CAB203 Project

This assignment is intended to give you opportunities to work with the ideas developed in the lectures. To this end, you are encouraged to discuss questions, approaches, and background material with your fellow students. However, *every student must prepare and submit their own solutions. The solutions you submit must be your own.* For some questions you will benefit from reading or online research. If you use sources outside of the unit you are expected to properly cite your sources. You can use any citation style you prefer, as long as you are consistent with it.

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.

In the next section there are four problem descriptions. You will choose any *one* of these problems to solve. These are given in every-day language. You can imagine that these might have come from someone who has not studied mathematics — they can describe what they want, but lack a rigorous way of describing the context and the problem. Your job will be to provide a rigorous description by using the ideas and notation developed in this unit. Your will then make use of the ideas from this unit to develop a description of how the problem can be solved. Finally, you will translate that description into Python code that implements your solution.

All of the problems can be considered to be a group of similar problem instances. An instance from the problem set needs to be specified in some way. For example, solving a Rubick's cube is the problem, and solving a specific Rubick's cube is an instance of that problem. So, given a Rubick's cube, you will need to devise a way of encoding the particular arrangement of the coloured squares on cube into a mathematical object (e.g. number, set, tuple, relation, function, etc. or some combination of these). There may be many ways of encoding the problem. Some may be easier to work with than others, so it pays to think about this a bit.

Similarly, the solution to the problem, which might be as easy as a number, or something more complex like a path on a graph, or a finite state automaton, needs to be encoded in some way. Finally, you will need a way of relating a given problem to the solution, again using mathematical notation. How do recognise a solution to a particular instance? How do you find such a solution?

Given the mathematical description, you will need to translate all of the parts into Python code. Your encoding of the problem needs to be implemented as Python data structures: sets, lists, dictionaries, functions, etc.. Your method of finding the solution needs to become code that processes the problem and outputs the solution. The solution, naturally, will be in some Python data structures corresponding to your mathematical encoding of the solution.

Finally, you need to print out the solution in a way that might make sense to the person posing the original question. This must be in a format that is easy to read for someone who is not familiar with the mathematical concepts from this unit, or the way that Python prints out its data structures. In some cases this will be straightforward, but in other cases there will be hints in the question as to what sort of format is suitable.

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

This assignment is worth 50% of your final grade.

Due date: Friday, 4 June 2021, 11:59:00pm

2 Topics

Choose *one* of the following topics for your project. Sample instances of each of problem are available in Section 6.

2.1 2×2 Rubick's cube

The famous Rubick's cube has a smaller cousin, the 2×2 Rubick's cube, for which each face has four squares rather than the usual nine. The squares are coloured red, blue, yellow, white, green or orange. The cube can be manipulated by performing a *move* which is to twist the one of the faces of the cube by 90°, either clockwise or anti-clockwise. The picture below shows a cube part way through a move, in this case twisting the top face.

The typical game played with such a cube is to scramble it by performing moves randomly for some time to arrive at a starting point. The goal then is to perform moves so that the cube is *solved*, meaning that each face's squares all have the same colour.

For your project, your goal will be: given a scrambled cube, determine the *smallest number* of moves required to solve the cube. Note that you don't have to output the sequence of moves, just the number required.

Note that there are a variety of "algorithms" available online for solving 2×2 cubes that are intended to be easy for people to get to a solved state, but typically do not do so by the smallest number of steps. These approaches are also much more complicated than necessary for a computer since they need to work around a human's inability to remember complex sequences or to reason several steps into the future. They also

won't solve the problem as they will typically use more than the minimal number of steps. Use a strategy discussed in the unit.

As a hint, there is a bit of a complication here, in that you can look at the cube from various points of view, and your description of it might change because of it. For example, maybe the top is all red and the bottom is blue, but turn the cube over and the blue face is on top and red is on the bottom. The easiest way to deal with this complication is to ignore it. As long as you can recognise legal moves, and when the cube is solved, that is sufficient.

2.2 Assembly line

A new company is building a factory for fabricating widgets. Widgets are rather complex, containing many parts such as gadgets and doodads. The company is planning to use an assembly line. The widgets move along a conveyor line with robots along the side. Each robot picks up a partially completed widget, adds a particular part, and sets it back on the conveyor belt where the widget will move along to the next robot, which will add another part, and so on. The plan is to have one assembly line that does everything, with finished parts coming out the end.

The engineers are having trouble figuring out where to put all the robots. "The doodad attaches to the gadget," says the lead engineer, "so we need to have the robots placed so that the gadget is already there before the doodad goes on. But we also need to make sure that the whatsits and the gadgets are there before the whatchamacallits because the whatchamacallit connects to the gadget and whatsit. Well, you get the idea." The engineer hands you list of parts along with which other parts each part attaches to.

But there's more! "The other problem is that the doohickey gets in the way of the T800 robot that attaches the thingamabob, so if the doohickey is already attached, the T800 can't attach the thingamabob." The engineer hands you *another* list of robots and which parts block it.

Help out the engineer by figuring out a way of arranging the robots along the conveyor belt so that the widgets can be assembled correctly, or determining that it is not possible to do so.

2.3 Family tree

Bob lives in an isolated community on board a generation ship — a spaceship travelling to a distant star over several lifetimes. Naturally over the generations the community has come a point where everybody is related to everybody. Bob has recently become interested in genealogy and has constructed a complete family tree for the community going back to the first generation, using the ship's meticulous records. His tree records the parents and children of every person, from which it is possible to work out how any two people on the ship are related.

Since then Bob has become obsessed with understanding how he is related to the other people on the ship, and these relationships are quite complicated! For example, Alice is not only his second cousin (his mother's father's mother's son's daughter's daughter) but also his first cousin once removed (his father's mother's mother's son's daughter), in additional to numerous other relationships.

Bob has decided to simplify his understanding of these relationships by adding annotations to his family tree by marking the parent-child relationships that link people most directly to him. Here he considers a

parent-child relationship (which we'll call a *step*) to be direct, and other relationships are measured by how many parent-child relations it takes to get from a person back to Bob. For example, Alice's relationship to Bob takes 5 steps via Alice's father, and 6 steps via her mother, so — assuming there are no more direct relationships — Bob's annotated tree will mark the relationship between Alice and her father, and there will be no mark on the relationship between Alice and her mother. This way Bob can quickly see the most direct relationship that he has with any person.

Bob realises that there may be more than one relationship between himself and another person that are equally direct, in which case he is happy with any one of them. Further, Bob has realised that his definition of relationship includes some cases that others might not consider as being a family relationship, particularly through half-siblings. However, Bob has decided to stick with his definition. Basically, just don't worry about these cases.

Help Bob to create his simplified family tree: given the complete family tree, find a simplified tree where there is a unique way of relating Bob to each other person, and that relationship is most direct among all possible relationships in the complete family tree.

Bob's family tree is in the form of *cards*. Each card lists a person's name, father, mother, and children (plus other information such as date of birth that we can ignore here.) Bob would like to have new cards that show the same information, but have a * in front of the names of people where the relationship needs to be marked. For example, on Alice's card, Alice's father's name will have a * in front, and on Alice's Father's card there will be a * in front of Alice's name.

2.4 Complicated wrist watch

Your cousin, a collector of old watches, has come to you with a rather complex digital watch that has multiple functions: timer, alarm, stopwatch, multiple time zones, etc.. The watch came with a users' manual, which unfortunately is not very intuitive. For example, instead of giving instructions for common tasks like setting the alarm, it says things like "If you are in Alarm mode, long press the Set button to enter Alarm Setting mode. If you are in Timer mode, press the Mode button to enter Stopwatch mode." Your cousin, after studying the manual for some time, says "I just want to go to the Date setting mode so that I can set the date, but I can't figure it out! At least it goes back to the Time display mode automatically after a bit, so I if I mess it up I can at least see the time."

Help your cousin to set the date by finding a sequence of buttons presses that will put the watch into Date setting mode, starting from the Time display mode. Your solution should generalise: given a similar user manual for a device with modes and buttons, a starting mode and a desired mode, find a sequence of button presses that takes the device from the starting mode to the desired mode.

3 Report

Your report must include the following sections:

1. **Introduction:** Using *your own words*, describe, in plain language (avoid mathematical terms) the problem. 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 are not relevant (eg. the fact that Bob is on a spaceship).

2. **Instance model:** Describe a mathematical framework for encoding a specific instance of the problem. Link it back to the original problem by using words from your introduction. This should be sufficient so that someone can understand how to construct the mathematical objects required given a plain language description. For example, use sentences like: "Given two cities A and B (say, Brisbane and Sydney) we include edge (A, B) whenever there is a direct flight between A and B."

3. **Solution model:** Similar to the instance model, describe a mathematical framework for encoding a specific *solution* to an instance of a problem. Again, link it to the plain language description in the introduction so that the reader can understand how to interpret the solution back in the original context. For example, use sentences like "The solution will be a sequence of vertices $v_1, v_2, \ldots, v_n$, where each v_j is a city, and the cities should be visited in the order listed." In some cases the solution model may be extremely simple (eg. a single number), in which case this section will be short!

4. **Problem model:** Describe the relationship between the problem instance and its solution (if it exists). Use mathematical terms and notation from the unit. For example, use sentences like "The solution is a shortest path between A and B in the instance graph G."

5. **Solution method:** Describe, using mathematical terms and notation, how to find a solution for a given instance. In most cases you will utilise methods described in the lectures or tutorials, in which case use notation consistent with the notation used in the unit material, but describe it using your own words. For example, if using breadth first search, use the notation D_j and V_j as in the slides, but use your own words to describe how it the method works.

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.

3.1 Python implementation

The problems are constructed so that a complete implementation of a solution is possible within about 20 lines of Python code (not including test data or helper functions from the unit). You are allowed to use any functions defined throughout the lectures, tutorials, and assignment solutions. Your solution should be a reasonable implementation of the mathematical description described in your report. Use code constructs used in the unit. For example, use set comprehensions instead of loops where possible to match the mathematical notation better. Avoid Python language syntax and constructions that aren't used in the unit, such as classes. The problems are all solvable using the Python concepts used in the unit, with the possible exception of formatted printing which may be useful for printing out a solution.

- Work out how to use Python data structures (sets, tuples, dictionaries, functions, ...) to implement your mathematical models of problem instances and solutions.
- Define a variable called `testData` containing the test data available for your topic at the end of this document, encoded in a way that your `solve` function can use.
- Define any helper functions that you use in the same file (aside from standard library functions). Include any functions from the unit material that you use so that your file is self-contained.
- Write a function called `solve` that takes a problem instance as a parameter and returns a solution data structure.

- Define a function `printSolution` that prints out a solution object in a reasonable format.
- Call your `solve` function on your `testData` and print the result to show that it arrives at a suitable solution.

Your file should be arranged like so:

```
testData = ...

#define any helper functions here, including any from slides/tutorials

def solution(problemInstance):
    ...

def printSolution(solutionInstance):
    ...

printSolution(solution(testData))
```

Your code submission will be a single file that can be run directly with Python3.

4 Marking criteria

4.1 Presentation

- Report is typeset throughout (no hand-written equations or hand-drawn diagrams). (1 mark)
- Report is well organised, using sections as mentioned above. (1 mark)
- Report formatting is consistent and appropriate (eg. larger fonts for headings) (1 mark)
- Minimal typos and grammatical errors. (1 mark)
- Submission instructions (below) followed correctly, including correct file types and file names. (1 mark)

4.2 Mathematical description

- Introduction is a reasonable description of the problem. (2 marks)
- Introduction uses plain language and is easy to understand for someone without a mathematical background. (2 marks)

- Instance model is a reasonable encoding of the problem using mathematical notation and terminology. (1 mark)
- Instance model explains how the mathematical model relates to the plain language description from the introduction. (1 mark)
- Solution model is a reasonable encoding of the problem using mathematical notation and terminology. (1 mark)
- Solution model explains how the mathematical model relates to the plain language description from the introduction. (1 mark)
- Problem model gives a correct mathematical description of the relationship between an instance and a valid solution. (2 marks)
- Problem model uses rigorous mathematical notation and terminology. (1 mark)
- Solution method is appropriate to the problem. (1 mark)
- Solution method description is reasonable and correct. (2 marks)

4.3 Implementation

- Python file runs without errors and prints out a solution. (1 mark)
- `solve` function gives a correct solution on the test data in a non-trivial way (i.e. computes the answer rather than just returning a fixed solution). (1 mark)
- `solve` function gives correct solution in general. (2 marks)
- `solve` function implements solution found in report. (1 mark)
- `printSolution` function prints out the solution in an easy to understand format. (1 mark)

Total marks: 25

5 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 `cab203-n1234567.py` with your student ID in place of `n1234567`

Links will be available on Blackboard under Assessment. You can make as many submissions as you like up to the due date, but only your most recent submission will be saved. Please note that Blackboard considers the deadline to be 11:59:00pm. If your clock says 11:59pm, then it is probably past the deadline. We will generally accept these even though they are marked late, but you will get locked out at 11:59:00pm if you have a previous submission. Plan ahead and submit on time to avoid complications!

Extensions: Extension requests are processed by student services. There is a link on Blackboard under Assessment 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 (`cab203@qut.edu.au`) to delete your previous submission so that you can make a late submission.

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.

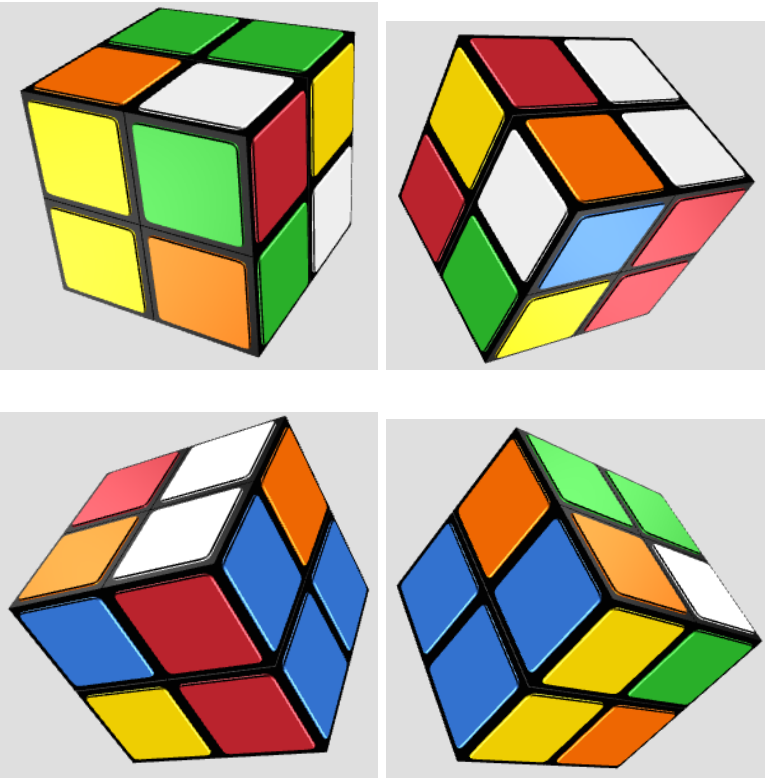
For code, please include your citation as a comment within the code.

6 Test data

Use this test data for testing your Python code. You must include your encoding as the variable `testData` in your source code.

6.1 2×2 Rubick's cube

The following are pictures of a single Rubick's cube from multiple angles.



You may find the following link helpful: https://www.grubiks.com/puzzles/rubiks-mini-cube-2x2x2/

6.2 Assembly line

The engineer has given the following lists of parts and robots:

Part	Attaches to
chasis	None
gadget	whatsit
doodad	gadget, doohickey
whatsit	thingamabob
whatchamacallit	doodad, whatsit
doohickey	chasis
thingamabob	chasis

Robot	Part it attaches	Parts which block it
Johnny-V	chasis	None
TARS	gadget	whatchamacallit
Marvin	doodad	None
Gerty	whatsit	None
Ava	whatchamacallit	None
Lore	doohickey	whatsit
T800	thingamabob	doohickey, doodad

6.3 Family Tree

Below are the cards in Bob's family tree. Bob's records end with the first generation aboard the ship, so their parents are noted as "Unknown".

Name:	Alice
Father:	Arlo
Mother:	Madeline
Children:	None

Name:	Bob
Father:	Charlie
Mother:	Eve
Children:	None

Name:	Eve
Father:	Oliver
Mother:	Aurora
Children:	Bob

Name:	Charlie
Father:	Jack
Mother:	Luna
Children:	Bob

Name:	Madeline
Father:	Jack
Mother:	Aurora
Children:	Alice

Name:	Arlo
Father:	Oscar
Mother:	Isla
Children:	Alice

Name:	Oliver
Father:	Hugo
Mother:	Rose
Children:	Eve

Name:	Luna
Father:	Oscar
Mother:	Isla
Children:	Charlie

Name:	Aurora
Father:	Hugo
Mother:	Rose
Children:	Eve, Madeline

Name:	Jack
Father:	Oscar
Mother:	Rose
Children:	Charlie, Madeline

Name:	Hugo
Father:	Unknown
Mother:	Unknown
Children:	Oliver, Aurora

Name:	Rose
Father:	Unknown
Mother:	Unknown
Children:	Oliver, Aurora, Jack

Name:	Isla
Father:	Unknown
Mother:	Unknown
Children:	Luna, Arlo

Name:	Oscar
Father:	Unknown
Mother:	Unknown
Children:	Luna, Jack, Arlo

6.4 Complicated wrist watch

Here is your cousin's users' manual. By trial and error, you have discovered that if a button press is not mentioned in the manual for a particular mode, then then that button press does not change the mode. Also by trial and error, if the watch is left alone for a while will go back to Time mode, so you can consider that the start state. Your cousin wants to get to the Date set mode.

Clarsio F911W-1000 User's manual

In Time mode press Mode to change to Timer mode. In Stopwatch mode press Mode to switch to Alarm1 mode. In Alarm2 mode press Mode to switch to Time mode. In Alarm1 mode long press Set to switch to Alarm1 set mode. In Time set mode press Set to switch to Time mode. In Alarm1 mode press Mode to switch to Alarm2 mode. In Time set mode long press Set to switch to Date set mode. In Time mode press Set to switch to TimeZone2 mode. In Alarm2 mode long press Set to switch to Alarm2 set mode. In Alarm1 set mode press Set to switch to Alarm1 mode. In Timer set mode press Set to switch to Timer mode. In Date set mode press Set to switch to Time set mode. In Timer mode press Mode to switch to Stopwatch mode. In Alarm2 set mode press Set to switch to Alarm2 mode. In TimeZone2 mode press Set to switch to Time mode. In Time mode long press Set to switch to Time set mode. In Timer mode press Set to switch to Timer set mode.