Assignment 2: Algorithm Analysis and Graphs

COSC 3020: Algorithms and Data Structures

Lars Kotthoff
larsko@uwyo.edu

Instructions

Solve the following tasks. You may work in teams of up to two people. For the theoretic part, submit a PDF with your solution, for the practical part, submit the Javascript file(s) to WyoCourses. The name and student ID of *all* partners must be clearly visible on each page of your PDF and in each source code file. Only one partner needs to submit. You have until Friday, 28 October 2022, 23:59h.

You may *not* use external libraries in your code unless explicitly stated.

I will test your code on Linux with `node.js`, checking whether your code works with empty inputs in addition to functionality. Please test it on the lab machines, where you have the same environment – if your code does not run, you may get no points for this part. If you submit a file other than a PDF or if your code is not in a separate Javascript file you may get no points for it. If you submit scanned or photographed handwritten notes, or hand-drawn graphs, I will take off 10%.

Acknowledge *all* sources you used – this includes help from the TA, other students, websites, etc.

1 Theory vs. Practice (9 points)

(a) List 3 reasons why asymptotic analysis may be misleading with respect to actual performance in practice. (3 points)

(b) Suppose finding a particular element in a binary search tree with 1,000 elements takes 5 seconds. Given what you know about the asymptotic complexity of search in a binary search tree, how long would you guess finding the same element in a search tree with 10,000 elements takes? Explain your reasoning. (3 points)

(c) You measure the time with 10,000 elements and it takes 100 seconds! List 3 reasons why this could be the case, given that reasoning with the asymptotic complexity suggests a different time. (3 points)

2 Graph Properties (9 points)

(a) Prove that if two graphs A and B do not have the same number of nodes, they cannot be isomorphic. (3 points)

(b) Prove that if two graphs A and B have the same number of nodes and are completely connected, they must be isomorphic. (3 points)

(c) Prove that if two graphs A and B are isomorphic they do *not* have to be completely connected. (3 points)

You need to give complete, formal proofs – state *all* your assumptions and make sure that you've explained every step of your reasoning.

Hint: A good way to start is by writing down the definitions for everything related to what you want to prove.

3 Ford-Fulkerson – Augmenting Paths (10 points)

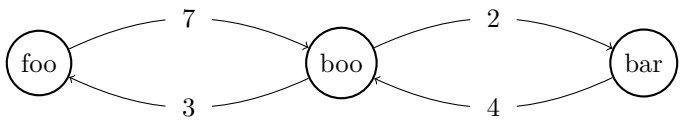
When we talked about the Ford-Fulkerson algorithm to find the maximum flow through a graph, I mentioned the "find an augmenting path" function. You're going to implement this function.

The prototype of the function must be the following:

```
function augmentingPath(graph, start, end);
```

where `graph` is the adjacency list of a directed weighted graph, `start` the name of the start node, and `end` the name of the end node (note that they are node names, not indices). The function returns a list of node names, starting with `start` and finishing with `end`. If there is no path from `start` to `end`, you must return an empty list.

To illustrate, here's an example graph:



This graph would be represented as follows to call `augmentingPath`:

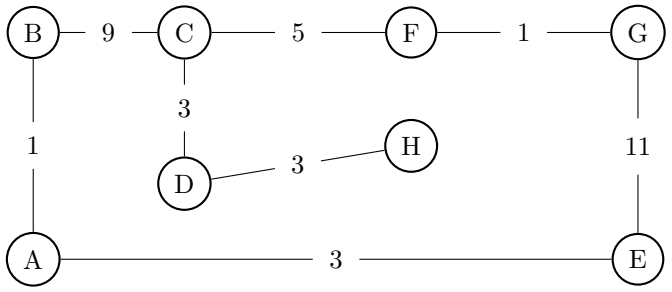
```
var graph = {'foo': {'boo': 7},
             'boo': {'foo': 3, 'bar': 2},
             'bar': {'boo': 4}};
augmentingPath(graph, 'foo', 'bar');
```

The call would return `['foo', 'boo', 'bar']`.

Test your code with a few different inputs, ideally using an automated testing framework. Make sure to include evidence of testing in your submission. To get full points, your code style and design must be good. This means in particular avoiding unnecessary steps and good encapsulation and abstraction.

Do *not* implement the Ford-Fulkerson algorithm, only the augmenting path function.

4 Kruskal's Algorithm (5 points)



Run Kruskal's algorithm on the above graph to produce a minimum spanning tree (MST). The weights are the numbers drawn on top of each edge. Draw the minimum spanning tree Kruskal's algorithm finds, indicate the order in which edges are added to it, and note the total weight of the MST. You may break ties arbitrarily.

Provide the output, not code that produces the output.