Answer Script

Question No. 01

Question: 1

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)

Dijkstra's algorithm pseudocode:

```
- create a distance array "d"
- initialize all values of "d" to infinity
- d[src] = 0
- create a visited array and mark all nodes as unvisited

- for i = 0  to n - 1:
    - pick the "unvisited" node with minimum d[node]
    - visited[node] = 1
    - for all adj_node of node:
       - if d[node] + c(node, adj_node) < d[adj_node]:
          - d[adj_node] = d[node] + c(node, adj_node)

- output array "d"
```

b)

The Limitations of the Dijkstra algorithm:

1. The graph should be weighted.
2. The weights should be non-negative.
3. It is inefficient for large graphs.
4. It does not work well with disconnected graphs.
5. It cannot handle graphs with cycles.

Question No. 02

Question: 2

a) What is Topological Sort?Explain with Example. 15

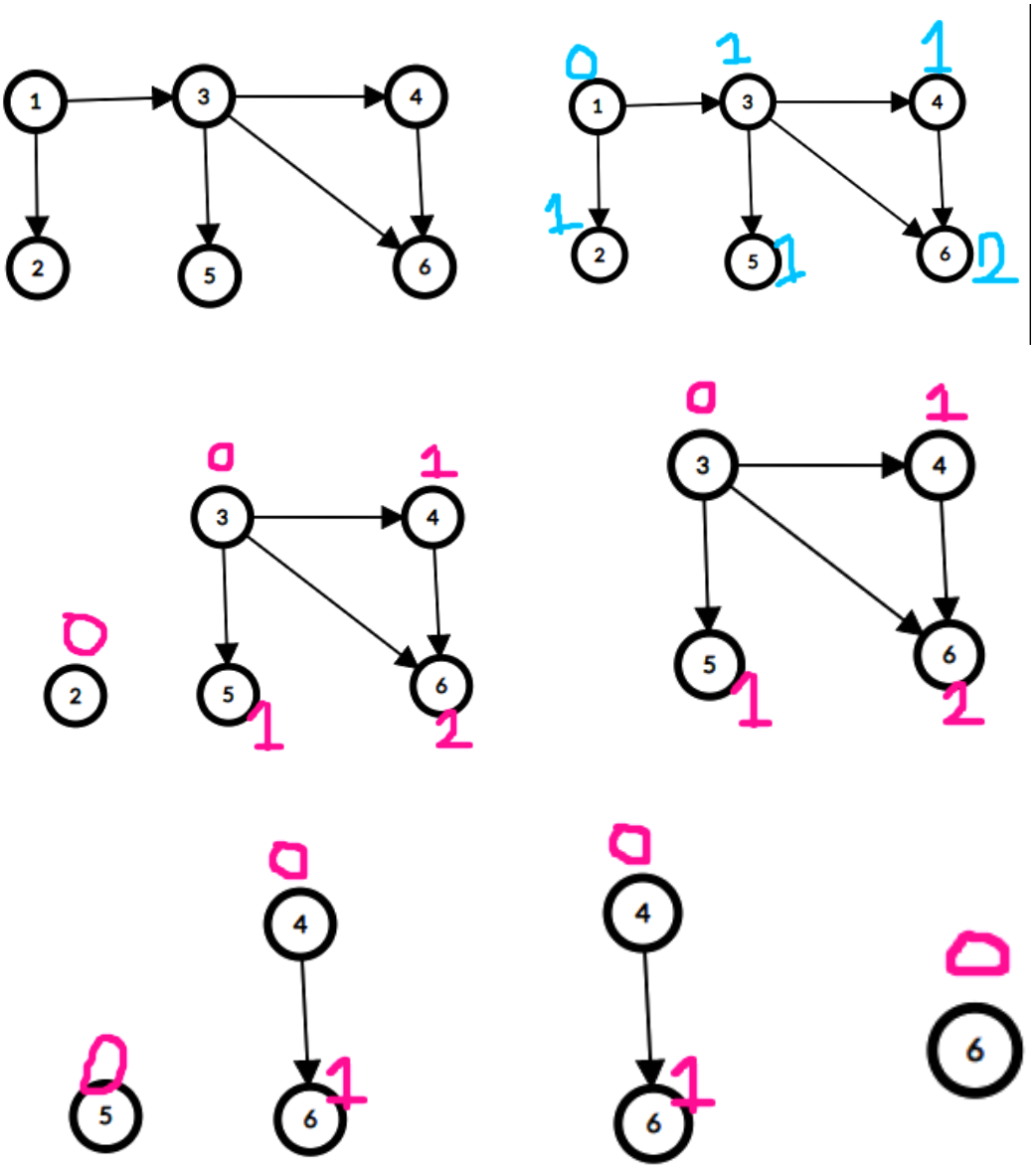
b) Can a topological sort be implemented in a Directed cyclic graph (DCG)?If so, explain How would you do it? 5

Answer No. 02

a)

Topological Sort:

A topological sort of a directed graph is a linear ordering of its vertices in which u occurs before v in the ordering for every directed edge uv from vertex u to vertex v.



This example has 6 nodes and 6 edges.
1 has no incoming edges that's why node 1 indegree 0. Node 2 has 1 incoming edge that's why it's indegree 1. Same way node 3, 4, 5 has 1 incoming edge that's why their indegree 1. Node 6 has 2 incoming edges that's why it's indegree 2.

Now 1st print which has 0 indegree. Node 1 has indegree 0. So, print it.
Topological Ordering: 1

Now Node 2 and 3 have no incoming edges that's why their indegree 0. Node 4 and 5 have 1 incoming edge that's why their indegree 1. Node 6 has 2 incoming edges that's why it's indegree 2.
Now print which has 0 indegree. Node 2 has indegree 0 so print it.
Topological Ordering: 1, 2
Now Node 3 has 0 indegree and prints it.
Topological Ordering: 1, 2, 3

Now, Node 4 and 5 have no incoming edges so it's indegree 0. 1st print node 5.
Topological Ordering: 1, 2, 3, 5

Now, Node 4 has 0 indegree. 1st print 4.
Topological Ordering: 1, 2, 3, 5, 4

Now, Node 6 has no incoming edges. So, it's indegree 0. Print it.
Topological Ordering: 1, 2, 3, 5. 4, 6

b)
No, a topological sort cannot be implemented in a Directed cyclic graph (DCG) because a DCG contains one or more cycles.
A topological sort of a directed graph is a linear ordering of its vertices in which u occurs before v in the ordering for every directed edge uv from vertex u to vertex v.

A cycle is a set of nodes where each node can be reached from another node in the same set by following a path through directed edges. In such a case, there is no way to determine a linear ordering of the nodes that satisfies the requirement of a topological sort.
A directed acyclic graph (DAG), which is a directed graph without any cycles, is necessary for a topological sort. By continually choosing a node in a DAG with no outgoing edges, adding it to the sorted list, and taking away all incoming edges from this node, a topological sort can be carried out. Until all nodes have been added to the sorted list, this process is repeated.

Question No. 03

Question: 3

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

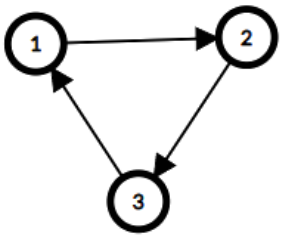
b) Explain the differences between Directed Cyclic Graphs (DCG) and Directed Acyclic Graphs (DAG). (At least 3) 10

Answer No. 03

a)

Directed Cyclic Graph:

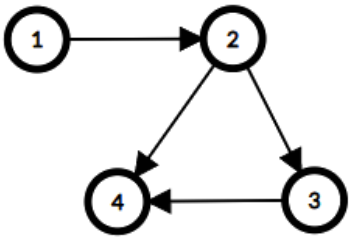
A graph data structure called a directed cyclic graph (DAG) is composed of nodes and edges, where the edges are directed and create cycles. Each edge in a directed graph has a corresponding direction, therefore it can only be traversed in that direction. When a path that follows the course of a node's edges leads back to that node, it is said to be in a cycle.



This is a DCG because there is a cycle in the graph. For example, we can follow the path 1 → 2 → 3 → 1, which starts and ends at the same vertex, 1.

Directed Acyclic Graph:

A directed acyclic graph (DAG) is a directed graph with no directed cycles, which means that there is no path that starts and ends at the same vertex.



This is a DAG because there are no cycles in the graph. We can see that there is no way to follow a path that starts and ends at the same vertex without reversing the direction of at least one edge.

b)

The differences between Directed Cyclic Graphs and Directed Acyclic Graphs:

i. A DCG contains at least one directed cycle, whereas a DAG does not contain any directed cycles. A path that begins and finishes at the same vertex is referred to as a directed cycle.

ii. In contrast to DAGs, which can only express acyclic relationships or dependencies, DCGs can represent cyclic interactions or dependencies between the graph's nodes. This means that DAGs are used to describe processes that have a rigid order or a linear flow, whereas DCGs are frequently employed in instances where cyclic interactions exist, such as when modeling processes with feedback loops.

iii. There is no way to order the vertices of DCGs in a linear sequence that respects the direction of the edges and meets all of the dependencies between the nodes since DCGs cannot be topologically sorted. DAGs, on the other hand, can be topologically sorted, which makes it possible to process the graph quickly by resolving dependencies in a particular order.

Question No. 04

Question: 4

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)

The purpose of a base case in a recursive function is to define the end condition for the recursion. It provides a stopping point for the function and prevents it from infinitely calling itself, which could cause an infinite loop and crash the program.

If a base case is not included in a recursive function, the function will continue to call itself indefinitely, resulting in an infinite loop. This can consume all available memory and CPU resources on the system.

Example:

```
int factorial(int n) {
   if(n == 0) {    // base case
      return 1;
   } else {
      return n * factorial(n-1);
   }
}
```

This is a recursive function to calculate the factorial. Here if condition is base case. When n is equal to 0 this recursive function ends.

b)

Recursion is a technique where a function calls itself repeatedly until it reaches a base case, which is a condition that stops the recursion. Each time the function calls itself, it creates a new instance of the function with its own set of parameters and local variables.

Iteration is a process where a loop is used to repeatedly execute a block of code until a certain condition is met. The loop runs for a fixed number of times or until a specific condition is satisfied.

The method used to solve the problem is where recursion and iteration diverge. Iteration employs a loop to constantly run a block of code until the problem is solved, while recursion uses a function that calls itself to break the problem into smaller subproblems. For some sorts of problems, recursion can be more elegant and succinct than iteration, but it can also be less effective and more difficult to debug in particular circumstances.

Question No. 05

Question:5

a) What is a bipartite graph? 5

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

Answer No. 05

a)

A bipartite graph is a graph in which the nodes can be partitioned into two sets such that no two nodes within the same set are adjacent.

b)

Graph coloring is used to detect whether a graph is bipartite or not.

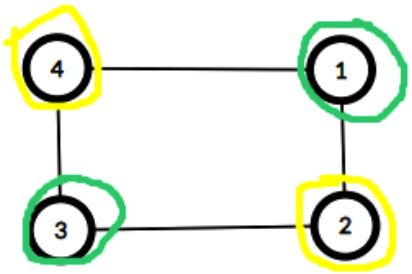
The basic idea behind this approach is to color each node of the graph either green or yellow, such that no two adjacent nodes have the same color. If it is possible to color all the nodes in the graph using only two colors, then the graph is bipartite; otherwise, it is not.

Here's how the algorithm works:

1. Choose an arbitrary node in the graph and color it yellow.
2. Color all its neighbors green.
3. Repeat step 2 for all neighbors of the green nodes, coloring them yellow. Continue this process until all nodes are colored or a conflict arises where a node has two different colors.

The graph is either bipartite or not bipartite depending on when we come across a node with two different colors during the process described above.

Bipartite:



Not Bipartite: