CSCI 3104 Spring 2022
Instructors: Profs. Chen and Layer

Problem Set 2

Due Date .... February 1, 2022
Name .... **Your Name**
Student ID .... **Your Student ID**
Collaborators .... **List Your Collaborators Here**

Contents

1 Instructions

- The solutions **should be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Canvas page** only. Please submit one PDF file, compiled using this LaTeX template.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit this document with no fewer pages than the blank template (or Gradescope has issues with it).
- You are welcome and encouraged to collaborate with your classmates, as well as consult outside resources. You must **cite your sources in this document. Copying from any source is an Honor Code violation. Furthermore, all submissions must be in your own words and reflect your understanding**

of the material. If there is any confusion about this policy, it is your responsibility to clarify before the due date.

- Posting to **any** service including, but not limited to Chegg, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.
- You **must** virtually sign the Honor Code (see Section 2). Failure to do so will result in your assignment not being graded.

2 Honor Code (Make Sure to Virtually Sign)

Problem 1.
- My submission is in my own words and reflects my understanding of the material.
- Any collaborations and external sources have been clearly cited in this document.
- I have not posted to external services including, but not limited to Chegg, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

Agreed (signature here). I agree to the above, Chengming Li □

3 Standard 5- BFS and DFS

3.1 Problem 2

Problem 2. Consider a Modified Connectivity problem:

- Instance: Let $G(V, E)$ be a simple, undirected graph. Let $s, t \in V(G)$.
- Decision: Given an integer $k \geq 1$, is there a shortest path from s to t in G that consists of k edges? Here the length/weight of a path is defined as the number of edges of the path.

Do the following. [**Note:** There are parts (a) and (b). Part (b) is on the next page.]

(a) Design an algorithm to solve the Modified Connectivity problem. Your solution should provide enough detail that a CSCI 2270 student could reasonably be expected to implement your solution.

Answer for Part (a). The way to solve finding the shortest path from s to t in G is using the algorithm of BFS.

- The main concept is that visiting all of the unvisited neighbors of the current node before visiting further away.
- Each node in the graph should at least have following components: neighbor, visited-status, prev-node.
- (1) This algorithm should create an empty set, Q(queue), which used to stored the visited nodes in the future and traverse the graph based on the first item in the Q. Moreover, the nodes will be placed in the queue, but they will also be removed from the queue when the algorithm is in the process.
- (2)The first node will be placed in the Q is s and its status will be marked as visited, its prev-node should be initialized to $NULL$.
- (3)The algorithm will jump into a *while* loop, and will end until the Q is an empty set, or queue.
- (4)At the beginning of each iteration, the first item in the Q will be polled from the Q, and then stored in a temporary variable, $current$. Then there is another for loop after that. The for loop will visited every unvisited neighbor node of $current$. In the order words, the algorithm will never visit the visited neighbor node again.
- (5)Once the algorithm jump into the current for loop, the algorithm should mark each unvisited neighbor node as *visited*, and put those unvisited neighbor of $current$ node into Q.
- (6)Inside the for loop, the algorithm will always check if we hit the target node t. If it is the target node, the algorithm will break the *while* loop.
- (7) If it is not the target node, the current for iteration should finish, the program will jump back to the *while* loop to check whether the condition, Q is not an empty set, still hold. If it still holds , repeated the step$(4) - (7)$ as stated above.
- (8) The shortest path can be found by back-traversed the prev-node of target node, because the shortest path is connected by the prev-node element from t back to s.

□

(b) We say that the graph G is *connected* if for every pair of vertices $s, t \in V(G)$, there exists a path from s to t. Design an algorithm to determine whether G is connected. Your algorithm should only traverse the graph once - this means that you should **not** apply BFS or DFS more than once. Your solution should provide enough detail that a CSCI 2270 student could reasonably be expected to implement your solution.

Answer for Part (b). The algorithm to solve this problem is using DFS to traverse the graph.

- The main concept to check if the graph is connected is making sure every node in the graph has been visited.
- For every node in the graph, they should at least have the following components: name, visited-status.
- (1) The DFS algorithm takes two input, Graph G and Vertex s. At the beginning of the function,the algorithm mark the current input Vertex s as visited.
- (2) Then, the function will traverse every unvisited neighbor node, v of s using for loop.
- (3) Inside the for loop, the algorithm will recursive call the DFS function itself with 2 input arguments, Graph G and Vertex v.
- (4) The algorithm will repeat $step(1) - (3)$ until there is no more unvisited neighbor node v of s in the graph. And the DFS algorithm function is finished.
- (5)Last,we should check the status of each node in the graph to see whether it has been visited. If every node in the graph has been visited, the conclusion is the graph G is connected. However, if there is node has not been visited, the conclusion is the graph G is not connected because there must be exist a pair of vertices $s, t \in V(G)$ that doesn't exist a path between them.

$\square$

3.2 Problem 3

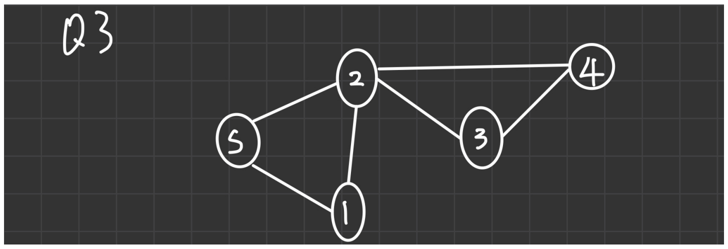
Problem 3. Give an example of a simple, undirected, and unweighted graph $G(V, E)$ that has a single source shortest path tree which a **depth-first traversal** will not return for any ordering of its vertices. Your answer must

(a) Provide a drawing of the graph G. [**Note:** We have provided TikZ code below if you wish to use LaTeX to draw the graph. Alternatively, you may hand-draw G and embed it as an image below, provided that (i) your drawing is legible and (ii) we do not have to rotate our screens to grade your work.]

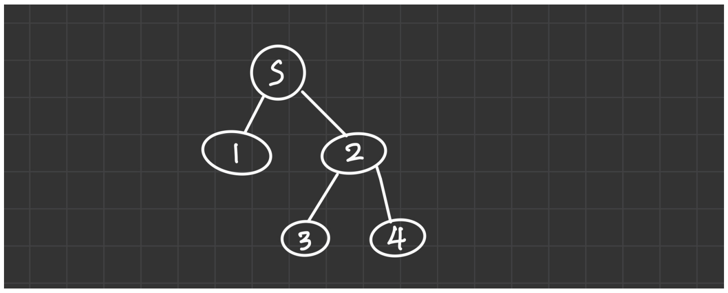
(b) Specify the single source shortest path tree $T = (V, E_T)$ by specifying E_T and also specifying the root $s \in V$. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

(c) Include a clear explanation of why the depth-first search algorithm we discussed in class will never produce T for any orderings of the vertices.

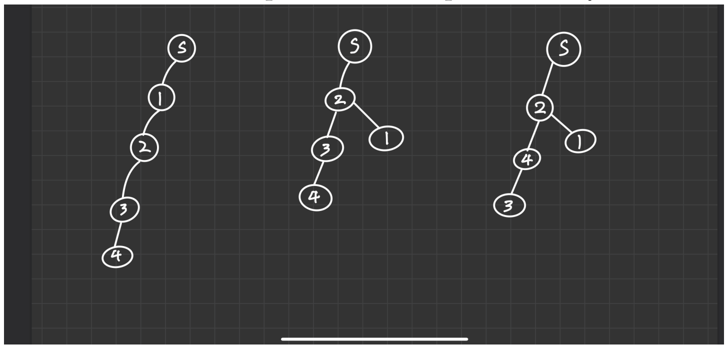
Answer. **Part a)**



Part b)



Part c) By the DFS, the single source shortest path tree will never produce T that seen in Part b), because the DFS algorithm will recursively traverse the unvisited neighbors of s until there is no more unvisited neighbors, or reach the dead-end. Even two vertices at the same depth of T will have different length from the tree. The DFS algorithm will traverse some unnecessary path to get to a specific vertex, which make the path from s to t is not the shortest.
In the order words, DFS algorithm aims to find the path as deep as possible first from the source vertex s even if there exist another way to hit the target vertex t. Moreover, the DFS algorithm will not end the recursion if it find the shortest path to a vertex t. The only condition to terminate the algorithm to produce tree is the dead-end is reached (i.e., a node whose neighbors have all be explored).
Here are a few example trees that produced by DFS based on the Graph in Part a):



□

3.3 Problem 4

Problem 4. Give an example of a simple, undirected, weighted graph such that a breadth-first traversal outputs a search-tree that is not a single source shortest path tree. (That is, BFS is not sufficiently powerful to solve the shortest-path problem on weighted graphs. This motivates Dijkstra's algorithm.) Your answer must

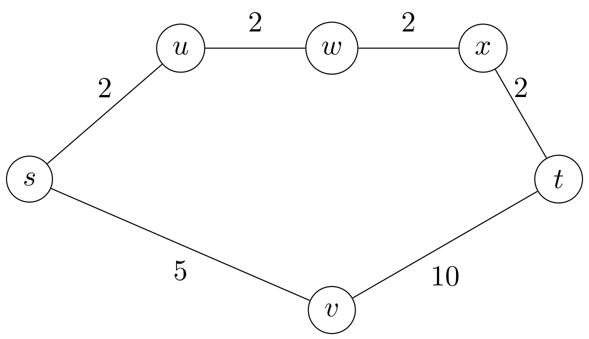
(a) Draw the graph $G = (V, E, w)$ by specifying V and E, clearly labeling the edge weights. [**Note:** We have provided TikZ code below if you wish to use LaTeX to draw the graph. Alternatively, you may hand-draw G and embed it as an image below, provided that (i) your drawing is legible and (ii) we do not have to rotate our screens to grade your work.]

(b) Specify a spanning tree $T(V, E_T)$ that is returned by BFS, but is not a single-source shortest path tree. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

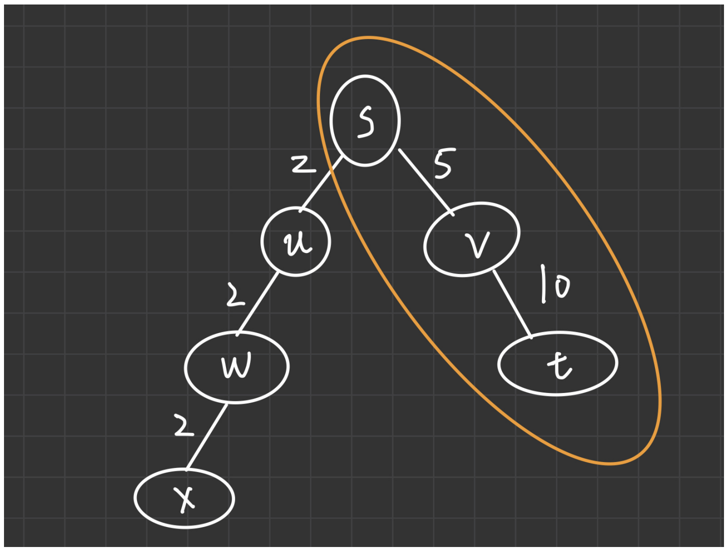
(c) Specify a valid single-source shortest path tree $T' = (V, E_{T'})$. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

(d) Include a clear explanation of why the search-tree output by breadth-first search is not a valid single-source shortest path tree of G.

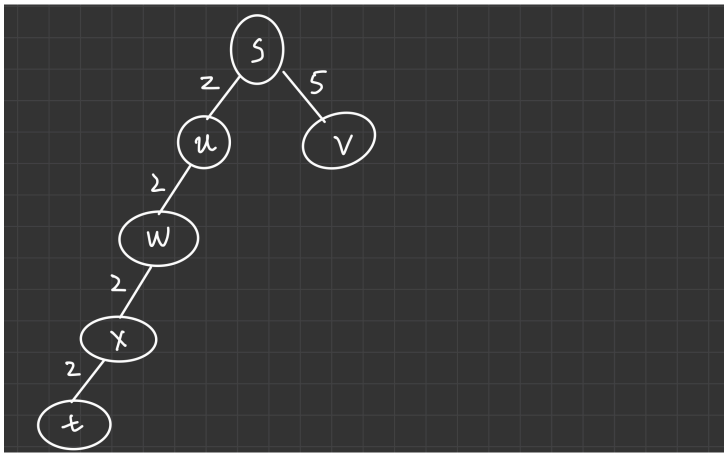
Answer. **Part a)**



Part b)



Part c)



Part d) By breadth-first search tree, the graph shown in above will not generate a valid single-source shortest path tree of G. Because, the path $s- > v- > t$ with weight = 15 will hit the vertex, t earlier than the path $s- > u- > w- > x$ with weight = 8. And since the path $s- > v- > twithweight = 15$ find the target vertex, t, earlier, it will break the loop and finish the function so that single-source shortest path tree $T' = (V, E_{T'})$ wouldn't be able to found it by the BFS. The changing of queue in BFS at each iteration can be seen below:

- Q = $\{s\}$
- Q = $\{u, v\}$
- Q = $\{v, w\}$
- Q = $\{w, t\}$
- Iteration break due to target vertex t has been found. And the spanning tree $T(V, E_T)$ that is returned by BFS is not a single-source shortest path tree, ie.the path $s- > v- > t$ with weight = 15 will be returned by BFS.

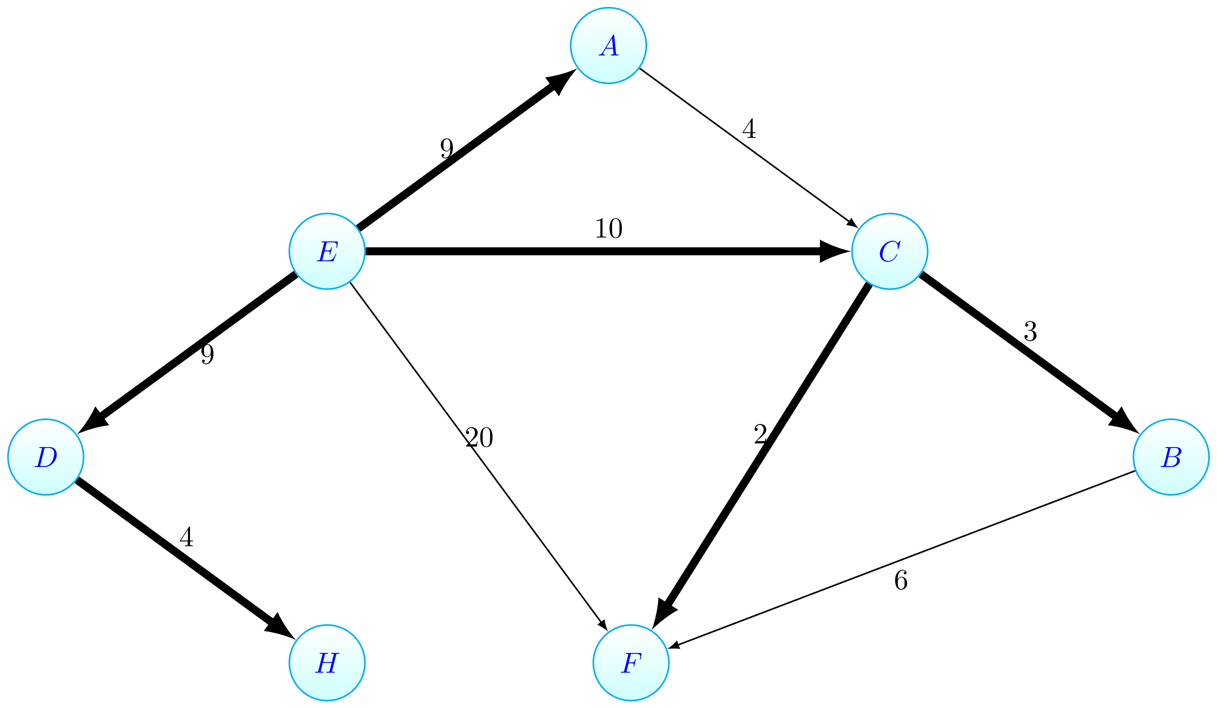
□

4 Standard 6- Dijkstra's Algorithm

4.1 Problem 5

Problem 5. Consider the weighted graph $G(V, E, w)$ pictured below. Work through Dijkstra's algorithm on the following graph, using the source vertex E.

- Clearly include the contents of the priority queue, as well as the distance from E to each vertex at each iteration.
- If you use a table to store the distances, clearly label the keys according to the vertex names rather than numeric indices (i.e., `dist['B']` is more descriptive than `dist['1']`).
- You do **not** need to draw the graph at each iteration, though you are welcome to do so. [This may be helpful scratch work, which you do not need to include.]



Answer.

Vertex	Dist from E	Prev-Vertex
A	∞	NULL
B	∞	NULL
C	∞	NULL
D	∞	NULL
E	0	NULL
F	∞	NULL
H	∞	NULL

PQ = [(E,0)]

Vertex	Dist from E	Prev-Vertex
A	9	E
B	∞	NULL
C	10	E
D	9	E
E	0	NULL
F	20	E
H	∞	NULL

PQ = [(A,9),(D,9),(C,10),(F,20)]

Vertex	Dist from E	Prev-Vertex
A	9	E
B	∞	NULL
C	10	E
D	9	E
E	0	NULL
F	20	E
H	∞	NULL

PQ = [(D,9),(C,10),(F,20)]

Vertex	Dist from E	Prev-Vertex
A	9	E
B	∞	NULL
C	10	E
D	9	E
E	0	NULL
F	20	E
H	13	D

PQ = [(C,10),(F,20),(H,13)]

Vertex	Dist from E	Prev-Vertex
A	9	E
B	13	C
C	10	E
D	9	E
E	0	NULL
F	12	C
H	13	D

PQ = [(F,12),(H,13),(B,13)]

Vertex	Dist from E	Prev-Vertex
A	9	E
B	13	C
C	10	E
D	9	E
E	0	NULL
F	12	C
H	13	D

PQ = [(H,13),(B,13)]

Vertex	Dist from E	Prev-Vertex
A	9	E
B	13	C
C	10	E
D	9	E
E	0	NULL
F	12	C
H	13	D

PQ = [(B,13)]

Vertex	Dist from E	Prev-Vertex
A	9	E
B	13	C
C	10	E
D	9	E
E	0	NULL
F	12	C
H	13	D

PQ = [ ] Algorithm terminates

□

4.2 Problem 6

Problem 6. You have three batteries, with capacities of 40, 25, and 16 Ah (Amp-hours), respectively. The 25 and 16-Ah batteries are fully charged (containing 25 Ah and 16 Ah, respectively), while the 40-Ah battery is empty, with 0 Ah. You have a battery transfer device which has a "source" battery position and a "target" battery position. When you place two batteries in the device, it instantaneously transfers as many Ah from the source battery to the target battery as possible. Thus, this device stops the transfer either when the source battery has no Ah remaining or when the destination battery is fully charged (whichever comes first).

But battery transfers aren't free! The battery device is also hooked up to your phone by bluetooth, and automatically charges you a number of dollars equal to however many Ah it just transfered.

The goal in this problem is to determine whether there exists a sequence of transfers that leaves exactly 10 Ah either in the 25-Ah battery or the 16-Ah battery, and if so, how little money you can spend to get this result.

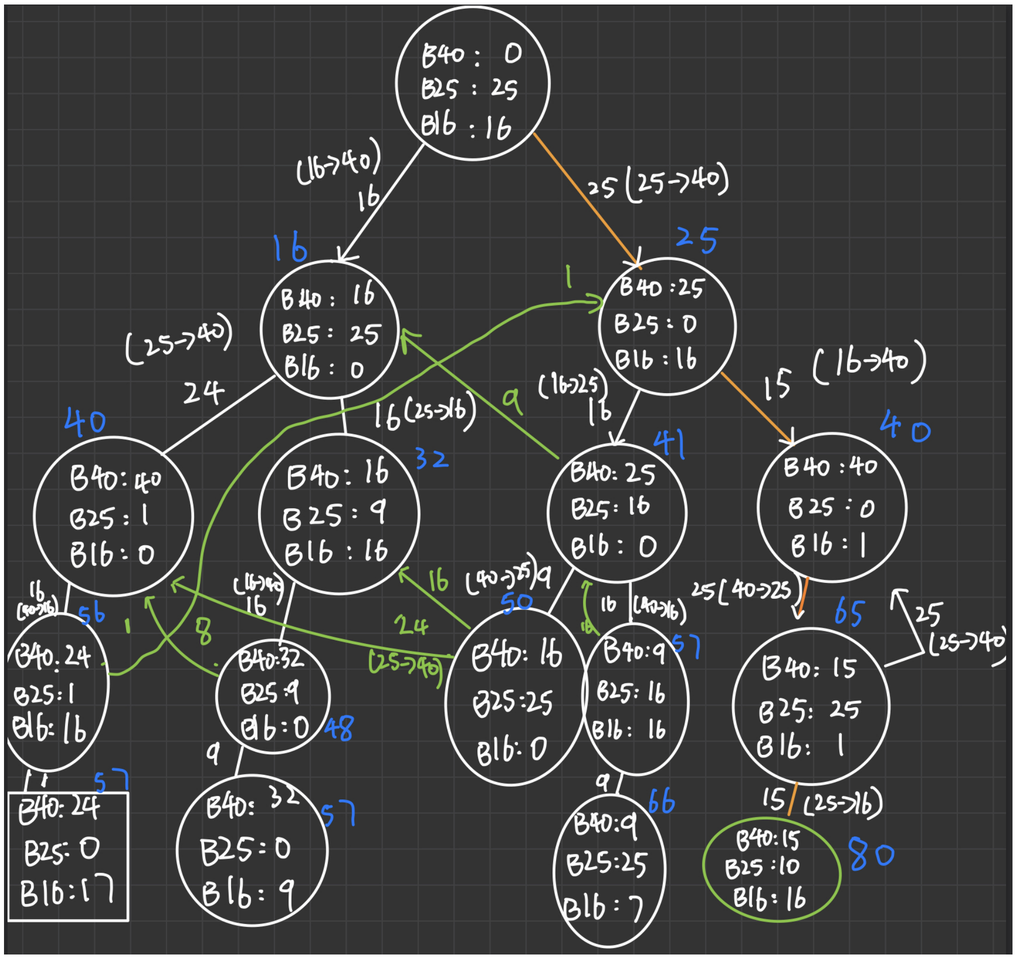
Do the following.

4.2.1 Problem 6(a)

(a) Rephrase this is as a graph problem. Give a precise definition of how to model this problem as a graph, and state the specific question about this graph that must be answered. [**Note:** While you are welcome to draw the graph, it is enough to provide 1-2 sentences clearly describing what the vertices are and when two vertices are adjacent. If the graph is weighted, clearly specify what the edge weights are.]

Answer.

- In the graph, source node can be defined$(0, 25, 16)$ as 40Ah battery has 0Ah, 25Ah battery has 25Ah and 16Ah battery has 16Ah. And target node can be defined$(15, 10, 16)$ as 40Ah battery has 15Ah, 25Ah battery has 10Ah and 16Ah battery has 16Ah
- Each node contains the power left in each battery, there would be three values in each node. And Each node should also contains an element that stores the information of visited status.
- The edge weights can be considered as the cost of number of dollars, or the Ah it just transfered between two vertices.
- When there is a transfer between two vertices, we could consider it as two vertices are adjacent.
- The question can be considered as finding the shortest path from source node to target node.
- The Graph can be seen below:


B40: 0
B25: 25
B16: 16
(16->40) 16
25 (25->40)
16
B40: 16
B25: 25
B16: 0
25
B40: 25
B25: 0
B16: 16
(25->40) 24
16 (25->16)
9
(16->25) 16
15 (16->40)
40
B40: 40
B25: 1
B16: 0
32
B40: 16
B25: 9
B16: 16
41
B40: 25
B25: 16
B16: 0
40
B40: 40
B25: 0
B16: 1
56
B40: 24
B25: 1
B16: 16
B40: 32
B25: 9
B16: 0
48
50
B40: 16
B25: 25
B16: 0
57
B40: 9
B25: 16
B16: 16
65
B40: 15
B25: 25
B16: 1
57
B40: 24
B25: 0
B16: 17
57
B40: 32
B25: 0
B16: 9
66
B40: 9
B25: 25
B16: 7
80
B40: 15
B25: 10
B16: 16
25 (40->25)
25 (25->40)
15 (25->16)
(40->25) 9
(40->16) 16
(25->40) 24
(16->40) 16
9
9
8
1
1
16 (40->16)

□

4.2.2 Problem 6(b)

(b) Clearly describe an algorithm to solve this problem. If you use an algorithm covered in class, it is enough to state that. If you modify an algorithm from class, clearly outline any modifications. Make sure to explicitly specify any parameters that need to be passed to the initial function call.

Answer.

- The key idea to solve this problem is the same as using Dijkstra Algorithm to find the shortest path from source vertex to a target vertex.
- This algorithm require the Graph G is connected and has no negative-weight edges. The weighted Graph(V, E, w) and source Vertex s need to be passed to the initial function call.
- (1)First, the algorithm will initialize an empty Priority Queue, Q. Then, for each vertex in the graph, the algorithm will initialize the $dist$ to be ∞, $processed$ to $false$ and $predecessor$ to $NULL$. The first element will be placed in the Q is the source node s, as (0,25,16). The distance of s would be 0.
- Next, the algorithm should create a $While$ loop to iterate until the Priority Queue Q becomes an empty set. Inside the $while$ loop, (2) the algorithm will poll the first element,$current$ from the Q and make it to be processed.
- Another loop (3)inside the $while$ loop is a for loop, which will iterate each unprocessed neighbor v of $current$. In this for loop, we will check if the $current$ vertex $dist + dist(current, v)$ is less than the v vertex $dist$. (4)If it is less than the v $dist$, the vdist will be updated to the $current$ vertex $dist + dist(current, v)$,$vpredecessor$ to $current$, and those information will also updated to the Priority Queue Q If it is not, the for loop will keep iterate until $step(4)$ condition appearing or there is no more unprocessed neighbor vertex v.
- Repeated $step(2) - (4)$ Until the Priority Queue is an empty set. Then, the algorithm will return the updated Graph,G.
- The cost of finding this transfer can be found on the $dist$ element of target vertex(that leaves exactly 10 Ah either in the 25-Ah battery or the 16-Ah battery). And the path of transfer can be found by back-traversing the $predecessor$ of target vertex.

□

4.2.3 Problem 6(c)

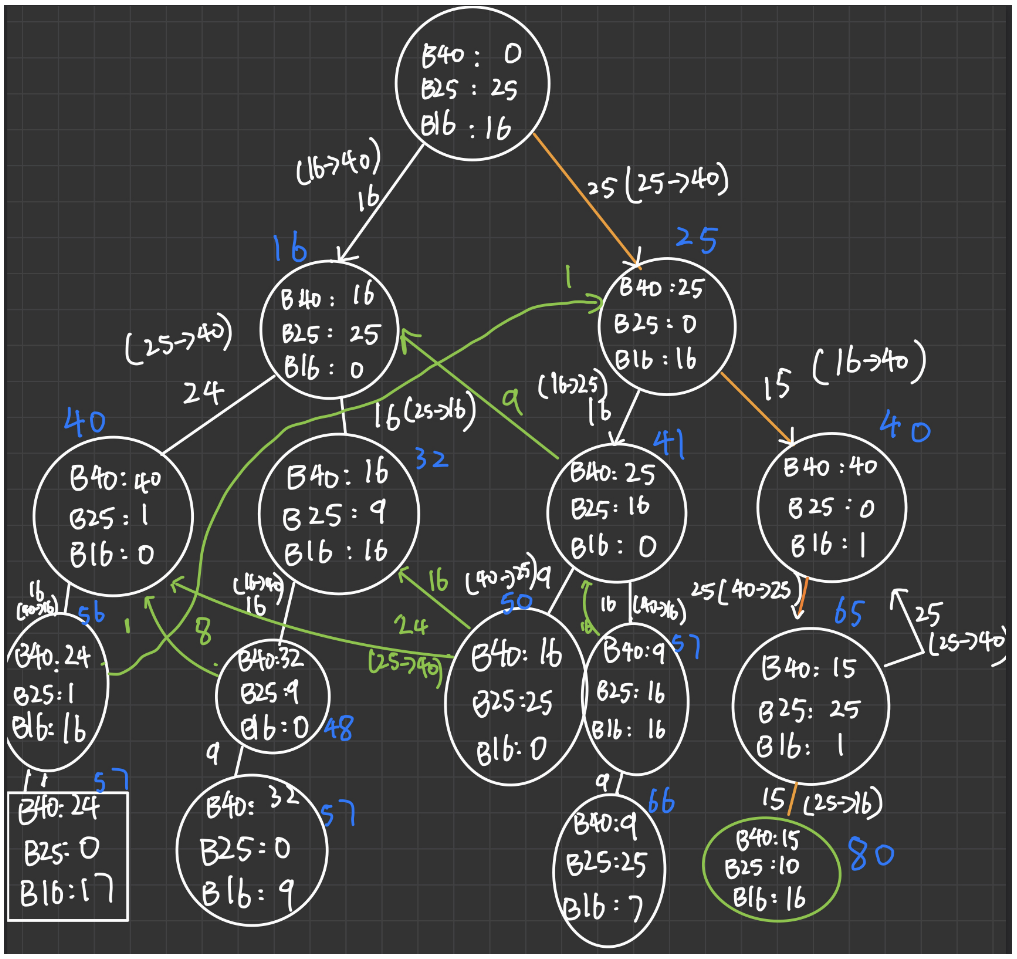
(c) Apply that algorithm to the question. Report and justify your answer. Here, justification includes the sequences of vertices visited and the total cost.

Answer. The sequences of vertices visited

- **PQ** = [**((0,25,16),0)**]
- **PQ** = [**((16,25,0),16) ,((25,0,16),25)**]
- **PQ** = [**((25,0,16),25),((16,9,16),32) ,((40,1,0),40)**]
- **PQ** = [**((16,9,16),32) ,((40,1,0),40),((40,0,1),40),((25,16,0),41)**]
- **PQ** = [**((40,1,0),40),((40,0,1),40),((25,16,0),41),((32,9,0),48)**]
- **PQ** = [**((40,0,1),40),((25,16,0),41),((32,9,0),48),((24,1,16),56)**]
- **PQ** = [**((25,16,0),41),((32,9,0),48),((24,1,16),56),,((15,25,1),65)**]
- **PQ** = [**((32,9,0),48),((24,1,16),56),((15,25,1),65),((16,25,0),50),((9,16,16),57)**]
- **PQ** = [**((24,1,16),56),((15,25,1),65),((16,25,0),50),((9,16,16),57),((32,0,9),57)**]
- **PQ** = [**((15,25,1),65),((16,25,0),50),((9,16,16),57),((32,0,9),57),((24,0,17),57)**]
- **PQ** = [**((16,25,0),50),((9,16,16),57),((32,0,9),57),((24,0,17),57),((15,10,16),80)**]
- **PQ** = [**((9,16,16),57),((32,0,9),57),((24,0,17),57),((15,10,16),80)**]
- **PQ** = [**((32,0,9),57),((24,0,17),57),((15,10,16),80),((9,25,7),66)**]
- **PQ** = [**((24,0,17),57),((15,10,16),80),((9,25,7),66)**]
- **PQ** = [**((15,10,16),80),((9,25,7),66)**]
- **PQ** = [**((9,25,7),66)**]
- **PQ** = [ ] Algorithm terminates

The total cost to find vertex(15,10,16) = 80Ah

After applying the algorithm to the question, the graph can be found below:



B40 : 0
B25 : 25
B16 : 16
(16->40)
16
25 (25->40)
16
B40 : 16
B25 : 25
B16 : 0
25
B40 : 25
B25 : 0
B16 : 16
(25->40)
24
40
B40: 40
B25 : 1
B16 : 0
B40: 16
B25 : 9
B16 : 16
32
16 (25->16)
9
(16->25)
16
41
B40: 25
B25: 16
B16 : 0
15 (16->40)
40
B40 : 40
B25 : 0
B16 : 1
25 (40->25)
65
B40: 15
B25: 25
B16: 1
25
(25->40)
56
B40: 24
B25: 1
B16: 16
B40:32
B25:9
B16:0
48
50
B40: 16
B25:25
B16: 0
57
B40:9
B25: 16
B16: 16
57
B40: 24
B25: 0
B16: 17
57
B40: 32
B25 : 0
B16 : 9
66
B40:9
B25:25
B16 : 7
15 (25->16)
80
B40:15
B25 :10
B16: 16

□