SNJB's Late Sau. Kantabai Bhavarlaji Jain College of Engineering
Department of AI & DS Engineering
University End Sem QP Solution (Academic Year : 2024-25 - Semester 2)
Course Name: Data Structures And Algorithms

Q-1 (a): Define the following terms:

1. Complete Graph
2. Strongly Connected Graph
3. Weighted Graph

Answer:

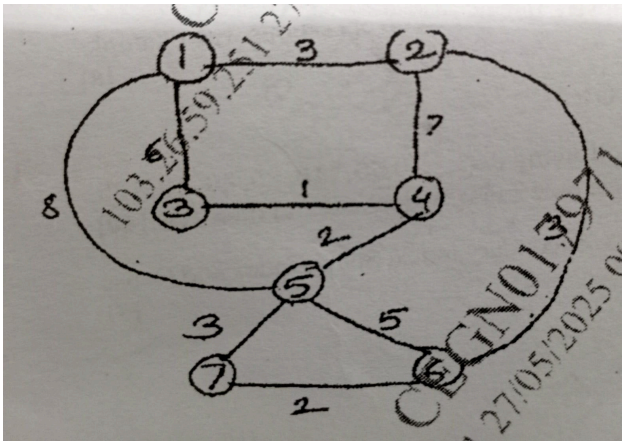
1. **Complete Graph:** A graph in which every pair of distinct vertices is connected by a unique edge.
2. **Strongly Connected Graph:** In a directed graph, a graph is strongly connected if there is a path from every vertex to every other vertex.
3. **Weighted Graph:** A graph where each edge has an associated numerical value (weight).

Q-1 (b):Write a pseudo C/C++ code for depth traversal of a graph.

Answer (DFS – Pseudocode in C++ Style):

```
void DFS(int node, bool visited[], vector<int> adj[]) {
   visited[node] = true;
   cout << node << " ";
   for (int i : adj[node]) {
      if (!visited[i]) {
         DFS(i, visited, adj);
      }
   }
}
```

Q-1 (c): Find MST (Minimum Spanning Tree) for the following graph using Prim's algorithm.



Vertices:

The graph contains **7 vertices**:
1, 2, 3, 4, 5, 6, 7

Edges and Weights (from image):

Edge	Weight
(1 - 2)	3
(1 - 3)	10
(1 - 4)	5
(2 - 4)	7
(3 - 4)	1
(3 - 5)	2
(4 - 5)	2
(4 - 6)	5
(5 - 6)	4
(5 - 7)	7
(6 - 7)	2

Prim's Algorithm Steps:

Start from any vertex — let's choose **vertex 1**

Step 1:

- MST Set: {1}
- Available edges from 1: (1-2,3), (1-3,10), (1-4,5)
- **Choose edge (1 - 2), weight 3**

Step 2:

- MST Set: {1, 2}
- Available edges: (1-4,5), (1-3,10), (2-4,7)
- **Choose edge (1 - 4), weight 5**

Step 3:

- MST Set: {1, 2, 4}
- Available edges: (4-3,1), (4-5,2), (4-6,5), (2-4,7), (1-3,10)
- **Choose edge (4 - 3), weight 1**

Step 4:

- MST Set: {1, 2, 3, 4}
- Available edges: (3-5,2), (4-5,2), (4-6,5)
- **Choose edge (3 - 5), weight 2**

Step 5:

- MST Set: {1, 2, 3, 4, 5}
- Available edges: (5-6,4), (5-7,7), (4-6,5)
- **Choose edge (5 - 6), weight 4**

Step 6:

- MST Set: {1, 2, 3, 4, 5, 6}
- Available edges: (6 - 7, 2), (5 - 7, 7)
- **Choose edge (6 - 7), weight 2**

Final MST Edges:

Edge Weight

1 - 2 3

1 - 4 5

4 - 3 1

Edge Weight

3 - 5 2

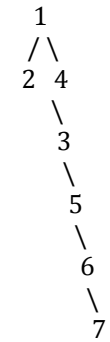
5 - 6 4

6 - 7 2

Total Weight of MST:

3 + 5 + 1 + 2 + 4 + 2 = 17

MST Diagram (Text Form):



Q-2 (a):Write an algorithm for BFS traversal of graph.

Answer (BFS – Pseudocode):

```
void BFS(int start, vector<int> adj[], int n) {
   vector<bool> visited(n, false);
   queue<int> q;
   visited[start] = true;
   q.push(start);

   while (!q.empty()) {
      int node = q.front(); q.pop();
      cout << node << " ";
      for (int i : adj[node]) {
         if (!visited[i]) {
            visited[i] = true;
            q.push(i);
         }
      }
   }
}
```

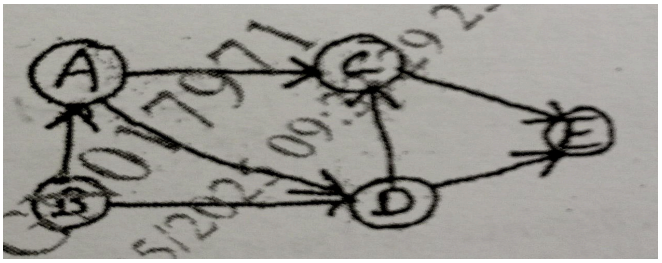
Q-2 (b): Give difference between Prim's & Kruskal's Algorithm.

Answer:

Feature	Prim's Algorithm	Kruskal's Algorithm
Approach	Grows MST from a starting node	Sorts edges by weight globally
Data Structure Used	Min-heap / Priority queue	Union-Find (Disjoint Set)
Graph Type	Works better on dense graphs	Efficient on sparse graphs
Cycle Handling	Avoids cycles via visited set	Uses union-find to detect cycle

---

Q-2 (C): **What is topological sorting ? Find topological sorting of given graph**



Observed Directed Edges:

From the image, we can identify the following directed edges:

- A → B
- A → C
- B → D
- C → D
- C → E
- D → E

Possible Solutions Depending on Question:

1. Topological Sorting (for Directed Acyclic Graph)

If the question is to find the **topological order**, here's one possible order:

1. A (no incoming edges)
2. B, C (after A)
3. D (after B and C)
4. E (after C and D)

One valid topological sort:
A → B → C → D → E
Another:
A → C → B → D → E

Topological sort is not unique; multiple valid orders may exist.

2. Adjacency List Representation

A → B, C
B → D
C → D, E
D → E
E → —

3. If it's a Shortest Path Problem (e.g., from A to E)

If weights were provided, we could apply **Dijkstra's Algorithm**.

Please confirm the actual question:

- Do you want:
  - Topological sort?
  - Shortest path from A to E?
  - Adjacency matrix/list?
  - Path enumeration?
  - DFS/BFS traversal?

Q-3 (a): What is topological sorting? Find topological sorting for the graph shown below.

(*Note: Graph not available; general answer provided.*)

Answer:
Topological Sorting is the linear ordering of a directed acyclic graph (DAG) such that for every directed edge u → v, vertex u appears before v.

Algorithm (Kahn's BFS-based):

1. Compute in-degree of all vertices.
2. Enqueue all vertices with in-degree 0.
3. While queue is not empty:
   a. Dequeue vertex and add to topological order.
   b. Decrease in-degree of all adjacent vertices.
   c. If in-degree becomes 0, enqueue them.

Q-3 (b):Explain static and dynamic tree tables with suitable examples.

Answer:

- **Static Tree Tables:** Tree structure is predefined and doesn't change. Ex: Syntax trees in compilers.
- **Dynamic Tree Tables:** Nodes can be added or deleted dynamically. Ex: Binary Search Trees, AVL Trees used in databases.

Q-3 (c):Explain with example Red-Black tree.

Answer:
Red-Black Tree is a self-balancing BST with these properties:

1. Every node is either red or black.
2. Root is always black.
3. Red nodes can't have red children.
4. Every path from a node to its descendant NULL nodes has the same number of black nodes.

Example:
Insert 10, 20, 30: Re-balancing is needed, rotations applied.

Q-4 (a):Write functions for LL and LR rotation with respect to AVL tree.

Answer:

LL Rotation (Right Rotation):

```
Node* rightRotate(Node* y) {
   Node* x = y->left;
   Node* T2 = x->right;

   x->right = y;
   y->left = T2;

   updateHeight(y);
   updateHeight(x);

   return x;
}
```

LR Rotation (Left-Right Rotation):

```
Node* leftRightRotate(Node* z) {
   z->left = leftRotate(z->left);
   return rightRotate(z);
}
```

Q-4 (b):Explain with example K-dimensional tree.

Answer:
K-D Tree is a binary tree used for multi-dimensional searches like range search, nearest neighbor.
Each level splits the space on one dimension.

Example: 2D Tree insertion based on x and y coordinates alternatively.

Q-4 (c):Construct AVL tree for following data: 15,20,24,10,13,7,30,36,25

Answer:
Steps:

1. Insert 15 → Balanced
2. Insert 20 → Balanced
3. Insert 24 → **LL Rotation**
4. Insert 10, 13, 7 → **RL and LL Rotations**
5. Insert 30, 36, 25 → **RR and LR Rotations**

Final AVL tree is height-balanced with rotations applied during insertion.

Q-5 (a):Write an algorithm to delete data from B-Tree.

Answer (Pseudocode):

```
function deleteKey(x, k):
  if x is leaf:
    if k in x: delete it
    else: return "Not found"
  else:
    if k in x:
      if predecessor or successor node has >= t keys:
        replace k
      else:
        merge children and recurse
    else:
      recurse in child with k
```

Q-5 (b):Explain Trie data structure to insert, delete, search.

Answer:
Trie is a prefix tree for string keys.

Insert:

```
for char in word:
  if not node[char]: create node
  move to node[char]
mark end of word
```

Search: Traverse each character; if any missing → return false.
Delete: Unmark end of word and recursively delete unused nodes.

Q-6 (a): Create B+ Tree of Order 5

Given Data:
5, 30, 110, 80, 40, 10, 120, 60, 20, 70, 100, 35, 90
Order = 5 → Max 4 keys per node and Max 5 children.

Insertion:

1. Start inserting in sorted order:
   → Insert: 5, 10, 20, 30
   → Insert 35 → Overflow, split into: [5, 10] | 20 | [30, 35] → root becomes 20

   Structure:

   ```
     [20]
    /   \
  [5,10] [30,35]
   ```
2. Insert 40 → goes to [30,35] → [30,35,40]
3. Insert 60 → [30,35,40,60]
4. Insert 70 → Overflow → Split [30,35,40,60,70] → [30,35] | 40 | [60,70]
   Root remains [20, 40]
5. Continue adding: 80, 90, 100, 110, 120
   → Final B+ Tree before deletion:

   ```
        [20, 40, 80]
      /   |    |   \
  [5,10] [30,35,60,70] [90,100,110] [120]
   ```

Deletion:

- **Delete 90** → From leaf node [90,100,110] → becomes [100,110] (no merge needed)
- **Delete 100** → [110] (still fine as min=2 for leaf nodes with 4 capacity)

Final Tree remains balanced.

Q-6 (b): Index Types

1. **Primary Index**: Index built on the primary key.
   Example: If EMPID is primary key, index stores EMPID and pointer to record.
2. **Secondary Index**: Built on non-primary attributes.
   Example: Index on EMPNAME.
3. **Sparse Index**: Index contains entries for **some** records.
   Example:

   Index: [1000, 2000, 3000]
   Points to: Record blocks
4. **Dense Index**: Contains entries for **every** record.
   Example:

   Index: [1001, 1002, 1003]
   Points to each individual record

Q-7 (a): Sequential vs Direct Access File Organization

Feature	Sequential	Direct Access
Access Method	Record by record	Any record directly
Efficiency	Good for batch processing	Good for real-time apps
Performance	Slower for random access	Fast for random access
Use Case	Payroll, logs	Databases, airline reservation

Q-7 (b): Multilist Structure

- **Concept**: Used when a record belongs to more than one list.
- Each record maintains multiple pointers for different lists.

Example: Coral Ring (Multilist in DB)

- Employee record belongs to departments & projects.
- One pointer points to next employee in the same department.
- Another pointer points to next employee in same project.

Q-8 (a): Linked File Organization & Variants

- **Linked Organization**: Records stored in linked list form. Each record has pointer to next.
- **Variants:**

1. **Inverted File**:
    - Index maintains list of pointers for attributes.
    - Efficient for search by non-primary keys.
2. **Cellular Partition**:
    - Disk divided into partitions (cells), linked by pointers.
    - Efficient space utilization and direct linking.

Q-8 (b): Sequential vs Indexed Sequential File Organization

Feature	Sequential	Indexed Sequential
Structure	Linear sequence	Sequential with index on key
Searching	Linear scan	Binary search on index
Update/Insert/Delete	Slower	Faster due to index
Storage	Simpler	Needs extra space for index

Advantages (Indexed Sequential):

- Faster search
- Supports both batch and online

Disadvantages:

- Extra storage for index
- Index maintenance overhead