CSCI 332 Fall 2023 - Software Assignment #1

(Version 3 – changes since last version shown in purple font)

Prompt: Implement Kruskal's algorithm for finding the minimum spanning tree of a weighted graph. For full credit, you must utilize a union-find data structure, as described in Section 4.6 of the Kleinberg/Tardos textbook; i.e., as part of this assignment you must read and understand Section 4.6. Please note that they prescribe several optimizations, and all of these should be present in your implementation as well. You must also implement a mergesort algorithm.

You can only utilize the following primitive python data structures: python lists, python dictionaries, and numpy arrays. You can only utilize primitive python and numpy functions, such as sum, multiplication, division, exponentiation, etc. If you are in doubt, then please ask. Regardless of what operations and data structures you use, *you must conform to the API provided below.* An executable .py file is included along with its expected output (as pdf and html files) to test the I/O of your code.

I have provided some example input/output below, primarily aimed at ensuring that your code is compatible with the API. Although it is not explicitly required, I strongly suggest that you create some auxiliary code to generate new graphs and test that your code produces the correct solution.

Submission

- Submit your python code as a single .py file. I may deduct points if you submit it in parts.
- I may deduct points if you do not use *exactly* the class/method names below
- Submission on Moodle by 11pm MST, 11/5/2023. I recommend submitting early - at 11:01 pm it will be considered a day late.
- 5% off per day that the assignment is late, up to 7 days. 0% credit beyond 7 days
- Reminder: you are not permitted to drop a software assignment

Additional Software Specifications and I/O:

- All code must be written in Python.
- All numeric values should be floats, unless otherwise stated.
- You must use *exactly* the function names described below.
- Code must be delivered as a Python class, and include the methods specified below.
- You can assume graphs will have unique weights, and will be fully-connected.

obj = kruskalClass()

- **Input**:
- **Output**: an object instantiated from your class. Your class should have the name 'kruskalClass'.

T = findMinimumSpanningTree(A)

- **Input**: 'A' is an NxN numpy array representing an adjacency matrix representation of a graph. If $A_{ij} = w > 0$ it implies there is an edge between nodes i and j with weight w. We will only consider undirected graphs, and therefore 'A' will be an upper right triangular matrix. You can assume no two edge weights are the same.

- **Output:** ‘T‘ is an NxN numpy array representing the minimum spanning tree of ‘A’. We will only consider undirected graphs, and therefore ‘T’ should be an upper right triangular matrix.

[b,inds] = mergesort(a)

- **Input**: ‘a’ must be a 1xK numpy array.
- **Output**: ‘b’ must be a 1xK numpy array with elements in *ascending* order (i.e., lowest value to highest). ‘inds’ must be a 1xK numpy array with the index in ‘a’ of each entry in the sorted output array ‘b’.

u = makeUnionFind(N)

- **Input:** ‘N’ is the number of nodes in a graph.
- **Output:** ‘u’ is a 1XN python dictionary, where the keys are numerical labels for the nodes, and the values are numpy arrays with the 0th entry being a pointer to a node in the same dictionary. The numpy arrays can be more than one 1-D if you want, but they must be numpy arrays, and the 0th entry must contain pointers to other nodes that are used in the ‘find’ and ‘union’ functions.

s = find(u,v)

- **Input:** ‘u’ is a union-find data structure. ‘v’ is a numerical index for a graph node.
- **Output:** ‘s’ is the numerical value corresponding to the label for the set of connected nodes to which ‘v’ belongs.

u_out = union(u_in,s1,s2)

- **Input**: ‘u_in’ is a union-find data structure. ‘s1’ and ‘s2’ are numerical values corresponding to the labels of two groups of graph nodes.
- **Output**: ‘u_out’ is the same as ‘u-in’, except that the two groups of nodes, ‘s1’ and ‘s2’, have been merged into a single group with a single label.

Grading criteria

- **Completeness** (50%): the code is complete. It does not throw an error when I run it with the I/O listed above, even if it does not produce the correct answer. It only uses primitive operations and data structures. The code should also be well commented with original commentary (i.e., it was not copied - you wrote it) - each method outlined in the I/O should have a brief description, and there should be (on average) one comment for every ~5 lines of code, if not more.
- **Correctness** (50%)**:** I will input several graphs into your code to verify that it produces the correct output for the top-level function, as well as the intermediate steps (e.g., ‘find’ and ‘union’).
    - You will lose 10% if you do not conform strictly to API above
    - You should be able to handle non-empty graphs of varying size.

A Note on Teamwork and Cheating:

- You are welcome to discuss the assignment and your algorithmic strategy with others, but for the reasons outlined above, you should be cautious and discuss it primarily at a broad level. As outlined below, you are responsible for being capable of explaining your code, in detail, if I were to ask you about it.
- I may manually inspect or use software to check the similarity of your code to software online (e.g., github), or among other submitted sets of code. If I find similarities in your code to other existing/submitted code, I reserve the right to ask you questions about your submitted software. *If asked, you should be able to explain all portions of your code in detail.* If not, I may dock a substantial portion of points from your software grade, and potentially take more serious action.
- For these reasons, I strongly discourage you from looking at implementations of the algorithm created by someone on the internet, or by someone else in the class, before you do it yourself. It is crucial that you attempt to write the code yourself to ensure that you understand it well, and can explain it, if asked.