Data Structures and Algorithms in Python

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Study Guide: Hints to Exercises

WILEY

Chapter 14 Graph Algorithms

Hints

Reinforcement

R-14.1) Use the drawing style of the book for the figure.

R-14.2) Recall that an n-vertex component of a simple undirected graph can have no more than $n(n-1)/2$ edges.

R-14.3) Fix a canonical ordering of the nodes.

R-14.4) Fix a canonical ordering of the nodes.

R-14.5) When drawing the Euler tour notice that every in has an out.

R-14.6) Recall the properties of inserting and removing elements in a doubly linked list.

R-14.7) Don't forget to update the edge list E.

R-14.8) Don't forget to update the edge list E.

R-14.9) How can the edges method be implemented?

R-14.10) How can the edges method be implemented?

R-14.11) The point of this exercise is to explore the trade-offs of the various representations. If you are having trouble, please review these trade-offs.

R-14.12) Review the pseudo-code while observing the different running times needed to implement each step when the graph is represented with an adjacency matrix.

R-14.13) Use a pencil and paper.

R-14.14) Work out an example with a complete graph of six vertices.

R-14.15) Work out an example with a complete graph of six vertices.

R-14.16) Follow the algorithm descriptions and drawing styles given in the book.

R-14.17) Mimic the drawing style used in the book.

R-14.18) If you wish, you may run our implementation to determine the order.

R-14.19) Are there any pairs of vertices that are not related?

R-14.20) Design a counting scheme that charges each level in T with the transitive closure edges that go from that level to lower levels.

R-14.21) Use the transitive closure algorithm given in the book.

R-14.22) Model the courses and their prerequisites as a directed graph.

R-14.23) Using the drawing style given in the book.

R-14.24) The essential property of course is to observe the edge directions. So work out the pseudo-code with this in mind.

R-14.25) Use the drawing style given in the book, and show the order in which the edges are added to the MST.

R-14.26) Use the drawing style given in the book, and show the order in which the edges are added to the MST.

R-14.27) Use an MST algorithm.

R-14.28) The black lines represent unexplored edges. What about the others?

R-14.29) Edges in the DFS tree are represented by solid lines. What about the others?

R-14.30) The black lines represent unexplored edges. What about the others?

R-14.31) Solid black lines are edges in the original input graph. What about the others?

R-14.32) Traversed edges are shown with dashed blue lines. What about the others?

R-14.33) Solid black edges have not been relaxed yet. What about the others?

R-14.34) Edges being considered in the current iteration for the MST are shown in thick blue lines. What about the others?

R-14.35) Edges being considered in the current iteration for the MST are shown in thick blue lines. What about the others?

R-14.36) Note how much time is spent in each step of George's algorithm.

Creativity

C-14.37) Make sure to remove any incident edges.

C-14.38) Make sure to remove the edge from the adjacency map for each incident vertex.

C-14.39) Use a data structure described in this book.

C-14.40) Suppose that such a nontree edge is a cross edge, and argue based upon the order the DFS visits the end vertices of this edge.

C-14.41) The DFS method must be changed so that the repetition stops when discovering v.

C-14.42) There has to be a good traversal that does this.

C-14.43) Maintain a set of all vertices upon which a recursive call to DFS is currently active.

C-14.44) Modify the dfs_complete function in order to tag each vertex with a component number when it is first discovered.

C-14.45) The solution involves a traversal discussed in this chapter.

C-14.46) Number the vertices 0 to $n-1$.

C-14.47) Suppose there is a forward edge or back edge and show why it would not be a nontree edge.

C-14.48) Suppose there is such an edge and show why it would not be a nontree edge.

C-14.49) Justify this by induction on the length of a shortest path from the start vertex.

C-14.50) Take each part in turn and use induction or contradiction as justification techniques.

C-14.51) Starting with the source vertex in the queue, repeatedly remove the vertex from the front of the queue and insert any of its unvisited neighbors to the back of the queue.

C-14.52) Do a BFS search in G.

C-14.53) Start out greedy and patch in the places this approach misses.

C-14.54) Perform "baby" searches from each station.

C-14.55) Try to compute the *diameter* of the tree T by a pruning procedure, starting at the leaves (external nodes).

C-14.56) Perform a traversal of T to compute distances to leaves.

C-14.57) What is the in-degree of the vertex labeled i in a compact graph?

C-14.58) Change the function to be optimized, but keep it indexed in terms of the a fixed listing of vertices.

C-14.59) Think about a shortest path with negative edges.

C-14.60) Be greedy.

C-14.61) Give G lots of edges and make every edge count for lots updates in the heap.

C-14.62) Examine closely the justification for why Dijkstra's algorithm works correctly and note the place where it breaks if negative-weight edges.

C-14.63) The greedy algorithm presented in this problem is not guaranteed to find the shortest path. Why not?

C-14.64) Maintain an auxiliary set of vertices that have already been discovered.

C-14.65) Think about the important fact about minimum spanning trees.

C-14.66) How does the number of connected components change from one iteration of Barůvka's algorithm to the next?

C-14.67) Use the tree-based union/find data structure from Chapter 10 along with one of the MST algorithms given in Chapter 12 and a good sorting algorithm.

C-14.68) This is not a shortest-path problem.

C-14.69) Use a greedy algorithm.

C-14.70) Model this problem as an optimal path problem that goes between two vertices in a directed graph without cycles.

C-14.71) Define an augmented graph from the input graph and perform the appropriate computation on the augmented graph.

C-14.72) If $M^2(i,j) = 1$, then there is a path of length $<= 2$ (a path traversing at most 2 edges) from vertex i to vertex j in the graph G. Alternatively, if $M^2(i,j) = 0$, then there is no such path. Now generalize from here...

C-14.73) The running time is more than linear.

Projects

P-14.74) Test the basic structure by having a method that visualizes the adjacency matrix.

P-14.75) Test the basic structure by having a method that visualizes the edge list.

P-14.76) Test the basic structure by having a method that visualizes the adjacency list.

P-14.77) Implement the static undirected version first, then add the extra methods.

P-14.78) Have some way of printing out each phase of the algorithm for debugging purposes.

P-14.79) Use uniform and Gaussian distributions for the edge weights.

P-14.80) You might use the `cs1graphics` Python module to implement the visualization.

P-14.81) Do a Breadth-First Search from each node in the network.