Computational Thinking – Modelling with Graphs

Summative Assignment

Lecturer: George Mertzios
Assessment: summative
Hand-in method: DUO
Document format: PDF
Deadline: Friday 2nd March 2018

Description of the Assignment: The Assignment is divided into Part A, Part B, Part C. Part A concerns *graph coloring*, Part B concerns *graph traversing* and Part C concerns *randomized aspects* of Parts A and B. The marks for the assignment are divided as follows: 45% for Part A, 40% for Part B, 15% for Part C. Each of these parts is sub-divided into Questions.

In each of the three parts of the Assignment, I provide you with 5 python (i.e. **.py*) files, each of them containing one specific 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*.

In each of Parts A, B and C you should write some code in Python. Then you should execute your code for each Part providing as input the graphs that are created by the corresponding graph files. For each Question of each Part, you are asked to submit:

- 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, instead of implementing any additional functions (such as "file.write" or similar) that automatically print the output into a file.

Details follow in the specific description of each Part 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.**

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 (45%)

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 a graph. Furthermore, you are given the file *brute_3_col.py*, which includes an implementation of the "brute-force" algorithm for the 3-coloring problem. In particular, *brute_3_col.py* contains the following functions:

- *recursion(G,i)*: this function is the recursive part of the brute force (which computes exhaustively all possible assignments of three colors (1, 2, 3) to the nodes of *G*).
- *checkcol(G)*: this function checks whether the currently computed coloring is proper, i.e. whether there exist two adjacent nodes with the same color.
- *brute_force(G)*: this function is the main part of the brute force algorithm, which makes the initial call of the function *recursion(G,i)*, in order to decide whether the given graph *G* has a 3-coloring, and to return such a coloring of the nodes if such one exists.

After the code for each of these functions, you can find (in the file *brute_3_col.py*) the calls of the function *brute_force()* for all of the graphs returned by the files *graph1.py*, … *graph5.py*. Therefore, if you run the file *brute_3_col.py* (for example by pressing F5), you see the output of the function *brute_force()* for each of these 5 graphs.

Question A.1 (15%).
Create a new file *brute_k_col.py* (by appropriately modifying the given file *brute_3_col.py*) such that, when you run this new file:

- it asks the user to type an integer *k* (one integer for each one of the graphs *graph1.py, ... graph5.py*) and then
- it solves the *k-coloring problem* using brute-force on these graphs (i.e. it returns a proper coloring of the nodes of the input graphs with at most *k* different colors 1, 2, …, k, or it returns that such a coloring does not exist for the input graph).

For each of the graphs *graph1.py*, … *graph5.py*, your file ***brute_k_col.py*** should ask the user for a value *k*, and then it should solve the *k*-coloring problem by brute force on the corresponding graph. Furthermore, the whole output of *brute_k_col.py* should be copied to a text file called "***output_brute_k_col_XXXXXX.txt***", where *XXXXXX* is your CIS-username. This text file with your output should be submitted together with your file *brute_k_col.py*.

Hint: Use the Python's function *input()*to ask the user for the value *k* (see pages 57-59 in *"Introduction to Computing using Python"*, Ljubomir Perkovic, 2012).

Question A.2 (15%).
You are given a file *greedy_col_basic_incomplete.py*. This file has two (incomplete) functions *find_smallest_color(G,i)* and *greedy(G)*. Complete these two functions and save your code in a file that is called *greedy_col_basic.py* such that, when you run this new file:

- it visits the vertices of the input graph 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 color (where every color is a positive integer 1, 2, 3, …) such that no two adjacent nodes receive the same color, and finally
- it outputs the constructed coloring and the number *kmax* of the different colors in this coloring (using the printing commands that are given within the function *greedy(G)* in the file *greedy_col_basic_incomplete.py*).

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

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

Question A.3 (15%).
You are given a file *greedy_col_variation_incomplete.py*. This file has three (incomplete) functions *find_next_vertex(G)*, *find_smallest_color(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 the smallest one (i.e. the node that is labeled with the smallest integer),
- at every step it assigns (in a greedy fashion) to the currently visited node the smallest possible color (where every color is a positive integer 1, 2, 3, …) such that no two adjacent nodes receive the same color, and finally
- it outputs the constructed coloring and the number *kmax* of the distinct colors in this coloring (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_color(G,i)* returns the color to be assigned to *i*. Note that the function *find_smallest_color(G,i)* is **exactly the same** as in Question A.2.

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*.

PART B (40%)

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 a 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 $u=1$.

Question B.1 (20%).
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 the shortest possible 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*).

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%).
You are given a file *depth_first_pair_nodes_incomplete.py*. This file has one (incomplete) function *dfs(G,a,b,u)*, where *G* is the input graph and *a,b* are two given nodes of *G*. This function *dfs(G,a,b,u)* is called recursively to continue the DFS-traversing of *G*, starting at node *u*. Complete this function and save your code in a file that is called *depth_first_pair_nodes.py* such that, when you run this new file:

- it performs a Depth-First Search (DFS), starting from node *a* and ending when it reaches node *b*,
- in addition it adds a label to each of the visited nodes, such that if a vertex v receives the label *i*, then this Depth-First Search has reached vertex *v* with a path of length *i* (starting form *a*); note that the label of vertex *a* is 0.

Hint: Recall that in the given file *depth_first.py* we used the integer variable *visited_counter* in order to count how many nodes we have visited so far (we stop our DFS when we have visited all nodes of the graph, i.e. when *visited_counter* = *n*). However, in the file *depth_first_pair_nodes.py* we do not need such a counter: we now

stop our DFS when we have visited the specific node *b* (i.e. we do not necessarily need to visit all nodes of the graph).

The output of ***depth_first_pair_nodes.py*** should be copied to a text file called "***output_depth_first_pair_nodes_XXXXXX.txt***". This text file with your output should be submitted together with your file *depth_first_pair_nodes.py*.

PART C (15%)

You are given the files *graph6.py*, *graph7.py*, *graph8.py*, *graph9.py* and *graph10.py*, i.e. the same graphs as in Part B. Each of these files contains a function *Graph()* which returns a graph. Furthermore, you are given a file *random_distance_incomplete.py*. This file has one (complete) function *bfs(G,a,b)*, one (complete) function *max_distance(G)* and one (incomplete) function *random_distance(G)*, where *G* is the input graph.

Note that the given (incomplete) function *bfs(G,a,b)* must be filled in exactly in the same way as in Question B.1, so you just need to *copy-paste* your code of *bfs(G,a,b)* from there.

After the code for the functions *max_distance(G)* and *random_distance(G)*, you can find (in the main part of *random_distance_incomplete.py*) the calls of these functions for all of the graphs returned by the files *graph6.py*, …, *graph10.py*. Complete the function *random_distance(G)* and save your code in a file that is called *random_distance.py* such that, when you run this new file:

- it outputs the greatest distance between any two nodes in the input graph (this is also called the *diameter* of the input graph), and
- it computes the distances of five randomly selected pairs of nodes, and it returns the maximum among these distances.

For the output returned by *random_distance.py*, just use the printing commands from the main part of the file *random_distance_incomplete.py*. The function *max_distance(G)* returns the *diameter* of the graph *G*. Furthermore, the function *random_distance(G)* returns the maximum among the distances of 5 randomly selected pairs of nodes in *G*.

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

Remarks:

- As we did in the lecture, ensure that in your function *random_distance(G)* you never calculate the distance of a randomly chosen vertex from itself.
- Note that, in the main part of *random_distance_incomplete.py*, we initialize the graphs *G6*, *G7*, *G8*, *G9* and *G10* before every call of *max_distance(G)* and of

random_distance(G). This is done in order to initialize again the attributes of these graphs, such that the calls of *breadth_first_plain.bfs(G,a,b)* return the correct values.

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

- *brute_k_col.py* (Question A.1)
- *output_brute_k_col_XXXXXX.txt* (Question A.1)
- *greedy_col_basic.py* (Question A.2)
- *output_greedy_col_basic_XXXXXX.txt* (Question A.2)
- *greedy_col_variation.py* (Question A.3)
- *output_greedy_col_variation_XXXXXX.txt* (Question A.3)
- *breadth_first.py* (Question B.1)
- *output_breadth_first_XXXXXX.txt* (Question B.1)
- *depth_first_pair_nodes.py* (Question B.2)
- *output_depth_first_pair_nodes_XXXXXX.txt* (Question B.2)
- *random_distance.py* (Question C)
- *output_random_distance_XXXXXX.txt* (Question C)