Understanding Input and Output in Data Structures and Algorithms

A.TULASI **22BDS004** **B.RAGHU VAMSI** **22BDS014** **RAJ KUMAR** **22BDS055** **CH.GANDHI RAJ** **22BDS017**

ABSTRACT:

This report analyses the input and output of various data structures, including Linked Lists, Stacks, Queues, Trees, and searching algorithms like BFS, DFS, and Prim's, Kruskal's, Dijkstra's, and Bellman-Ford algorithms. It examines their efficiency, applicability, and performance in solving different problems, providing insights into their strengths and limitations for different use cases.

PRIM'S ALGORITHM:

Prim's algorithm is a greedy algorithm to find the Minimum Spanning Tree in a connected, weighted graph. It starts with a vertex and adds the minimum-weight edges until all vertices are included.

```
INPUT: Enter the number of vertices in the graph:4
Enter the adjacency matrix for the graph:
0 2 3 0
2 0 0 0
3 0 0 4
0 0 4 0
OUTPUT:
Edge    Weight
0 - 1     2
0 - 2     3
2 - 3     4
```

EXPLANATION: The program successfully applied Prim's algorithm to find the Minimum Spanning Tree (MST) of the input graph. It printed the edges and their weights(2 2 4) that form the MST, demonstrating the algorithm.

KRUSKAL'S ALGORITHM:

Kruskal's algorithm is a greedy algorithm for finding the Minimum Spanning Tree in a connected, weighted graph by adding minimum-weight edges without forming cycles.

```
INPUT: Enter the number of cities (vertices) in the map: 4
Enter the number of roads (edges) in the map: 4
Enter the roads and their costs (source destination cost):
1 2 2
1 4 4
4 3 3
2 3 6
OUTPUT: Minimum cost to connect all cities in the map: 9
```

EXPLANATION: Once Kruskal's algorithm has been applied, connected components are tracked using the Union-Find data structure, which
iteratively adds edges to the MST if they don't form cycles. Finally, it figures out how much it would

cost to connect each city on the map at the lowest possible cost. Here it is 9.

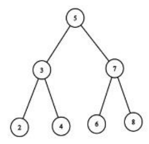
BREADTH-FIRST SEARCH:

BFS is a tree traversal algorithm that explores nodes level by level, using a queue for node expansion. It visits all nodes at the same depth before moving deeper.

```
INPUT : Enter the elements of the tree (enter 0 to exit):
5 3 7 2 4 6 8 0
```

OUTPUT : Elements in the tree using breadth-first traversal (BFS): 5 -> 3 -> 7 -> 2 -> 4 -> 6 -> 8 ->

EXPLANATION: The tree structure formed for the given input would be:



DEPTH-FIRST SEARCH:

DFS an algorithm for traversing or searching tree or graph data structures, exploring as far as possible along each branch before backtracking.

```
INPUT: Enter the number of vertices in the graph:4
Enter the number of edges in the graph: 5
Enter the edges (source destination) of the graph:
0 1
1 2
2 3
3 1
1 0
OUTPUT: The directed graph contains a cycle.
```

EXPLANATION: This C program checks for cycles in a directed graph using Depth-First Search (DFS). It creates a graph, reads the number of vertices and edges, then adds edges. It performs DFS on each vertex, marking visited nodes and using a stack to detect cycles. Finally, it prints whether the graph contains a cycle or not.

STACK SPAN PROBLEM USING STACK:

A stack is a linear data structure that follows the Last In First Out (LIFO) principle. It allows adding and removing elements from one end, known as the top, making it suitable for various applications like recursion, expression evaluation, and backtracking.

```
INPUT: Enter the number of days: 3
Enter the prices for each day:
100
120
140
OUTPUT: Stock Span:
Day 1: 1
Day 2: 2
Day 3: 3
```

EXPLANATION: This an efficient solution to the Stock Span Problem using a stack data structure. It computes the "span" of a stock's price for each day, indicating the number of consecutive previous days (including the current day) with prices less than or equal to the current day's price.

CIRCULAR QUEUE IMPLEMENTATION:

The code implements a Circular Queue in C, allowing insertion and removal of elements from the front and rear. It handles queue fullness and emptiness, and displays the front element. Circular queues efficiently manage data by overwriting old elements when full.

```
INPUT: Enter elements to enqueue (enter -1 to stop):
10
20
30
-1
OUTPUT: Element removed: 10
Enter elements to enqueue (enter -1 to stop):
40
50
-1
Element at front: 20
----------------------
Index : 5 4 3 2  1  0
----------------------
Queue:  20 40 50
```

EXPLANATION: The circular queue's contents are printed in reverse order of their enqueuing, i.e., 20 40 50.

LINKED LIST:

A linked list is a linear data structure where each element (node) contains data and a reference (pointer) to the next node. It provides dynamic memory allocation for flexible size and easy insertion/deletion.

```
INPUT: Enter numbers (enter -1 to stop):
10
20
30
-1
OUTPUT: Sum of the numbers: 60
```

EXPLANATION: The user inputs three numbers (10, 20, 30) to create a linked list. The input is terminated with -1.

BINARY HEAP IMPLEMENTATION:

A binary heap is a binary tree with a special property: the value of each node is either greater than or equal to (max heap) or less than or equal to (min heap) its child nodes.

```
INPUT:
Enter the number of elements to insert into the binary heap:
3
Enter the elements: 10 30 20
OUTPUT:
Binary Heap after insertion: 10 30 20
Removing minimum element: 10
Binary Heap after removal: 20 30
Removing minimum element: 20
Binary Heap after removal: 30
Removing minimum element: 30
Binary Heap after removal:
```

EXPLANATION: The output shows the binary heap's status after each insertion and removal, reflecting the binary heap's structural properties.

DIJKSTRA'S SHORTEST PATH ALGORITHM:

It operates by iteratively selecting the vertex with the minimum distance from the source and updating the distances of its neighbours, until all vertices are visited. The algorithm is greedy and can handle non-negative edge weights.

```
INPUT:
Enter the number of nodes in the graph: 5
Enter the number of edges in the graph: 7
Enter the adjacency matrix of the graph:
0 10 0 5 0
0  0  1 2 0
0  0  0 0 4
0  3  9 0 2
7  0  6 0 0
Enter the source vertex: 0
OUTPUT:
Vertex    Distance        Path
0              0               0
1              8               0 3 1
2              9               0 3 1 2
3              5               0 3
4              7               0 3 4
```

BELLMAN-FORD ALGORITHM:

Bellman-Ford is a single-source shortest-path algorithm that handles negative edge weights. It iterates through all edges, relaxing them by updating distances until the optimal shortest paths are achieved or negative cycles detected.

```
INPUT:
Enter the number of vertices in the graph: 5
Enter the number of edges in the graph: 6
Enter the edges and their weights (source destination weight):
0 1 3
0 2 6
1 3 2
2 3 1
3 4 4
4 2 7
Enter the source vertex: 0
OUTPUT:
Shortest distances from vertex 0 to all other vertices:
Vertex 0: Distance = 0, Parent = -1
Vertex 1: Distance = 3, Parent = 0
Vertex 2: Distance = 6, Parent = 0
Vertex 3: Distance = 5, Parent = 1
Vertex 4: Distance = 9, Parent = 3
```

INPUTS AND OUTPUTS OF CODES

Q1. Create a class of Data structures for Students and storing data in linked lists

Student list:

INPUT:-

Roll Number	**Name**	**Marks**
101	RAGHU	85.50
102	TULASI	78.20
103	RAJ KUMAR	92.00

Student List after deleting record

OUTPUT:-

Roll Number	**Name**	**Marks**
101	RAGHU	85.50
103	RAJ KUMAR	92.00

Q2) A. Concatenate two given linked list into a single linked lists

INPUT:

List 1:-	4	5	6
List 2:-	7	8	9

OUTPUT:

Concatenated List:	4	5	6	7	8	9

Q2) B. Insert an element in a linked list in sorted order. The function will be called for every element to be inserted.

INPUT:

List Given:-	4	6	8	7	2	3

OUTPUT:

Sorted List	2	3	4	6	7	8

Q2) C. Always insert elements at one end, and delete elements from the other end (FIRST-IN FIRST-OUT QUEUE).

INPUT:

Deque after insertion at the front:	6	5	4
Deleted element from the rear:	4		

OUTPUT:

Deque after deletion from the rear:	**6**	**5**