear explanation about how you obtained the answer/code, including (if appropriate):
  - a reference to the material (book/slides), why this applies here, what you changed, and a reference to the relevant part of your code, and
  - a reference to the students with whom you conferred to obtain this solution.

Assessment and feedback We will check both your report as well as your code on

- correctness,
- clarity, and
- completeness with respect to the requirements above.

[1] If you use Eclipse, please use the export-to-jar function, or otherwise refer to the Java Tutorial `http://java.sun.com/docs/books/tutorial/deployment/jar/appman.html`.

Story telling

Together with a group of friends you are developing an adventure computer game. You are in charge of the story, while for example Alice and Bob are developing a number of scenes. However, together you have ambitiously decided not to have just one story line, but to let some scenes follow upon other scenes depending on a combination of user's actions and chance.

Task A (1 point)

As a first step, describe how you could model the possible sequences of scenes by a graph. What do the nodes and edges represent? Why did you chose an (un)directed graph? Given the number of nodes n, what is a tight upper bound on the number of edges m: how many of them can you get at most?

Task B (2 points)

To Alice and Bob it is important that every scene can be reached. Given a certain starting scene s, give an efficient algorithm in pseudocode that prints 1 if there is a possibility to get from s to every other scene, and 0 otherwise. Briefly explain in your own words why your algorithm is correct. Also give the tightest upper bound on the run time of your algorithm as a function of n and m, and show why this holds.

Task C (3 points)

Often it is undesirable to go through the same scene twice. Give the pseudocode of an algorithm that prints 1 if it is possible to find a sequence of at least two scenes that starts and ends with the same scene, and 0 otherwise. Briefly explain in your own words why your algorithm is correct. Also give the tightest upper bound on the run time of your algorithm as a function of n and m, and show why this holds.

Task D (3 points)

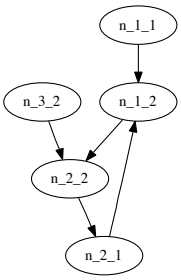
A *strong component* of a scene s is defined as the set of all scenes v such that s and v are mutually reachable. Give the pseudo-code of an efficient algorithm that prints the size of the largest strong component. Briefly explain in your own words why your algorithm is correct. Also give the tightest upper bound on the run time of your algorithm as a function of n and m, and show why this holds.

Task E (3 points)

One of the ways to measure the connectivity of a graph is the diameter. We define the *diameter* of a directed graph to be the length (in our case the number of edges) of the shortest path from one vertex u to another v *that are furthest away* from each other, excluding pairs of vertices u and v for which there is no path from u to v. Give the pseudocode of an efficient algorithm that prints the diameter of a given graph. Also give the tightest upper bound on the run time of your algorithm as a function of n and m, and show why this holds.

Task F (8 points)

Write a program that includes an implementation of the algorithms given in Tasks B and D and E. Your program should be able to read a graph of scenes from the input stream in the following format. First read "`digraph G {`" and then read each edge represented by "`<identifier> -> <identifier>;`". Finally, read a starting point followed by a newline. See the figure below for an example.




```
digraph G
{
n_1_1 -> n_1_2;
n_1_2 -> n_2_2;
n_2_1 -> n_1_2;
n_2_2 -> n_2_1;
n_3_2 -> n_2_2;
}
n_3_2
```

The output of your program should be exactly the following.

1. On the first line, print a 1 if from the starting scene all other scenes can be reached, or a 0 otherwise (Task B), followed by a newline.
2. Print a 1 if it is possible to find a sequence of at least two scenes that starts and ends with the same scene, or a 0 otherwise (Task C), followed by a newline.
3. Print the size of the largest strong component (Task D), followed by a newline.
4. Print the diameter of the given graph (as defined in Task E), followed by a newline.

Bonus Task (3 points)

For each scene in a strong component it is possible to find a sequence to each other scene in this set. In practice, this is a desired property for the complete set of scenes. Therefore, give the pseudo-code of an efficient algorithm that prints a minimal set of connections that can be added to be able to reach every scene from every other scene. Explain briefly in your own words why your algorithm is correct.