CSCI 3104 Spring 2022
Instructors: Profs. Chen and Layer

Problem Set 2

Due Date ..... February 1, 2022
Name ..... **Julia Troni**
Student ID **109280095**
Collaborators ..... **Me, myself and I**

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.

I agree to the above, Julia Troni. □

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 following pseudo code shows an algorithm to solve the Modified Connectivity Problem. The function *ModifiedConnectivity* uses a modified BFS algorithm to find the shortest path and check if it is equal to k. It takes input of the graph G, the starting vertex s, the vertex we wish to traverse to t, and k, the number of edges of the path. It will return true the path is equal to k, otherwise it will return false.

- Begin by initializing 3 arrays a state array to hold if a vertex has been visited, a predecessor array that will track the parent node along the path, and a distance array to save how many edges it took to reach that vertex. Set all vertices to NEW (unvisited), all predecessors to NULL, and all distances to infinity
- Then set the distance of the start vertex, s, to 0.
- Make a FIFO queue with the start vertex and that predecessor, which is NULL.
- Then while the queue is not empty, dequeue a vertex, if it is marked as NEW, set it to OLD, save the predecessor, and increment the distance by 1. Then for each neighbor of that vertex, add it to the queue. In this way we are traversing all the connected elements of graph, and due to the FIFO queue, we are traversing layer by layer and hence finding the shortest path.
- Once the while loop completes, check if the goal vertex, t, has been marked OLD and if that the distance is equal to k. If so, return true. Otherwise, return false.

```
#G is a simple undirected graph
#s is the starting vertex in G
#t is the vertex that we wish to traverse to
#k is the length of the path from s to t
#ModifiedConnectivity will return true if there is a shortest path from vertex s to t with
                                                    k edges
def ModifiedConnectivity(Graph G, Vertex s, Vertex t, Integer k) {
  for i to n {
    state[i] = NEW  #mark all vertices as unvisited
    pred[i] = NULL    #initialize array to hold predecessors
    dist[i] = infinity  #set all distances to infinity
  }

  dist[s] = 0   #distance of s set to 0
  Q= newQueue(NULL, s)    #make a queue, with pred[s] and s. Very important that queue is
                                                  FIFO

  while Q is not empty {
    (p,x)= deQueue(Q)   #get vertex x out of Q
    if state[x]==NEW {
      state[x]=OLD  #mark it as visited
      pred[x]=p   #save how we got to x
```

```
      dist[x]=dist[p]+1   #store the distance to x
      for each neighbor y of x {
        enqueue(Q, (x,y))   #add y to the Q
      }
    }
  }

  if (state[t] == OLD && dist[t] ==k) {   #if t has been visited and the distance to t is
                                          the same as k, then there does exist a
                                          path from s to t in G that consists of k
                                          edges
    return true
  }
  else {
    return false
  }
}
```

□

(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 following pseudo code shows an algorithm to determine if G is connected. The function *CheckConnected* takes input of the graph G, the starting vertex, and the total number of nodes in the graph and it will return true if the graph is connected, otherwise it will return false.

- Begin by initializing a count variable to 0. Then initialize a state array where the state of all the vertices in the graph is NEW, meaning unvisited.
- Make a queue containing the start vertex, the queue can be FIFO or LIFO because either BFS or DFS will traverse all connected elements.
- Then while the queue is not empty, dequeue a vertex. If it marked NEW, set it to OLD and increment the count variable by 1 and for each neighbor of that vertex, add it to the queue. In this way we are traversing all the elements of graph that are connected to the start vertex.
- Once the while loop completes, check if the count is equal to the number of nodes. If so, it indicates that we were able to reach every vertex from the starting vertex and thus the graph is connect so return true. If not, return false.

```
#G is a graph that may or may not be connected
#s is the starting vertex in G
#NumNodes is the total number of nodes in a graph
#CheckConnected will return true if the graph is connected, otherwise it will return false
def CheckConnected(Graph G, Vertex s, Integer NumNodes) {
  Int count = 0      #initialize counting variable that will hold the count of the number
                                                  of visited nodes
  for i to NumNodes {
    state[i] = NEW  #mark all vertices as unvisited
  }

  Q= newQueue(s)    #make a queue with s

  while Q is not empty {
    x= deQueue(Q)   #get vertex x out of Q
    if state[x]==NEW {
      state[x]=OLD  #mark it as visited
      count+=1    #increment the count of visited nodes
      for each neighbor y of x {
        enqueue(Q, (y))   #add y to the Q
      }
    }
  }

  if (count ==NumNodes) {   #if the count of visited nodes is the same as the total number
                                                  of nodes in the graph, then the graph is
                                                  connected
    return true
  }
  else {
    return false
  }
}
```

□

3.2 Problem 3

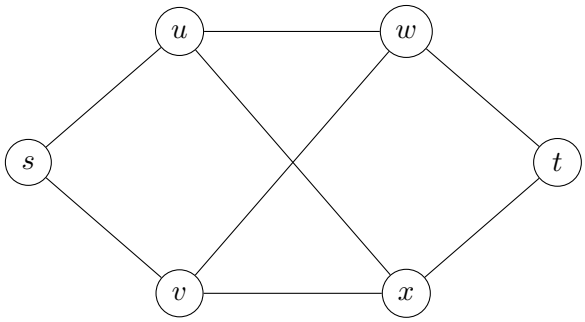
Problem 3. Give an example of a simple, undirected, and unweighted graph $G(V, E)$ that has a single source shortest path tree which a **depth-first traversal** will not return for any ordering of its vertices. Your answer must

(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.]

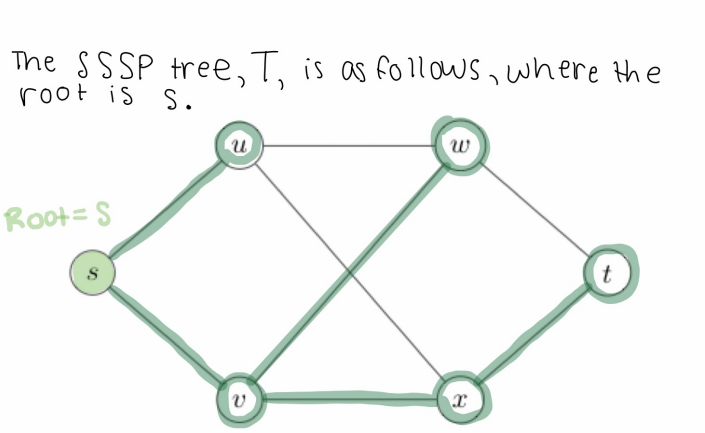
(b) Specify the single source shortest path tree $T = (V, E_T)$ by specifying E_T and also specifying the root $s \in V$. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

(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.

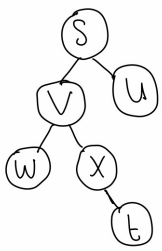
Answer. (a) Let the graph G be the following



(b) Let the root be s and the single source shortest path tree, T is depicted below. The edges E_T are highlighted in dark green



Or more clearly...



(c) A traditional DFS algorithm will never produce T for any ordering of the vertices due to the way in which DFS traverses a graph. DFS uses a LIFO queue so it starts from a starting node, s, and recursively traverses the unvisited neighbors of s. This looks like: start at s, arbitrarily choose a neighbor to visit, call it m, then choose an arbitrary neighbor of m, say n to visit, and repeats until it cannot find any more unvisited neighbors. At that point, backtrack until finding another unvisited neighbor. The result of this are long stringy trees.

□

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

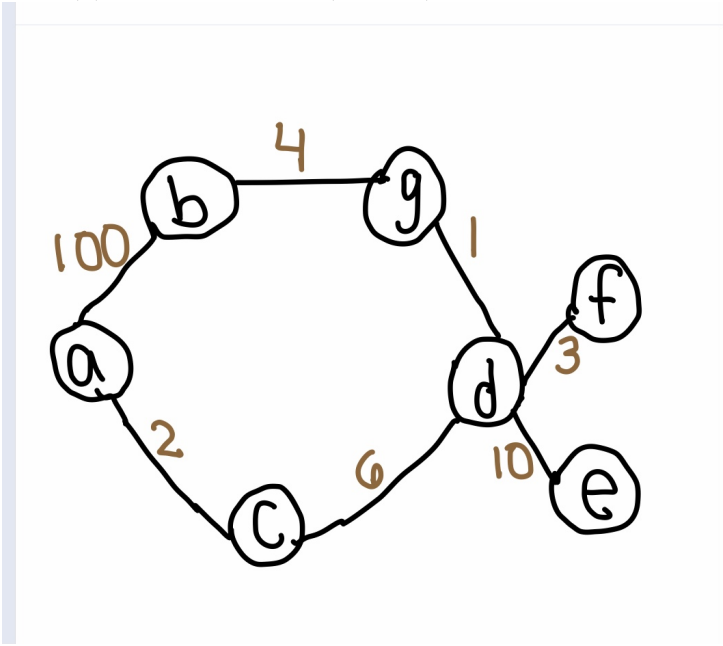
(a) Draw the graph $G = (V, E, w)$ by specifying V and E, clearly labeling the edge weights. [**Note:** We have provided TikZ code below if you wish to use LaTeX to draw the graph. Alternatively, you may hand-draw G and embed it as an image below, provided that (i) your drawing is legible and (ii) we 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.]

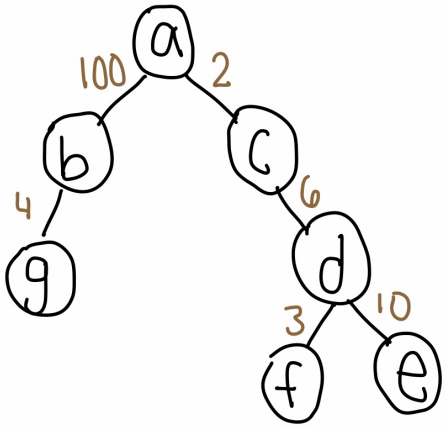
(c) Specify a valid single-source shortest path tree $T' = (V, E_{T'})$. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

(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.

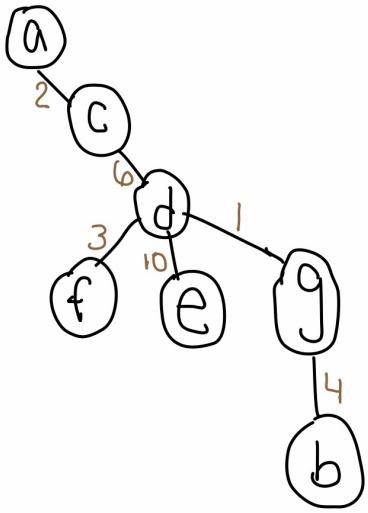
Answer. (a) The graph $G = (V, E, w)$ is depicted below



(b) A spanning tree $T(V, E_T)$ that is returned by BFS, but is not a single-source shortest path tree is depicted below. Let vertex A be the starting vertex.



(c) A valid single-source shortest path tree $T' = (V, E_{T'})$ is depicted below. Let vertex A be the starting vertex.



(d) The search-tree output by breadth-first search is not a valid single-source shortest path tree of G because BFS only produces valid SSSP trees for UNWEIGHTED graphs, or if all of the weights of edges are the same. This is because BFS counts the number of edges, rather than the weight of the edges. BFS finds all the vertices that are 1 edge away from the starting vertex, then 2 edges away, etc until it has visited all vertices. Hence, BFS minimizes the number of edges that are traversed, rather than considering the weight of the edges.
So, in the graph above, where A is the starting vertex, BFS finds the following paths:

- A to B: $A - B$ with a weight of 100
- A to G: $A - B - G$ with a weight of 104
- A to C: $A - C$ with a weight of 2
- A to D: $A - C - D$ with a weight of 8
- A to F: $A - C - D - F$ with a weight of 11
- A to E: $A - C - D - E$ with a weight of 18

However, the correct shortest paths (which Dijkstra's correctly finds) are:

- A to B: $A - C - D - G - B$ with a weight of 13
- A to G: $A - C - D - G$ with a weight of 9
- A to C: $A - C$ with a weight of 2
- A to D: $A - C - D$ with a weight of 8
- A to F: $A - C - D - F$ with a weight of 11
- A to E: $A - C - D - E$ with a weight of 18

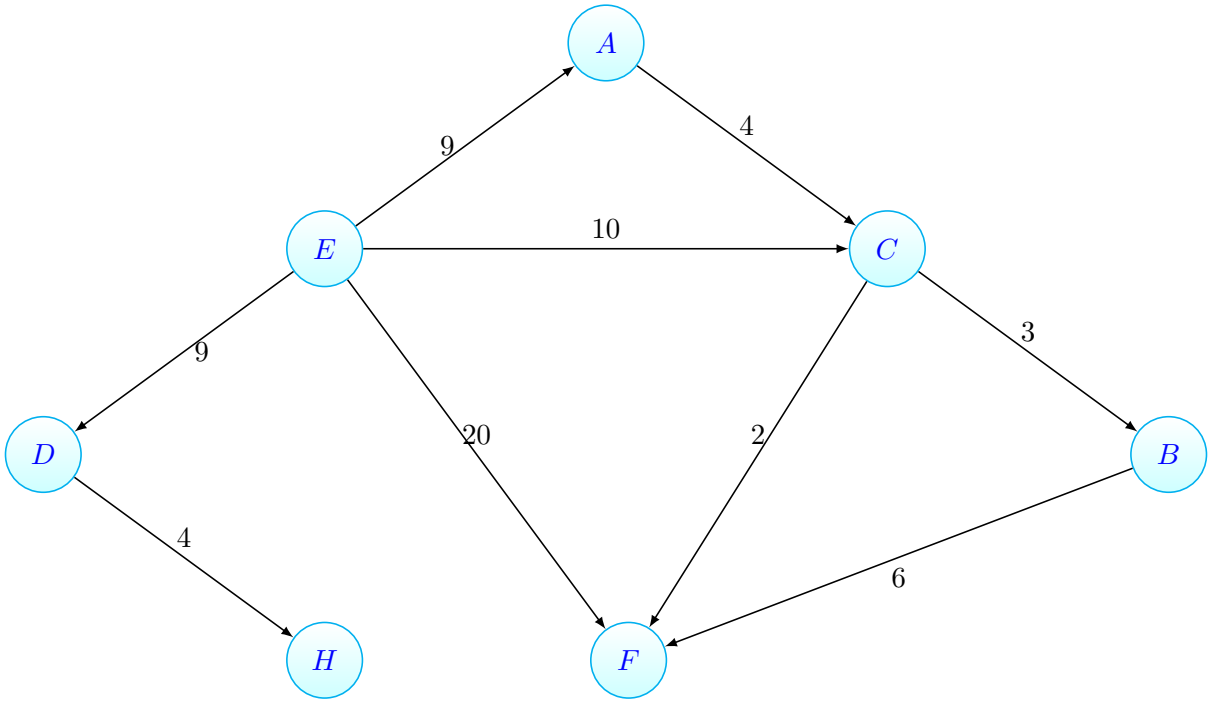
□

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. We invoke $Dijkstra(G, E)$, where E is the starting vertex

1. For all vertices except E, set the distance to ∞. Set $E.dist = 0$ and push E on the priority queue so that $Q = [(E, 0)]$
2. Poll E. Set E.processed= true. The unprocessed neighbors of E are A, C, F, D.
   - $w(E, A) = 9 < \infty$ so $A.dist = 9$ and $A.predecessor = E$
   - $w(E, C) = 10 < \infty$ so $C.dist = 10$ and $C.predecessor = E$
   - $w(E, F) = 20 < \infty$ so $F.dist = 20$ and $F.predecessor = E$
   - $w(E, D) = 9 < \infty$ so $D.dist = 9$ and $D.predecessor = E$
     Push A, C, F, D to the priority queue so $Q = [(A, 9), (D, 9), (C, 10), (F, 20)]$
3. Poll A from the priority queue and set A.processed=true. The unprocessed neighbor of A is C.
   - $dist(E, A) + w(A, C) = 9 + 4 > 10$ so we make no further changes to C
     Now the priority $Q = [(D, 9), (C, 10), (F, 20)]$
4. Poll D from the priority queue and set D.processed=true. The unprocessed neighbor of D is H.

- $dist(E, D) + w(D, H) = 9 + 4 < \infty$ so $H.dist = 13$ and $H.predecessor = D$
  Push H to the priority queue so $Q = [(C, 10), (H, 13), (F, 20)]$

5. Poll C from the priority queue and set C.processed=true. The unprocessed neighbors of C are B and F.

   - $dist(E, C) + w(C, B) = 10 + 3 < \infty$ so $B.dist = 13$ and $B.predecessor = C$
   - $dist(E, C) + w(C, F) = 10 + 2 < 20$ so $F.dist = 12$ and $F.predecessor = C$
     Push B to the priority queue and reorder so now $Q = [(F, 12), (H, 13), (B, 13)]$

6. Poll F from the priority queue and set F.processed=true. The unprocessed neighbor of F is B.

   - $dist(E, F) + w(F, B) = 12 + 6 > 13$ so we make no further changes to B
     Now $Q = [(H, 13), (B, 13)]$

7. Poll H from the priority queue and set H.processed=true. There are no unprocessed neighbors of H so $Q = [(B, 13)]$

8. Poll B from the priority queue and set B.processed=true. As there are no unprocessed neighbors of B and the queue is empty, so the algorithm terminates

Dijkstra's found the shortest paths from E to each vertex.

- E to A: $E - A$ which has a weight 9.
- E to D: $E - D$ which has a weight 9.
- E to C: $E - C$ which has a weight 10.
- E to F: $E - C - F$ which has a weight 12.
- E to B: $E - C - B$ which has a weight 13.
- E to H: $E - D - H$ which has a weight 13.

□

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.

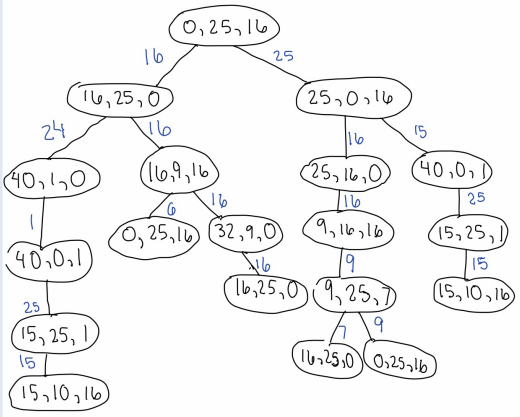
Do the following.

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. Given a graph $G(V, E, W)$ where each vertex V is a tuple holding the charge of the batteries such that each V = [Charge of $40Ah$, Charge of $25Ah$, Charge of $16Ah$]. When two vertices are adjacent, this represents a batter transfer so each edge, E represents 1 battery transfer such that the weight of the edge indicates the number of Ah transferred. So each edge weight = amount of Ah transferred.
The goal is to find the shortest weight path from the start vertex $[0, 25, 16]$ to a vertex where the 2nd or the 3rd element is 10. That is, find the shortest path from $[0, 25, 16]$ to $[xx, 10, xx]$ or $[xx, xx, 10]$, where xx represents some arbitrary number.

A representation of the graph is seen below



□

4.2.2 Problem 6(b)

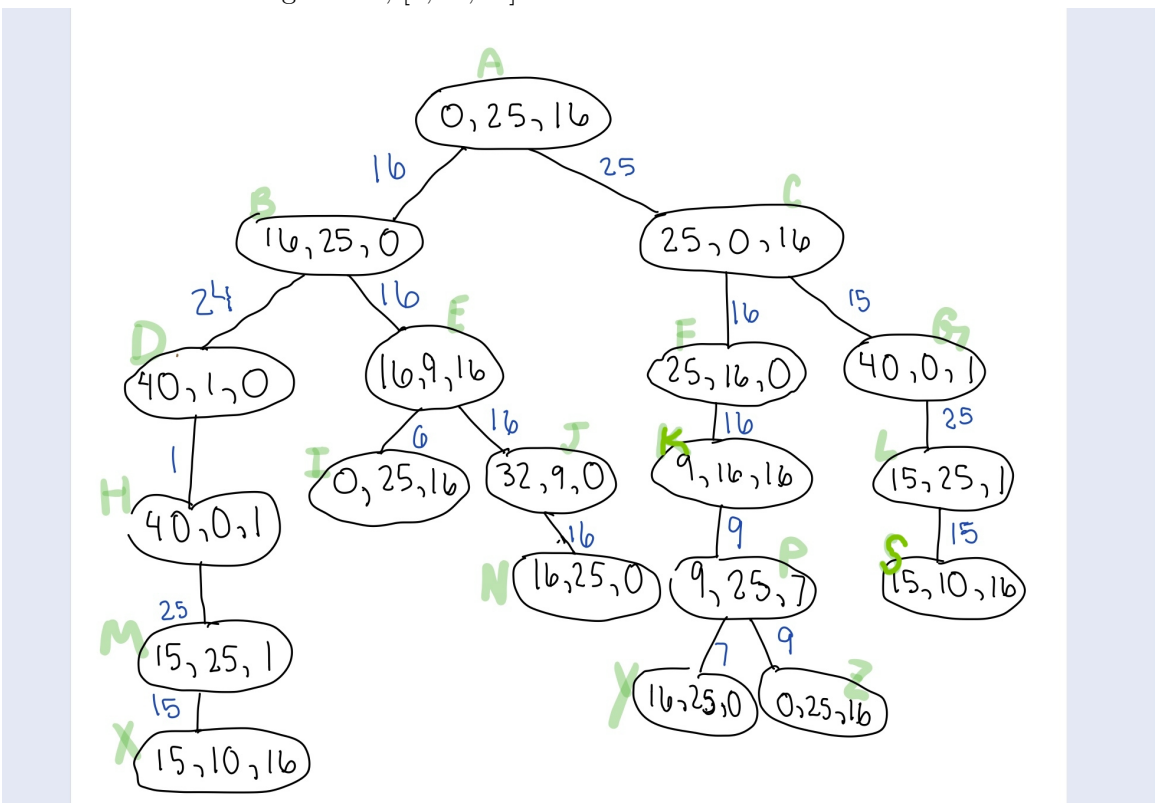
(b) Clearly describe an algorithm to solve this problem. If you use an algorithm covered in class, it is enough to state that. If you modify an algorithm from class, clearly outline any modifications. Make sure to explicitly specify any parameters that need to be passed to the initial function call.

Answer. Dijkstra's algorithm can be used to solve this problem. Call Dijkstra's from the starting vertex $[0, 25, 16]$. After Dijkstra's terminates, we will have the shortest path to all vertices. But since there will be multiple vertices with the form $[xx, 10, xx]$ or $[xx, xx, 10]$, we will compare the weight of each of them and choose the lowest one. □

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. Using the following graph, where I have labeled the vertices for simplicity, we invoke $Dijkstra(G, A)$, where A is the starting vertex, $[0, 25, 16]$



1) For all vertices except A, set the distance to ∞. Set $A.dist = 0$ and push A on the priority queue so that $Q = [(A, 0)]$
2) Poll A. Set A.processed= true. The unprocessed neighbors of A are B, C.
   - $w(A, B) = 16 < \infty$ so $B.dist = 16$ and $B.predecessor = A$
   - $w(A, C) = 25 < \infty$ so $C.dist = 10$ and $C.predecessor = A$ Push B, C to the priority queue so $Q = [(B, 16), (C, 25)]$
3) Poll B from the priority queue and set B.processed=true. The unprocessed neighbors of B are D, E.
   - $dist(A, B) + w(B, D) = 16 + 24 < \infty$ so $D.dist = 40$ and $D.predecessor = B$
   - $dist(A, B) + w(B, E) = 16 + 16 < \infty$ so $E.dist = 32$ and $E.predecessor = B$
     Now the priority $Q = [(C, 25), (D, 40), (E, 32)]$
4) Poll C from the priority queue and set C.processed=true. The unprocessed neighbors of C are F, G.
   - $dist(A, C) + w(C, F) = 25 + 16 < \infty$ so $F.dist = 41$ and $F.predecessor = C$
   - $dist(A, C) + w(C, G) = 25 + 15 < \infty$ so $G.dist = 40$ and $G.predecessor = C$
     Push F, G to the priority queue so $Q = [(D, 40), (E, 32), (G, 40), (F, 41)]$
5) Poll D, mark it as processed, add H to the priority queue. $Q = [(E, 32), (G, 40), (F, 41), (H, 41)]$
6) Poll E, mark it as processed add I and J to the priority queue. $Q = [(I, 38), (G, 40), (F, 41), (H, 41), (J, 48)]$
7) Poll I, mark it as processed, since it has no neighbors, nothing is added. $Q = [(G, 40), (F, 41), (H, 41), (J, 48)]$

8) Poll G, mark it as processed, add L to the priority queue. $Q = [(F, 41), (H, 41), (J, 48), (L, 65)]$
9) Poll F, mark it as processed, add K to the priority queue. $Q = [(H, 41), (J, 48), (K, 57), (L, 65)]$
10) Poll H, mark it as processed, add M to the priority queue. $Q = [(J, 48), (K, 57), (L, 65), (M, 66)]$
11) Poll J, mark it as processed, add N to the priority queue. $Q = [(K, 57), (N, 64), (L, 65), (M, 66)]$
12) Poll K, mark it as processed, add P to the priority queue. $Q = [(N, 64), (L, 65), (M, 66), (P, 66)]$
13) Poll N, mark it as processed, since it has no neighbors, nothing is added. $Q = [(L, 65), (M, 66), (P, 66)]$
14) Poll L, mark it as processed, add S to the priority queue. $Q = [(M, 66), (P, 66), (S, 80)]$
15) Poll M, mark it as processed, add X to the priority queue. $Q = [(P, 66), (S, 80), (X, 81)]$
16) Poll P, mark it as processed, add Y and Z to the priority queue. $Q = [(Y, 73), (Z, 75), (S, 80), (X, 81)]$
17) Poll Y, mark it as processed, since it has no neighbors, nothing is added. $Q = [(Z, 75), (S, 80), (X, 81)]$
18) Poll Z, mark it as processed, since it has no neighbors, nothing is added. $Q = [(S, 80), (X, 81)]$
19) Poll S, mark it as processed, since it has no neighbors, nothing is added. $Q = [(X, 81)]$
20) Poll X, mark it as processed, since it has no neighbors, nothing is added. Now the queue is empty and the algorithm terminates.

Dijkstra's found the shortest paths from A, which is $[0, 25, 16]$ to each vertex in the graph. Since we are only interested in the vertices that are $[xx, 10, xx]$ or $[xx, xx, 10]$, we will only examine those. The vertices of that form correspond to X and S, which are $[15, 10, 16]$ and $[15, 10, 16]$.

- A to X, which is $[0, 25, 16]$ to $[15, 10, 16]$: has a weight 81 and the path is $[0, 25, 16] - [15, 25, 0] - [40, 1, 0] - [40, 0, 1] - [15, 25, 1] - [15, 10, 16]$.
- A to S, which is $[0, 25, 16]$ to $[15, 10, 16]$: has a weight 80 and the path is $[0, 25, 16] - [25, 0, 16] - [40, 0, 1] - [15, 25, 1] - [15, 10, 16]$.

Since $80 < 81$, the cheapest sequence of transfers to result with 10Ah in the 25Ah or 16Ah battery is $[0, 25, 16] - [25, 0, 16] - [40, 0, 1] - [15, 25, 1] - [15, 10, 16]$. Thus 80 is the cheapest way to achieve the goal. □