06-26945 & 06-26944
Distributed and Parallel Computing + Extended
Spring Semester 2019

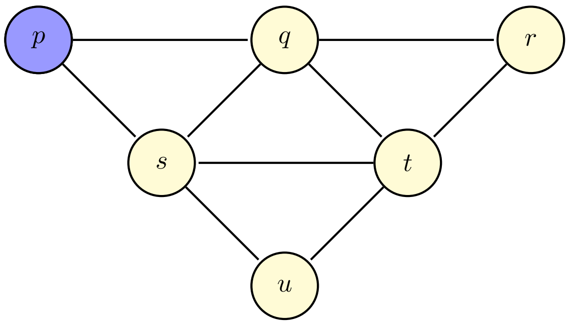
The University of Birmingham
School of Computer Science
© Alan P. Sexton 2019

Assignment 02

Write an *Erlang* program to execute the Tarry distributed algorithm as taught in this module.

The algorithm should read, from standard input, the topology and connectivity of the network and the name of the initiator of the algorithm.

The text file contains the name of the initiator on the first line of the file, then, on each following lines the name of one of the nodes followed by the list of nodes that that node can send messages to. For example, consider the distributed system topology on slide 5 of the `12-distrib-wave-traversal-tarry.pdf` lecture slide set:



In this case, assuming that `p` is the initiator, the input text file would contain:

```
p
p q s
q p s t r
r q t
s p q t u
t q r u s
u s t
```

Note that node names can be any string of letters and digits and the names on a line are separated by space characters.

Your program `MUST NOT` be interactive: it must not output any questions or prompts and expect corresponding input. Nor should it require **ANY** arguments. Instead it should simply read the lines of the input file from standard input (**NOT** from a named file) and output the results to standard output. The reason for this is so that your submission can be executed in a batch script on several different distributed system topologies for marking purposes.

Example Erlang code for reading from a file and parsing the line into space-separated tokens can be found at
`http://learnyousomeerlang.com/functionally-solving-problems#heathrow-to-london`
Please note that you do not need to read and understand the whole chapter, the appropriate part to read can be found by searching on that page for "`file module`". Unlike in that example, I recommend using `io:get_line("")` (i.e. with a prompt of an empty string), to read each line of the standard input.

The only output allowed from your program should be from the main process. The actual algorithm to implement and the output required is as follows:

The distributed algorithm to implement is the *Tarry* algorithm. The program should have the main process spawn off all the nodes, give them the process ids of the nodes that they have channels to (according to the input described above) and then tell the initiator to start the algorithm.

The token passed around the network should collect the names of all processes it passes through in a list and, on return to the initiator, the initiator should report that list back to the main process, which should print out the list to standard output.

IMPORTANT: do not remove duplicates from this list: the list should show, in order, the sequence of nodes that the token passes through in the order that they were passed through. Since nodes are visited more than once, those node names should appear on the list more than once.

No processes other than the main process should write anything out (please remove or comment out any debug print statements before submitting your program).

One possible output for the above input example is

```
p q r t q s p s u t u s t s q t r q p
```

0.1 Extended Version

For students on the extended version of the module, the algorithm implemented should be the second depth-first version, as described on the last slide of the `distrib-03` set of slides, where the information in the token is used to avoid sending the token along frond edges. That is, **NOT** the version where the token can be sent along frond edges but is immediately sent back along the same edge on finding out that the node it was sent to was already visited by the token. Instead you should implement the version where you use the information stored in the token detailing which nodes have already been visited and only send it to nodes that have not been visited yet, or, if no such node exists, send it back to the node that the token was initially received from. I recommend getting the basic algorithm for `Tarry` working before adapting it to the depth-first version.

One depth-first possible output for the above input example is

```
p q s t r t u t s q p
```

Please note. Students on the non-extended version of the module are allowed to join into groups with students from the extended version of the module. However, any group with even one student from the extended version of the module MUST submit the extended version of this assignment, and all students in the group will be marked on this extended version.

1 Teams

You will work in teams of 3 or 4 students. You should already have self-selected your teams on Canvas. There is a group set called "A2 Groups" for this purpose. Any students not in a team will be added to existing teams. If you choose not to participate in the assignment (and therefore earn zero marks for it!) please let me know so that I can avoid assigning you to a team that you will not contribute to. You can do this by adding yourself to the special "Not Participating" group or by emailing me.

2 Erlang Version

The program should work on Linux computers in the School lab and must not require any proprietary libraries: you can use any modules from Erlang's standard libraries.
Note that the erlang version (which is different from the "erl" shell version) can be found with the command:

```
erl -eval 'erlang:display(erlang:system_info(otp_release)), halt().'  -noshell
```

In the school labs, this is "R16B03-1". While newer versions of erlang are mostly backward compatible with this version, there are new library functions that will NOT work in the older versions so ensure you only use library calls that work in version "R16B03-1". The library documentation for this version can be found at: `https://erldocs.com/r16b03-1/`

3 Marks

This assignment will be marked out of 10 and contributes 10% of the total module marks both for the extended and the non-extended versions of the module.
Marks will be assigned as follows.

- 6 marks for correctness of the code
- 2 marks for clarity and simplicity of the code
- 2 discretionary marks (imagination, originality, clever ideas, insight, etc.)

4 Submission

Your submission should be in the form of a single zip file which should include:

- A short report (.txt or .pdf — NOT word) containing:
  a. The team name and the names and student ids of all students in the team
  b. The Linux commands to compile and run your program
  c. A text copy of the terminal output when run on the sample input file described above
  d. A short description of how far you got. For example:
     - "Assignment fully completed and correct"
     - "Basic algorithm working but couldn't get results printing from main process"
     - "Input and setup of topology complete but couldn't fix a bug in the algorithm"
     - (for extended students only:) "Fully complete and correct except couldn't get depth first version working"
     - etc.
  e. A note about anything you consider especially good, original or unique about your solution that might be worth considering for extra discretionary marks.
- Your source code (sources only: no compiled files, .beam files, binary files, compiled libraries etc.).