AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

19AIE205
Data Structures and Algorithms - II
&
19AIE202
Operating Systems

Deadlock detection using Topological Sorting

PRESENTED BY :-

Aadharsh Aadhithya - CB.EN.U4AIE20001

Anirudh Edpuganti - CB.EN.U4AIE20005

Madhav Kishor - CB.EN.U4AIE20033

Onteddu Chaitanya Reddy - CB.EN.U4AIE20045

Pillalamarri Akshaya - CB.EN.U4AIE20049

Acknowledgment

We would like to express our special thanks of gratitude to our teacher (Dr. Premjith Sir and Ms. Ganga Gowri Mam) who gave us the golden opportunity to do this wonderful project on the topic (Deadlock detection using Topological Sorting), which also helped us in doing a lot of research and we came to know about so many new things. We are thankful for the opportunity given.

Declaration

We declare that the Submitted Report is our original work and no values and context of it have been copy-pasted from anywhere else. We take full responsibility, that if in future, the report is found invalid or copied, the last decision will be of the faculty concerned. Any form of plagiarism will lead to the disqualification of the report.

Table of Contents

Abstract 5

Introduction 5

Topological sorting 6

Deadlock detection 9

Resource allocation 9

Deadlock Detection Using Topological Sorting 11

Applications 12

References 12

Abstract

This report discusses the concepts of Deadlock detection and Topological sorting and integrates both of them. We dive deep into the working principle and conceptual part of topological sorting and how it is implemented using the Depth First Search (DFS) approach. However, it's not constrained to proceed with DFS but, since the main motive is to integrate it with the deadlock detection and DFS would be the optimal choice for it. Then, we move on to the concept of Deadlock detection conditions which covers the Operating System subject and integrates it with the Data Structures and Algorithms-2, hence giving rise to the topic Deadlock detection using Topological sorting.

Introduction

There are a lot of places where we find a deadlock situation from our house to the assembly. In simple terms, deadlock is nothing but being locked up in an infinite loop which will have no use. Similar is the situation within an operating system. No process can be carried out without the resource being assigned to it. So, it is obvious that each and every process waits for its turn to get the resources. However, in a practical case, it is not so simple for a process to execute without waiting for resources. Sometimes, this waiting time becomes infinite giving rise to a deadlock condition.

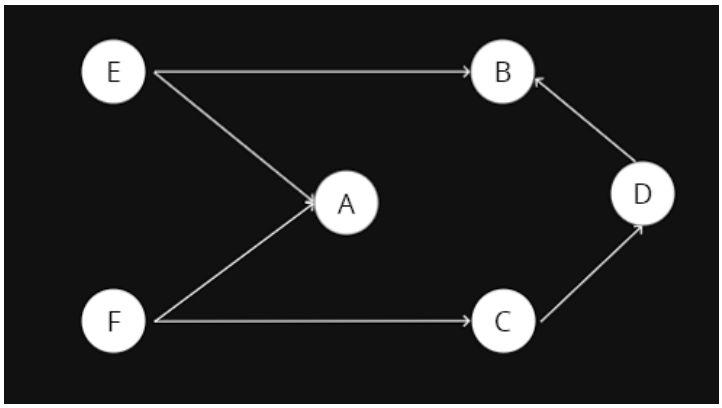
This condition always needs to be detected for the smooth functioning of an operating system. So, in this project, we are going to detect this condition of deadlock with the help of an algorithm known as topological sorting. We will move to the next section to know the detailed working of topological sorting and further finding deadlock with this algorithm.

Topological sorting

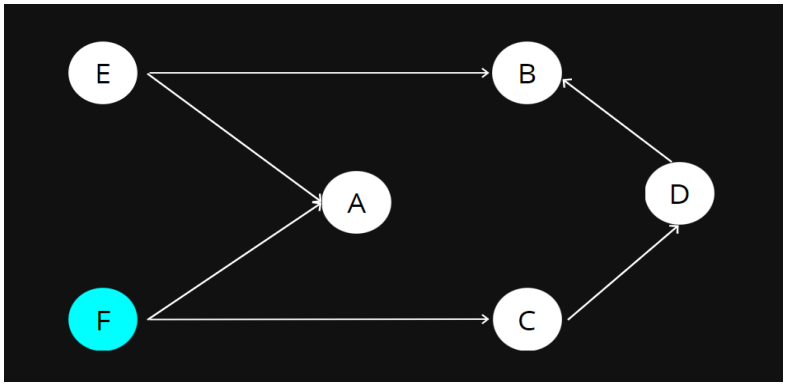
For a Directed Acyclic Graph (DAG), topological sorting is a linear ordering of vertices in which vertex u occurs before v in the ordering for any directed edge (u,v). If the graph is not a DAG, topological sorting is not possible.

Using DFS:

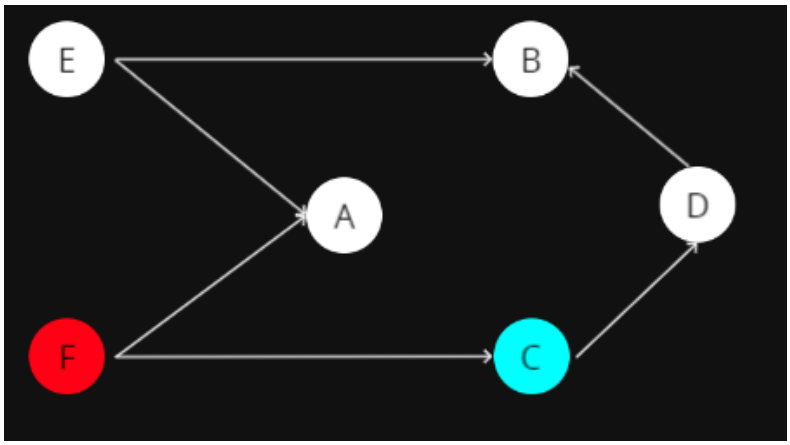
In this project we will Depth First Search (DFS) algorithm to implement topological sorting. Consider the below given graph



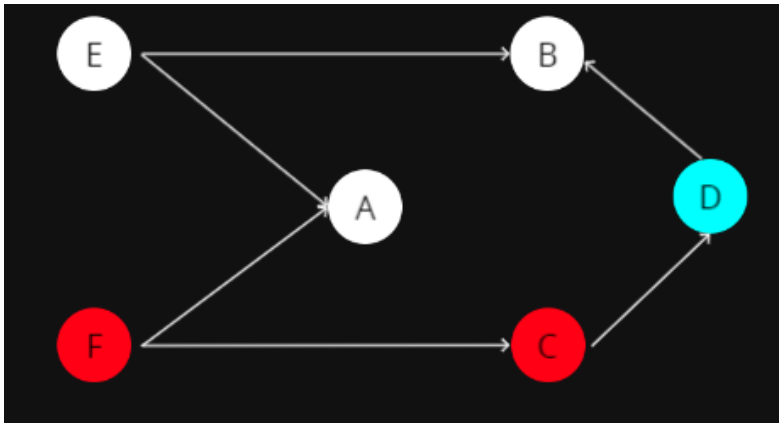
Now we will choose a node which has only outgoing edges. Here, let's say we choose F.



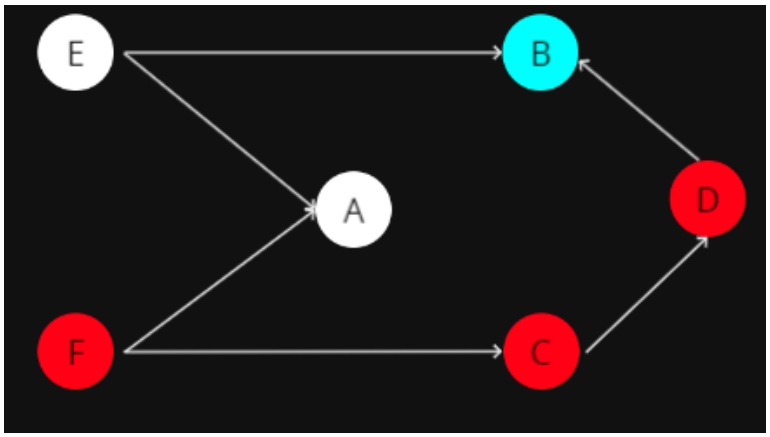
From F we can either go to A or C. Now we will go to C



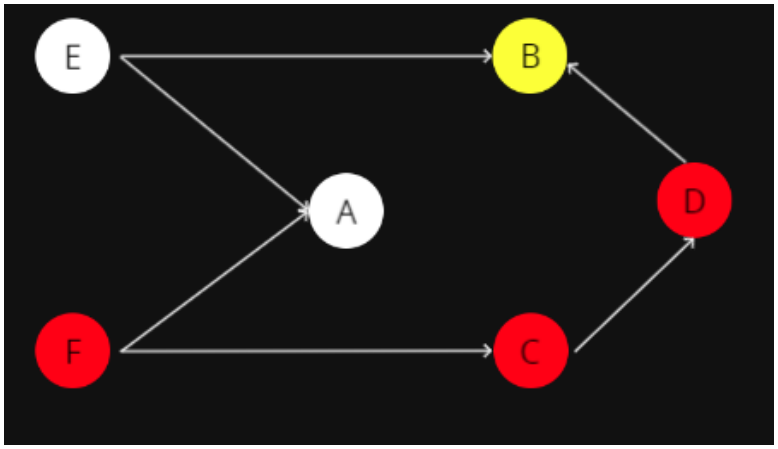
Now C has an outgoing edge D



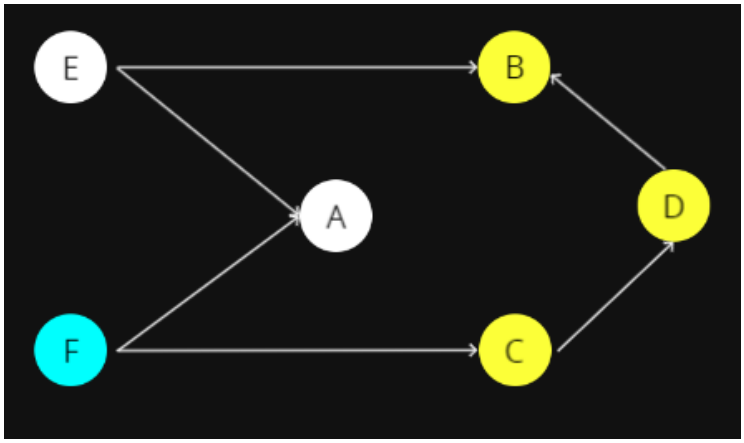
from D we can go to its outgoing edge B.



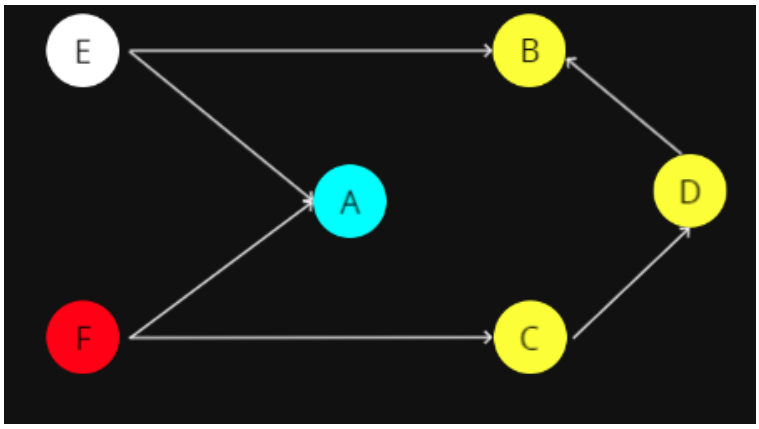
Since B doesn't have any outgoing edge it will go into the stack



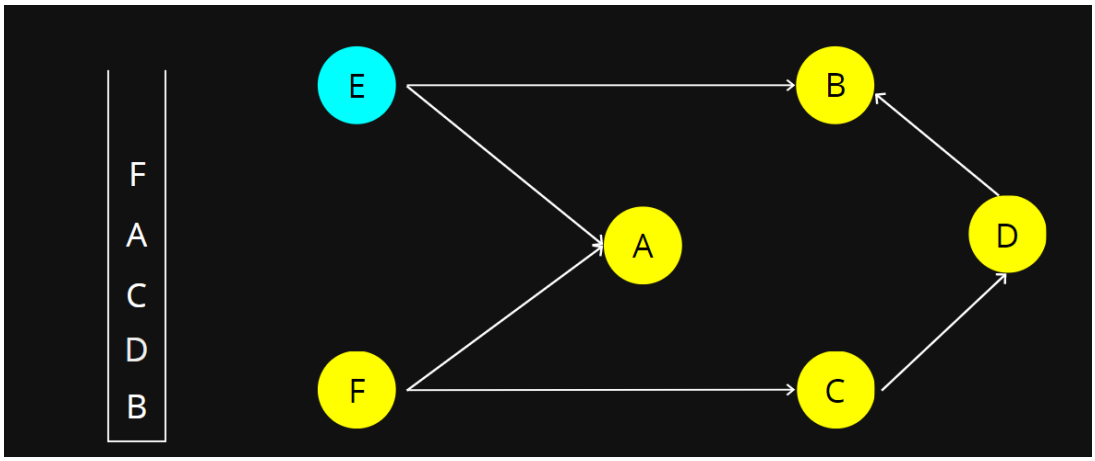
From B it will trace back to D then D doesn't have any outgoing edges except B, it will go into the stack, similarly C will go into stack .



F has another outgoing edge A



A does not have any outgoing edges so it will go into the stack and now since F doesn't have any outgoing edges other than A and C, F will be stacked. E is the last remaining node and since it doesn't have a path to take, it will be pushed into the stack.



Now all the elements in the graph are stacked. So, we will pop each element in the stack and that will be our order of Topological sort for the given Directed Acyclic Graph.
A topological order can be seen as aligning the vertices on a straight line where all edges are from left to right.
Here the topological sorting order is E, F, A, C, D, B

Deadlock detection

A set of two or more processes are deadlocked if they are blocked, each holding a resource and waiting to get a resource.

- ❖ If graph contains no cycles, then no deadlock
- ❖ If graph contains a cycle

When is Deadlock detected?
When the below conditions hold simultaneously, deadlock is detected.

- ***Mutual exclusion***: Resource is allocated to only one process at a time.
- ***Hold and wait***: A process which is holding at least 2 resources is waiting to acquire additional resources held by another process.
- ***No pre-emption***: A resource can be released voluntarily by process holding it, after this process the task is completed
- ***Circular wait***: there exists a set {P0, P1, ... Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ... Pn-1 is waiting for a resource that is held by Pn and Pn is waiting for a resource that is held by P0.

Resource allocation

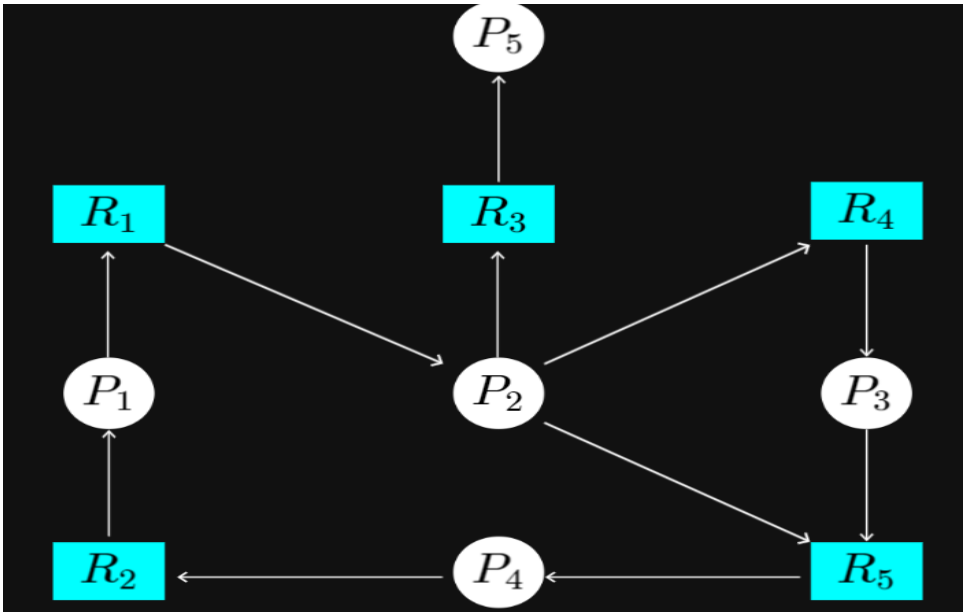
The resource allocation graph explains to us what is the state of the system in terms of processes and resources. Like how many resources are available, how many are allocated and what is the request of each process.
A set of vertices V and a set of edges E.
V is partitioned into two types:

- $P = \{P_1, P_2, ... Pn)$, the set consisting of all the processes in the system.
- $R= \{R_1, R_2, ... Rn\}$, the set consisting of all resource types in the system.

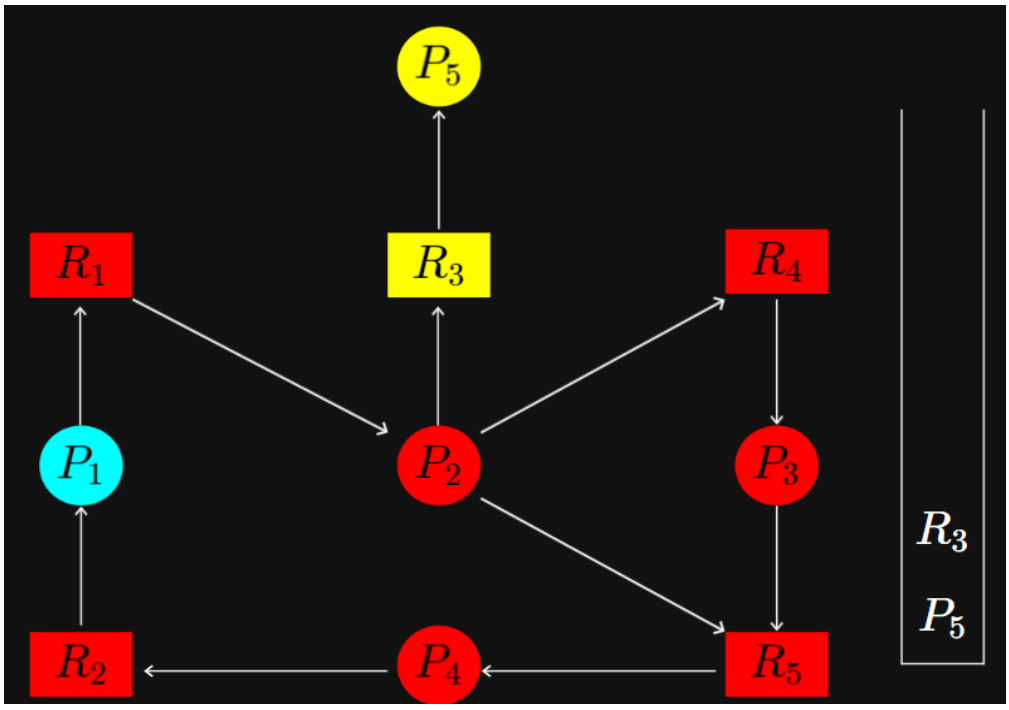
In the Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle.

There will be two edges:
Request edge - directed edge P1 → Rj
Allocation edge - directed edge Rj → Pi

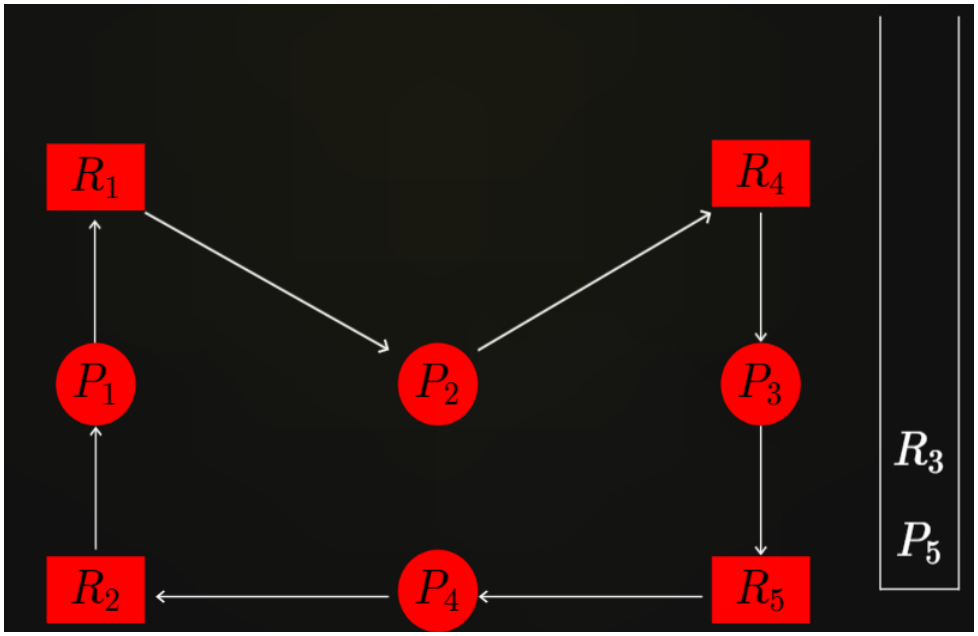
There are 5 resources and 5 processes



We will select any resource from these 5 resources
For suppose here we select R1 and R1 is allocated to P2 .P2 can request resources from R3,R4,R5 ,Now we will select R3 then R3 is allocated to P5 ,P5 doesn't have any resource to request so P5 will go into stack then R3 also goes into stack.P2 can request from R4 and R 5 here we will select R4 then R4 is allocated to P3 and so on it continues.
It forms a circle
So there is a deadlock



This is the path of Deadlock.



Deadlock Detection Using Topological Sorting

Whenever there is a cycle in the Resource Allocation graph, we can say that the given graph is in a deadlock state. Now our problem turns out to be detecting a cycle in a Resource Allocation Graph.

We will be having 3 arrays named List, Visited and Stack. List contains all the nodes that are present in the Resource Allocation Graph (RAG). Visited array contains the information of all the Visited Nodes. Stack array contains information about the edges which don't have any outgoing node or a node from which we can’t go further.

Whenever a node is visited, we will remove that node from the list and append that node to the visited array. Now we will again see the outgoing edges from that node and move to that node. If there is no outgoing edge from the particular node which we have visited then we will remove that node from the visited array and push that node into the stack and trace back our path. If the next node from the particular node is already in visited and we have no other outgoing edge then since the next node is already in visited there is a cycle present in Graph. If the next element is already in the stack, we will trace back our path. This procedure continues either until all the nodes are in stack or the cycle is detected in the given Resource Allocation Graph (RAG)

Applications

- Finding a cycle in graph
- Dependency Resolution
- Sentence Ordering
- Critical path analysis etc

References

- https://www.youtube.com/watch?v=rKQaZuoUR4M
- https://www.youtube.com/watch?v=Q9PIxaNGnig