Q5> Algorithm assignment -6.5

Let's assume that each vertex in the graph which is formed by processes have an additional category information that it holds, which represents the category to which a process belongs and priority associated with it. Initially all the processes will be assigned -1 as their priority.

In order to achieve this grouping instead of using a Queue to hold independent processes we can use an min priority queue to hold the processes by assigning a priority to processes. Let (p1, a,-1), (p2, b,-1), (p3, a,-1), (p4, c,-1), (p5, b,-1) where p1, p2, p3,p4,p5 represents processes and a, b and c be the category. Let -1 be the initial priority assigned

We can use an hashmap to keep track of priority assigned to each category,
Each time we are adding a process to the priority queue we check if priority is assigned to that category by using hashmap. If assigned then we set the priority of the node to it's corresponding value in hashmap else we set it's priority to next priority value.

Min priority queue will ensure that the processes are grouped based on category and ensures that dependency is maintained, if possible else it returns the processes sorted based on their dependencies.

```
// Each vertex has priority and category assigned to it
Struct Vertex
{
        String category;
        Integer priority;
}

Input: Let V be nodes in the graph, E be edges in the graph
Output : resStack containing processes in sorted order also grouped if grouping
           is possible, empty if not possible
FetchProcessesGroupedAndSorted (V, E)
{

        Priority is set to -1 initially for all vertices     v ∈ V
        for each((u,v) in E)
        {
                v.priority =-1
        }



        Initialize indegree for all vertices v ∈ V as 0

        // Calculating indegrees for all the vertices in the graph
        for each((u,v) in E)
        {
                indgree[u] ++;

        }
```

Let minQueue be a minimum priority queue to keep track of all the independent processes sorted based on their priority. Root has process with minimum priority

// HashMap to store the priority value for each category
Let groupMap be an empty hashmap

//Initial priority assigned
priorityVal = 1;

```
// Fetching all nodes with indegree 0 initially
for( v in v)
{
        If(indegree[v] ==0)
        {
                // If priority is already assigned to that category then we
                // fetch the corresponding value from hashmap and assign // it to the
                process and then add that process to minimum  priority queue
                If( groupMap.contains( v.category)
                {
                        v.priority = groupMap.get(v.category)
                        minQueue.add(v)
                }
                // else we assign the next priority and add it to the
                // minimum priority queue
                else
                {
                        v.priority = priorityVal
                        minQueue.add(v)
                        priorityVal++
                }

        }

}
```

Let resStack be a stack which holds all the processes in sorted order

// count of visited processes
Int count=0

```
// Processing all the process which are independent to get sorted order
while ( minQueue is not empty)
{
        u = minQueue.remove()  // first process with minimum priority In
                                        //the queue
```

```
            resStack.push( u)

            // For each adjacent edge of u
            for( (u, v)   in E)
            {
                    // removing the edge (u,v)
                    Indegree[v] - -

                    If(indegree[v] == 0)
                    {

                      // If priority is already assigned to that category then we
                      // fetch the corresponding value from hashmap and assign
                      // it to the process and then add that process to minimum
                      // priority queue
                      if( groupMap.contains( v.category)
                      {
                            v.priority = groupMap.get(v.category)
                            minQueue.add(v)
                      }
                     // else we assign the next priority and add it to the
                     // minimum priority queue
                     else
                      {
                            v.priority = priorityVal
                            minQueue.add(v)
                            priorityVal++
                       }

                    }
            }
            Count ++;
      }

      // cycle exists if count is not equal to the number of processes
      If(count != V.length)
            Return empty array


      // resStack has all the processes in the sorted order, will also be grouped
      // based on category if grouping is possible
     return resStack;

}
```