Data Structures Lab 12 Graphs 2 Name:____________

Objectives: To understand how a graph can be used to solve graph algorithms.

To start the lab: Download and unzip the file lab12.zip

Part A: In IDLE open the `PriorityQueue` class file: `lab12/priorityQueue.py`. The changes to our familiar `BinHeap` class that we discussed in lecture are included:

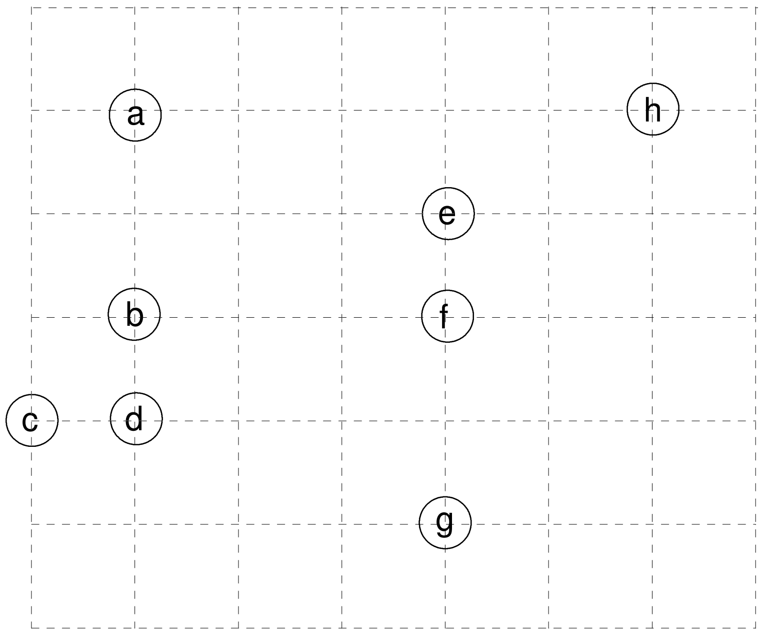
- `self.heapArray` is a list of tuples with the first tuple value (tuple index 0) being the "priority" and second tuple value (tuple index 1) being its associated "key" value.
- a `__contains__` method is added to check if a value is in the priority queue.
- a `decreaseKey` method is added to allow a priority value to be reduced (i.e., increasing its priority).

a) Why do the methods `percDown`, `minChild`, and `percUp` used the tuple index 0 when comparing tuples in the `self.heapArray`?

b) Why do the methods `decreaseKey` and `__contains__` used the tuple index 1 when comparing tuples in the `self.heapArray`?

c) Run the `lab12/make_min_spanning_tree.py` program which uses Prim's algorithm on the graph from lecture. Does it give the expected output?

d) Predict the order of edges added by Prim's algorithm if we start at vertex "e":

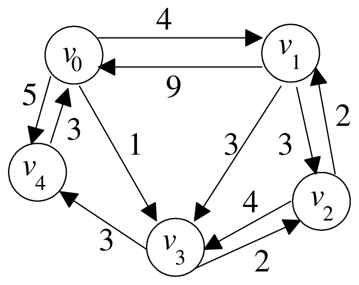


d) Modify the `lab12/make_min_spanning_tree.py` program to verify your prediction. NOTE: This is a very easy modification. You just need to start Prim's algorithm starting at the vertex labeled "e".

After you have answered the above questions and completed the code, raise your hand and explain your answers.

<u>Part B:</u> The textbook's Dijkstra's Algorithm code (Listing 7.11 p. 341 and same on-line) is in the `lab12/graph_algorithms.py` file.

a) Run the `lab12/test_dijkstra.py` program which uses Dijkstra's algorithm on the graph from lecture. Does it give the expected output?



b) Modify the `dijkstra` function in the `lab12/graph_algorithms.py` file by comparing it to the similar `prim` function.

After you have fixed the dijksta function in `lab12/graph_algorithms.py`, raise your hand and demonstrate your code by running the `lab12/test_dijkstra.py` program.

<u>Part C:</u> In IDLE open the `PriorityQueue` class file: `lab12/Part_C/priorityQueue.py`. This version adds a data attribute `self.keyToIndexDict` dictionary where the keys are all the second tuple values in the `self.heapArray` and their values are there corresponding index locations in is the `self.heapArray`.

The methods `buildHeap`, `percDown`, `minChild`, and `percUp` have all been modified to correctly update the `self.keyToIndexDict` dictionary.

Your task for Part C is to complete the methods `decreaseKey` and `__contains__` so that they use the `self.keyToIndexDict` dictionary. Thus, greatly improving the efficiency of `decreaseKey` and `__contains__` methods.

After you have completed the `decreaseKey` and `__contains__` methods in `lab12/Part_C/priorityQueue.py`, raise your hand and demonstrate your code by running the `lab12/Part_C/make_min_spanning_tree.py` program.

The EXTRA CREDIT Opportunities related to Part B:
1. Add code to the end of the `test_dijkstra.py` program to print the shortest paths from v_0 to each of the other vertices. One line of output might look something like:
   "Shortest path from v0 to v4 is v0 > v3 > v4 with a total distance of 4"