

Python Programming Assignments

One-Person

1. **BFS Tree.** Given a dictionary L representing a directed graph G , return a dictionary T that represents a BFS spanning tree of G , where the keys are the vertices of G , the values are the parents in the BFS tree, and the root has itself as the parent. Note that the list L should be thought of as representing an undirected graph, in the usual fashion.
2. **Sort Edges by Weight.** Given a dictionary of dictionaries L representing an directed weighted graph G , return a list of the edges of the underlying undirected graph G sorted by weight, with an edge from u to v of weight w represented as $[u, v, w]$. For example, if $L = \{1:\{2:1, 3:4\}, 4:\{2:7, 3:2\}, 3:\{5:12\}, 2:\{\}, 5:\{\}\}$, then it would return $[[1, 2, 1], [3, 4, 2], [1, 3, 4], [2, 4, 7], [3, 5, 12]]$. Note that since the graph is undirected, each edge may be represented in two possible ways, so that $[[2, 1, 1], [3, 4, 2], [4, 2, 7], [1, 3, 4], [3, 5, 12]]$ would also be a valid list to return.
3. **Topological Sort.** Given a dictionary L representing a directed graph which may be assumed to be acyclic, return a list $[v_1, v_2, \dots, v_n]$ of the vertices so that all arcs of the graph have their head at a vertex of higher index than the tail. This corresponds to an ordering of the vertices so that all arcs go “left-to-right.”
4. **Dijkstra’s Algorithm.** Given a dictionary of dictionaries L representing an directed weighted graph G , and two vertices v and w of G , return a list of vertices giving a shortest path from v to w in the underlying undirected graph. Note that this list should start with v and end with w . If $v = w$, then the list should have a single element in it.
5. **Girth.** Given a dictionary L representing a directed graph, return the length of the smallest cycle in the underlying undirected graph. If the graph has no cycle, return -1 .
6. **Longest Tree Path.** Given a dictionary L representing a directed tree, return the length of the longest path in the underlying, undirected tree. Recall that the length of a path is the number of edges in that path.

Two-Person

7. **Linked List DSDS.** Implement a Disjoint Set Data Structure using linked lists, as in the text. You should create a class that provides the three usual functions, so that if d is a DSDS, one may call $d.Find_Set(x)$, $d.Make_Set(x)$ and $d.Union(x, y)$.
8. **Tree DSDS.** Implement a Disjoint Set Data Structure using rooted trees, as in the text, complete with path compression and union by rank. You should create a class that provides the three usual functions, so that if d is a DSDS, one may call $d.Find_Set(x)$, $d.Make_Set(x)$ and $d.Union(x, y)$.
9. **Kruskal.** Implement Kruskal's Algorithm, making use of the functions $Make_Set$, $Find_Set$ and $Union$, as described in the text. Your code should be sufficiently flexible to make use of others' implementations of these functions by importing that module.
10. **Min Priority Queue.** Implement the usual functions for a min priority queue built on a heap data structure. When the queue is first constructed a comparator function should be passed to the constructor so that the queue knows how to compare items.

12. **Strongly Connected Components.** Given a dictionary L representing a directed graph, return a DSDS giving the strongly connected components of the directed graph.
14. For future use...

One Person

20. **Prim's Algorithm.** Implement Prim's algorithm, making use of a min priority queue to find the light edge connecting a tree vertex to a non-tree vertex. Your code should be sufficiently flexible to make use of others' implementations of a min priority queue.
21. **Average Distance.** Given a dictionary representing a directed graph, return a list [avg, max] giving the average distance between two vertices in the graph and the maximum distance between two vertices in the graph.
22. **All Pairs Shortest Path.** Given a dictionary of dictionaries L representing an directed weighted graph G , return a dictionary of dictionaries giving a shortest path from v to w in the underlying undirected graph for every pair of vertices v and w . Thus the user ought to be able to construct the graph, call the function, and then ask for `d[3][12]` to find the length of the shortest path from vertex 3 to vertex 12 in the graph.
23. **Bipartite Graph.** Given a dictionary L representing a directed graph, decide whether the underlying, undirected graph is bipartite or not. If it is not, return -1 . If it is, return a list of two lists of vertices representing the two parts of the bipartition.
24. **Hamilton Circuit.** Given a dictionary L representing a directed graph, decide whether the underlying, undirected graph has a Hamilton circuit or not. If it does, return a list of vertices in the order of the circuit, starting and ending with the same vertex. If it does not, return -1 . (Note that the problem of deciding whether a graph has a Hamilton circuit or not is NP-complete, so that you cannot expect whatever code you write to work quickly for large graphs.)
25. **Euler Circuit.** Given a dictionary L representing a directed graph, decide whether the underlying, undirected graph has an Euler circuit or not. If it does not, return -1 . If it does, return a list of vertices representing an Euler circuit. Note that the list may repeat vertices, and that the last vertex in the list should be the same as the first vertex.
26. **Chinese Postman.** Given a dictionary of dictionaries representing a weighted, directed graph, find the length of the shortest tour in the graph that crosses each edge of the graph at least once. Note that this will require you to cooperate with the solver of problem 22, who will be required to provide you with an easy-to-use method to obtain the all-pairs-shortest-paths distance matrix. Your program does not need to produce the tour, just the length. I will be glad to help you devise an algorithm for this problem.
27. **Tree Center.** Given a dictionary L representing a directed tree, return the vertex that is the center of the underlying, undirected tree. The *center* of a tree is the vertex whose maximum distance to the other vertices in the tree is as small as possible.
28. **Tree Median.** Given a dictionary L representing a directed tree, return the vertex that is the median of the underlying, undirected tree. The *median* of a tree is the vertex whose average distance to the other vertices in the tree is as small as possible.

29. **Random Tree.** Write a program that generates a random tree, and outputs in our usual fashion — as a dictionary representing a (directed) graph. The module should provide a function `rand_tree(n)` that produces a random tree with n vertices. To test this function, you should have it generate 100 or so random trees on 7 vertices, and observe that all possible tree structures have appeared.

30.