Answer Script

Question No. 01	
a) Write pseudocode for Dijkstra's algorithm? (Naive Approach)	10
b) Write some limitations of the Dijkstra algorithm? (at least 3).	10

Answer No. 01

a) Pseudocode for Dijkstra's algorithm:

```
function Dijkstra(graph, start_node):
distance = {}
visited = {}

for node in graph:
distance[node] = infinity
visited[node] = false
distance[start_node] = 0

while not all nodes have been visited:
current_node = null
min_distance = infinity

for node in graph:
if not visited[node] and distance[node] < min_distance:
        current_node = node
        min_distance = distance[node]
        visited[current_node] = true

 for neighbor in graph[current_node]:
 new_distance = distance[current_node]+graph[current_node][neighbor]
  if new_distance < distance[neighbor]:
        distance[neighbor] = new_distance

  return distance
```

b) Limitations of Dijkstra's algorithm:

1) It works with non-negative edge weights.
2) It can be slow for large graphs
3) It finds shortest path to one destination only.

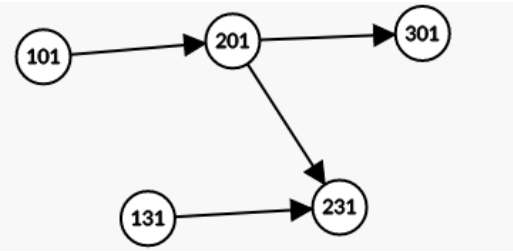
Question No. 02

a) What is Topological Sort? Explain with Example. 15
b) Can a topological sort be implemented in a Directed cyclic graph (DCG)? If so, explain How would you do it? 5

Answer No. 02

a) Topological Sort: Topological sorting is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering.

Suppose we have a graph that represents the prerequisites for a set of courses:



Here,
Course 101 has no prerequisites
Course 201 has one prerequisites (101)
Course 301 has one prerequisites (201)
Course 131 has no prerequisites
Course 231 has one prerequisites (131)

To find a topological sort of this graph, we can follow the following steps:

1. Identifying all vertices with no incoming edges (i.e., no prerequisites).
2. Adding these vertex to the ordering.
3. Removing these vertex and all its outgoing edges from the graph.
4. Repeat steps 1-3 until there are no vertices left in the graph.

Applying these steps, we get the following topologitical sort.
101, 201, 301, 131, 231
101, 201, 131, 231, 301
They are not all possible topological sort for this graph, There are more possibilities.

b) Can a topological sort be implemented in a Directed cyclic graph (DCG)?If so, explain How would you do it?

Ans:

A topological sort cannot be implemented on a directed cyclic graph (DCG) because one of the fundamental requirements of topological sorting is that the graph must be a directed acyclic graph (DAG).

So, in order to implement a topological sort we have to remove the edges that is making the graph cyclic. Only removing the cycle we can implement topological sort.

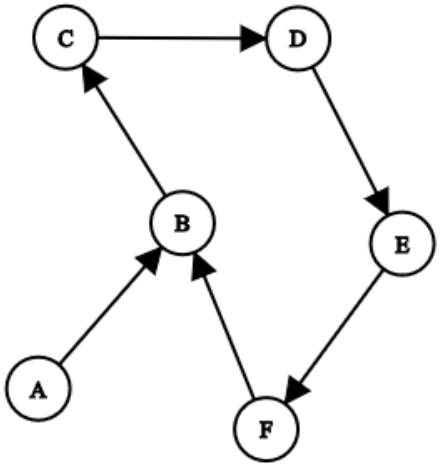
Question No. 03

a) Draw a Directed Cyclic Graph(DCG) and Directed Acyclic Graph(DAG) 10

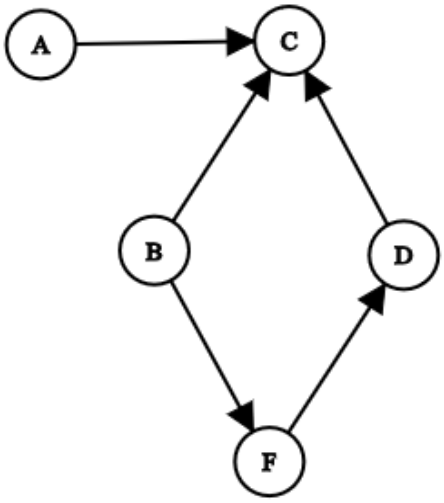
b) Explain the differences between Directed Cyclic Graphs (DCG) and Directed Acyclic Graphs (DAG). (At least 3) 10

Answer No. 03

a) Directed Cyclic Graph:



Directed Acyclic Graph:



b) Difference between Directed Cyclic Graphs (DCG) and Directed Acyclic Graphs (DAG)

Directed Cyclic Graphs (DCG)	Directed Acyclic Graphs (DAG)
It contains cycle.	It doesn't contain any cycle.
Notion of reachability is complicated	Notion of reachability is clear
It is useful for modeling systems that have feedback loops or cycles	It is used to model dependencies between tasks or events

Question No. 04

a) What is the purpose of a base case in a recursive function, and what happens if it is not included? 10

b) What is recursion, and how is it different from iteration? 10

Answer No. 04

a) What is the purpose of a base case in a recursive function, and what happens if it is not included?

Ans: The purpose of a base case in a recursive function is to provide a stopping condition for the recursive process. Without a base case, the recursive function will continue to call itself indefinitely, leading to an infinite loop.

The base case is typically the simplest possible input or inputs for the function, which can be solved without recursion. When the function reaches the base case, it returns a result or stops the recursive process.

b) What is recursion, and how is it different from iteration?

Ans: Recursion is a technique in which a function calls itself with different inputs, breaking down the problem into smaller subproblems until a base case is reached. The function then returns a result for the base case, and the recursion stops.

On the other hand, iteration is a technique in which a loop is used to repeatedly execute a block of code until a specific condition is met. Iteration is used to solve problems that require repetitive computation, such as searching or sorting algorithms.

The key difference between recursion and iteration is the way they solve problems. Recursion breaks down the problem into smaller subproblems and calls itself to solve them, while iteration repeats a block of code until a condition is met.

Question No. 05

a) What is a bipartite graph? 5
b) How can you detect whether a graph is bipartite? 15

Answer No. 05

a) Bipartite Graph: A bipartite graph is a graph in which the nodes can be divided into two disjoint sets such that every edge connects a node from one set to a node from the other set.

b) How can you detect whether a graph is bipartite?

Ans:

There are several ways to detect whether a graph is bipartite. We can detect a bipartite graph by using DFS, BFS. There are two ways to check for a bipartite graph.

1) **A graph is a bipartite graph if and only if it is 2–colorable:**
   If there exists an edge connecting the current vertex to a previously colored vertex with the same color, then we can safely conclude that the graph is not bipartite.

2) **A graph is a bipartite graph if and only if it does not contain an odd cycle:**
   If a graph is not bipartite, it must contain an odd cycle. This is because every cycle of odd length necessarily has an odd number of edges and thus connects an odd number of vertices, making it impossible to split these vertices into two sets with no adjacent vertices. Therefore, an odd cycle can not present in a bipartite graph.