Graduation Time

Sri Harsha Pasupuleti

SUMMARY:

Directed Acyclic Graph is one of the popular concepts in computer science. We have given a problem to find the estimated amount of time the student takes to graduate. The problem given is to find the longest time for a student to graduate from a university with completing all the courses with prerequisites. The inputs that are given are in the text file format that we have to parse it using the file reader to form the nodes and graph. Our starting approach is to parse the input file and pass the values as the inputs to the topological sort, which is done by using the DFS approach also recursion to sort the data in the proper order. As far as topological sorting is concerned, the graph must not contain any cycles. With the graph provided for the input, it's pretty clear that the process works. In topological sort, print the values at the stack to get the topological sorting order. We use a concept called a temporary stack. With the topological sort, we are going to write the algorithm, print the vertex in the following ways. Taking as output as the input to find the longest path. Tracking the maximum length of each path and storing the values of the nodes in the list. When the complete traversing is done, the length and the path will be printed.

Pseudocode for topological sort:

```
Step 1: Create the graph by calling new graduation _time();
Step 2: call the read file using Graph.readfile()
Step 3: Read the text file using a buffered reader("graph.txt")  (Parsing)
         While (line != null){
          If (line==''#")
          Increment }
          If (value==0){
          Interger value}
          Else If (value==1){
          Insert vertex}
          Else if (value==2){
          Insertedge(str(0),str(1))

Step 4: Call the topological sort using graph.topologicalsort()
Step 5: Create a stack and a boolean array named as visited[ ] which can keep track
of which node is visited or not and set them initially to false.
          Begin
          mark u as visited
          for all vertices v which is adjacent to u, do
          if v is not visited, then
          topoSort(c, visited, stack)
          While stack.empty == false
          Add string and do stack pop.
          Done
          push u into a stack using stack.push(str)
Step6: initially mark all nodes as unvisited so we can make it true in the array if
visited.
```

```
        for all nodes v of the graph, do
        if v is not visited, then
       Topological Sort(i, visited, stack)
       done
       Stack pop and print all elements from the stack
```

Pseudocode for the longest path:

```
Function labelNodes(sortedNodes)
n=sortedNodes of length;
index=0;
for nodes to sortedNodes
    do labelnodes[index]=node;
    index++
return labelnodes

Function clalulateDistance(number_of_nodes, sortedNodes)
distance=sortedNodes.size;

for i from 1 to n
    do for j 1 to n
        do if adjList contains labelNode[i]
            get_adjListlabelNode[i] contains labelNode[j]
                if  distance[j] < distance[i] + 1
                distance[j] = distance[i] + 1;
         if destSourceMap contains labeledNodes[j]
           destSourceMap replace labeledNodes[j] labeledNodes[i]
         else
           destSourceMap put labeledNodes[j] labeledNodes[i]


Function getEndNode  (n , labeledNodes
    endNode = empty;
   max = Integer.MIN_VALUE;
   for index  from 1 to n
        do  i distance[index] > max
     max = distance[index];
     endNode = labeledNodes[index];
length = max;
return endNode;

Function longestPath (node)
   if  !destSourceMap contains node
      then add node to longestPath;
     return
   longestPath destSourceMap get (node);
   add node to longestPath;
```

Pseudocode for the entire algorithm:

Step 1: Create the graph by calling new graduation _time();

Step 2: call the read file using Graph.readfile()

Step 3: Read the text file using a buffered reader(“graph.txt”) (Parsing)

```
While (line != null){
 If (line==”#”)
 Increment }
 If (value==0){
 Interger value}
 Else If (value==1){
 Insert vertex}
 Else if (value==2){
 Insertedge(str(0),str(1))
```

Step 4: Call the topological sort using graph.topologicalsort()

Step 5: Create a stack and a boolean array named as visited[ ] which can keep track of which node is visited or not and set them initially to false.

```
Begin
mark u as visited
for all vertices v which is adjacent to u, do
if v is not visited, then
topoSort(c, visited, stack)
While stack.empty == false
Add string and do stack pop.
Done
push u into a stack using stack.push(str)
```

Step6: initially mark all nodes as unvisited so we can make it true in the array if visited.

```
 for all nodes v of the graph, do
 if v is not visited, then
Topological Sort(i, visited, stack)
done
Stack pop and print all elements from the stack
```

Step7: For finding the longest path,

```
        Function labelNodes(sortedNodes)
    n=sortedNodes of length;
index=0;
for nodes to sortedNodes
  do labelnodes[index]=node;
  index++
return labelnodes

Function clalulateDistance(number_of_nodes, sortedNodes)
distance=sortedNodes.size;
for i from 1 to n
  do for j 1 to n
```

```
                do if adjList contains labelNode[i]
                        get_adjListlabelNode[i] contains labelNode[j]
                                if  distance[j] < distance[i] + 1
                                        distance[j] = distance[i] + 1;
                         if destSourceMap contains labeledNodes[j]
                            destSourceMap replace labeledNodes[j] labeledNodes[i]
                         else
                            destSourceMap put labeledNodes[j] labeledNodes[i]
  Function getEndNode  (n , labeledNodes
           endNode = empty;
      max = Integer.MIN_VALUE;
      for index  from 1 to n
          do  i distance[index] > max
           max = distance[index];
           endNode = labeledNodes[index];
length = max;
return endNode;

Function longestPath (node)
      if  !destSourceMap contains node
          then add node to longestPath;
         return
      longestPath destSourceMap get (node);
      add node to longestPath;
```

The description on how to run the code:

In Tuffix Environment:

- Save the .java and all the .txt files into the local drive.
- Copy the path of Graph01.txt,Graph02.txt and biggraph.txt and paste in the file reader.
- Run the .java file in the command prompt.
- Then take the .class file and run with javac.
- The outputs of the file will be shown in the console.

In Java IDE:

- Graph01.txt,Graph02.txt and biggraph.txt are the input files that have to be parsed and fed into the code. Just keep the file in the local disk and indicate the exact path in the code.
- Open the Graduation_time.java file using any java IDE like eclipse, sublime editor to execute the code.
- Hit enter or run to execute the code to get the desired output.
- The outputs of the file will be shown in the console.

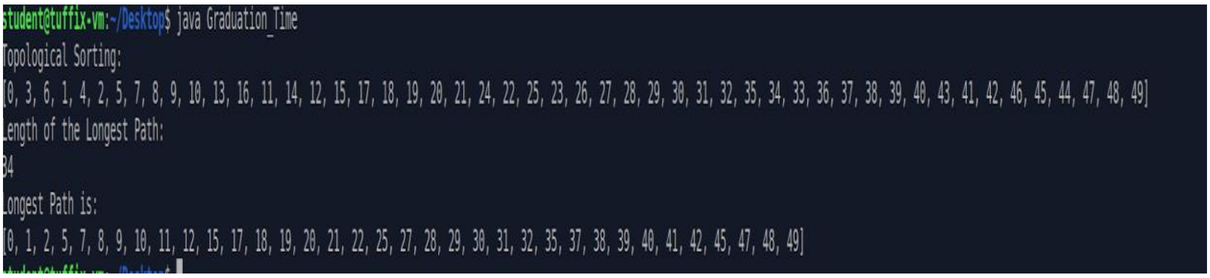
Steps:

- Download the .txt input files on the hard disk and copy the path and paste it into the read file function.
- Then hit the run button so as to code execute the code to get the output of topological sorting and longest path as well.

Screenshots corresponding to three input files and project members:

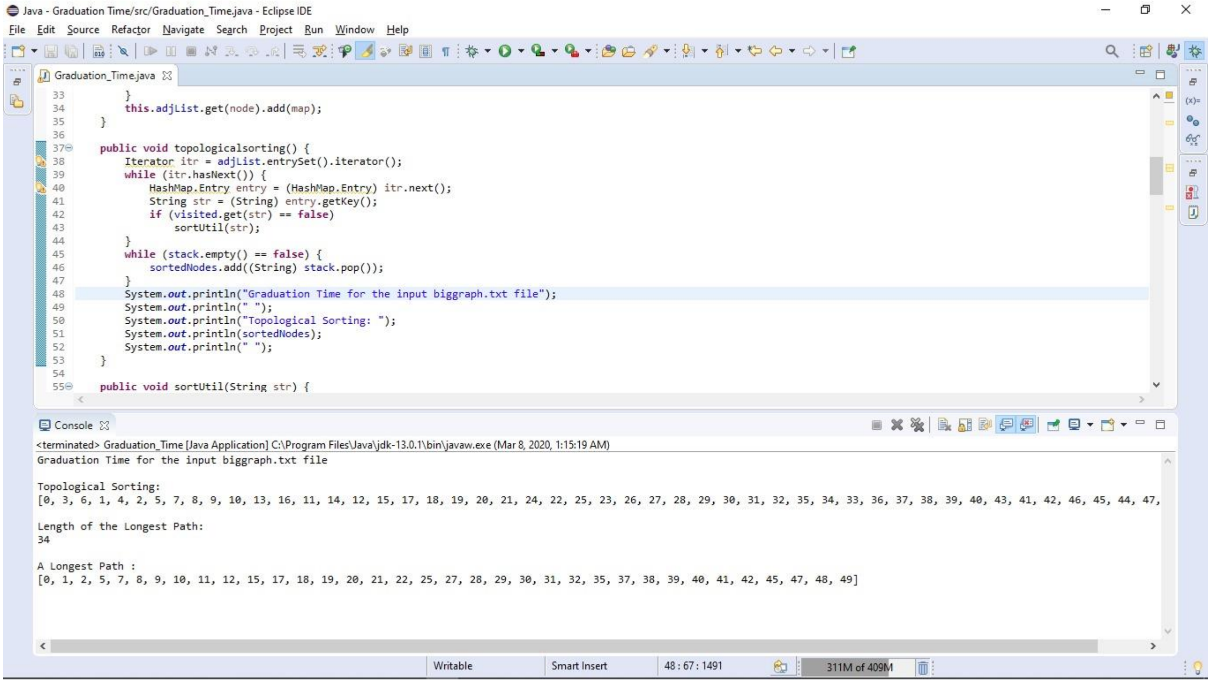
Biggraph.txt output using tuffix environment and Java IDE:

Tuffix Environment:



```
student@tuffix-vm:~/Desktop$ java Graduation_Time
Topological Sorting:
[0, 3, 6, 1, 4, 2, 5, 7, 8, 9, 10, 13, 16, 11, 14, 12, 15, 17, 18, 19, 20, 21, 24, 22, 25, 23, 26, 27, 28, 29, 30, 31, 32, 35, 34, 33, 36, 37, 38, 39, 40, 43, 41, 42, 46, 45, 44, 47, 48, 49]
Length of the Longest Path:
34
Longest Path is:
[0, 1, 2, 5, 7, 8, 9, 10, 11, 12, 15, 17, 18, 19, 20, 21, 22, 25, 27, 28, 29, 30, 31, 32, 35, 37, 38, 39, 40, 41, 42, 45, 47, 48, 49]
student@tuffix-vm:~/Desktop$
```

Java IDE:

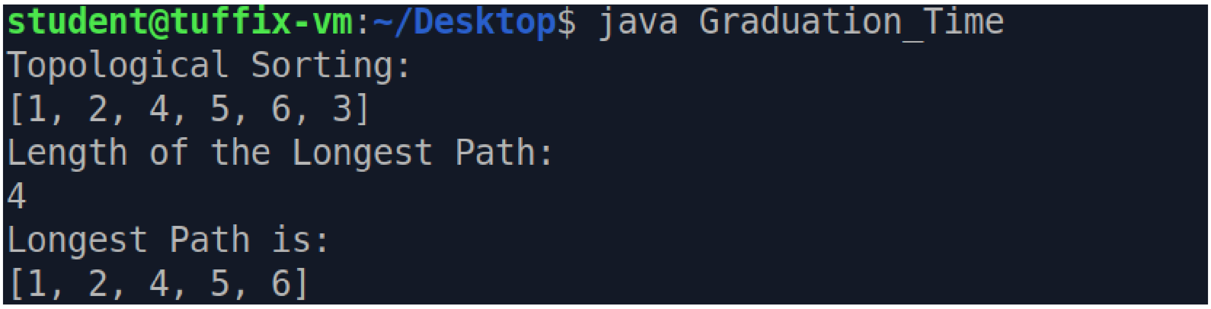

Java - Graduation Time/src/Graduation_Time.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Graduation_Time.java
Console
Writable
Smart Insert
48 : 67 : 1491
311M of 409M

```
        }
        this.adjList.get(node).add(map);
    }

    public void topologicalsorting() {
        Iterator itr = adjList.entrySet().iterator();
        while (itr.hasNext()) {
            HashMap.Entry entry = (HashMap.Entry) itr.next();
            String str = (String) entry.getKey();
            if (visited.get(str) == false)
                sortUtil(str);
        }
        while (stack.empty() == false) {
            sortedNodes.add((String) stack.pop());
        }
        System.out.println("Graduation Time for the input biggraph.txt file");
        System.out.println(" ");
        System.out.println("Topological Sorting: ");
        System.out.println(sortedNodes);
        System.out.println(" ");
    }

    public void sortUtil(String str) {
```

```
<terminated> Graduation_Time [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Mar 8, 2020, 1:15:19 AM)
Graduation Time for the input biggraph.txt file

Topological Sorting:
[0, 3, 6, 1, 4, 2, 5, 7, 8, 9, 10, 13, 16, 11, 14, 12, 15, 17, 18, 19, 20, 21, 24, 22, 25, 23, 26, 27, 28, 29, 30, 31, 32, 35, 34, 33, 36, 37, 38, 39, 40, 43, 41, 42, 46, 45, 44, 47,

Length of the Longest Path:
34

A Longest Path :
[0, 1, 2, 5, 7, 8, 9, 10, 11, 12, 15, 17, 18, 19, 20, 21, 22, 25, 27, 28, 29, 30, 31, 32, 35, 37, 38, 39, 40, 41, 42, 45, 47, 48, 49]
```

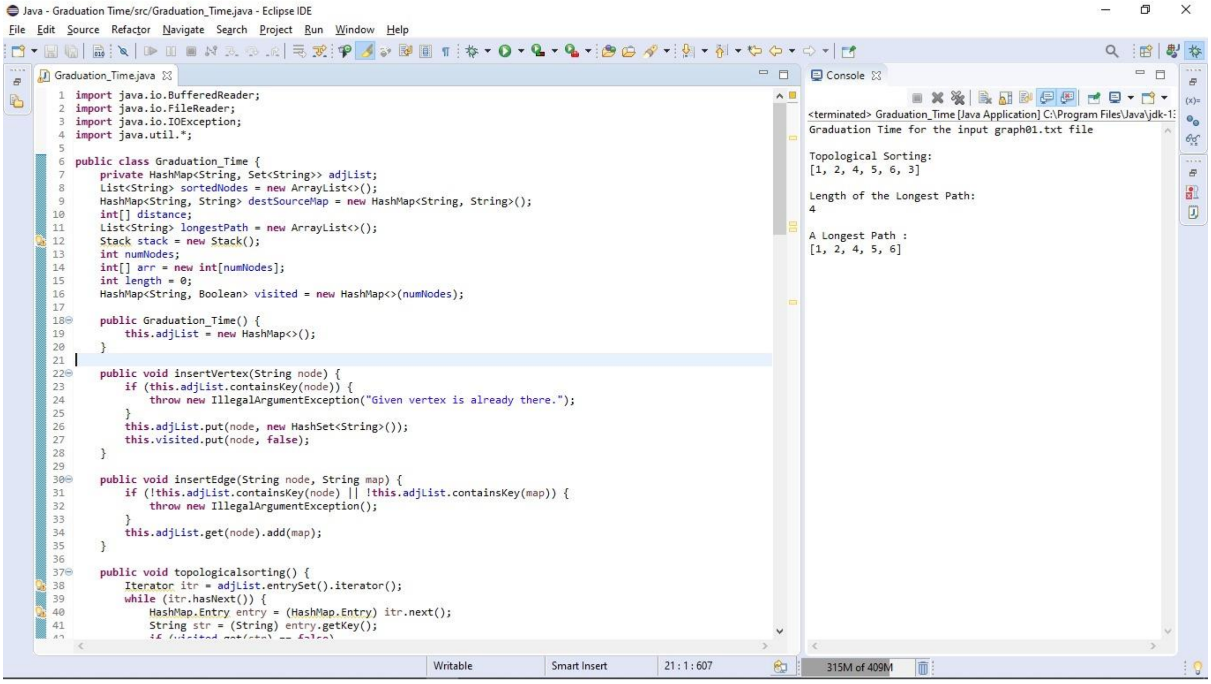
Graph01.txt output through using and Java IDE:

Tuffix Environment:



```
student@tuffix-vm:~/Desktop$ java Graduation_Time
Topological Sorting:
[1, 2, 4, 5, 6, 3]
Length of the Longest Path:
4
Longest Path is:
[1, 2, 4, 5, 6]
```

Java IDE:



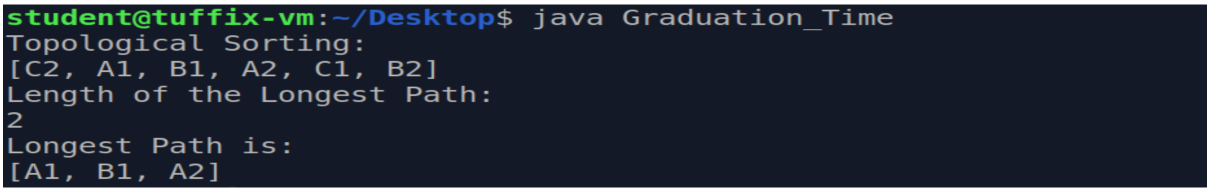
Java - Graduation Time/src/Graduation_Time.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Graduation_Time.java
Console

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

public class Graduation_Time {
    private HashMap<String, Set<String>> adjList;
    List<String> sortedNodes = new ArrayList<>();
    HashMap<String, String> destSourceMap = new HashMap<String, String>();
    int[] distance;
    List<String> longestPath = new ArrayList<>();
    Stack stack = new Stack();
    int numNodes;
    int[] arr = new int[numNodes];
    int length = 0;
    HashMap<String, Boolean> visited = new HashMap<>(numNodes);

    public Graduation_Time() {
        this.adjList = new HashMap<>();
    }

    public void insertVertex(String node) {
        if (this.adjList.containsKey(node)) {
            throw new IllegalArgumentException("Given vertex is already there.");
        }
        this.adjList.put(node, new HashSet<String>());
        this.visited.put(node, false);
    }

    public void insertEdge(String node, String map) {
        if (!this.adjList.containsKey(node) || !this.adjList.containsKey(map)) {
            throw new IllegalArgumentException();
        }
        this.adjList.get(node).add(map);
    }

    public void topologicalsorting() {
        Iterator itr = adjList.entrySet().iterator();
        while (itr.hasNext()) {
            HashMap.Entry entry = (HashMap.Entry) itr.next();
            String str = (String) entry.getKey();
            if (visited.get(str) == false)
```

```
<terminated> Graduation_Time [Java Application] C:\Program Files\Java\jdk-1:
Graduation Time for the input graph01.txt file

Topological Sorting:
[1, 2, 4, 5, 6, 3]

Length of the Longest Path:
4

A Longest Path :
[1, 2, 4, 5, 6]
```

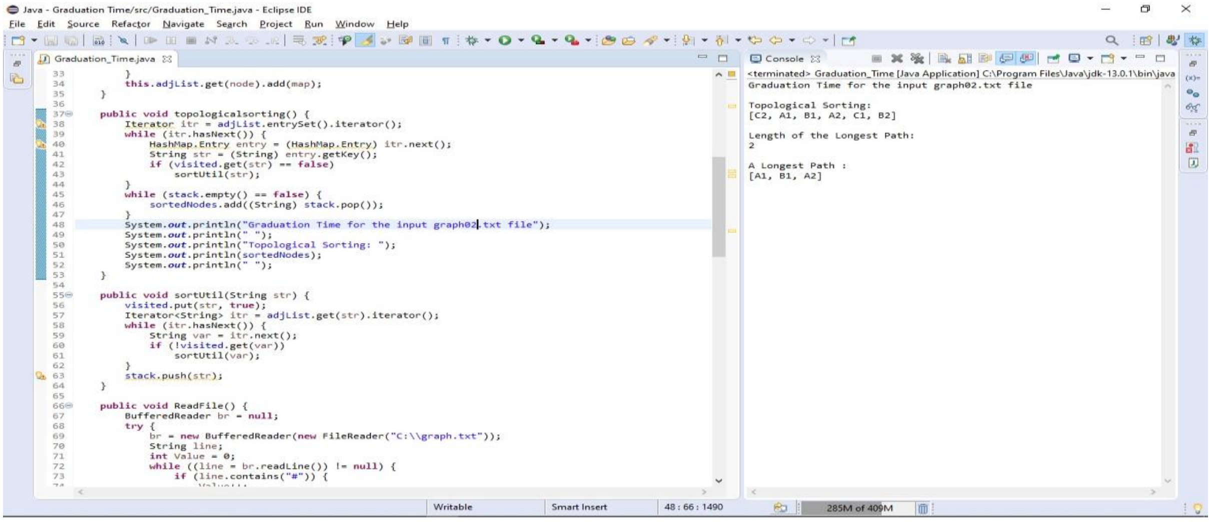
Graph02.txt output using tuffix and Java IDE:

Tuffix Environment:



```
student@tuffix-vm:~/Desktop$ java Graduation_Time
Topological Sorting:
[C2, A1, B1, A2, C1, B2]
Length of the Longest Path:
2
Longest Path is:
[A1, B1, A2]
```

Java IDE:

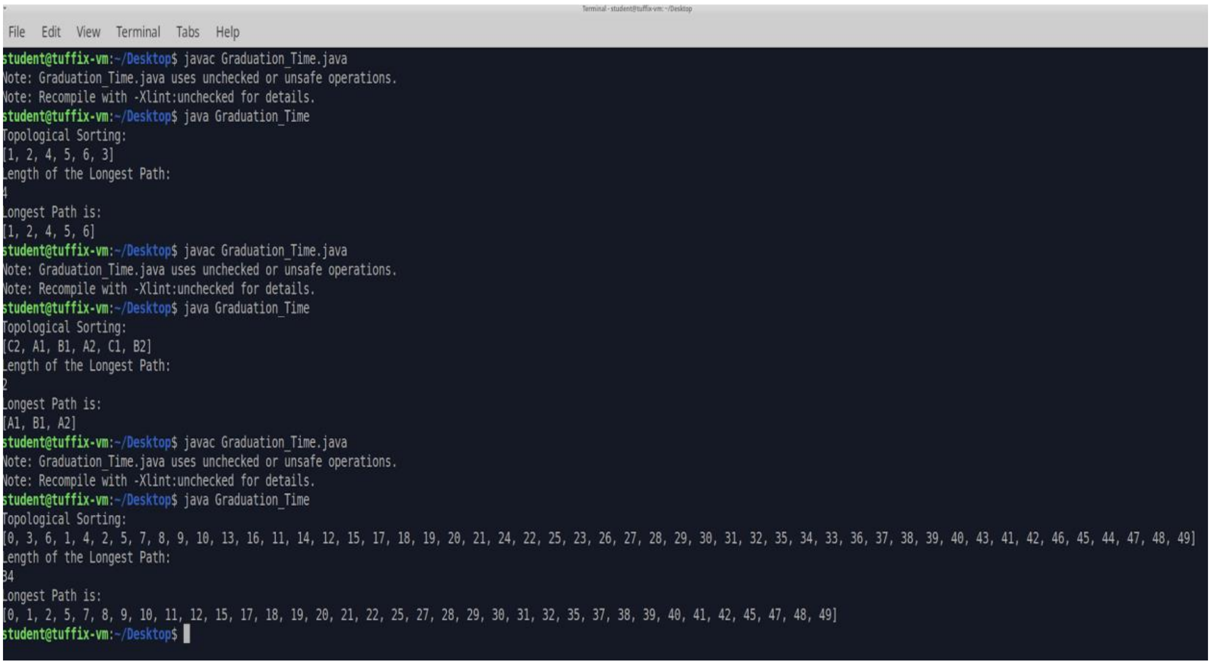


Java - Graduation Time/src/Graduation_Time.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Graduation_Time.java
Console
Writable
Smart Insert
48 : 66 : 1490
285M of 409M

```java
        }
        this.adjList.get(node).add(map);
    }

    public void topologicalsorting() {
        Iterator itr = adjList.entrySet().iterator();
        while (itr.hasNext()) {
            HashMap.Entry entry = (HashMap.Entry) itr.next();
            String str = (String) entry.getKey();
            if (visited.get(str) == false)
                sortUtil(str);
        }
        while (stack.empty() == false) {
            sortedNodes.add((String) stack.pop());
        }
        System.out.println("Graduation Time for the input graph02.txt file");
        System.out.println(" ");
        System.out.println("Topological Sorting: ");
        System.out.println(sortedNodes);
        System.out.println(" ");
    }

    public void sortUtil(String str) {
        visited.put(str, true);
        Iterator<String> itr = adjList.get(str).iterator();
        while (itr.hasNext()) {
            String var = itr.next();
            if (!visited.get(var))
                sortUtil(var);
        }
        stack.push(str);
    }

    public void ReadFile() {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader("C:\\graph.txt"));
            String line;
            int Value = 0;
            while ((line = br.readLine()) != null) {
                if (line.contains("#")) {
```

```
<terminated> Graduation_Time [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\java
Graduation Time for the input graph02.txt file

Topological Sorting:
[C2, A1, B1, A2, C1, B2]

Length of the Longest Path:
2

A Longest Path :
[A1, B1, A2]
```

Combined output:



Terminal - student@tuffix-vm: ~/Desktop
File Edit View Terminal Tabs Help

```
student@tuffix-vm:~/Desktop$ javac Graduation_Time.java
Note: Graduation_Time.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
student@tuffix-vm:~/Desktop$ java Graduation_Time
Topological Sorting:
[1, 2, 4, 5, 6, 3]
Length of the Longest Path:
4
Longest Path is:
[1, 2, 4, 5, 6]
student@tuffix-vm:~/Desktop$ javac Graduation_Time.java
Note: Graduation_Time.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
student@tuffix-vm:~/Desktop$ java Graduation_Time
Topological Sorting:
[C2, A1, B1, A2, C1, B2]
Length of the Longest Path:
2
Longest Path is:
[A1, B1, A2]
student@tuffix-vm:~/Desktop$ javac Graduation_Time.java
Note: Graduation_Time.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
student@tuffix-vm:~/Desktop$ java Graduation_Time
Topological Sorting:
[0, 3, 6, 1, 4, 2, 5, 7, 8, 9, 10, 13, 16, 11, 14, 12, 15, 17, 18, 19, 20, 21, 24, 22, 25, 23, 26, 27, 28, 29, 30, 31, 32, 35, 34, 33, 36, 37, 38, 39, 40, 43, 41, 42, 46, 45, 44, 47, 48, 49]
Length of the Longest Path:
34
Longest Path is:
[0, 1, 2, 5, 7, 8, 9, 10, 11, 12, 15, 17, 18, 19, 20, 21, 22, 25, 27, 28, 29, 30, 31, 32, 35, 37, 38, 39, 40, 41, 42, 45, 47, 48, 49]
student@tuffix-vm:~/Desktop$
```