Algorithm Analysis and Data Structures: Assignment 5 – Topological Sort

Write program to do Topological sort (DFS).

your graph must have at least 10 nodes and 15 edges. It must have multiple edges going in and also coming out from some of the edges.

1. Run the program on a graph with no cycles.  Your print out would show all the edges of the graph.  Then it will show the list of vertices in the topological order.

2. Run the program on a graph that has a cycle.  Your program catches the cycle and print the list of edges that form the cycle.

Code:

**package** Graphs;

**import** java.util.\*;

**import** java.io.\*;

**public** **class** TopologicalSorting {

**private** **int** vertex;

**private** LinkedList<Integer> adj[];

**private** **boolean** hasCycle = **false**;

**private** **int** cycleVetex;

TopologicalSorting(**int** v)

{

vertex = v;

adj = **new** LinkedList[v];

**for** (**int** i=0; i<v; ++i)

adj[i] = **new** LinkedList();

}

**void** addEdge(**int** v,**int** w) { adj[v].add(w); }

**public** **static** **void** main(String args[])

{

TopologicalSorting gAcyclic = **new** TopologicalSorting(10);

TopologicalSorting gCyclic = **new** TopologicalSorting(10);

gAcyclic.addEdge(0, 1);

gAcyclic.addEdge(0, 2);

gAcyclic.addEdge(1, 4);

gAcyclic.addEdge(1, 5);

gAcyclic.addEdge(2, 3);

gAcyclic.addEdge(2, 4);

gAcyclic.addEdge(3, 4);

gAcyclic.addEdge(3, 6);

gAcyclic.addEdge(3, 8);

gAcyclic.addEdge(4, 5);

gAcyclic.addEdge(5, 7);

gAcyclic.addEdge(5, 9);

gAcyclic.addEdge(6, 7);

gAcyclic.addEdge(7, 9);

gAcyclic.addEdge(8, 9);

System.***out***.println("Acyclic Graph \n");

*DisplayGraph*(gAcyclic);

gAcyclic.Sort();

gCyclic.addEdge(0, 1);

gCyclic.addEdge(0, 2);

gCyclic.addEdge(1, 4);

gCyclic.addEdge(1, 5);

gCyclic.addEdge(2, 3);

gCyclic.addEdge(2, 4);

gCyclic.addEdge(3, 4);

gCyclic.addEdge(3, 6);

gCyclic.addEdge(3, 8);

gCyclic.addEdge(4, 5);

gCyclic.addEdge(5, 7);

gCyclic.addEdge(9, 5);

gCyclic.addEdge(6, 7);

gCyclic.addEdge(7, 9);

gCyclic.addEdge(8, 9);

System.***out***.println("\n");

System.***out***.println("Cyclic Graph \n");

*DisplayGraph*(gCyclic);

gCyclic.Sort();

}

**static** **void** DisplayGraph(TopologicalSorting graph)

{

**for**(**int** v = 0; v < graph.vertex; v++)

{

System.***out***.println("Adjacency list of vertex "+ v);

System.***out***.print("("+ v+")");

**for**(Integer i: graph.adj[v]){

System.***out***.print(" -> "+i);

}

System.***out***.println("\n");

}

}

**public** **void** StackElementsToSort(**int** vertex, **boolean** visited[], Stack stack)

{

**if**(!hasCycle)

{

visited[vertex] = **true**;

Integer i;

Iterator<Integer> it = adj[vertex].iterator();

**while** (it.hasNext())

{

i = it.next();

**if** (!visited[i])

StackElementsToSort(i, visited, stack);

**else** **if**(visited[i] && stack.search(i) < 0 )

{

hasCycle = **true**;

cycleVetex = i;

**break**;

}

}

stack.push(**new** Integer(vertex));

}

}

**public** **void** Sort()

{

Stack stack = **new** Stack();

**boolean** visited[] = **new** **boolean**[vertex];

**boolean** isCycleVertex = **false**;

**for** (**int** i = 0; i < vertex; i++)

visited[i] =**false**;

**for** (**int** i = 0; i < vertex; i++)

**if** (visited[i] == **false** && !hasCycle)

StackElementsToSort(i, visited, stack);

**if**(!hasCycle)

**while** (!stack.empty())

System.***out***.print(stack.pop() + " ");

**else**

{

String sVertices = ""+cycleVetex;

**while** (!stack.empty())

{

**if**(!isCycleVertex)

{

**if**(stack.pop().equals(**new** Integer(cycleVetex)))

isCycleVertex = **true**;

}

**else**

sVertices += ","+stack.pop();

}

System.***out***.println("There is a cycle in this graph at " +sVertices+ " vertices");

}

}

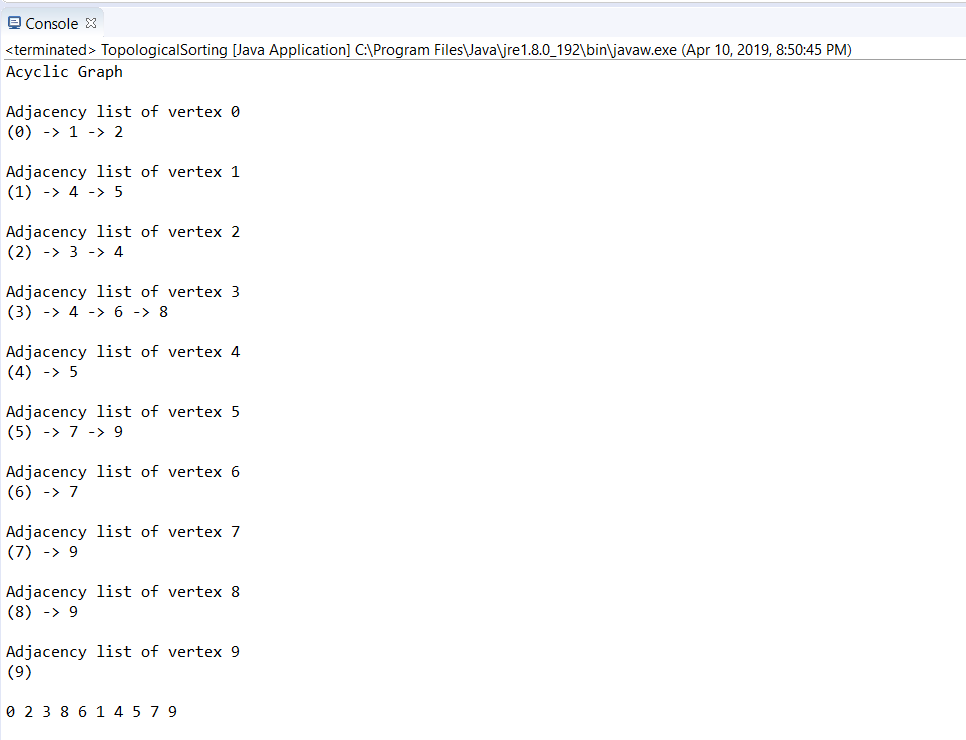
}

Instructions to compile:

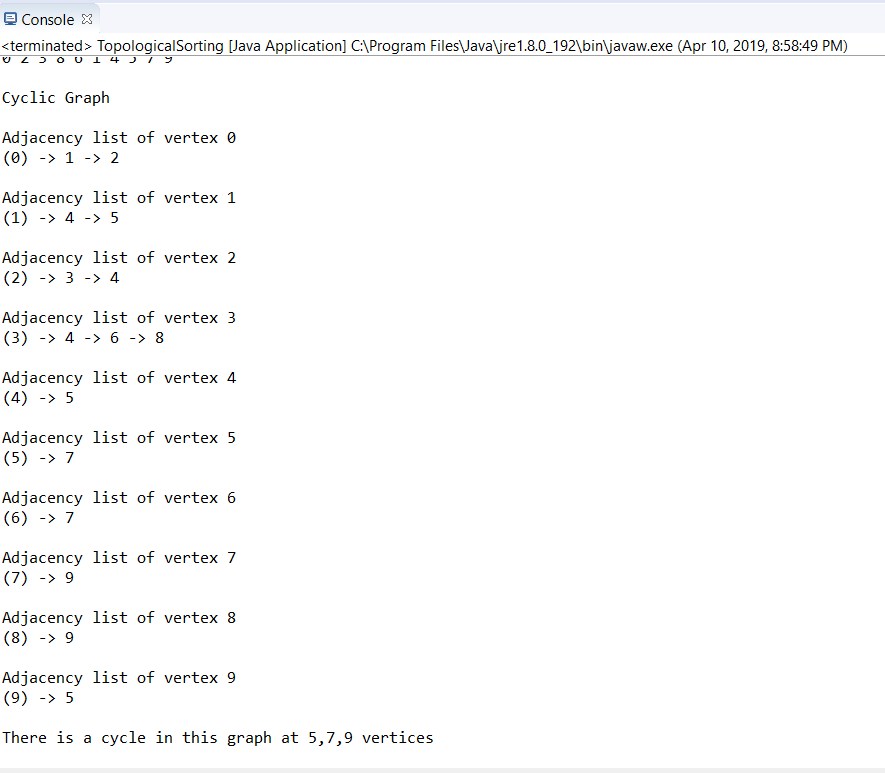
1. Create a new java project in eclipse named Graphs
2. Create a new package in the source folder named Graphs
3. Create a java class in the same package with the name TopologicalSorting
4. Place the code and save
5. Run the code

Screen Shot of the output

Output of an Acyclic Graph

****

Output of a Cyclic Graph

****