Algorithm

Problem Statement: Consider that you are working for a delivery service, where you can go to the warehouse and pick items to deliver. Each item i has a cost c_i and weight w_i, as well as the address to where to deliver. You have a limit W on how much weight your van can carry. For every item you deliver, you earn 10% of its cost

You are also provided with a map of the town with all the possible addresses of the items in the warehouse, and the distances between them. Every mile you travel costs you $1.

For simplicity you can consider the items are packaged and cannot be divided further. Also there is a direct path between all pairs of addresses, although the distances may differ. Each address receives exactly one item.

Step 1: Get Input of the list of items into a list container or array of type Item where the class item contains three members of integer type; cost, weight and addressID, populate the member for each element of the array of items. Get the input of graph edges create an array of type edge, the class edge has three members: source, destination and cost for direct cost between two nodes, populate the members for each element in the array of edges.

Step 2: Create a set of unique addresses/address IDs extracted from the list of edges. Iterate through the list of edges and add to the set the destination and source node of each edge, the edge will only add unique elements. The set can then be used for its size to declare the new structures for problem solving.

Step 3: Create the adjacency matrix as a two dimensional array of type integer. Both dimensions of the array are equal the size of the set of addressIDs which are represented as nodes in the graph which the adjacency list represents.

- Construct the adjacency matrix with the list of edges; Initialize the matrix to infinity. populate the adjacency matrix by populating the source and destination index intersect element with the edge cost. Graph[source, destination] = cost.
- Do the same for the reciprocal elements in the adjacency matrix by flipping the indices to source, destination. Graph[destination, source] = cost.

Step 4: Extract the set of shipped items from the knapsack algorithm.

1. Initialize the knapsack table as a two dimensional array of rows which equals 1 greater the size of the list of items and columns 1 greater than the size of the capacity. The

columns represent the weights for which the knapsack is dynamically computed and the rows represent the items included.

2. Sort the array of items in ascending order. So that the knapsack can be performed to get the optimal profit of values.
3. Compute the optimum value for each item in each weight column. Iterate through each row of items in the knapsack table, and for each row iterate through the columns of weights. Fill each element using the equation:

   $V[i, w] = \max(V[i - 1, w], P_i + V[i - 1, w - w_i])$.
4. The equation gets the maximum of the previous optimum value in the previous item row and the new optimum calculated by adding the profit to the item from the previous row if the item exceeds the maximum optimum for the weight column.
5. If the weight of the item being considered exceeds the weight column that the item is being considered for. Get the optimal maximum profit in the last row $V[i, j] = V[i - 1, w]$.
6. The optimum value for the total weight capacity is the last element in the last row of the table.
7. Extract the set of included items from the knapsack table by reverse engineering what items were included. Start from the last element in the last row of the table. Keep track of the weight being considered, initialize it to the total capacity.

   i) Check if the optimal value for the weight is the same in the last row. If the value is same in the previous row go to the previous row and check if the previous row has the same optimal value of profit.

   ii) If it is not the same then that means the change was caused by the item being considered included the item in the set, subtract the item weight from the weight being considered and subtract the item cost from optimal cost being considered for the corresponding weight.

   iii) Loop till the weight considered is greater than 0.
8. Return the set of shipped items. These set of items give the maximum profit for the knapsack

Step 5: Compute the trip starting from the address of the first item in the set of items. The trip is computed by performing dijkstra's algorithm to get the minimum path from the source node to all other nodes in the graph.

1. The dijkstra's algorithm computes the minimum cost minimum spanning tree starting from a source node to minimum cost path to every other node.
2. The dijkstra algorithm creates an array of minimum distances for every other node and is initialized to infinity except for the distance of source node to itself which is 0. It also keeps a boolean array of added vertices in computing the shortest path. And keeps track of the parent nodes that are used to reach any given node. The Parent for source is assigned a sentinel value because it has no parent.
3. Dijkstra's computes the distance to each node, it updates with the minimum distance array values with the direct cost to each connected link all others are kept infinite.
4. Dijkstra's iterates through the set of vertices and adds the minimum distance vertex to be included and it then updates the distances for the direct cost links of the included node

with (direct cost + cost to reach the node) if they are less than the previous computed shortest distances.

5. The algorithm continues until all nodes are included and the minimum distance from each node is computed.

Step 6: The output of dijkstra's is then used to compute the path to the next address which is in the array of shortest distance, the array element Shortest-Distance[destination] gives the shortest path. And then the next address is reached and add the cost of traveling to the address to the total cost of the trip.

1. The algorithm also traces the path by tracing from destination to its parent node all the way back to parent using recursion.

Step 7: Compute dijkstra’s again for this address and then travel to the next address adding the cost of travel to the total cost. Continue this process until there are no items left and there is no address to travel to. Output the total cost at the end of the trip and calculate the profit by subtracting the total cost from the optimal value computed in the knapsack.

Example Problem:

Edges:

Source 1	Destination 2	Cost 2
Source 1	Destination 3	Cost 5
Source 1	Destination 4	Cost 1
Source 2	Destination 4	Cost 2
Source 2	Destination 3	Cost 3
Source 4	Destination 3	Cost 3
Source 4	Destination 5	Cost 1
Source 3	Destination 5	Cost 1
Source 3	Destination 6	Cost 5
Source 5	Destination 6	Cost 2
Source 5	Destination 1	Cost 8
Source 5	Destination 2	Cost 8
Source 6	Destination 1	Cost 8
Source 6	Destination 2	Cost 8
Source 6	Destination 4	Cost 9

Item Data:

Profit: 2;	Weight: 1;	AddressId: 3;
Profit: 5;	Weight: 2;	AddressId: 2;
Profit: 4;	Weight: 2;	AddressId: 4;
Profit: 7;	Weight: 2;	AddressId: 6;
Profitt: 10;	Weight: 3;	AddressId: 5;
Profit: 1;	Weight: 5;	AddressId: 1;

The table includes profit instead of cost Profit is obtained by dividing the cost of the item by 10 so since profit is 10%

Knapsack Table:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	0	0	0	0	0
1	0	2	2	2	2	2	2	2
2	0	2	5	7	7	7	7	7
3	0	2	5	7	9	11	11	11
4	0	2	7	9	12	14	16	18
5	0	2	7	10	12	17	19	22
6	0	2	7	10	12	17	19	22

Sent: Cost: 10;	Weight: 3;	AddressId: 5;
Sent: Cost: 7;	Weight: 2;	AddressId: 6;
Sent: Cost: 5;	Weight: 2;	AddressId: 2;

Graph Adjacency Matrix

	1	**2**	**3**	**4**	**5**	**6**
1	0	2	5	1	8	8
2	2	0	3	2	8	8
3	5	3	0	3	1	5
4	1	2	3	0	1	9
5	8	8	1	1	0	2
6	8	8	5	9	2	0

Dijkstra's Algorithm: Source 5

	1	**2**	**3**	**4**	**5**	**6**
5	8,5	8,5	1,5	1,5	0,5	2,5
5,3	6,3	4,3	1,5	1,5	0,5	2,5
5,3,4	2,4	3,4	1,5	1,5	0,5	2,5
5,3,4,6	2,4	3,4	1,5	1,5	0,5	2,5
5,3,4,6,2	2,4	3,4	1,5	1,5	0,5	2,5

Address	**Distance**	**Path**
5 → 6	2	5 6

Dijkstra's Algorithm: Source 6

	1	**2**	**3**	**4**	**5**	**6**
6	8,6	8,6	5,6	9,6	2,6	0,6
6,5	8,6	8,6	3,5	3,5	2,6	0,6
6,5,3	8,6	6,3	3,5	3,5	2,6	0,6
6,5,3,4	4,4	5,4	3,5	3,5	2,6	0,6
6,5,3,4,1	4,4	5,4	3,5	3,5	2,6	0,6

Address	**Distance**	**Path**
6 → 2	5	6 5 4 2

Cost of the entire trip: 7

Profit of the entire trip: 15