UL Course Schedule Question

Question:

Given:

- Number of courses
- An array of the courses and their prerequisites in the form [Course number, This course is required to have been completed (Prerequisite)]

Task:

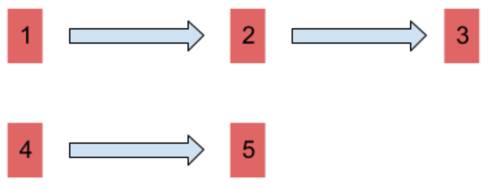
Output true if all the courses can be taken given its prerequisite requirements otherwise print false

Creating a Solution:

The courses can never be completed if a cycle exists between them. See the examples below.

Example 1:

numCourses = 5, prerequisites = [[1,2],[2,3],[4,5]]

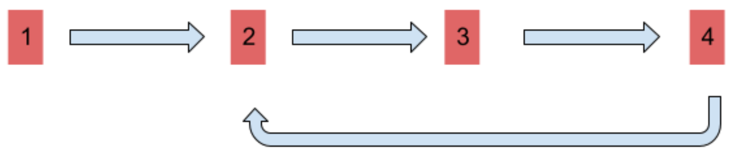


All courses can be taken. Course 1 is dependent on course 2 which is dependent on course 3. Course 4 is dependent on course 5.

Output → True

Example 2:

numCourses = 4, prerequisites = [[1,2],[2,3],[3,4],[4,2]]



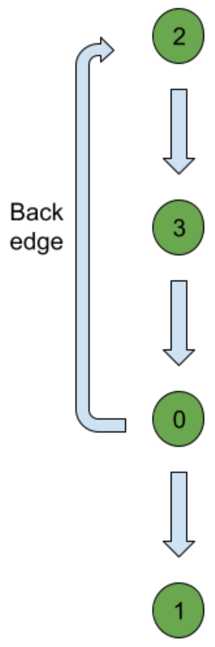
All courses cannot be taken. Course 1 is dependent on course 2 which is dependent on course 3. Course 4 is dependent on course 2. The cycle that is created between the courses and their prerequisites makes this course progression impossible.
Output → False
*This may also be referred to as detecteing a cycle in a directed graph.

Arriving at a solution

This diagram can also be visualised a tree diagram where the nodes are the course numbers and the courses that each course is dependent on can be viewed as its cold nodes or the nodes below it. See the following example.

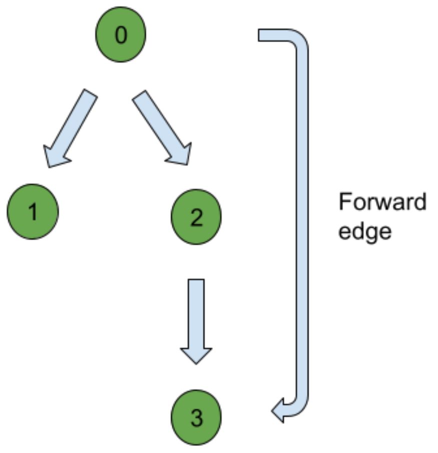
Example 3:

For the example numCourses = 4, prerequisites = [[0,1], [0,2], [2,3], [3,0]] in the above format this can be visualised as the following. The back edge arrow that is created in this loop dictates that this formation will not work.

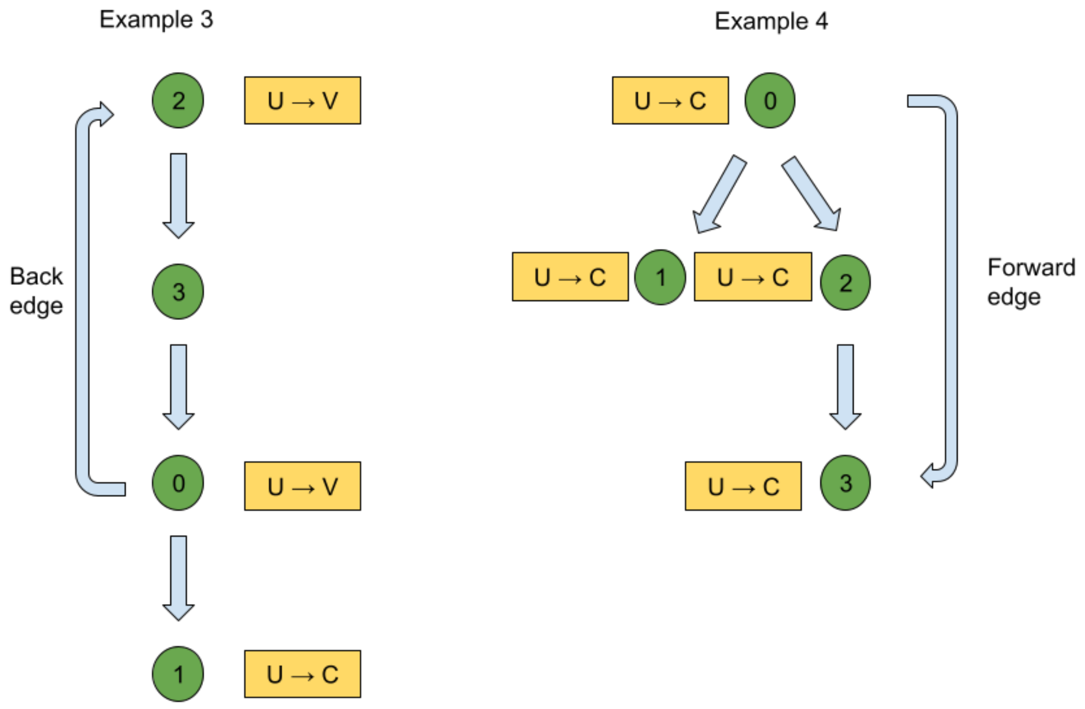


Example 4:

On the other hand for the example numCourses = 4, prerequisites = [[0,1], [0,2], [0,3], [2,3]]. This formation creates a forward edge and now cycle is created. This means that this formation is okay.



To check whether a cycle exists using code we can use a univisited, visited or complete labelling system. At the beginning all of the nodes are marked unvisited by default. We can start marking the nodes from bottom to top. If we are investigating a node it goes into a visited state. In all of a nodes neighbours are in a completed state or node has no neighbours then that node becomes completed. If theres is a situation where nodes cannot because completed the formation is false. See the following diagram.



For programming purposes the states are indicated as follows:
0 → unvisted

1 → completed
2 → visited - being explored

<u>Solution Steps (using example 3):</u>

Step 1: Creating an adjacnecy list

Node	Nodes adjacent
0	1, 2
1	null
2	3
3	0

Step 2: Begin the program

Create a program that takes in the necessary information from the question numCourses and a multidimaensional array prerequisites, and allow it to return a boolean value

```
public boolean canFinish(int numCourses, int[][] prerequisites){
}
```

Step 3: Creating the prerequisite adjacency structure

```
ArrayList<ArrayList<Integer>> adj = new ArrayList<>(numCourses);

for(int i = 0; i < numCourses; i++){
   adj.add(new ArrayList<Integer>());
}

for(int i = 0; i < prerequisites.length; i++){
   adj.get(prerequisites[i][0]).add(prerequisites[i][1]);
}
```

- Create a new array list for the adjacency list called adj. Its the size of the number of courses.
- Create a new array list for each of the elements of the array list.
- Fill the array list with the prerequisites and add edges to adjacency list

Step 4: Checking for a cycle in the adjacency list

```
int[] visited = new int[numCourses];
for(int i = 0; i < numCourses; i++){
   if(visited[i] == 0){
       if(isCycle(i, adj, visited)){
           return false;
```

```
            }
        }

    }
    return true;
```

Check for a cycle in the adjacency list, if there is one there the course schedule is impossible otherwise the course schedule in possible.

Step 5: Create the function icCycle to check for a cycle

```
    private boolean isCycle(int currNode, ArrayList<ArrayList<Integer>> adj,
    int[] visited){
        if(visited[currNode] == 2){
            return true;
        }

        visited[currNode] = 2;
        for(int neighbour: adj.get(currNode)){
            if(visited[neighbour]!= 1){
                if(isCycle(neighbour, adj, visited)){
                    return true;
                }
            }
        }
        visited[currNode] = 1;
        return false;

    }
```

- Check if the current node is in the processing state if it was then there is a cycle return true
- Else change the state of the node to visited i.e 2
- Create a recursive call that completes the same function for the nodes neighbours. Check if the eighbours are marked completed or not.
- If they are not there is a cycle present. If they are change state of current node to processed i.e 1and return false for no cycle being detected.

Completed Solution

```
import java.util.ArrayList;

public class CourseSchedule {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        ArrayList<ArrayList<Integer>> adj = new ArrayList<>(numCourses);

        for(int i = 0; i < numCourses; i++){
            adj.add(new ArrayList<Integer>());
        }
        for(int i = 0; i < prerequisites.length; i++){
            adj.get(prerequisites[i][0]).add(prerequisites[i][1]);
        }

        int[] visited = new int[numCourses];
        for(int i = 0; i < numCourses; i++){
            if(visited[i] == 0){
                if(isCycle(i, adj, visited)){
                    return false;
                }
            }

        }
        return true;
    }

    private boolean isCycle(int currNode, ArrayList<ArrayList<Integer>> adj,
int[] visited){
        if(visited[currNode] == 2){
            return true;
        }

        visited[currNode] = 2;
        for(int neighbour: adj.get(currNode)){
            if(visited[neighbour]!= 1){
                if(isCycle(neighbour, adj, visited)){
                    return true;
                }
            }
        }
        visited[currNode] = 1;
        return false;
    }
}
```