Computational Thinking – Modelling with Graphs

Summative Assignment

Lecturer: Dr Eleni Akrida

Assessment: summative

Hand-in method: DUO

Document format: PDF

Deadline: Friday 6th March 2020, 14:00

Description of the Assignment: The Assignment is divided into two parts, Part A and Part B. Part A concerns graph colouring (60% of the total marks) and Part B concerns graph traversing (40% of the total marks). Each part is subdivided into Questions, each having an associated percentage of the total marks.

In each part of the Assignment, you are provided with 5 python (i.e. **.py*) files, each of them containing one specific input graph. Please copy all these files to your working Python folder (such that you can access them in IDLE). Each one of these files contains a function *Graph()* which returns the appropriate graph. Furthermore, these graphs will already have some initialization of all the necessary attributes for the nodes. Suppose that, for example, the file is called *graph1.py*. In this case, if you write the following in IDLE:

```
>>> import graph1
>>> G = graph1.Graph()
```

then G will be the graph that is constructed by the function *Graph()* of the file *graph1.py*.

For Questions A.1 and A.2 of Part A, and B.1 of part B you should write some code in Python. Then you should execute your code providing as input the graphs that are created by the corresponding graph files. For each of those questions, you are asked to submit both:

- your code and
- the output of this code in a separate .txt file. To produce this separate text file with the output of your code, please just copy-paste your output into this .txt file. Please note that you should **not** implement any additional functions (such as "file.write" or similar) that automatically prints the output into a file.

Details follow in the specific description of each Part (and question within) below. Also, **to avoid unnecessary loss of marks, please do not change at all the way the output is produced in the *.py* files that are provided to you.** It is also important that you prepare and run your code in **Python 3.6** (not in other python versions). Otherwise you may risk unnecessary loss of marks, as I will test your code in Python 3.6.

Your answers will be marked as either correct or wrong. For Questions A.1, A.2, and B.1, this means that if your code for one of the questions works, then you will receive the full 20% associated with that question; **if your code does not work, you will receive 0% for that question**. For the sub-questions within Questions A.3 and B.2, if your written answer is correct you will receive the full percentage associated with that sub-question; **if your answer is wrong, you will receive 0% for that sub-question**.

In the specific description of each Part below, whenever you are asked to submit a file (containing either code or text), it is specified how to include **your 6-digit CIS-username *XXXXXX*** in the name of the file (this code is of the form *abcd12* and you can find it also on your Campus-card).

PART A (60%)

Question A.1 (20%)

You are given the files *graph1.py, graph2.py, graph3.py, graph4.py* and *graph5.py*. Each of these files contains a function *Graph()* which returns an input graph.

You are also given a file *greedy_col_incomplete.py*. This file has two (incomplete) functions *find_smallest_colour(G,i)* and *greedy(G)*. Complete these two functions and save your code in a file that is called *greedy_col.py* such that, when you run this new file:

- it visits the vertices of the input graph sequentially in the order 1, 2, 3, 4, … (i.e. regardless of whether the next visited node is adjacent or not to any of the previously visited nodes),
- at every step it assigns (in a greedy fashion) to the currently visited node the smallest possible colour (where every colour is a positive integer 1, 2, 3, …) such that no two adjacent nodes receive the same colour, and finally
- it outputs the constructed colouring and the number *kmax* of the different colours in this colouring (using the printing commands that are given within the function *greedy(G)* in the file *greedy_col_incomplete.py*).

Each time the algorithm visits a new node *i* of a graph *G*, the function *find_smallest_colour(G,i)* returns the colour to be assigned to *i*.

The **output** of ***greedy_col.py*** should be copied to a text file called **"*output_greedy_col_XXXXXX.txt*"**, where *XXXXXX* is your CIS-username. This text file with your output should be submitted together with your file *greedy_col.py*.

Question A.2 (20%)

Consider again the files *graph1.py, graph2.py, graph3.py, graph4.py* and *graph5.py* of Question A.1. Recall that each of these files contains a function *Graph()* which returns an input graph.

You are also given a file *greedy_col_variation_incomplete.py*. This file has three (incomplete) functions *find_next_vertex(G), find_smallest_colour(G,i)* and *greedy(G)*. Complete these three functions and save your code in a file that is called *greedy_col_variation.py* such that, when you run this new file:

- it visits the vertices of the input graph in such an order that always the next visited node is adjacent to at least one previously visited node,
- among all such vertices, the algorithm visits at every step the smallest one (i.e. the vertex that is labelled with the smallest integer index),

- at every step it assigns (in a greedy fashion) to the currently visited vertex the smallest possible colour (where every colour is a positive integer 1, 2, 3, …) such that no two adjacent nodes receive the same colour, and finally
- it outputs the constructed colouring and the number *kmax* of the distinct colours in this colouring (using the printing commands that are given within the function *greedy(G)* in the file *greedy_col_variation_incomplete.py*).

At every iteration of the algorithm, the function *find_next_vertex(G)* computes and returns the next visited node. Furthermore, each time the algorithm visits a new node *i* of a graph *G*, the function *find_smallest_colour(G,i)* returns the colour to be assigned to i. Note that the function *find_smallest_colour(G,i)* is exactly the same as in Question A.1.

The **output** of ***greedy_col_variation.py*** should be copied to a text file called "***output_greedy_col_variation_XXXXXX.txt***", where *XXXXXX* is your CIS-username. This text file with your output should be submitted together with your file *greedy_col_variation.py*.

Question A.3 (20%)

This question does **not** require you to program. You are instead required to demonstrate your understanding of how to transform a real-world problem into a graph colouring problem and go through the steps of a colouring algorithm on the resulting graph. The real-world problem we consider is the following:

Problem A.3

The University timetabling team has four available time-slots and an unlimited number of lecturers on 6th March 2020 to schedule classes $C_1, \dots, C_7$ which contain the following students:

$$C_1 := \{Amy, Brian, Clare, Ellen\}$$

$$C_2 := \{Clare, David, Ellen, Mary, Thomas\}$$

$$C_3 := \{Ellen, Fred, Eva\}$$

$$C_4 := \{Gertrude, Henry, Isla\}$$

$$C_5 := \{Isla, John, Amy, Christopher, Lee, Bob, Jack\}$$

$$C_6 := \{Brian, David, Fred, Henry, John\}$$

$$C_7 \coloneqq \{Christopher, Mary, Lee, Jack, Thomas\}$$

Is it possible to timetable all the classes on that day?

A.3.1 (5%) Transform Problem A.3 into a graph colouring problem, i.e. describe the instance of the problem as a graph, G, and then ask the relevant question, which if answered would give you the answer to Problem A.3.

A.3.2 (7%) Consider the graph, G, that you constructed in A.3.1. What is the order in which the algorithm of Question A.2 visits the vertices of G to find a proper colouring?

A.3.3 (5%) What colour is assigned to each vertex of G according to the algorithm of Question A.2? Use colours with labels 1, 2, 3... as in Question A.2.

A.3.4 (3%) What is the chromatic number of $G, \chi(G)$? Why?

The answers to Questions A.3.1 – A.3.4 should be *neatly* written using Word or LaTeX; you may use Paint (or any other similar application, or even use the Shapes provided within the Word editor, or a LaTeX package, e.g. TikZ) to create the graph G and insert the image in your Word or LaTex source file. Marks will be deducted if your answers are not clear or neatly written. You should then export your final Word or LaTeX document into PDF format and submit your solutions in a pdf file called **"scheduling_solution_*XXXXXX.pdf*"**, where *XXXXXX* is your CIS-username.

PART B (40%)

Question B.1 (20%)

You are given the files *graph6.py, graph7.py, graph8.py*, *graph9.py* and *graph10.py*. Each of these files contains a function *Graph()* which returns an input graph. Furthermore, you are given the file *depth_first.py*, which includes an implementation of the "Depth-First Search (DFS)" algorithm for traversing a graph. In particular, *depth_first.py* contains only one recursive function *dfs(G,u)*, which is called recursively to continue the DFS-traversing of the graph *G*, starting at node *u*.

After the code for the function *dfs(G,u)*, you can find (in the main part of *depth_first.py*) the calls of this function for all of the graphs returned by the files *graph6.py*, …, *graph10.py*. Therefore, if you run the file *depth_first.py* (for example by pressing F5), you see the output of the function *dfs()* for each of these 5 graphs, each time for *u*=1.

You are given a file *breadth_first_incomplete.py*. This file has one (incomplete) function *bfs(G,a,b)*, where *G* is the input graph and *a,b* are two given nodes of *G*. Complete this function and save your code in a file that is called *breadth_first.py* such that, when you run this new file:

- it performs a Breadth-First Search (BFS), starting from node *a* and ending when it reaches node *b*,
- it outputs the *distance* (i.e. the length of a shortest path) between the given pairs of nodes for the 5 input graphs (as specified in the main part of the file *breadth_first_incomplete.py*).

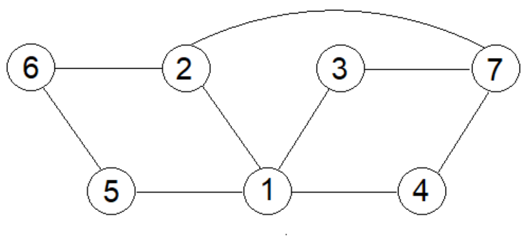
To avoid loss of marks, please do not use ready-made functions from networkx (or other imported packages) to compute distances or shortest paths between vertices.

The **output** of ***breadth_first.py*** should be copied to a text file called "***output_breadth_first_XXXXXX.txt***", where *XXXXXX* is your CIS-username. This text file with your output should be submitted together with your file *breadth_first.py*.

Question B.2 (20%)

This question does not require you to program. You are instead required to demonstrate your understanding of different algorithms that we have seen in the lectures.

B.2.1 (3%) Consider the graph:



Write down the adjacency matrix and adjacency linked lists specifying this graph. *The rows and columns in the adjacency matrix, as well as the vertices in the adjacency linked lists, should follow the (numerical) order of the vertex labels.*

B.2.2 (8%) Starting at vertex 1 and resolving ties by the vertex numerical order, traverse the graph of Question B.2.1 by depth first search (DFS) and construct the corresponding DFS tree. Give the order in which the vertices were reached for the first time (pushed onto the traversal stack) and the order in which the vertices become dead ends (popped off the traversal stack).

B.2.3 (9%) Consider again the graph of Question B.2.1 and add weights to its edges as follows: edge $\{i, j\}$ gets weight equal to $i \cdot j$, i.e. the weight of each edge is the product of the indices of its endpoints. Run Kruskal's algorithm on the resulting graph in order to find a minimum spanning tree. Describe the order in which edges are considered and added to the tree. Write down the produced spanning tree and its total weight (cost).

The answers to Questions B.2.1 – B.2.3 should be *neatly* written using Word or LaTeX; you may use Paint (or any other similar application, or even use the Shapes provided within the Word editor, or a LaTeX package, e.g. TikZ) to create any necessary graphs and insert the relevant image in your Word or LaTex source file. Marks will be deducted if your answers are not clear or neatly written. You should then export your final Word document into PDF format and submit your solutions in a pdf file called **"traversing_solution_*XXXXXX.pdf*"**, where *XXXXXX* is your CIS-username.

In summary, you are expected to submit the following files:

- *greedy_col.py* (Question A.1)
- *output_greedy_col_XXXXXX.txt* (Question A.1)
- *greedy_col_variation.py* (Question A.2)
- *output_greedy_col_variation_XXXXXX.txt* (Question A.2)
- scheduling_solution_*XXXXXX.pdf* (Question A.3)
- *breadth_first.py* (Question B.1)
- *output_breadth_first_XXXXXX.txt* (Question B.1)
- traversing_solution_*XXXXXX.pdf* (Question B.2)