Week 12

Dijkstra's Single-Source Shortest Path

Topic covered:

- Prim's algorithm for Minimum Spanning Tree
- Dijkstra's Single-Source Shortest Path algorithm

Supplemental materials

- Class lecture
- Heap.py (from week 3)
- Minimum Spanning Tree.pptx (from week 11)
- Single-Source Shortest Path.pptx
- village.zip containing the edge list of a simple test graph (for Prim's algorithm, from week 11)
- sssp.zip containing two weighted directed graphs for testing Dijkstra's algorithm. The provided solutions are with vertex 1 being the source vertex.

Goals

a) Develop a Python 3 program of Prim's algorithm and Diskstra's SSSP algorithm.
b) Understand the common mechanism of the two algorithms, and what distinguishes them.

1) Review, as necessary, the "heap" data structure studied in week 3. Example codes for utilizing the heap are also provided at the end of the Heap.py file.

2) Review the lecture note (or supplemental online resource) on how Prim's algorithm may be implemented. Writing a Python 3 program for Prim's algorithm requires the understanding of Python 3 class and heap data structure (to function as priority queue).

   Alternative online resource to review:
   - Prim's algorithm in 2 minutes - YouTube
   - Prims Algorithm for Minimum Spanning Trees - YouTube

   If these skills are acquired, you may proceed to write the program. If not, however, go back to step 1 and/or 5 until this condition is satisfied.

   The only required output is the total cost of the MST.

3) Test your program with the provided "village" test case.

4) Review the lecture note (or supplemental resource) on how Dijkstra's algorithm may be implemented. Develop your own program and test the correctness with test cases in sssp.zip. The solution to every test case is under assumption that vertex 1 is the source vertex.

   Alternative online resource to review:
   - Graph Data Structure 4. Dijkstra's Shortest Path Algorithm - YouTube
   - Dijkstra Algorithm Example - YouTube
   - Dijkstra's algorithm in 3 minutes - YouTube

5) Mechanism-wise, both Prim's and Diskstra's algorithms are equivalent, as both are specific applications of Uniform-Cost Search technique (For CS students, this will be studied further in Algorithm Design course). What is the logical difference between the two algorithms?

6) An advanced student who completes all steps above early should attempt on utilizing the provided "Priority_Queue" class given in priorityqueue.py, which has an additional "elevate_key" operation, for both Prim's and Dijkstra's programs. The elevate_key operation allows a node in the priority queue to increase its priority and consequentially puts the node in the position that satisfies the heap property. Accordingly, only one copy of each state is required to be enqueued and any subsequent change of its key value can be updated in the corresponding heap node directly (instead of enqueuing a copy of the state as a new node).

   The Priority_Queue requires additional auxiliary functions to be defined for the object class that will be its element.