Algorithm Design and Analysis

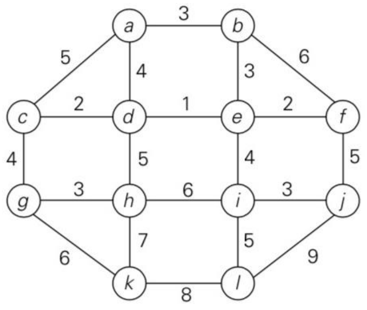
Assignment 04: Greedy Algorithms, Dynamic Programming

Total Points:	40
Assigned:	Wednesday, 13th March 2024
Due:	Sunday, 24th March 2024, 11:59pm.
Assignment Type:	Individual assignment
Submit:	Latex for Written Work

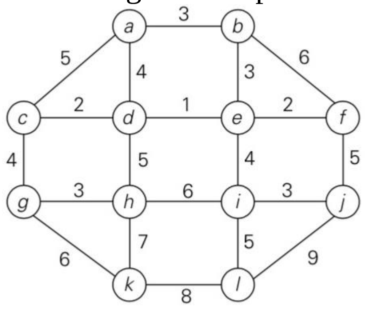
Preparation
Study the relevant sections of Chapters 8 & 9 of the textbook by Levitin
Instructions: Answer All questions.

PART A: Basic Practice with Key Algorithms

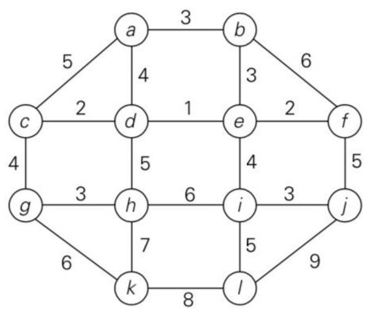
Problem 1 [3 points]: *Greedy Algorithms: Prim's Algorithm*
Execute Prim's algorithm to find the minimum spanning tree of the graph below. Break any ties among nodes alphabetically.



Problem 2 [3 points]: *Greedy Algorithms: Kruskal's Algorithm*
Execute Kruskal's algorithm to find the minimum spanning tree of the graph below. Break any ties among nodes alphabetically.



Problem 3 [3 points]: *Greedy Algorithms: Dijkstra's Algorithm*
Execute Dijkstra's algorithm on the graph below, using vertex ***a*** as the source vertex. Break any ties among nodes alphabetically.



.

Problem 4 [5 points]: *Greedy Approximation Algorithm*
Explain what adjustments, if any, that needs to be made in Dijkstra's algorithm and/or in an underlying graph to solve the following problems.

a. Solve the single-source shortest-paths problem for directed weighted graphs.

b. Find a shortest path between two given vertices of a weighted graph or digraph. (This variation is called the ***single-pair shortest-path problem***.)

c. Find the shortest paths to a given vertex from each other vertex of a weighted graph or digraph. (This variation is called the ***single-destination shortest-paths problem***.)

d. Solve the single-source shortest-paths problem in a graph with nonnegative numbers assigned to its vertices (and the length of a path defined as the sum of the vertex numbers on the path).

Problem 5 [8 points]: *Dynamic Programming: Knapsack Problem*

Item	Weight(g)	Profit
1	1	¢2
2	2	¢4
3	3	¢5
4	4	¢5
5	2	¢2
6	1	¢3

capacity W = 8.

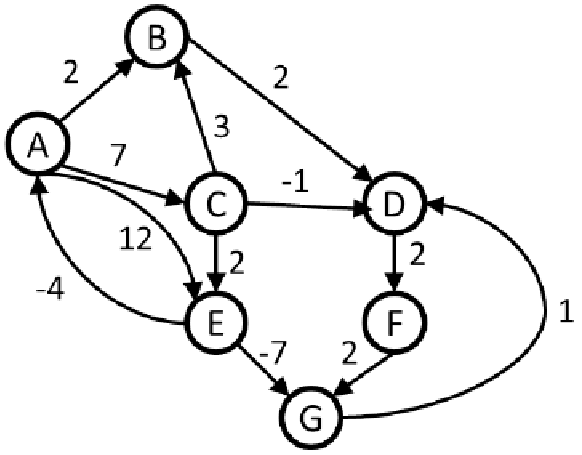
[4 points] Apply the bottom-up dynamic programming algorithm to the following instance of the knapsack problem:

a. [1 point] How many different optimal subsets does the instance of part (a) have?

b. [1 point] In general, how can we use the table generated by the dynamic programming algorithm to tell whether there is more than one optimal subset for the knapsack problem's instance?

c. [2 points] Instead of computing values in a bottom-up way, suppose we apply the memory function method to the instance of the knapsack problem given in part (a). Indicate the entries of the dynamic programming table that are (i) never computed by the memory function method, (ii) retrieved without a re-computation.

Problem 6 [8 points]



a. Even though the graph has negative weight edges, step through Dijkstra's algorithm to calculate supposedly shortest paths from A to every other vertex. Show your steps in the table below. Cross out old values and write in new ones, from left to right within each cell, as the algorithm proceeds. Also list the vertices in the order which you marked them known.

b. Dijkstra's algorithm found the wrong path to some of the vertices. For just the vertices where the wrong path was computed, indicate both the path that was computed and the correct path.

c. What single edge could be removed from the graph such that Dijkstra's algorithm would happen to compute correct answers for all vertices in the remaining graph?

Problem 7 [5 points]
Rod-cutting problem Design a dynamic programming algorithm for the following problem. Find the maximum total sale price that can be obtained by cutting a rod of *n* units long into integer-length pieces if the sale price of a piece *i* units long is *pi* for *i* = 1, 2, . . . , *n*. What are the time and space efficiencies of your algorithm?

Problem 8 [5 points]
Background
The following algorithm, from Section 12.3 of the Levitin textbook, is an example of a greedy approximation algorithm for the travelling salesman problem:

Multifragment-heuristic algorithm

Step 1 Sort the edges in increasing order of their weights. (Ties can be broken arbitrarily.) Initialize the set of tour edges to be constructed to the empty set.

Step 2 Repeat this step n times, where n is the number of cities in the instance being solved: add the next edge on the sorted edge list to the set of tour edges, provided this addition does not create a vertex of degree 3 or a cycle of length less than n; otherwise, skip the edge.

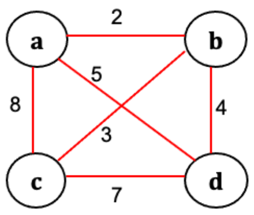
Step 3 Return the set of tour edges.

Task

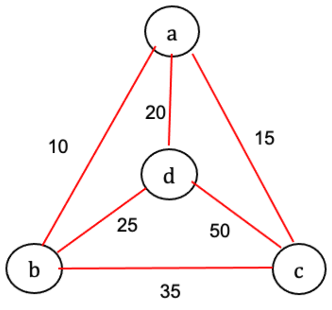
For each of the two instances of the traveling salesman problem given below, apply

(i) the greedy multifragment-heuristic algorithm listed above, and

(ii) the brute force approach, to solve the problem. In each case, compute the ratio between the value of the approximate solution found by the greedy approach and the optimal solution found by the brute force. This ratio is called the *accuracy ratio*.



(a)



(b)