FIT1045 Algorithmic Problem Solving – Workshop 4

Objectives

The **objectives of this workshop** are:

- To implement and manipulate data structures for graphs in Python.
- To implement algorithms on graphs in Python.
- To implement tables as lists of lists in Python.

Useful Material

Reading from files: Section 7.2.1
`https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files`

Lists: `https://docs.python.org/3/tutorial/datastructures.html`

Task 0: Prelab

NOTE: If using Windows, it is encouraged that you open any text files being used from input/output in either the IDLE editor that comes standard with any Python installation. Or use a text editor that is better suited than Notepad. A good example is Notepad++.

The purpose of this prelab is to raise your understanding of reading/writing from/to files.

Part 1 The file 'mod_haiku.txt' contains a haiku poem with the spaces replaced with ';' and newlines, '\n', replaced with tabs, '\t'. Using only the following functions, and any control structures or datatypes you have learned thus far, you should restore the haiku to its previous state and write it to a file called 'haiku.txt':

```
open()
write()
close()
split()
join()
```

Part 2 The file 'grades.txt' constains a list of (fictional) student grades for FIT1045. The file contains several lines, where each line represents a unique student. The comma separated values are in the following order: student ID, full name and final mark. You are to write a program that reads the file and displays each field to the user with final grade (N, P, C, D, HD) calculated and displayed.

Task 1:

One way of storing a graph is as an *adjacency list*, that is, for each vertex we store a list of vertices adjacent to it. For this task you can assume the vertices are numbered $0, 1, 2, \ldots, n-1$ where $n \geq 1$.

Write a Python program that takes as input the number of vertices, n, and the name of a file containing an edge list representing a graph. Each line of the file has exactly one edge represented by 2 vertices.

For example: The file containing the following lines:

```
0 1
2 1
0 2
1 3
```

represents the graph in Figure 1.

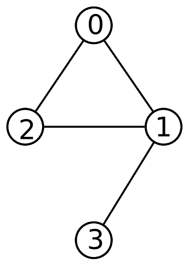

Figure 1: Graph G

Your program should read an edge list from the file and store it as an adjacency list. It should print the adjacency list, so that you can check that you have read the file correctly.

For example: If the file "testGraph.txt" contains the edges of the graph in Figure 1, your program might do the following:

```
Enter value for n: 4
Enter the filename: testGraph.txt
[[1,2],[0,2,3],[1,0],[1]]
```

Task 2:

Modify your program in Task 1 so that your program finds a spanning tree of G. You can assume G is a connected graph. Your program should print an adjacency list for the spanning tree graph.

For example: If the file "testGraph.txt" contains the edges of the graph in Figure 1, your program might do the following:

```
Enter value for n: 4
Enter the filename: testGraph.txt
[[1],[0,2,3],[1],[1]]
```

Task 3:

Another way of storing a graph is as an *adjacency matrix*, that is, an $n \times n$ table A where $A[i][j] = 1$ if vertex i is adjacent to vertex j. In this task vertices which represent football teams are numbered $0, 1, 2, \ldots, n-1$, where n is the number of football teams. Two teams are adjacent in the graph, if they have played a match together in 2015. You can assume that two teams played at most one match with each other in 2015.

Write a Python program that takes as input the number of teams, n, and the name of a file containing the edges in the graph. The file is formatted in the same way as the file in the previous tasks.

Your program should store the graph as an adjacency matrix. It should print the adjacency matrix, so that you can check that you have read the file correctly.

For example: If the file "testGraph.txt" contains the edges of the graph in Figure 1, your program might do the following:

```
Enter the number of teams: 4
Enter the filename: testGraph.txt
[[0,1,1,0],[1,0,1,1],[1,1,0,0],[0,1,0,0]]
```

Task 4:

Modify your program in Task 3, so that it outputs all pairs of teams that did not play a match in 2015. Your program should output each pair exactly once. A team cannot play itself in a match.

For example: If the file "testGraph.txt" contains the edges of the graph in Figure 1, your program might do the following:

```
Enter value for n: 4
Enter the filename: testGraph.txt
0,3
2,3
```

Task 5:

Modify your program in Task 4, so that it asks for an additional input file that contains the names of the teams. There is a single team's name on each line of the file.

For example: a possible input file for team names corresponding to the vertices in the graph in Figure 1 might be::

```
Collingwood
Essendon
Hawthorn
Richmond
```

Your program should now output the names of the pairs of teams that have not played a match in 2015.

For example: If the file "testGraph.txt" contains the edges of the graph in Figure 1 and the file "teamsNames.txt" is the example file above, your program might do the following:

```
Enter value for n: 4
Enter the filename: testGraph.txt
Enter the team-names file: teamNames.txt
Collingwood, Richmond
Hawthorn, Richmond
```