QUT

# *CAB203 Project*

## 1 Introduction

In this project will demonstrate your knowledge and skill with the material in the unit by developing a solution to a real world problem by translating it to mathematical language, relating the problem to well known problems on mathematical structures, and implementing software to solve the problem.

At the end of this brief there are three topics, each describing a different problem. You will choose any *one* of these problems to solve. Your job will be to use ideas from the unit to describe the problem and a solution mathematically in a report and to implement your solution in Python.

Your submission will consist of two parts: a report detailing the mathematical descriptions of the problem and solutions, and a Python file containing your implementation of the solution.

This assignment is worth 40% of your final grade.

**Due date: Wednesday, 25 May 2022, 11:59:00pm**

## 2 Report

Your report must include the following sections:

1. **Introduction:** Using *your own words*, describe the problem in plain language. This should include enough information from the topic description above to understand the rest of your report, but can exclude specific examples, or other information that is not relevant.
2. **Problem:** Use mathematical language, concepts and notation from the unit to describe the problem. You should make references to mathematical problems discussed in the unit, for example finding a shortest path in a graph. Describe what an instance of the problem is and how it relates to these mathematical problems.
3. **Solution:** Describe, using mathematical terms and notation, how to find a solution for a given instance of the problem. Make reference to solution methods discussed in the unit, for example breadth first traversal. Describe how the solution to the mathematical problem relates to the solution to the original problem.
4. **Implementation:** Describe your implementation in Python. Be sure to mention any of these that apply: rationale for choices of data structures, necessary transformations of data, how you use functions from external libraries like `graphs.py` or `digraphs.py`. Please note that this section should be about programming considerations, not on the algorithm that you choose (which belongs in the Solution section.)

Overall, report should be understandable by another student in CAB203 who knows the material but hasn't thought about your specific topic. It is not necessary to define terms already used in the unit, but you should point out the significance of particular details about the problem and the choices that you make in modelling and solving it.

Your report will be a single file, in PDF format. There is no maximum page length, but a concise, easy to understand report is better than a long wordy report. Three to four pages is about right.

### 2.1 Python implementation

Your solution should be a reasonable implementation of the mathematical solution described in your report. The problems are all solvable using the Python concepts and syntax used in the unit, although you may need to search for some specific Python functions like reading CSV files or reading the characters from a string.

You are allowed to use or modify any functions defined throughout the lectures, tutorials, and assignment solutions. Many of these functions are collected in Python files available on Blackboard on the Project page. You can import these files rather than copying from them. You can assume that these are available; there is no need to include them in your solution. A submission template file is available from Blackboard Project page for each topic. If your solution includes modified code from the unit, say so in a comment explaining where you obtained it and what modifications you made. One line is enough detail.

Each problem defines the formats for any input or output required. You are required to follow these precisely as your code will be marked automatically. The marking system will run a single function, specified in the topic. The included test harness for each topic simulates the marking system for you. The test harness tries a single instance only; the marking system will have multiple instances. You are responsible for developing your own test cases, for which you are free to modify the test harness.

The topics are all chosen so that they can be solved with a relatively short program. There is no limit on the length of your program. However, the marking system will impose a time limit of about 20 seconds to avoid problems with infinite loops. This should be plenty of time to solve the problems given.

Your code submission will be a single file. The file name is specified in each topic below.

## 3 Marking criteria

### 3.1 Report

- The introduction gives a good overview of the problem and is sufficient to understand the entirety of the following sections assuming knowledge of CAB203 material. (4 marks)

- The Problem section gives a rigorous mathematical description of the problem stated using mathematical terms and notation from the unit. The original problem is clearly restated in terms of a graph theoretic problem described in the unit. The relationship between the an instance of the original problem and the graph theoretic problem is given. Where relevant, choices for how the problem is represented are defended. (6 marks)

- The Solution section gives a correct and appropriate mathematical description of a solution to the problem. The solution is described using mathematical terms and notation from the unit. The choice of solution method is appropriate and well defended. The relationship between the solution to the mathematical problem and the original problem is clearly described. (6 marks)

- The overall presentation of the report is clear, including appropriate sections, with minimal grammatical or stylistic problems. Appropriate citations are given where relevant. Any equations are typeset and diagrams are created digitally. (4 marks)

### 3.2 Implementation

- The automated marking system will run your solution against a variety of test cases for your particular topic. You will receive one mark for each passed test case. Please note that these test cases will not be made available, other than the case given with the test harness for each topic. (15 marks)

- Your code is a reasonable implementation of the solution described in your report. The implementation section gives a good overview of your code and defends choices of data structures etc.. The reader should be able to understand why you chose this particular way of implementing your solution. (5 marks)

**Total marks: 40**

## 4 Submission

***Sumbission process:*** You will need to make two submissions:

- Your report, via Turnitin, in PDF format
- Your Python code, via Blackboard assignment, as plaintext Python source code. Please name your file according the specification in your chosen topic.

Links will be available on Blackboard on the Project page. You can make as many submissions as you like up to the due date, but only your most recent submission will be saved.

***Extensions:*** Extension requests are processed by student services (`science@qut.edu.au`). There is a link on Blackboard on the Project page where you can apply for an extension. If you make a submission before the due date then after the due date further submissions will no longer be accepted by Turnitin. In this case you will need to email the unit coordinator (`matthew.mckague@qut.edu.au`) to delete your previous submission so that you can make a late submission.

48-hour extensions are available. Because of this, the due dates for Turnitin and Blackboard will be set to 48 hours after the set due date to allow you to submit late if you have an extension.

***Citing your sources:*** You are welcome to source information and code from the internet or other sources. However, to avoid committing academic misconduct, you must cite any sources that you use.

You are welcome to use resources, including code, from within the unit. Please cite the unit like *CAB203, Tutorial 7* or similar. This is only necessary when explicitly quoting unit material. There is no need, for example, to cite the definition of a graph or similar, unless you are directly quoting the lecture's definition.

For code, please include your citation as a comment within the code. For example

```
# modified from CAB203 graphs.py
```

***Policy on collaboration:*** We encourage you to learn from your peers. However, for assessment you need to turn in your own work, and you learn best if you have spent some time thinking about the problem for yourself before talking with others. For this reason, talking with other students about the project is encouraged, as long as you are putting in the effort on the problems yourself as well. But do not share your code or your report with other students and do not copy from others. It is considered academic misconduct to copy from other students or to provide your work to others for the purposes of copying.

For Slack and other online discussions, please do not post about solutions. Keep your discussions private so that everyone gets a chance to get to the solutions on their own.

# 5 Topics

Choose *one* of the following topics for your project. If you are unsure which one to choose, we suggest Widgets.

## 5.1 Pegs

There are a variety of puzzles that involve pegs in a board with a pattern of holes. We will explore one such puzzle here.

The puzzle board is a line of regularly spaced holes. The number of holes can vary. The puzzle starts with some of the holes occupied by pegs, and some holes empty. The player proceeds by a series of jumps. In a jump, a peg moves over an adjacent peg into an empty hole. The peg which was jumped over is removed. The goal of the puzzle is to find a sequence of jumps so that the board ends up with a single peg and all other holes empty.

For this project, the game board's starting position is given as a Python string like so:

```
XoXoooXXoo
```

where `X` indicates a peg and `o` indicates an empty hole. An example of a valid jump would take this:

```
XooXX
```

to this:

```
XoXoo
```

where the rightmost peg has jumped to the left, removing the second from last peg.

Your goal is to write a Python function `pegsSolution(gameBoard)` which returns a sequence of jumps that results in a board with a single peg. The sequence of jumps should be a Python list like so:

```
[ (3, 'L'), (5, 'R'), (4, 'L') ]
```

where each item in the list is a pair indicating the position of the peg which is jumping (counting from 0 on the left of the board) and the direction (either L or R for left or right). If there is no sequence of jumps which wins the game, then your function should return `None`.

Promises:

- The input string will consist of a single line of the characters `X` and `o` only.
- The input string will be at least three characters long.

A template file for your solution is available on Blackboard. The template includes a test harness and a simple test case. Your submission should be only your solution in a file named `pegs.py`.

## 5.2 Text adventure game

Back in Ye Olde Days text adventure games were a popular genre of computer games. These games had an interface that is reminiscent of command line interfaces. During a project in your computer archaeology unit, you have discovered an interesting example of a text adventure game, called *Bork*. Bork seems to take place in a place with strange multi-dimensional space-time. The character sometimes has options to move north or west, but also sometimes to move future or past, or strange directions like noruture (north-future?) or wast (west-past?). Moving around is made especially difficult by the fact that it is often impossible to go back directly to a previous location. However, the documentation that came with Bork indicates that it is always *possible* to travel back to the starting location (and hence from any location to any other location), but it might involve a very different route than on the way out.

Your attempts to map out the locations in the game have so far been thwarted by the complexity of the space-time. You decide to automate the process. Fortunately, someone else has written a Python interface for Bork that allows you to move the character around programmatically. From the interface you can identify your current location (through the description of the location, which is unique to that location), get a list of exits from the current location, or move to a new location by choosing an exit. Now you need to decide how to map out the game.

The Python interface to the Bork automator is a Python object `bork` which has the following methods:

- `bork.restart()` restarts the game with the character in the starting location.

- `bork.description()` returns a string giving a description of the character's current location (which is unique)
- `bork.exits()` returns a Python set containing the names of the exits from the character's current location as strings
- `bork.move(exit)` moves the character by taking exit `exit` to a new location
- `bork.save()` saves the game and returns the save game data (treat this as opaque, you can't interpret or modify it in a meaningful way)
- `bork.restore(savegame)` restore a saved game `savegame` returned by a call to `bork.save()`

Later, to make use of this information in further automated processes, you need to have the final map in a useful data structure. You have decided on a structure like so:

```
exitsMap = {
   'Home':  { 'North': 'Town', 'Noruture':'Swamp' },
   'Town':  { 'South': 'Home' },
   'Swamp': { 'Pwest': 'Town' }
}
```

Your submission should be a Python function `traverseBork(bork)` which takes an instance of the Bork automator `bork` and returns a map as a Python dictionary similar to that shown above. A template Python file, including test harness, and code for a mock Bork automator can be found on Blackboard on the Project page. Do not include the Bork automator file in your submission, include only your solution, named `bork.py`.

Promises:

- It is always possible to get back to the starting location from any point in the game.
- The functions available for the Bork automator will work as described above.
- The map does not change over time. Unlike many games, there are no secret exits to unlock.

Please note that your solution should not depend on specifics of the Bork automator, only on the functions mentioned above. The interface will stay the same, but the Bork automator may change implementation details.

For the extra adventurous, Bork has a hard-core mode in which the save and restore functions are disabled. To play this version, define a function `traverseBorkHardCore(bork)`. The test harness in the template will run this function with the Bork automator in hard-core mode. If your function is successful in hard-core mode then you will earn a 1% bonus mark. Be aware that this requires a decidedly more subtle approach! You should not attempt this unless you have already completed the project in normal mode. Full marks are still possible without attempting hard-core mode.

## 5.3 Widget factory

You own a factory that makes widgets from PLP (poly-lactic-PLP). There are several known processes for making widgets and you own machines that process the steps for several different processes. For example,

the Telza process makes widgets from gadgets while gadgets can be made either directly from PLP (using the Fredison process) or from thingamabobs (via the Mosk process). You have several machines, each of which implements a different process and has a different capacity. You have only one machine for each process. Conveniently, one widget requires 1KG of PLP regardless of the process, so to keep everything consistent the capacity of each machine is measured in terms of the weight of product it produces in one hour.

You have collected together the specifications for your machines in a CSV file, formatted like so:

```
Machine,Process,Input,Output,Capacity
Telza-3000,Telza,gadget,widget,5
Gadgetron,Fredison,PLP,gadget,3
Thingamabob-X,Mosk,thingamabob,gadget,3
```

Note that Python has a standard library for processing CSV files and there is documentation widely available on the internet.

Each machine can be set to produce its product at any rate between 0 and its capacity, but to keep things simple we'll restrict the rates to integers. Your goal is to determine how each machine should be set to maximise the number of widgets produced per hour without any machine producing excess products. Since you aspire to manufacture other products as well, your program should not make any assumptions and instead determine everything from the CSV file. For example, the feedstock may be ABS or something else rather than PLP and the final product may not be widgets. The feedstock and final product should be deduced from information given in the CSV file.

The results of your optimisation will be fed directly into the control software which is written in Python and accepts inputs as a dictionary relating machine names to rates, like so:

```
1 { "Telza-3000": 2, "Gadgetron": 1, "Thingamabob-X": 3 }
```

To integrate with the control software, your code needs to include a function `optimiseWidgets(filename)` which takes a filename for the CSV input file and returns a dictionary like the one above. A sample CSV file and solution template including a test harness are available from Blackboard on the Project page.

Promises:

- There is only one feedstock (eg. PLP), which can be deduced from the CSV file.
- There is only one final product (eg. widgets), which can be deduced from the CSV file.
- Machines are not reversible. For example, there is a machine that makes widgets from gadgets, but there is no single machine that makes gadgets from widgets.
- The capacities are positive integers.
- Machine and process names are unique.

Your code submission should include the Python file `widget.py` *only*. Do not include the sample CSV file in your submission.