Chapter 9 – Final Exam – Kelling

* Read **Chapter 9** in the McAllister text. **Write a Java Program that does the following:**
  + Generate a graph with 100,000 nodes, where each node randomly has between 1 and 5 connections to other nodes.
* Each node should contain within it a random value between 1 and 300,000. (So generally about 1 in 3 searches will yield a query match)
  + Allow the user to enter a number to search for, and **implement each** of the following three types of searching algorithms.
* **Breadth-First.** **(30 points)**
* **Depth-First.** **(30 points)**
* **Dijkstra's Algorithm. (40 points)**
  + Do not allow back-tracking in your searches. (Mark nodes that you already searched as complete, and do not re-visit them in the same search)
  + Each search should return the following:
* The Success/Failure of your search.
* The length of the shortest path to the found node.
* The total number of nodes examined during the search.
* *Optionally you may return the exhaustive display of the shortest path, for testing and verification.*
* *Also remember since your graph is created with random values, try re-creating it for different results.*
* **Note:**Dijkstra's Algorithm will require augmentation of the graph to include connection weights, which can also be completely random.
* **Hint:** In each case, start with a smaller number of nodes to test that your code is working, then increase the size when you are sure it works.
* ***Grading will be calculated by the correctness of query results, the adherence to the algorithm definitions, and by code quality.***

Depth First Search:

* Goes DEEP to its children before going broad to its neighbors
* Implemented with a recursive algorithm
* Make sure not to fall into an infinite loop – FLAG NODES THAT HAVE BEEN VISITED ALREADY

Breadth First Search:

* Goes BROAD to its neighbors before going deep to its children
* Implemented using a queue

Dijkstra's Algorithm:

* minPQ (min Priority Queue) –
* add start vertex to PQ
* assign it a priority of the distance between the start node & itself

\*start node would be assigned 0 – it is 0 steps away from itself

* at each step we remove the smallest entry in min PQ & then add all of its neighbors
* visited Nodes – List of nodes already checked / searched