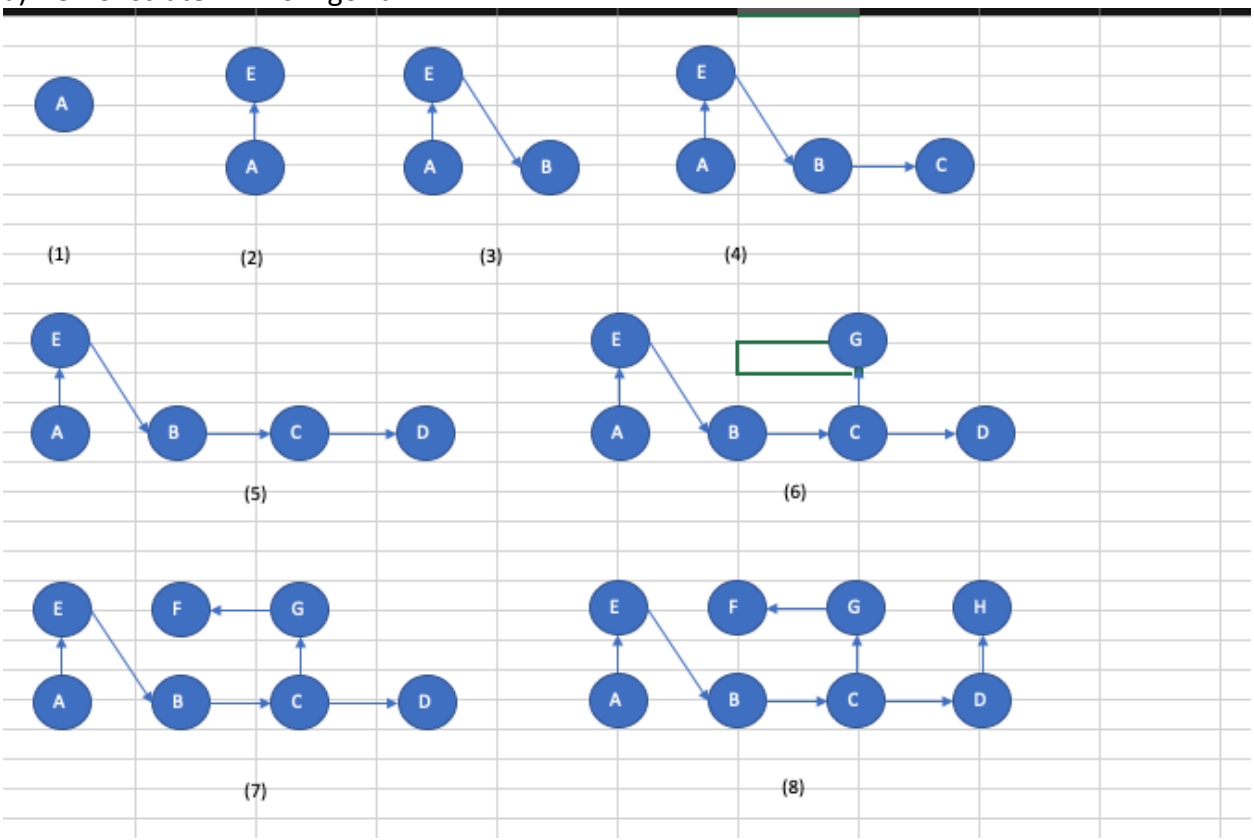
Question 1:

a) Demonstrate Prim's Algorithm:




Note: Arrows are only to emphasize order in which nodes were selected. This is not directional graph.

Order in which edges are added:

(A) -> (A, E) -> (E,B) -> (B,C) -> (C, D) -> (C, G) -> (G, F) -> (D,H)

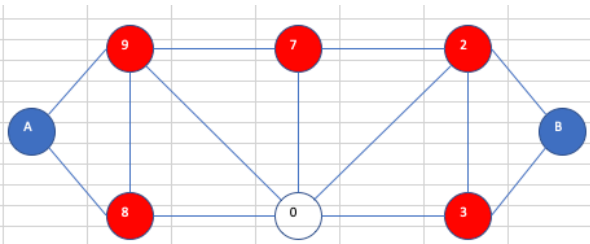
b) Yes. The overall weight of the MST will change by 1 * (n-1), where n is the number of vertices in the MST.

Question 2:

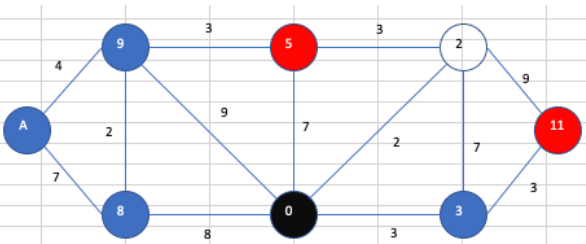
a) We will use Dijkstra's to calculate to solve the single source shortest path problem from G.

Note: For graph in question two:

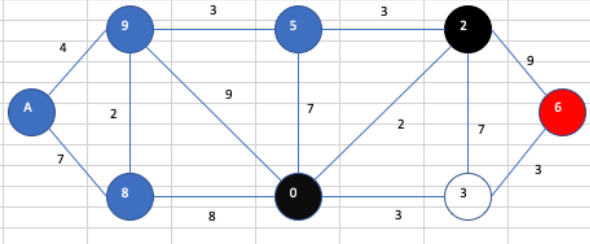
1. White colored Vertex is the current node being processed
2. Red indicates a change in the weight of vertices adjacent to the current vertex being inspected
3. Black colored vertex indicates the node has been inspected


9
7
2
A
B
8
0
3

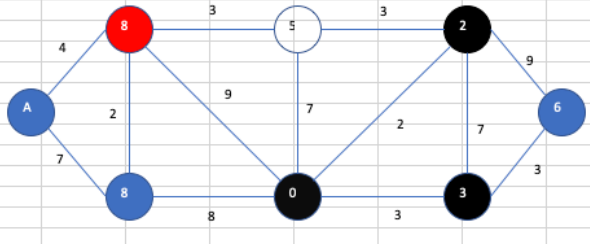
(1)


3
3
4
9
5
2
9
A
2
9
7
11
2
7
7
8
0
3
3
8
3

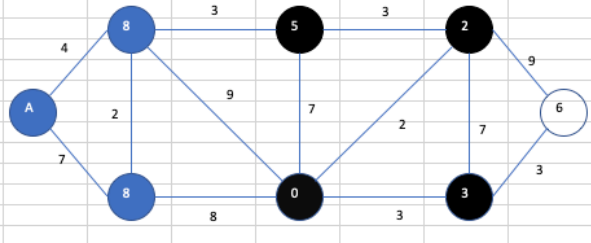
(2)


3
3
4
9
5
2
9
A
2
9
7
6
2
7
7
8
0
3
3
8
3
3

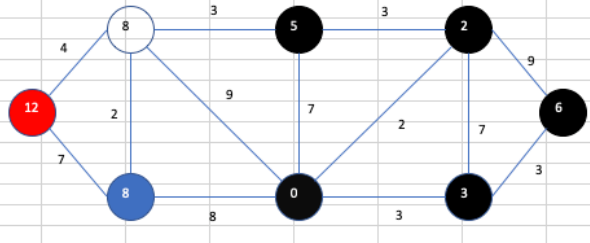
(3)


3
3
4
8
5
2
9
A
2
9
7
6
2
7
7
8
0
3
3
8
3

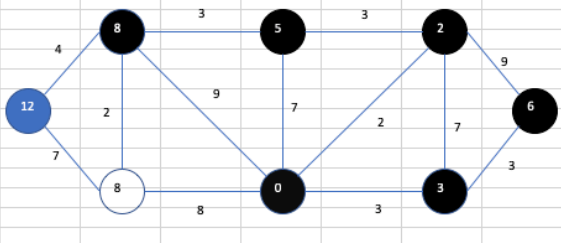
(4)


3
3
4
8
5
2
9
A
2
9
7
6
2
7
7
8
0
3
3
8
3

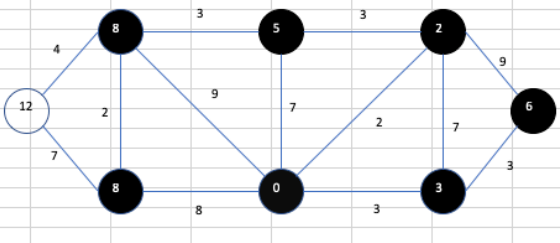
(5)


3
3
4
8
5
2
9
12
2
9
7
6
2
7
7
8
0
3
3
8
3

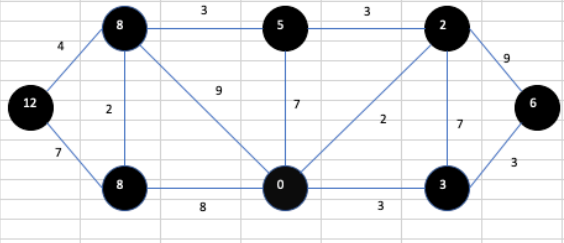
(6)


3
3
4
8
5
2
9
12
2
9
7
6
2
7
7
8
0
3
3
8
3

(7)


3
3
4
8
5
2
9
12
2
9
7
6
2
7
7
8
0
3
3
8
3

(8)

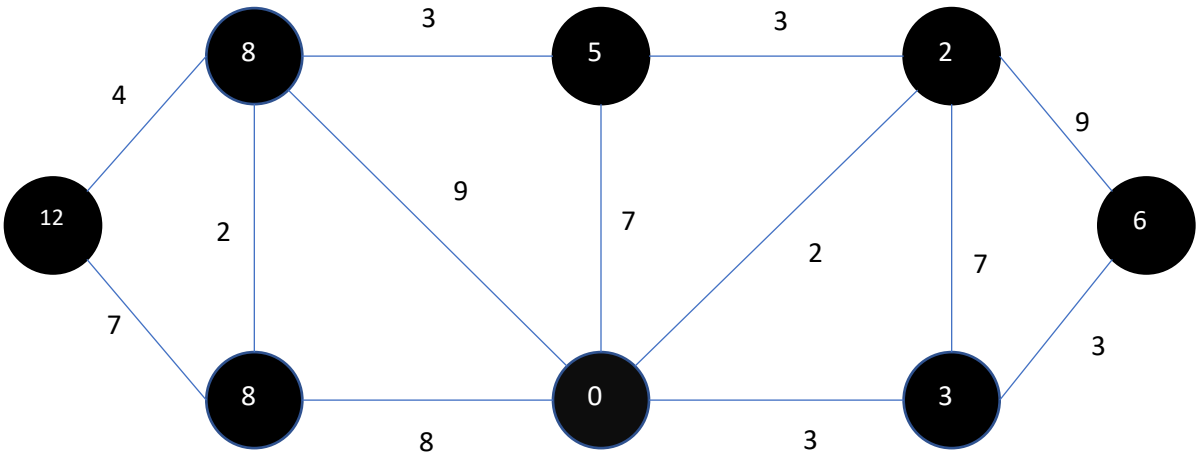

3
3
4
8
5
2
9
12
2
9
7
6
2
7
7
8
0
3
3
8
3

(9)

b) In order to find the “optimal”, or most central vertex, we need to find a vertex whose furthest distance node is the minimum of all the starting nodes. Our algorithm will run Dijkstra’s for each vertex and calculate the single source shortest path for each.

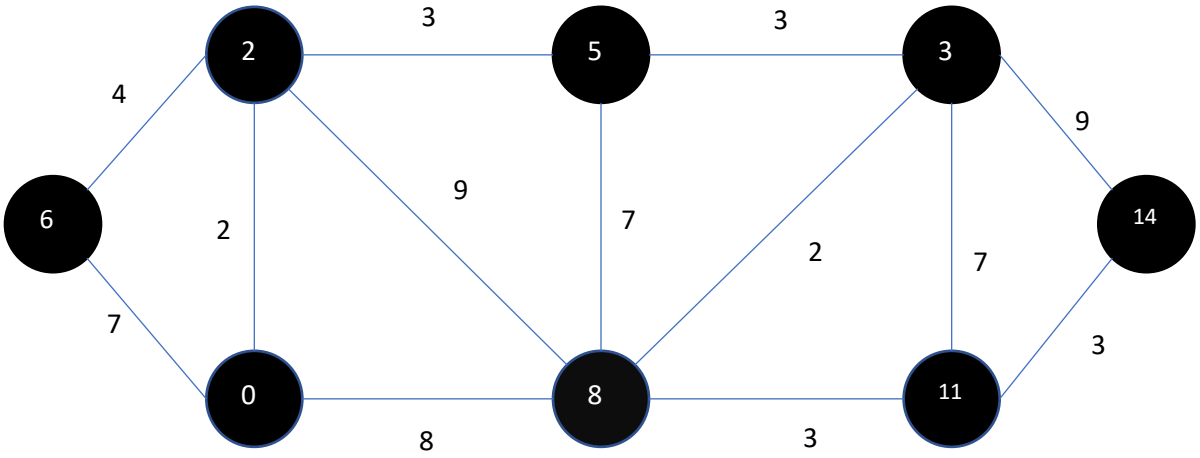
```
Optimal(G){
  Let curr − Min be the distance of furthest vertex of the optimal start vertex
  Let opt − vertex be the optimum vertex
  curr − Min = ∞
  opt − vertex = NIL
  for each v ∈ G.V{
    Let T = G
    furthest = 0
    Dijkstra(T, w, v)
    for each v ∈ T.V{
      if v.d > furthest:
        furthest = v.d
    }

    if furthest ≤ curr − Min{
      curr − Min = furthest
      opt − vertex = v
    }
  }
  return opt − vertex
}
```

c. Optimal Location:
Initially I chose the optimal vertex to be the one with lowest average distance. However, after consulting with my group, we determined that the optimal vertex is the vertex whose furthest node is smallest of all starting vertices.
Node E is the optimal distance since its furthest distant node was 10.



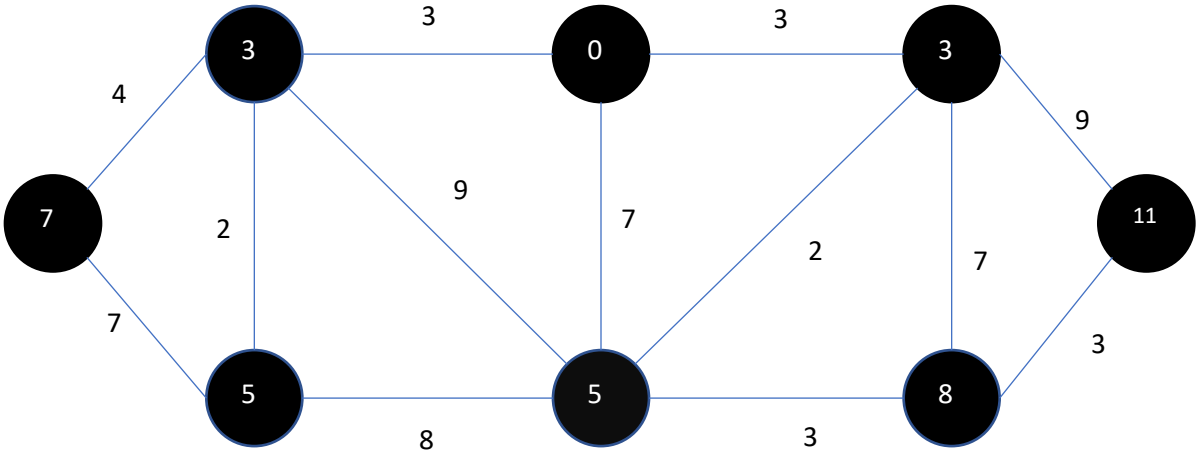

Dijkstra(G)
average_path=6.28

Dijstra(F)



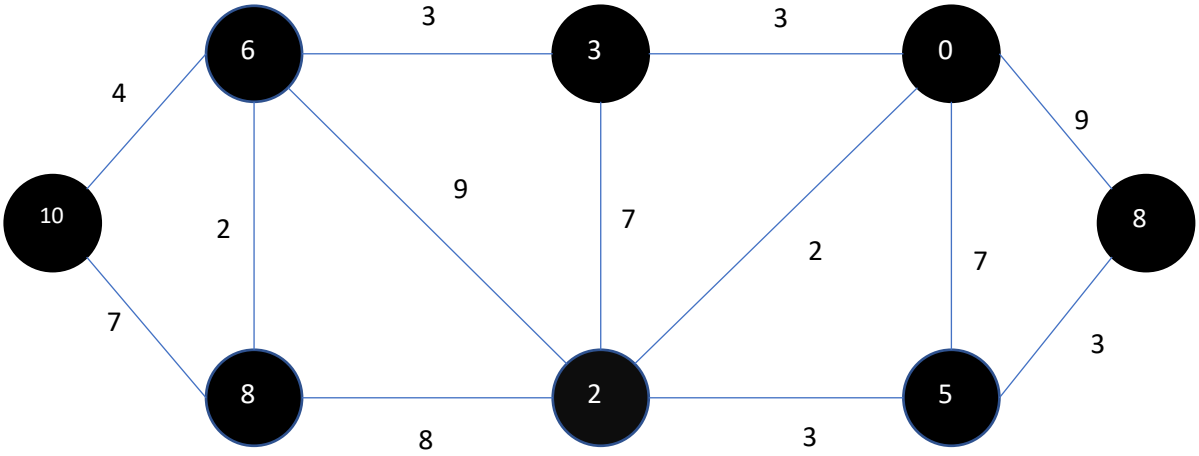

Dijkstra(F)
average_path=6.75

Dijkstra(D)



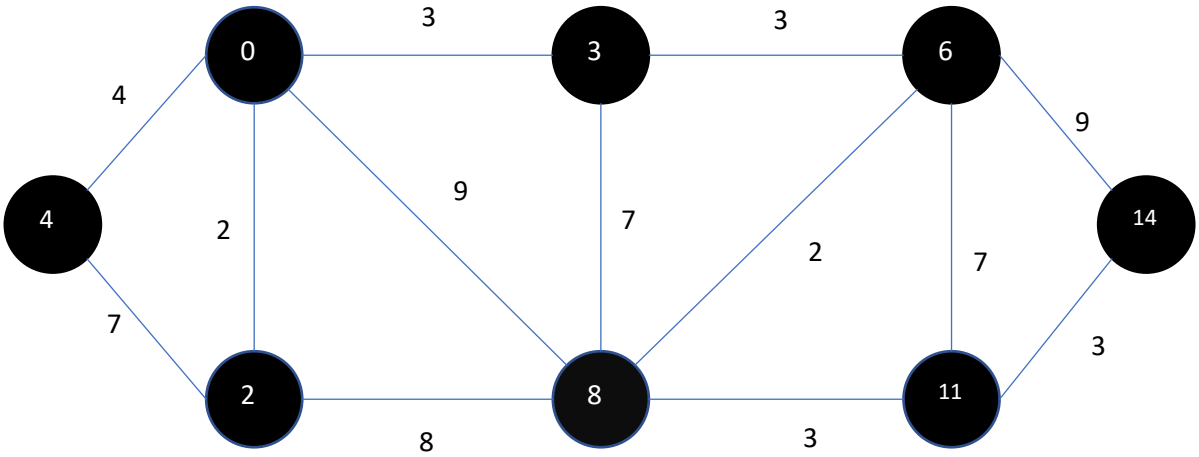

Dijkstra(D)
average_path=5.25

Dijkstra(E)



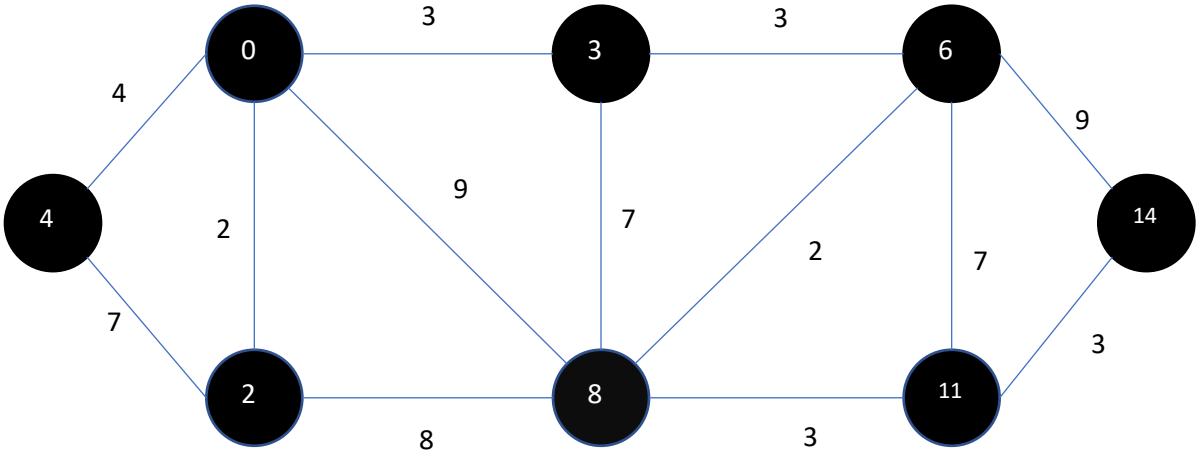

Dijkstra(E)
average_path=5.25

Dijkstra(C)



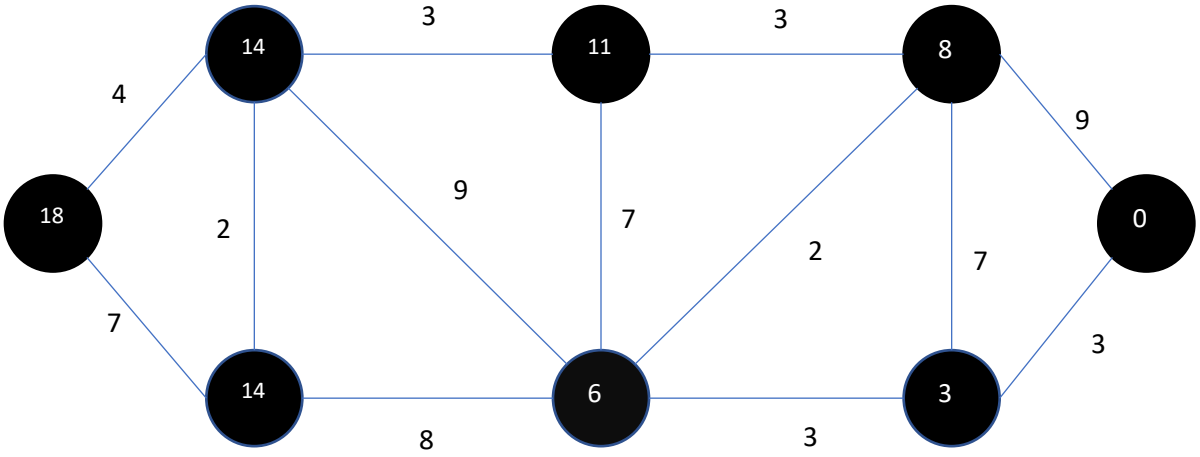

Dijkstra(C)
average_path=6

Dijkstra(A)



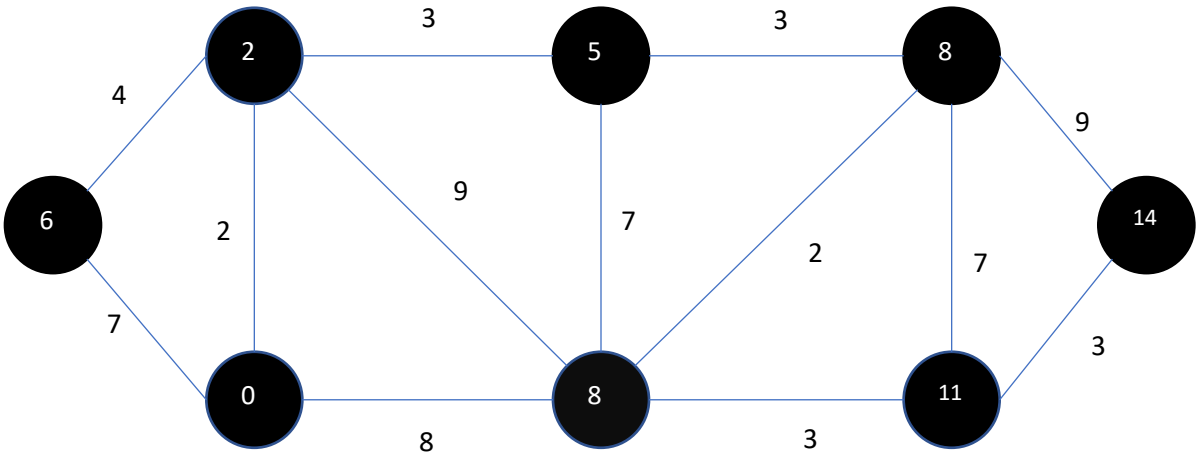

Dijkstra(C)
average_path=6

Dijkstra(B)




Dijkstra(B)
average_path=9.25

Dijkstra(F)




Dijkstra(F)
average_path=9.25

3. Wrestler Program

a. PseudoCode:

```
wrestler(G, s){
  Let isBipartite = True

  for each vertex v ∈ G.V{
    v.team = NIL
    v.isProcessed = False
    v.isTeamSet = False
  }
  s.team = "Babyfaces"
  s.isTeamSet = True

  Let Q be a Queue
  Q ← ∅
  Enqueue(Q, s)
  while Q is not empty{
    u = Dequeue(Q)

    for each v ∈ G.Adj(u)
      //Case where both parent and adjacent teams are set
      if u.isTeamSet and if v.isTeamSet{
        if u.team == v.team{
          isBipartite = False
        }
      //Case where parent is set and adjacent is not
      }else if u.isTeamSet and if not v.isTeamSet{
        set v.team to opposite of u.team
        v.isTeamSet = True
      //Case where parent is not set but adjacent team is set
      }else if not u.isTeamSet and if v.isTeamSet{
        set u.team to the opposite of v.team
        u.isTeamSet = True
      //Case where neither parent nor adjacent team is set
      }else{
        u.team = "Babyfaces"
        v.team = "Heels"
        u.isTeamSet = True
        v.isTeamSet = True
      }
      if not v.isProcessed{
        Enqueue(Q, v)
      }
    u.isProcessed = True
    if Q is empty and ∃ a vertex v ∈ G.V that is not processed {
      Enqueue(Q, v)
    }
  }
}
```

b. Running time:
For loop initializing the vertices: O(V)

While Q is not empty loop:
Each vertex will be enqueued: O(V)
For Each Vertex, the Adjacency list is searched: O(R) in worst case if a vertex is a Rival to every other vertex. Best Case there are no adjacent vertices, or rivalries, then it is constant.
Total for the While loop: O(V+R)

Searching for an unprocessed vertex v in G.V: O(V) in worst case if every vertex does not have any adjacent vertices

Total:
$T(n) = O(V) + O(V+R) + O(V)$
$T(n) = O(V+R)$