Code Challenges

General guidelines

Coding language is English, that is variables and documentation must be in English. All code must be documented. All code must be delivered executable. So please provide a command line call to invoke your solution and verify the result. Assume a *nix operating system. Code delivered without a command line call will *not* be considered a solution of the challenge. You must at least solve two of the three challenges and challenge two must be among them. Please solve the challenges yourself and provide only your original code.

Challenge One - Basic algorithms

Assume a directed acyclic graph (DAG) in which nodes can have multiple parents and there is a single unique root node. The *minimum* number of edges connecting any node *n* with the DAG's root node *r* is considered the depth *d*(*n*) of node *n*. Because a node can have many parents, there might be several routes, of varying number of edges, from the node to the root *r*. Provide a test (see Test Driven Development) that executes your solution and verifies, it works as expected. Then write the function that returns the depth of a node given the DAG and test it with your test. Assume the DAG, i.e. the input to your function, to be represented as an adjacency matrix. Another input argument is the root node, i.e. consider the root node to be known so that your algorithm does not need to find it. Use any programming language you like to solve this challenge. You will receive extra points if you implement the solution in either Brainfuck or Assembler, but because you will be considered not totally sane we will not invite you to dinner.

Code challenge Two - Web servers

Write a simple Server in NodeJS using the Express Framework. The server should respond to two routes "/foo" and "/bar". "foo" should write the response "Hello" and "bar" should write the String "World" to the response in JSON format, serializing the objects `{ response: "Hello" }` or `{ response: "World" }` respectively. Make sure the response is *exactly* as defined here. The important point in this exercise is to parametrize the server at startup depending on two environment variables "PORT" and "BASE_URL". "PORT" indicates on which port the express server will listen, and "BASE_URL" defines which will be the base path inserted before the two above routes "foo" and "bar". For example the shell command

```
export PORT="3000"
export BASE_URL="/conabio"
npm run express_challenge_two.js
```

Should start a server that responds to

`localhost:3000/conabio/foo`

with '{"response": "`Hello`"}', and to

`localhost:3000/conabio/bar`
with '{"response": "`World`"}'.
Consider that the *default values* for the above arguments should be `PORT=3333` and `BASE_URL=""` (empty string)
Your solution should include two Javascript files. The first "index.js" should create and start the Express server, the second "routes.js" shall use an Express Router (express.Router) in which the routes `'foo'` and `'bar'` are defined.

You will get extra points, if you deliver a Dockerfile to create an appropriate image for this express application. The Dockerfile must set the above environment variables to control PORT and BASE_URL.

Challenge Three - Containerization and cloudification

Use Docker and docker-compose to set up three containers:

- *Database*
- *Server A*
- *Server B*

The database container shall host a relational database, e.g. MySQL, Postgres, SQLite, etc., and reside inside the same network as *Server A*. Have a default database installed and running on the database server. *Server B* shall have no access to the database, but be able to connect to *Server A*. On startup *Server A* executes a simple shell-script that checks whether it can connect to the database and logs that to standard-out. *Server B* executes a similar script on startup and checks, e.g. using ping or nc, whether *Server A* is available, and logs the result to its standard-out. Consider using simple Debian or Alpine Linux images for *A* and *B*, and a standard relational database docker image for the database server. Write a simple Shell-Script that starts the network as described and prints out the respective outputs of the startup scripts.

You will receive extra points if you make *Server B* respond to requests from the host on port 3000, i.e. 'localhost:3000', with some simple greeting, e.g. "Hello World" (for inspiration see the Docker image nginxdemos/hello).

Enjoy your code challenges and thank you very much for your interest, time, and effort!