CS5800: Algorithms — Virgil Pavlu

Homework 10

Name:
Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. (15 points) Exercise 22.1-5.

Solution:
Using adjacency Matrix , this procedure will take $O(V^3)$ time becuase of the three for loops running from 1 to V

```
Procedure GSquare( g[ ] [ ] )
for ( i = 1 to V)
   for ( j = 1 to V )

        G' [ i ][ j ] = 0

           for ( k = 1 to V )

                if ( g[ i ][ k ] == 1 ) AND ( g [ k ][ j ] == 1 )
                        G'[ i ][ j ] == 1
                         break;


return G'
```

Using adjacency List :
1. for each u we traverse its adj list
2.for each adjacent v, we traverse its adj list . This will give us ”w” (vertex adj to v)

```
Procedure GSquare( V[G ] , E[G ] )
for each ( v in adj [u] )
   for ( w in adj [v ]  )
            G2.E = (u, w) ∈ G2
            insert w in adj2[ u ]



return G2
```

Time Complexity: $O(|V| * |E|)$.

---

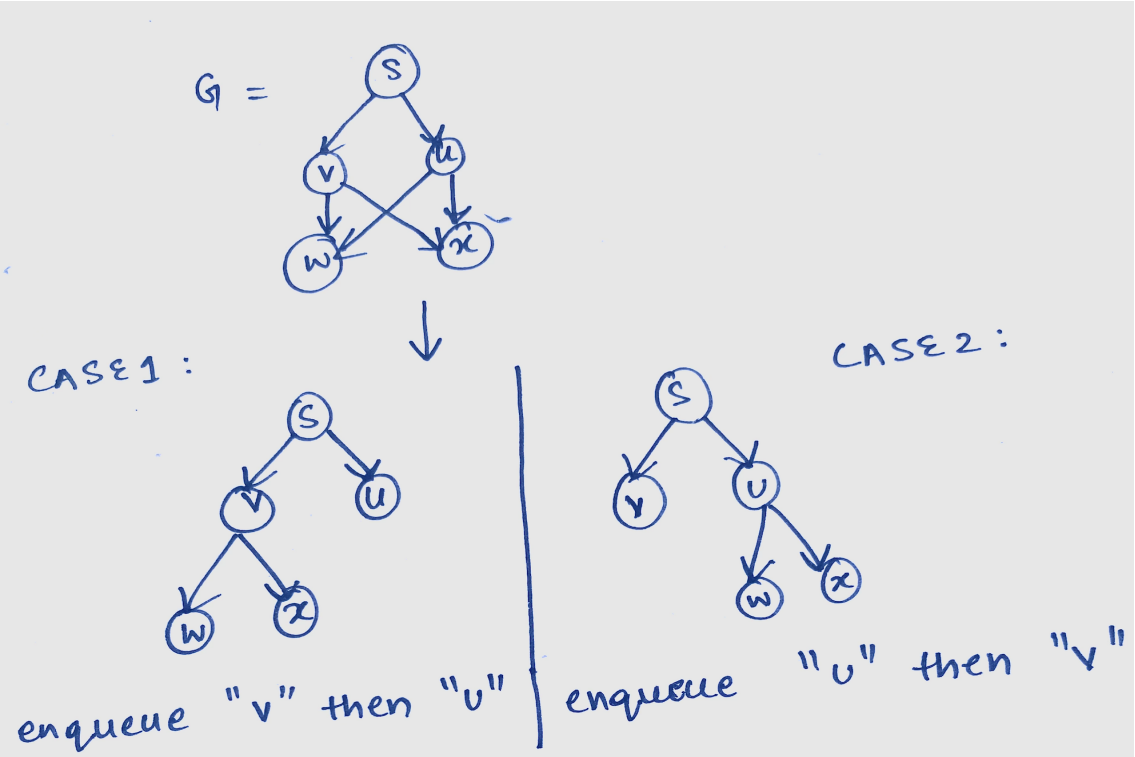
2. (15 points) Exercise 22.2-6.

Solution:

When we run BFS on G, we first enqueue ”s” .Then select all nodes connecting to ”s” , ie (s,v) and (s,u) .Now we deque ”s” and we have 2 cases of ordering:

CASE 1: visit ”v” first (enqueue ”v”)
enqueue (v) and (v,w)
CASE 2: visit ”u” first (enqueue ”u”)

enqueue (u,x) and (u,w)



From the diagrams of the resulting BFS Tree, it can be seen that at any point BFS will not put together (v,w) and (u,x) into BFS tree. This proves that set of edges cannot be produced by running BFS on G, no matter the ordering.

---

3. (15 points) Exercise 22.2-7.

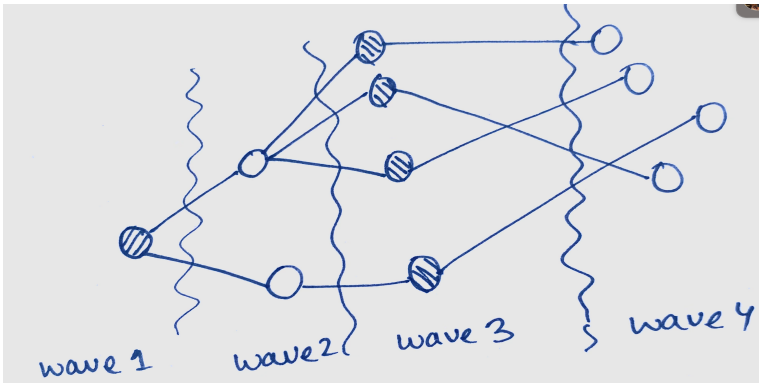
Solution:

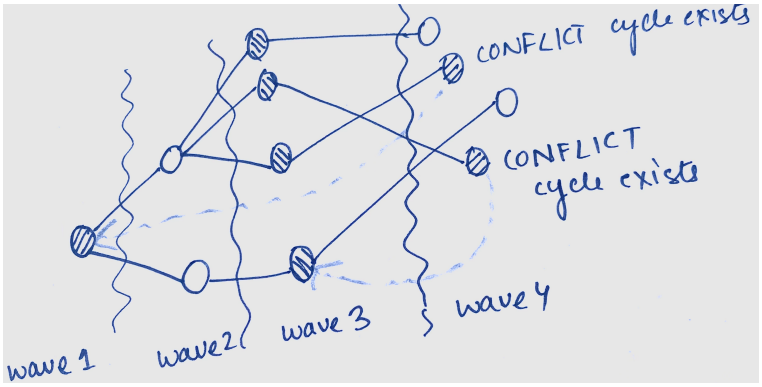
Consider ”babyface” and ”heel” as 2 colors in the graph with values 1 and 2 and Rivalry is the edges of the graph.

We have to ensure that the coloring of the graph is proper. To color the nodes, we will use BFS Algorithm , such that all the nodes in a wave have the same color, alternating between the waves, ie, say nodes in wave 1 have color1, nodes in wave 2 have color 2, nodes in wave 3 have color1 and so on..

We start by picking any node and assign it color 1 . Then move to its neighbor and assign it color 2. This coloring of nodes process is repeated till all nodes are colored. Note: We are alternating the color between 1 and 2.

”Conflict” : during coloring, if there is a conflict, that is, we are assigning color 1 , while the node is already colored 2, this means there exists a ”cycle” in the graph (rivalry exists) and there cannot be any solution.





Runtime Analysis: BFS will take O(n+r) time and the verifying if the coloring is proper will take O(r) .
Total Runtime: O(n+r)

---

4. (10 points) Exercise 22.3-7.

Solution:

Procedure DFSUsingStacks

```
for each ( u in V )
    u.color = white

for each ( u in V )
   if (  u.color== white )
       DFS-VISIT (G , u )
```

Procedure DFS-VISIT (S , u)

```
Initialise stack St
St.push( S )

while ( ! St.isEmpty( ) )
    curr = St.peak( )
    time = time +1
    curr.dt = time

if ( curr.color == white )
    curr.color = gray

      for ( u in adj[ curr ] )
            if ( u.color == white )
                St.push (u )
                 u.π = curr

elseif  ( curr.color == gray )
       curr.color= black
       time = time + 1
       u.ft = time
        St.pop ( )
```

---

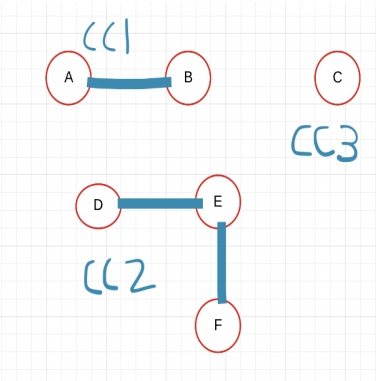
5. (10 points) Exercise 22.3-10.

Solution:

---

6. (15 points) Exercise 22.3-12.

Solution:



Algorithm:

1. In the beginning, assign (-1) to every vertex.
2. Start from any vertex of any CC having vertex as (-1) . Apply DFS on it.

3. If $v! = -1$, move to next v
4. In the end, count number of times DFS was applied.

```
Procedure CC ( int v, int G )

initialize k = 0        //number of CC
initialize  arr [v]     //number of vertices


FOR ( i=0 to v )
    arr [ i ] = -1      //assign (-1) to all vertices


FOR ( i= 0 to v )
{
     IF ( arr [ i ] == -1 )

       { DFS_CC( G, v )
          k + +
        }
}
```

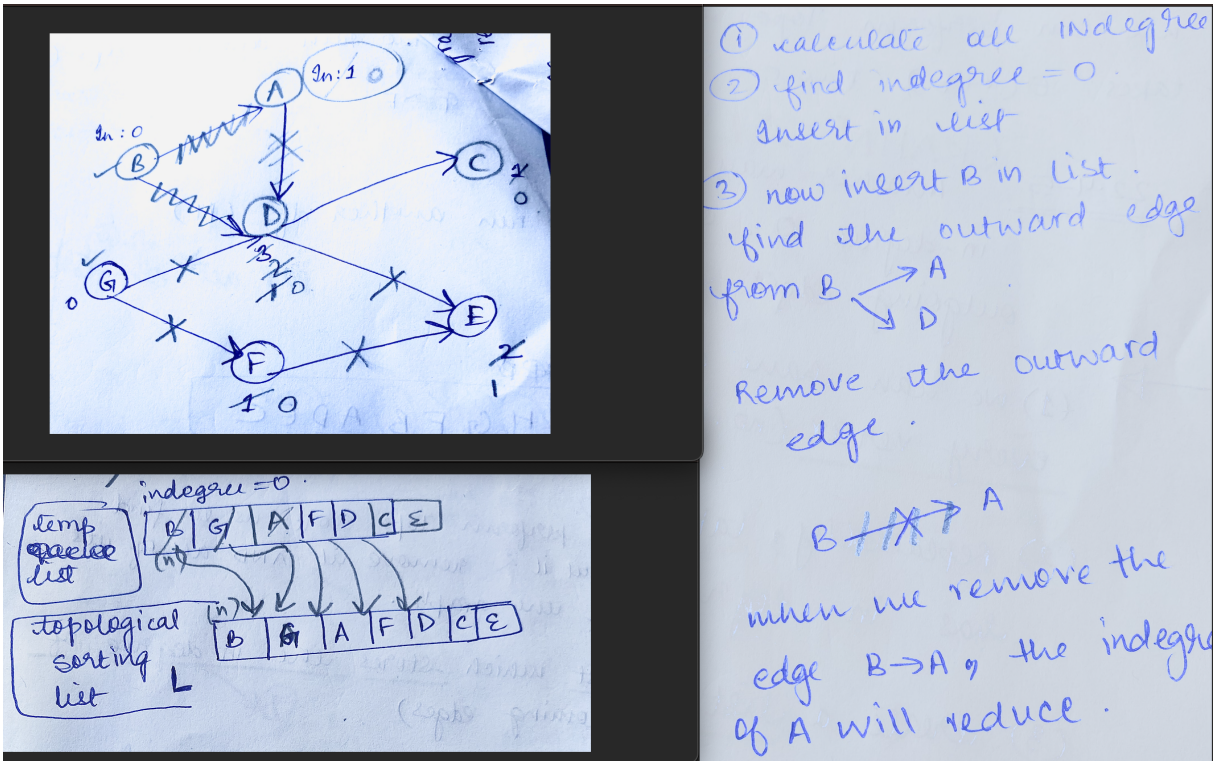
```
Procedure DFS-CC(G, v)

For each v ∈ G. adj [v]
    IF ( arr[v] == -1 )

       DFS-CC (G, v)

arr[v] ==0
```

---

7. (20 points) Exercise 22.4-5.

Solution:

(1) We can have a list which stores the IN degree of every vertex (no of incoming edges)
(2) Repeatedly, take out (extract) the node which has in-degree 0 and store it in another list. Lets call it ”temp-list”
(3) For every extracted vertex, we remove it from the temp-list, we visit its child and delete the edge connecting child and parent. this will reduce the degree by 1 of the child. (4) Create a list L which will contain the topologically sorted output




```
while ( ! temp_list.isEmpty( ) )
{

  remove a node (n)  from temp_list
  add (n) to L

  For each node (m) having (e) edge n -> m
     remove (e) from G
     IF m.degree== 0
        insert m to temp_list
}



IF G has edges
    return "cycle exists"

Else
   return L       //sorted list
```

Runtime : $O(|E|+|V|)$

---

8. (15 points) Two special vertices s and t in the undirected graph $G = (V, E)$ have the following property: any path from s to t has at least $1 + |V|/2$ edges. Show that all paths from s to t must have a common vertex v (not equal to either s or t) and give an algorithm with running time $O(V + E)$ to find such a node v.

Solution:

Let's run BFS on graph G from root "s" till "t" . In the BFS Tree, for every node we have levels (aka

waves "w") which is length of one of the shortest paths from root to node.

The number of waves (to find "t") > $|V|/2$ because the path between (s,t) has atleast $1 + |V|/2$ edges. Excluding s and t , the total number of nodes in wave 1 to V/2 is V-2. This means there is a wave level where there exists a node "v".
Starting from "s", BFS Tree will contain all the rechable vertices from "s" . This implies that for a path to exist between (s,t) there must be a common vertex "v" .

Let d_1[v] and d_2[v] be the distance from (s to v) and (t to v)
Then for each v :
d_1[v] + d_2[v] = d(s,t)

Running Time: Since BFS is being used, RT is O($|V| + |E|$)

---

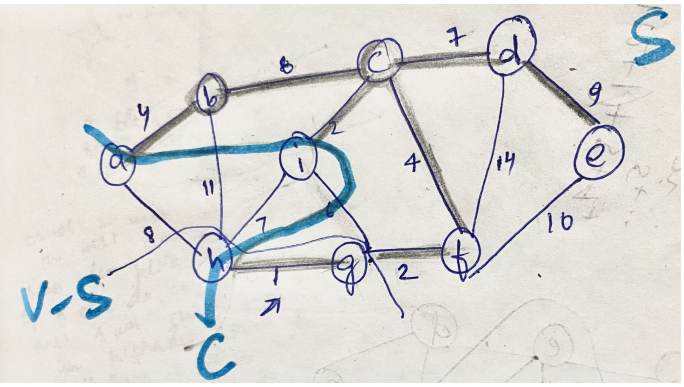
9. (Extra Credit 25 points) Problem 22-3.

10. (Extra Credit 25 points) Problem 22-4.

11. (25 points) Exercise 23.1-3.

Solution:

Let us assume T is some MST in a graph G which has edges (u,v). Adding one more edge to MST, will form a cycle because T is a connected tree
Let us assume there is a cut C which divides the graph into S and (V-S).



There is some edge (or path) which connects S to (V-S) . We know (u,v) is in T , so there is some edge connecting S to V-S. Let's call this edge (a,b) . We know T is an MST , this means it will have the **minimum possible total edge weight**. So , weight of (a,b) is greater than (u,v) . IF weight of (a,b) was smaller, it would be a part of the MST T .

This proves our point, **that in an MST if (u, v) is an edge , then it must be a light edge crossing some cut of the graph.**

---

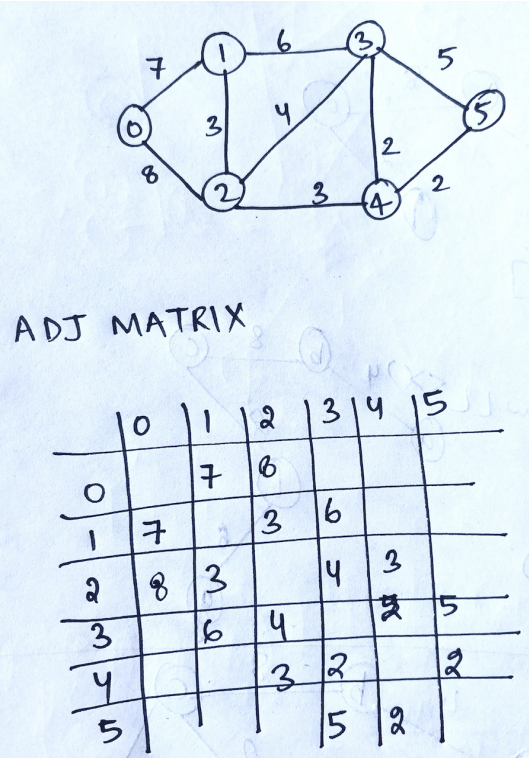
12. (25 points) Exercise 23.2-2.

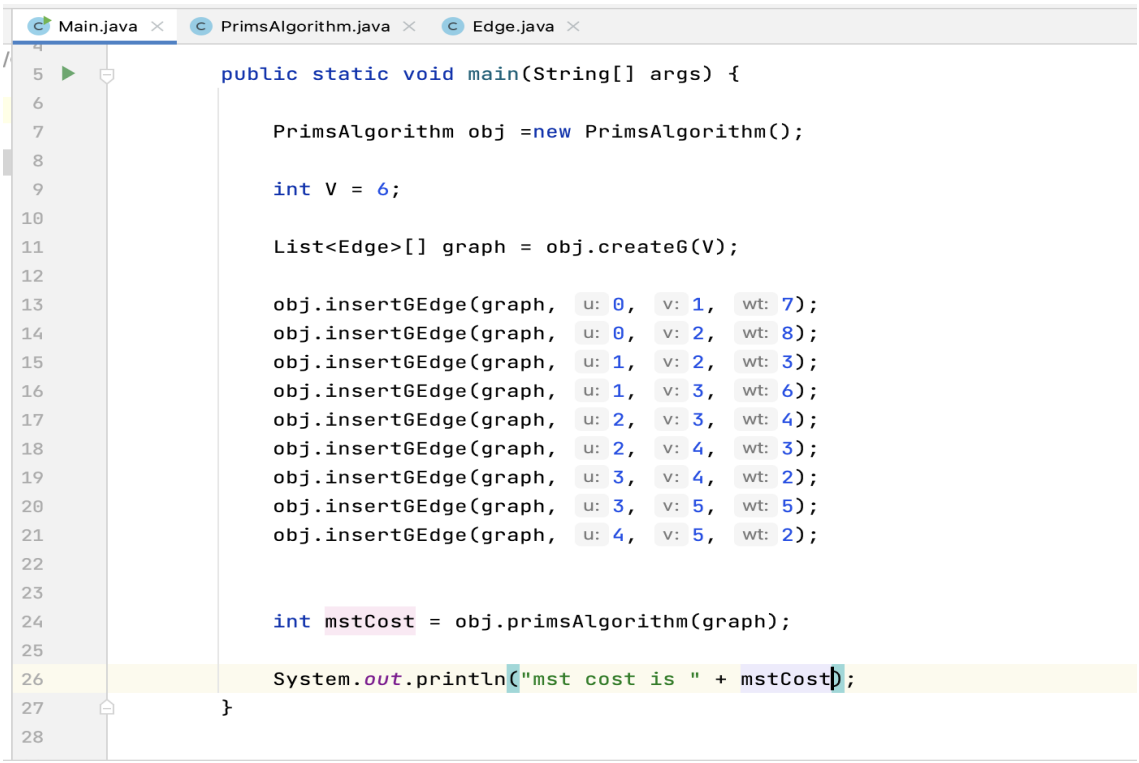
Solution:

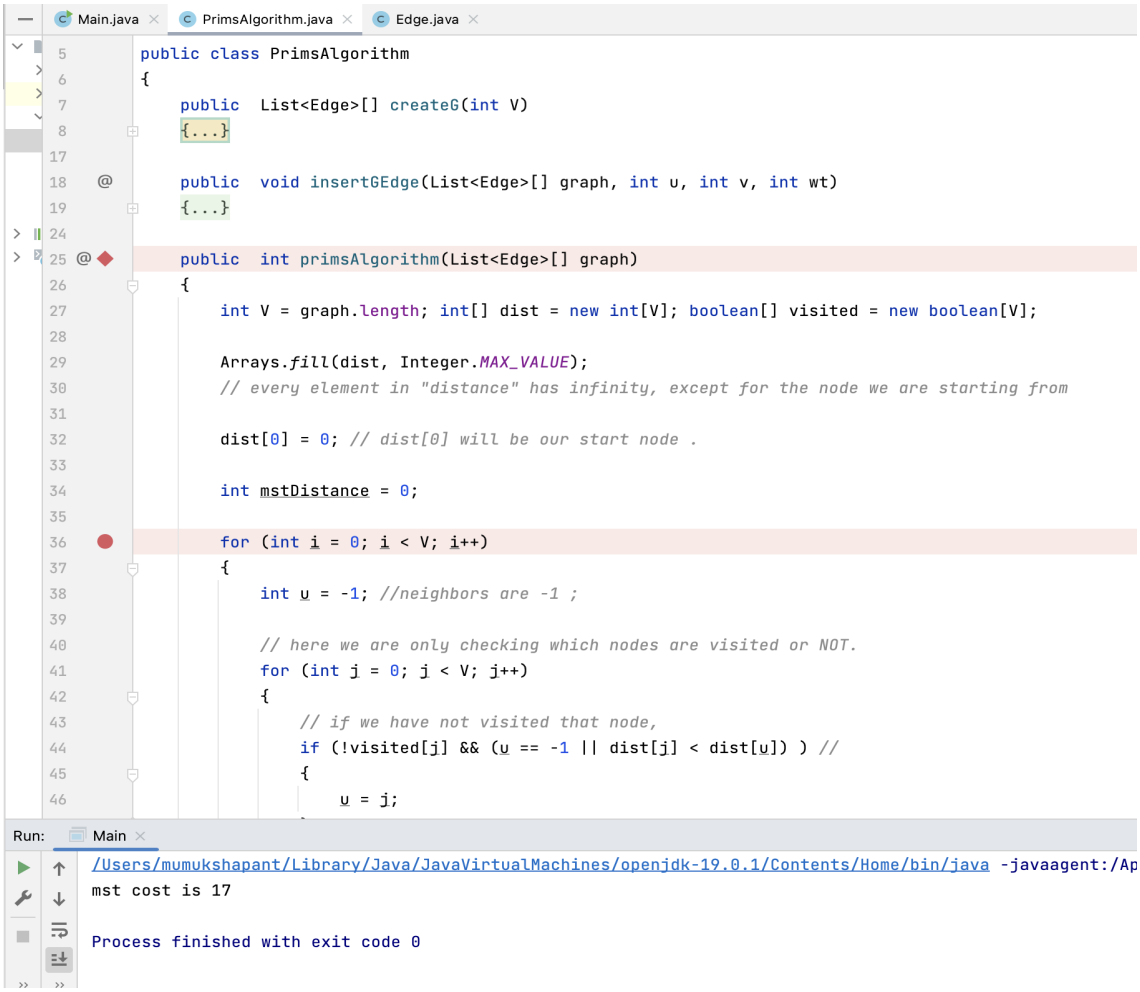
The Running Time of this implementation is O($|V|^2$), where V is the number of Vertices in the graph.

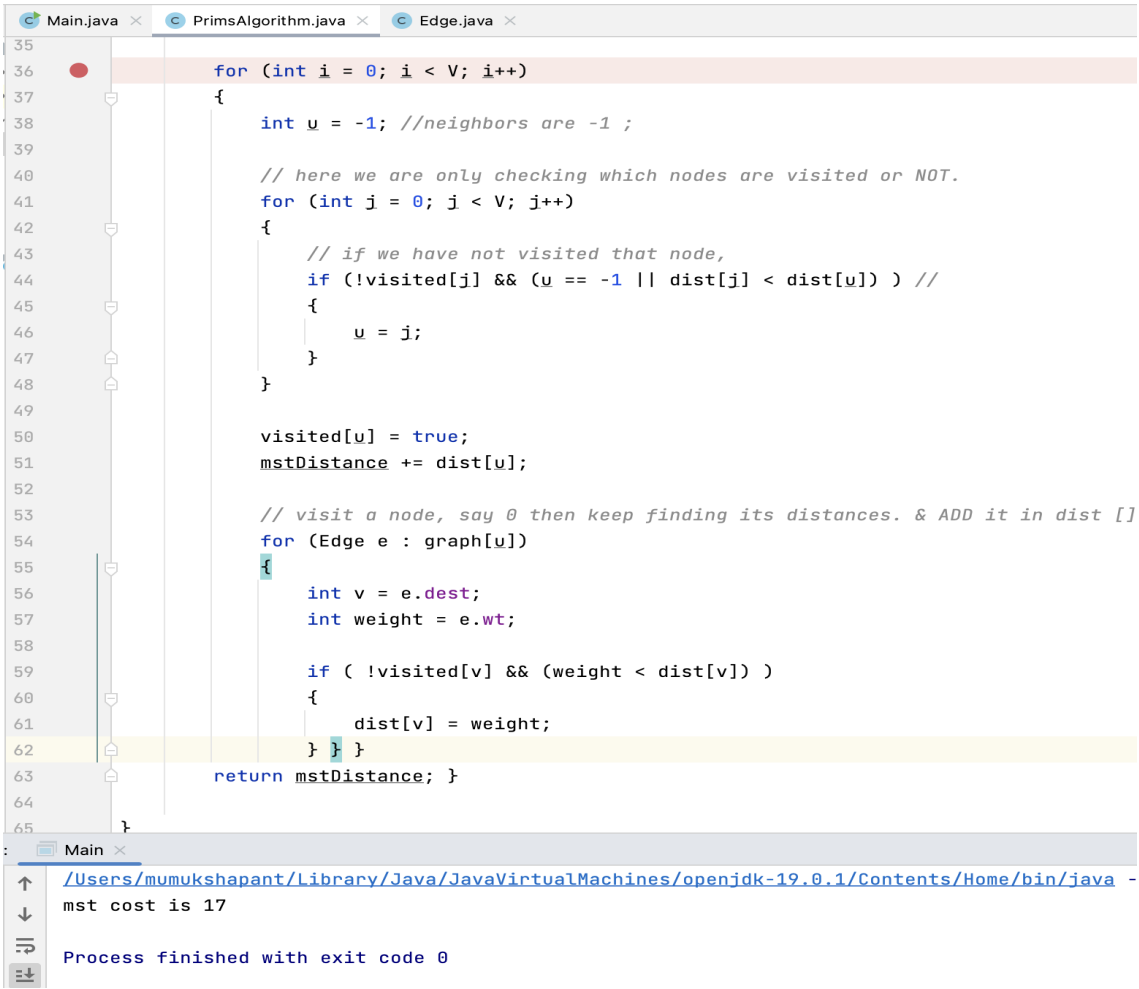
this is because of 2 for loops being used. The outer FOR loop runs V times to search "un- visited" vertex and its minimum weight.

In inner loop, for every adj vertex, it will check if its visited then update the dist array value.




Main.java
PrimsAlgorithm.java
Edge.java
public static void main(String[] args) {
PrimsAlgorithm obj =new PrimsAlgorithm();
int V = 6;
List<Edge>[] graph = obj.createG(V);
obj.insertGEdge(graph, u: 0, v: 1, wt: 7);
obj.insertGEdge(graph, u: 0, v: 2, wt: 8);
obj.insertGEdge(graph, u: 1, v: 2, wt: 3);
obj.insertGEdge(graph, u: 1, v: 3, wt: 6);
obj.insertGEdge(graph, u: 2, v: 3, wt: 4);
obj.insertGEdge(graph, u: 2, v: 4, wt: 3);
obj.insertGEdge(graph, u: 3, v: 4, wt: 2);
obj.insertGEdge(graph, u: 3, v: 5, wt: 5);
obj.insertGEdge(graph, u: 4, v: 5, wt: 2);
int mstCost = obj.primsAlgorithm(graph);
System.out.println("mst cost is " + mstCost);
}


Main.java
PrimsAlgorithm.java
Edge.java
public class PrimsAlgorithm
{
public List<Edge>[] createG(int V)
{...}
public void insertGEdge(List<Edge>[] graph, int u, int v, int wt)
{...}
public int primsAlgorithm(List<Edge>[] graph)
{
int V = graph.length; int[] dist = new int[V]; boolean[] visited = new boolean[V];
Arrays.fill(dist, Integer.MAX_VALUE);
// every element in "distance" has infinity, except for the node we are starting from
dist[0] = 0; // dist[0] will be our start node .
int mstDistance = 0;
for (int i = 0; i < V; i++)
{
int u = -1; //neighbors are -1 ;
// here we are only checking which nodes are visited or NOT.
for (int j = 0; j < V; j++)
{
// if we have not visited that node,
if (!visited[j] && (u == -1 || dist[j] < dist[u])) //
{
u = j;
Run:
Main
/Users/mumukshapant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java -javaagent:/Ap
mst cost is 17
Process finished with exit code 0



13. **(25 points)** Exercise 23.2-4.

Solution:

14. **(25 points)** Exercise 23.2-5.

Solution:

15. **(Extra Credit 40 points)** Problem 23-1.

16. **(Extra Credit 30 points)** Exercise 23.1-11.

17. **(Extra Credit 30 points)** Write the code for Kruskal algorithm in a language of your choice. You will first have to read on the disjoint sets datastructures and operations (Chapter 21 in the book) for an efficient implementation of Kruskal trees.