Department of Computer Science

COS284 Pratical and Assignment 6: Data structures

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Contents

1 Introduction

This document contains the specification for assignment 6. In general the assignments will build upon the work of the current practical.

1.1 Submission

You have until Wednesday the 2nd of November at 17:00 to complete the practical component. You have until Friday the 11th of November at 17:00 to complete the assignment component. **No late submissions will be accepted.** The upload slots for the practical and assignment will become available on the Tuesday the 25th of October at 11:00.

1.2 Plagiarism policy

It is in your own interest that you, at all times, act responsibly and ethically. As with any work done for the purpose of your university degree, remember that the University of Pretoria will not tolerate plagiarism. Do not copy a friend's work or allow a friend to copy yours. Doing so constitutes plagiarism, and apart from not gaining the experience intended, you may face disciplinary action as a result.

For more on the University of Pretoria's plagiarism policy, you may visit the following webpage: `http://www.library.up.ac.za/plagiarism/index.htm`

1.3 Practical component - 50 Marks [50%]

The first part is to construct a graph that consists of `Node` structs. The given main.c has the declaration of this struct. To construct the graph, you will have to implement two functions in assembly: `allocateNode` and `linkNodes`.

The purpose of `allocateNode` is to allocate space on the heap for a node, and initialise this node. The id of the node should be set to the string which is passed as the first parameter. All other members of the struct should be set to 0.

The purpose of `linkNodes` is to link two nodes to each other and assign a distance to this link. This link information is stored in the `links` array member of the node. This array consists itself of structs that store a pointer to the other node along with the distance of that link.

The expected output for the example main function should be:

```
Node B 5.4
Node C 2.1
Node D 3.3

Node A 5.4
Node C 1.2

Node A 2.1
Node B 1.2

Node A 3.3
```

The order of elements in the links array do not matter, Fitchfork will give you the marks as long as all of the links are present.

One further note on marks, Fitchfork will test if you dynamically grow your link arrays and it will test for memory leaks.

1.4 Assignment component - 50 Marks [50%]

The next step is to calculate the best route between each possible combinations of nodes and populate the `routes` array with this information. This requires a bit of a complicated process. You have to implement a function called `calculateRoutes`. This function will recieve a pointer to one of the nodes in the graph.

The first step is to make a list of all the nodes in the graph. So you will traverse recursively through the graph and add each node you visit to a list. Be careful of an infinite loop here. Once you have this list, you can work with the nodes easily.

The next step is to have each node add routes for its direct neighbours. It is also a good idea to have each node add a route to itself with a distance of 0 (This will simplify the next step).

Now all the nodes have a route to A, but longer routes like B to D still needs to be calculated. To do this, for each combination of nodes X and Y, have X ask all its neighbours if

they have a route to Y. If a neighbour does, then that route can be added to X's routes. Of course, we are only interested in the shortest routes. So if two neighbours have routes to Y, X should only add the shortest one.

After the previous step we will have routes between all the nodes in the example, but the route B-A-D is not the shortest route between B and D. To fix this, we simply have to repeat the previous step, since C has in the mean time gained a route to D. In fact, the previous step should be repeated as many times as there are nodes.

The expected output for the example main function should be:

```
Node A Node A 0.0
Node B Node C 3.3
Node C Node C 2.1
Node D Node D 3.3

Node A Node C 3.3
Node A Node A 0.0
Node C Node C 1.2
Node D Node C 6.6

Node A Node A 2.1
Node B Node B 1.2
Node C Node C 0.0
Node D Node A 5.4

Node A Node A 3.3
Node B Node A 6.6
Node C Node A 5.4
Node D Node D 0.0
```

The order of elements in the routes array do not matter, Fitchfork will give you the marks as long as all of the links are present.

Fitchfork will test if you dynamically grow your link arrays and it will test for memory leaks.

2 Mark Distribution

Activity	**Mark**
Part 1	50
Part 2	50
Total	**50**