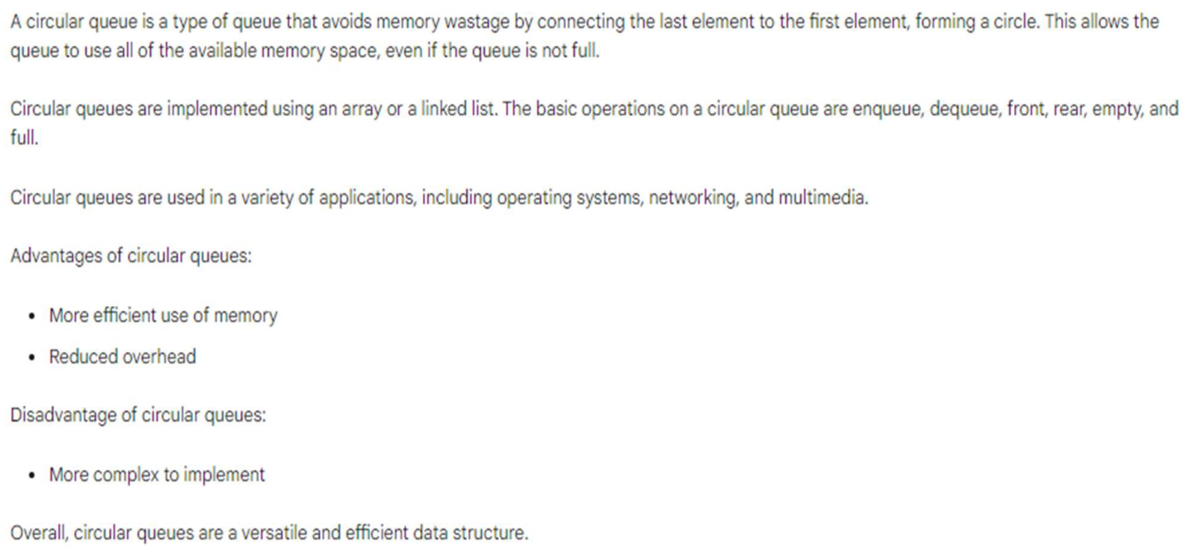
Answer 1



A circular queue is a type of queue that avoids memory wastage by connecting the last element to the first element, forming a circle. This allows the queue to use all of the available memory space, even if the queue is not full.

Circular queues are implemented using an array or a linked list. The basic operations on a circular queue are enqueue, dequeue, front, rear, empty, and full.

Circular queues are used in a variety of applications, including operating systems, networking, and multimedia.

Advantages of circular queues:

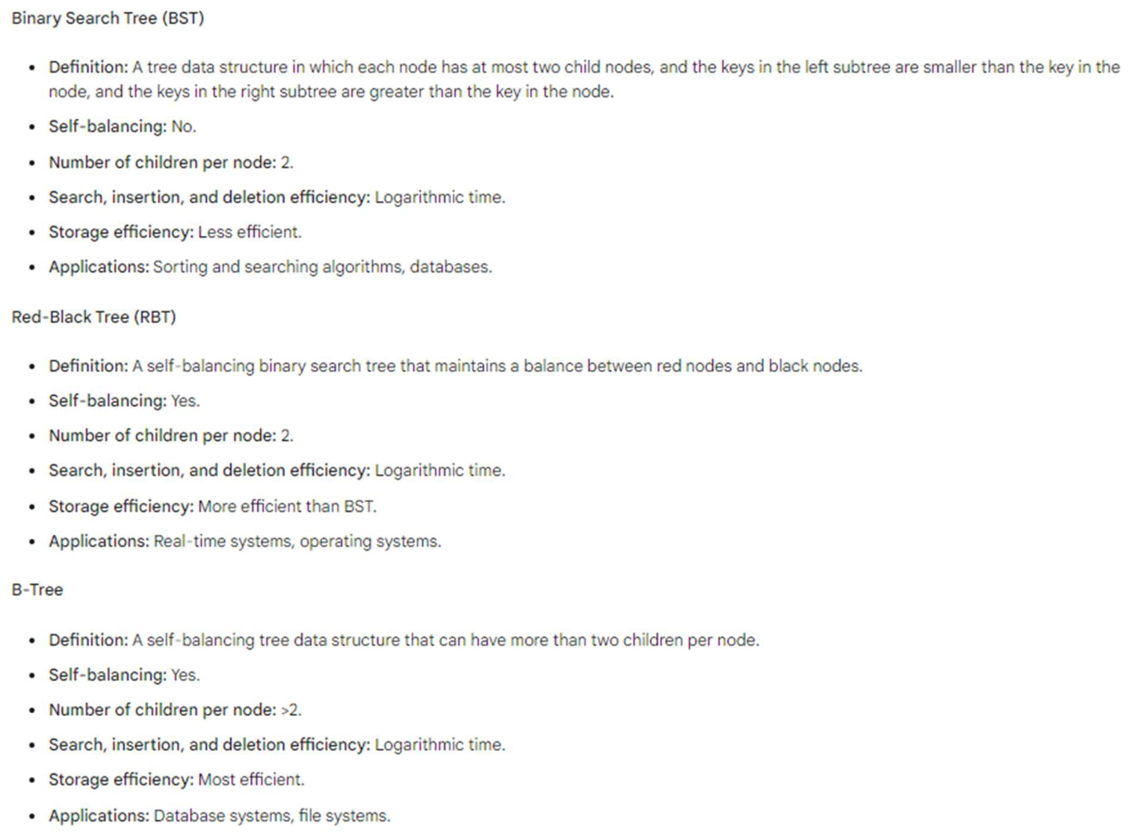
- More efficient use of memory
- Reduced overhead

Disadvantage of circular queues:

- More complex to implement

Overall, circular queues are a versatile and efficient data structure.

Answer 2



Binary Search Tree (BST)

- **Definition:** A tree data structure in which each node has at most two child nodes, and the keys in the left subtree are smaller than the key in the node, and the keys in the right subtree are greater than the key in the node.
- **Self-balancing:** No.
- **Number of children per node:** 2.
- **Search, insertion, and deletion efficiency:** Logarithmic time.
- **Storage efficiency:** Less efficient.
- **Applications:** Sorting and searching algorithms, databases.

Red-Black Tree (RBT)

- **Definition:** A self-balancing binary search tree that maintains a balance between red nodes and black nodes.
- **Self-balancing:** Yes.
- **Number of children per node:** 2.
- **Search, insertion, and deletion efficiency:** Logarithmic time.
- **Storage efficiency:** More efficient than BST.
- **Applications:** Real-time systems, operating systems.

B-Tree

- **Definition:** A self-balancing tree data structure that can have more than two children per node.
- **Self-balancing:** Yes.
- **Number of children per node:** >2.
- **Search, insertion, and deletion efficiency:** Logarithmic time.
- **Storage efficiency:** Most efficient.
- **Applications:** Database systems, file systems.

Answer 3

Inserting 2 into an empty AVL tree

```
        NULL
        /
        2
       /
       NULL
      /
      NULL
```

Rotation: None

Inserting 1 into the AVL tree

```
       2
       /
       1
```

Rotation: None

Inserting 4 into the AVL tree

```
       2
      / \
      1   4
```

Rotation: None

Inserting 5 into the AVL tree

```
       2
      / \
      1   4
     5
```

Rotation: None

Inserting 9 into the AVL tree

```
          2
         / \
        1   4
       /
      5
     / \
    NULL  9
```

Rotation: None

Inserting 3 into the AVL tree

```
          2
         / \
        1   4
       / \
      5   9
     /
    3
```

Rotation: Right rotation at node 4

```
          4
         / \
        2   9
       /   / \
      3   5   NULL
```

Inserting 6 into the AVL tree

```
          4
         / \
     2   9
    /   / \
   3   5   6
```

Rotation: None

Inserting 7 into the AVL tree

```
       4
      / \
     2   9
    /   / \
   3   5   6
  /
 7
```

Rotation: Left-right rotation at node 9

```
       4
      / \
     2   7
    / \
   3   5
  /
 6
```

Overall, the following rotations are performed after inserting 2, 1, 4, 5, 9, 3, 6, 7 into an initially empty AVL tree:

Right rotation at node 4

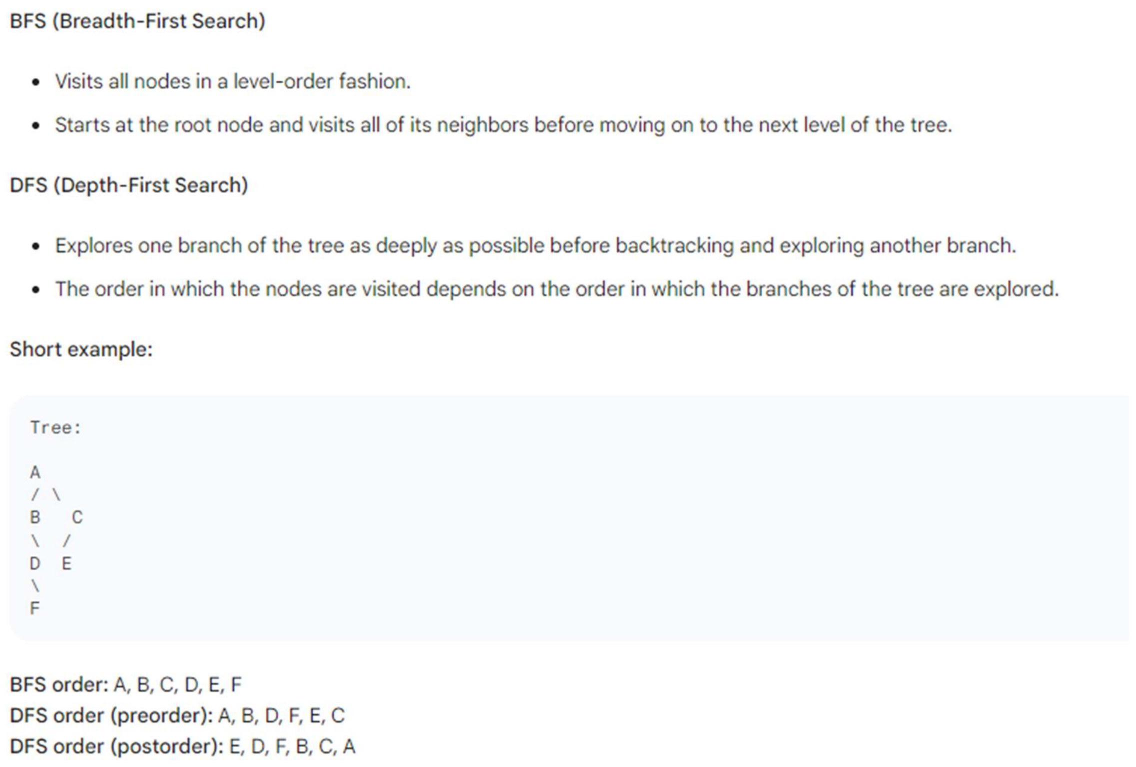
Left-right rotation at node 9

The resulting AVL tree is:

```
       4
      / \
     2   7
```

```
      / \
     3   5
    /
   6
```

Answer 4



BFS (Breadth-First Search)

- Visits all nodes in a level-order fashion.
- Starts at the root node and visits all of its neighbors before moving on to the next level of the tree.

DFS (Depth-First Search)

- Explores one branch of the tree as deeply as possible before backtracking and exploring another branch.
- The order in which the nodes are visited depends on the order in which the branches of the tree are explored.

Short example:

```
Tree:

A
/ \
B   C
\   /
D   E
\
F
```

BFS order: A, B, C, D, E, F
DFS order (preorder): A, B, D, F, E, C
DFS order (postorder): E, D, F, B, C, A

Answer 5

On next page

To solve the problem in the image, we can use Kruskal's algorithm to find the minimum spanning tree (MST) of the graph. Kruskal's algorithm works by sorting the edges of the graph in increasing order of weight and then adding them to the MST one by one, as long as they do not create a cycle.

The following is the sequence of steps that Kruskal's algorithm would take to solve the problem in the image:

1. Sort the edges of the graph in increasing order of weight:

```
Edge | Weight
-----|--------
A-B | 1
B-C | 2
C-D | 3
D-E | 4
E-F | 5
A-C | 6
```

2. Initialize the MST to be empty.

3. Add the edge with the smallest weight to the MST and check if it creates a cycle. If it does not create a cycle, then the edge is added to the MST. Otherwise, the edge is discarded.

4. Repeat step 3 until all of the vertices in the graph are connected in the MST.

The following is the MST of the graph in the image:

```
A
/ \
B   C
\  /
D  E
\
F
```

The total weight of the MST is 10.