Zülal Karın **1/04/2022**
12622989076 **Section 02**

CMPE 343 HW-2 REPORT

Problem Statement and Code Design

Task 1: In Task-1, I first needed to create a directed graph based on the given course names and prerequisite order. Next, I had to output true if the first of the two courses given in the last input line was the precondition of the second, and false otherwise. To check the precondition, I made use of the reachability feature in digraphs using dfs and the reverse postorder stack used in Topological sort. I have put a Structure Chart below for you to better understand the steps I followed.

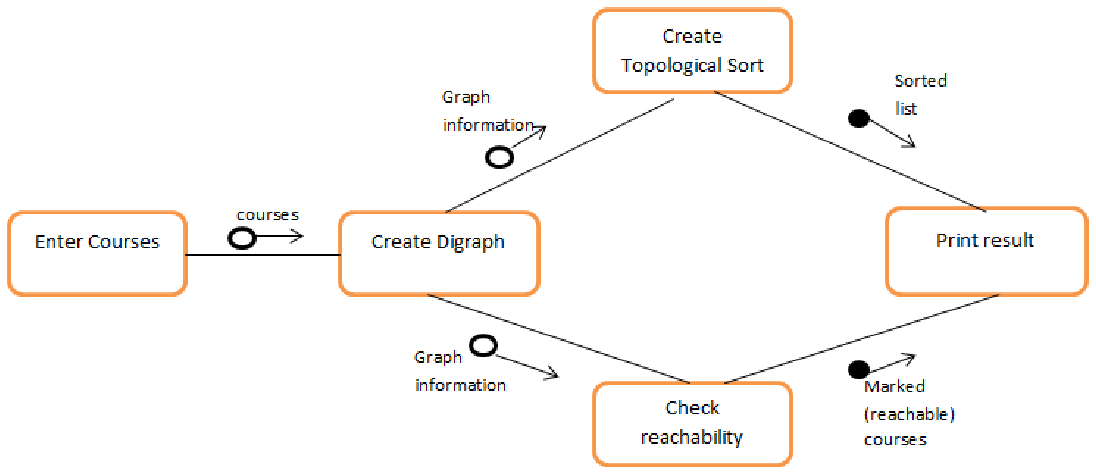


figure 1

Task 2: In Task-2, I needed to find the O's surrounded by S's and convert them to X's. For this, I made a directed graph implementation with the given input. When creating a Directed graph, it was very important to properly connect the "O"s and "S"s. Because my goal was to find the O's on the borders and run dfs for that O's. Then converting the other O's to X, except for all the O's connected to them.

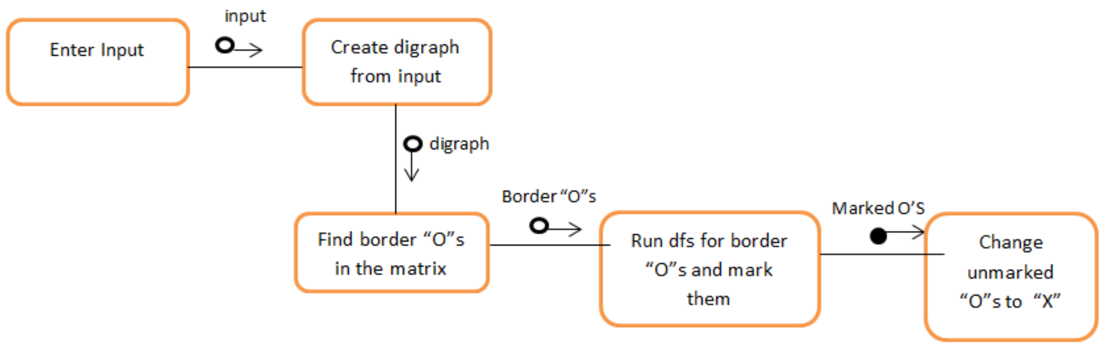

Enter Input
input
Create digraph from input
digraph
Border "O"s
Marked O'S
Find border "O"s in the matrix
Run dfs for border "O"s and mark them
Change unmarked "O"s to "X"

figure 2

Implementation and Functionality

Task 1:

For this task, mainly I created Digraph, DirectedDFS and Driver classes. I will describe these classes and the methods within them using the Structure Diagram in figure1.

1)Create Digraph: I separated the inputs I received according to the signs (","and "-") used in the driver class and created a directed graph using the digraph class. For this, I used the symbol table.

2)Create Topological Sort: The reason I applied Topological Sort was because I wanted to sort the courses in order of prerequisites. According to the reverse poster Stack used in topological sort, the index number of the prerequisite course is bigger than the first course to be taken. I used this topological order to compare index numbers of courses in main.

3) Check Reachability: Although I have determined the prerequisite order of the courses using topological sort, the index numbers here are not decisive for each test case. For this, I also needed to look at the reachability of the courses. For this, I just called dfs for the first given course and checked the other courses that were marked. If a vertex is not marked, it is not likely to be a prerequisite.

Task 2:

1)Create Digraph from input: Creating a digraph was one of the key points. To do this, I had to connect all the S and O's together as I drew them in the figure below. As seen in the figure, it can be seen that the vertexes on all 4 sides of the O's can be reached easily. This was the most important point in my implementation. To do this, I used the createEdges method in the Driver class.

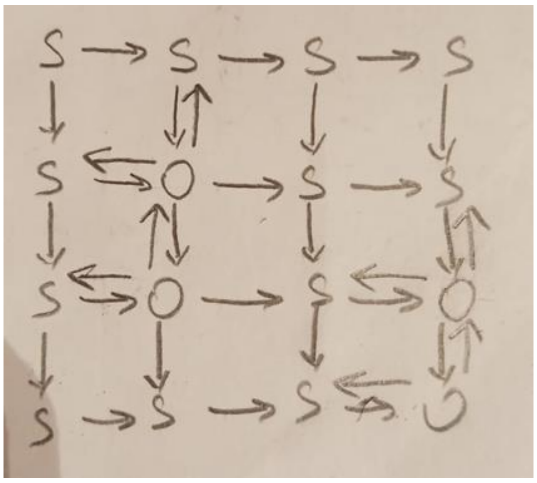


figure 3

2)Run DFS for "O"s at the border: I need to convert all O's surrounded by S's to X's. For O's to be completely surrounded by S's, they must not be on the border or connected to a bordering O. For this, I first need to find all the O's that are on the border and run dfs for them. So I will have found all the O's connected to the O's on the border.

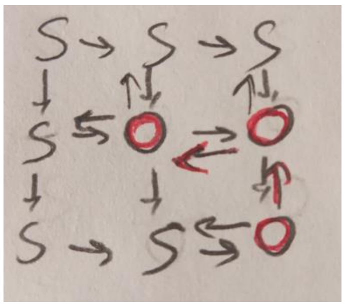


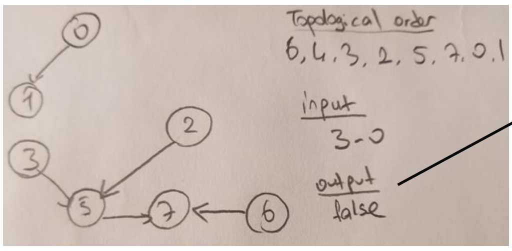
figure 4

For example, in this input I drew, I ran dfs from the O in the lower right corner and found and marked all the O's connected to it.

3) Change unmarked “O”s to “X”: I marked all O's not surrounded by S's in the previous step. That means all non-mark O's are surrounded by S's. I converted all unmarked O's to X and completed the task.

Testing

Task 1: My code passed all tests. But I want to talk about the test case that I had the most difficulty with. I will give the digraph I drew below as an example. Let's assume that the question asked is 3-0. If we look at the topological sort, the result will be true. However, when I use the reachability feature, it seems that I cannot reach vertex 0 from vertex 3. So the output will be "false".



Because,

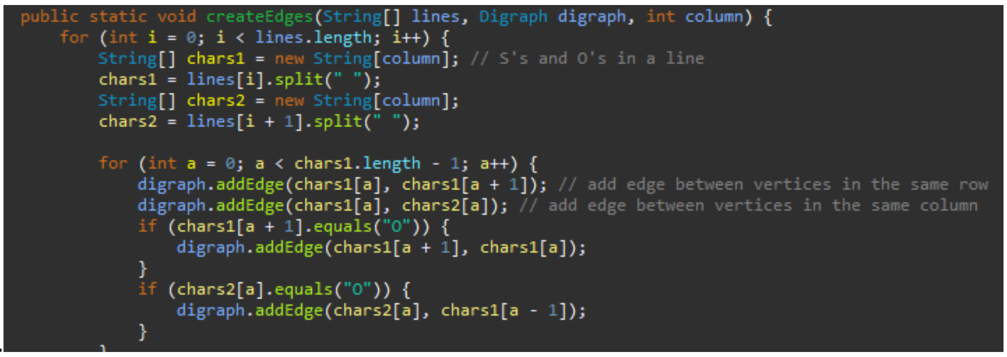
digraph.marked[0] = false

figure 5

Task 2:

I built the graph correctly and using dfs I found all the "O" ssurrounded by S's. I couldn't complete my code because I had mistakes in the implementation part, but I think the path I went is right.

In Figure 6, you can see the method I used to correctly connect S and O’s in the Driver class and create a digraph



```java
public static void createEdges(String[] lines, Digraph digraph, int column) {
    for (int i = 0; i < lines.length; i++) {
        String[] chars1 = new String[column]; // S's and O's in a line
        chars1 = lines[i].split(" ");
        String[] chars2 = new String[column];
        chars2 = lines[i + 1].split(" ");

        for (int a = 0; a < chars1.length - 1; a++) {
            digraph.addEdge(chars1[a], chars1[a + 1]); // add edge between vertices in the same row
            digraph.addEdge(chars1[a], chars2[a]); // add edge between vertices in the same column
            if (chars1[a + 1].equals("O")) {
                digraph.addEdge(chars1[a + 1], chars1[a]);
            }
            if (chars2[a].equals("O")) {
                digraph.addEdge(chars2[a], chars1[a - 1]);
            }
```

figure 6

Final Assessments

- For part 1, it was difficult for me to create the algorithm that determines the reachability for vertices.

- I really had a hard time in part 2. First of all, adapting the given question to graph was the hardest part. Because the question can be easily solved using only two-dimensional matrix, but it was very complicated to solve using graph.

- Both questions got me thinking about how to solve problems using directed graph and its properties. So it was useful for me.