**1. Hash Table Implementation**

**Output Explanation:**

* **Insert via Linear Probing**: A key-value pair is inserted based on the hash function, resolving collisions through sequential probing.
* **Insert via Quadratic Probing**: Collisions are resolved by incrementing the index quadratically.
* **Search**: Locates the key-value pair by following the probing method used for insertion.

**Theory:**

* **Algorithm**: The hash function maps keys to indices in a hash table. If a collision occurs, a probing technique is used to find the next available slot.
  + **Linear Probing**: Increment index by 1 until an empty slot is found.
  + **Quadratic Probing**: Use index=(hash+i2)%sizeindex = (hash + i^2) \% sizeindex=(hash+i2)%size, where iii starts at 0 and increases.
* **Time Complexity**:
  + Best Case: O(1)O(1)O(1)
  + Worst Case: O(n)O(n)O(n) (in case of full table or clustering)

**2. Set Implementation**

**Output Explanation:**

* **Add/Delete/Search**: Modifies or retrieves values in a set, ensuring no duplicates.
* **Operations**: Perform union, intersection, difference, and subset check on two sets.

**Theory:**

* **Algorithm**:
  + **Union**: Combine all elements of two sets without duplicates.
  + **Intersection**: Include only common elements.
  + **Difference**: Include elements in one set but not the other.
* **Data Structure**: List representation.
* **Complexity**:
  + Search: O(n)O(n)O(n)
  + Add/Delete: O(n)O(n)O(n) for duplicate check.

**3. General Tree**

**Output Explanation:**

* Constructs a tree structure (Book -> Chapters -> Sections).
* Displays the hierarchy in a structured format.

**Theory:**

* **Algorithm**:
  + Recursive creation and traversal of nodes.
* **Time Complexity**:
  + Creation: O(n)O(n)O(n) (where nnn is the total number of nodes).
  + Display: O(n)O(n)O(n).

**4. Binary Search Tree (BST)**

**Output Explanation:**

* Constructs a BST based on input data.
* Provides operations: search, longest path, mirror tree, and traversal.

**Theory:**

* **Algorithm**:
  + **Insert**: Place nodes based on comparison with parent nodes.
  + **Search**: Traverse left/right based on the key.
  + **Longest Path**: Calculate the depth recursively.
  + **Mirror**: Swap left and right children recursively.
* **Time Complexity**:
  + Insert/Search: O(log⁡n)O(\log n)O(logn) (average), O(n)O(n)O(n) (worst-case).
  + Longest Path: O(n)O(n)O(n).

**5. Expression Tree**

**Output Explanation:**

* Constructs an expression tree from a prefix expression.
* Traverses the tree in post-order non-recursively and deletes it.

**Theory:**

* **Algorithm**:
  + Parse the prefix expression.
  + Use a stack to build the tree.
  + Traverse in post-order (Left-Right-Root).
* **Complexity**:
  + Tree Construction: O(n)O(n)O(n).
  + Traversal: O(n)O(n)O(n).

**6. Graph Implementation**

**Output Explanation:**

* **Adjacency Matrix**: Represents the graph.
* **BFS and DFS**:
  + BFS explores level by level.
  + DFS explores depth first.

**Theory:**

* **Algorithm**:
  + BFS uses a queue.
  + DFS uses a stack or recursion.
* **Complexity**:
  + BFS/DFS: O(V+E)O(V + E)O(V+E) where VVV is vertices and EEE is edges.

**7. Minimum Spanning Tree (Prim’s Algorithm)**

**Output Explanation:**

* Constructs a minimum spanning tree by choosing the least-cost edge iteratively.

**Theory:**

* **Algorithm**:
  + Start with an arbitrary node.
  + Repeatedly add the smallest edge connecting the tree to an unvisited node.
* **Complexity**:
  + O(V2)O(V^2)O(V2) with adjacency matrix.
  + O(Elog⁡V)O(E \log V)O(ElogV) with priority queue.

**8. Priority Queue**

**Output Explanation:**

* Inserts and removes elements based on priority.
* Serious cases are served first, followed by non-serious and checkups.

**Theory:**

* **Algorithm**:
  + Enqueue: Insert elements in sorted order based on priority.
  + Dequeue: Remove the highest-priority element.
* **Complexity**:
  + Enqueue: O(n)O(n)O(n).
  + Dequeue: O(1)O(1)O(1).

**9. Sequential File Organization**

**Output Explanation:**

* Adds, deletes, and displays student records sequentially from a file.

**Theory:**

* **Algorithm**:
  + Append for addition.
  + Sequential search for retrieval and deletion.
* **Complexity**:
  + Search/Delete: O(n)O(n)O(n).

**10. Optimal Binary Search Tree (OBST)**

**Output Explanation:**

* Constructs an OBST based on given probabilities to minimize search cost.

**Theory:**

* **Algorithm**:
  + Dynamic Programming: Compute weight, cost, and root matrices.
  + Build tree using these matrices.
* **Complexity**:
  + O(n3)O(n^3)O(n3) for matrix computation.

Let me know if you need further clarifications on any of these concepts!

4o

**You said:**

No its good ahead with all other

**ChatGPT said:**

ChatGPT

Sure, let’s continue with the theoretical explanations and outputs for the remaining algorithms and programs from the file.

**11. BFS and DFS in Graph**

**Output Explanation:**

1. **Adjacency Matrix**: The program creates and displays the adjacency matrix for the graph.
2. **BFS (Breadth-First Search)**: Starts from a given vertex and explores neighbors level by level.
3. **DFS (Depth-First Search)**: Starts from a given vertex and explores as far as possible along a branch before backtracking.

**Theory:**

* **Breadth-First Search**:
  + **Algorithm**:
    1. Use a queue to manage nodes for exploration.
    2. Mark the starting node as visited and enqueue it.
    3. Dequeue a node, explore all unvisited neighbors, and mark them visited.
  + **Complexity**: O(V+E)O(V + E)O(V+E), where VVV is vertices and EEE is edges.
* **Depth-First Search**:
  + **Algorithm**:
    1. Use a stack or recursion to explore a vertex.
    2. For each unvisited neighbor, visit recursively.
  + **Complexity**: O(V+E)O(V + E)O(V+E).

**12. Minimum Spanning Tree (Prim’s Algorithm)**

**Output Explanation:**

* The program calculates the minimum spanning tree by connecting nodes with minimum cost edges while avoiding cycles.
* Displays the edges chosen and their respective costs.

**Theory:**

* **Algorithm**:
  1. Start from any vertex and include it in the MST.
  2. Choose the smallest edge that connects a new vertex to the tree.
  3. Repeat until all vertices are included.
* **Complexity**:
  1. O(V2)O(V^2)O(V2) with adjacency matrix.
  2. O(Elog⁡V)O(E \log V)O(ElogV) with a priority queue.

**13. Expression Tree from Prefix Expression**

**Output Explanation:**

* Constructs a tree from a prefix expression.
* Traverses it post-order and deletes nodes in the tree.

**Theory:**

* **Algorithm**:
  1. Parse the prefix expression from right to left.
  2. Push operands onto a stack.
  3. On encountering an operator, pop two operands, form a subtree, and push back the root node.
* **Complexity**: O(n)O(n)O(n) for tree construction and traversal.

**14. Priority Queue for Hospital Services**

**Output Explanation:**

* Implements a priority queue for hospital management.
  + **Insert**: Patients are added based on priority.
  + **Display**: Patients are shown in order of priority.
  + **Delete**: Removes the highest-priority patient.

**Theory:**

* **Algorithm**:
  + Insert elements in sorted order based on priority.
  + Remove the element with the highest priority.
* **Complexity**:
  + Insert: O(n)O(n)O(n) for shifting elements.
  + Delete: O(1)O(1)O(1) for removing the front element.

**15. Sequential File Organization**

**Output Explanation:**

* Manages student records with operations like adding, deleting, and searching records.
* Displays all records sequentially.

**Theory:**

* **Algorithm**:
  + Append new records for addition.
  + Sequential search for retrieval and deletion.
* **Complexity**:
  + Search/Delete: O(n)O(n)O(n).

**16. Optimal Binary Search Tree (OBST)**

**Output Explanation:**

* Constructs an OBST based on given probabilities of access.
* Displays the cost matrix, root matrix, and the final OBST structure.

**Theory:**

* **Algorithm**:
  1. Use dynamic programming to calculate the minimum cost for subtrees.
  2. Determine the optimal root for each subtree.
  3. Build the tree using the root matrix.
* **Complexity**:
  1. O(n3)O(n^3)O(n3) for constructing the cost matrix.
  2. O(n2)O(n^2)O(n2) for building the tree.

These explanations should cover the theoretical and output details for all the programs in the file. Let me know if there’s any specific topic or code you’d like to revisit!

4o

**You said:**

Now give me examples of output and input of these examples in sequence

**ChatGPT said:**

ChatGPT

Here are examples of input and output for each code in sequence:

**1. Hash Table Implementation**

**Input:**

mathematica

Copy code

Enter size of hash table: 5

1. Linear Probe

2. Quadratic Probe

Enter Choice: 1

\*\* 1.Insert \*\*

Enter phone number: 12345

Enter name: Alice

\*\* 1.Insert \*\*

Enter phone number: 67890

Enter name: Bob

**Output:**

css

Copy code

Data inserted at 0

[[12345, 'Alice'], None, None, None, None]

Data inserted at 1

[[12345, 'Alice'], [67890, 'Bob'], None, None, None]

**2. Set Implementation**

**Input:**

vbnet

Copy code

--- MAIN MENU ---

1 -> Add value to set

Choose an option (1-10): 1

Which set to operate on? 1

Total values to add: 3

Value to add: 5

Value to add: 10

Value to add: 15

**Output:**

python

Copy code

Set is: [5]

Set is: [5, 10]

Set is: [5, 10, 15]

**3. General Tree**

**Input:**

mathematica

Copy code

Enter name of book: Data Structures

Enter number of chapters in book: 2

Enter the name of Chapter 1: Arrays

Enter number of sections in Chapter Arrays: 2

Enter Name of Section 1: Basics

Enter Name of Section 2: Applications

Enter the name of Chapter 2: Linked List

Enter number of sections in Chapter Linked List: 1

Enter Name of Section 1: Implementation

**Output:**

yaml

Copy code

-----Book Hierarchy---

Book title: Data Structures

Chapter 1: Arrays

Sections:

Basics

Applications

Chapter 2: Linked List

Sections:

Implementation

**4. Binary Search Tree**

**Input:**

kotlin

Copy code

Enter the data for the node: 50

Do you want to continue? (y|n): y

Enter the data for the node: 30

Do you want to continue? (y|n): y

Enter the data for the node: 70

Do you want to continue? (y|n): n

**Output:**

mathematica

Copy code

Preorder Tree: 50 30 70

Inorder Tree: 30 50 70

Postorder Tree: 30 70 50

**5. Expression Tree**

**Input:**

mathematica

Copy code

Enter prefix Expression: +a\*bc

**Output:**

yaml

Copy code

Postorder Traversal: a b c \* +

Deleting node: +

Deleting node: \*

Deleting node: a

Deleting node: b

Deleting node: c

**6. Graph BFS and DFS**

**Input:**

yaml

Copy code

Enter number of vertices: 4

Enter number of edges: 3

Edges:

0 1

0 2

1 3

Enter initial vertex: 0

**Output:**

yaml

Copy code

The adjacency matrix of the graph is:

0 1 1 0

1 0 0 1

1 0 0 0

0 1 0 0

The BFS of the Graph:

0 1 2 3

The DFS of the Graph:

0 2 1 3

**7. Minimum Spanning Tree (Prim’s Algorithm)**

**Input:**

yaml

Copy code

Enter the number of branches: 4

Enter the number of connections: 4

Connections:

1 2 4

1 3 3

2 3 2

3 4 6

**Output:**

csharp

Copy code

Minimum cost connection is 1 -> 3 with charge: 3

Minimum cost connection is 2 -> 3 with charge: 2

Minimum cost connection is 3 -> 4 with charge: 6

The minimum total cost of connections: 11

**8. Priority Queue**

**Input:**

mathematica

Copy code

Enter patient name: John

Enter priority of patient (1 - Serious, 2 - Non-serious, 3 - General checkup): 1

Enter patient name: Jane

Enter priority of patient (1 - Serious, 2 - Non-serious, 3 - General checkup): 2

**Output:**

vbnet

Copy code

Patient's Name: John Priority: Serious

Patient's Name: Jane Priority: Non-serious

**9. Sequential File Organization**

**Input:**

mathematica

Copy code

1. Create

Enter Roll No of Student: 1

Enter Name of Student: Alice

Enter Division of Student: A

Enter Address of Student: NY

Do You Want to Add More Records? (Y/N): N

2. Display

**Output:**

less

Copy code

The Content of File:

Roll Name Div Address

1 Alice A NY

**10. Optimal Binary Search Tree**

**Input:**

mathematica

Copy code

Enter a number of identifiers: 3

Enter identifiers: a b c

Enter success probability for identifiers: 0.2 0.5 0.3

Enter failure probability for identifiers: 0.1 0.1 0.1 0.1

**Output:**

kotlin

Copy code

Weight Cost Root

0.1 0 0

0.3 0.3 1

0.6 1.1 2

The final OBST is:

b

a

c