Analysing and Exploring Graphs Data Structure and Searching Algorithms

SREESHANTH
22BDS016

KAMAL DAS
22BDS035

NAVADEEP
22BDS040

SUHAAS
22BDS056

BELLMAN FORD ALGORITHM

The program implements the Bellman-Ford algorithm to find the shortest paths in a graph with negative edge weights. It reads graph details, performs edge relaxation, and prints the minimum cost to reach each vertex from a given source vertex.

INPUT :
Enter the number of vertices: 5
Enter the number of edges: 7
Enter edges (start vertex end vertex weight):
0 1 4
0 2 5
1 2 3
1 3 6
2 3 7
3 4 2
4 2 1
Enter the source vertex: 0

OUTPUT :
Minimum cost to reach each vertex from the source vertex 0:
Vertex 0: 0
Vertex 1: 4
Vertex 2: 5
Vertex 3: 10
Vertex 4: 12

PRIM’S ALGORITHM

The program implements Prim's algorithm to find the Minimum Product Spanning Tree (MPST) of a weighted graph efficiently. It selects vertices with the minimum product values, updates the MPST, and prints the tree's edges and their weights, along with the total product of the edge weights.

INPUT :
Enter the number of vertices in the graph: 5
Enter the adjacency matrix of the graph:
0 2 3 0 6
2 0 4 5 0
3 4 0 1 0
0 5 1 0 5
6 0 0 5 0

OUTPUT:
Product of Minimum Product Spanning Tree:
30.000000

Edge	Weight
0 - 1	2
0 - 2	3
2 - 3	1
3 - 4	5

DIJKSTRA’S ALGORITHM

The program uses Dijkstra's algorithm to find the shortest path from a given starting node to all other nodes in a weighted graph represented by an adjacency matrix. It initialiSes data structures, iteratively selects the node with the smallest distance, and prints the shortest distances and paths to all vertices.

INPUT :
Enter the number of vertices in the graph: 5
Enter the adjacency matrix of the graph:
0 2 3 0 6
2 0 4 5 0
3 4 0 1 0
0 5 1 0 5
6 0 0 5 0

OUTPUT:
Distance of 1 = 2
Path = 1 <- 0
Distance of 2 = 3
Path = 2 <- 0
Distance of 3 = 4
Path = 3 <- 2 <- 0
Distance of 4 = 6
Path = 4 <- 0

QUEUE

The code implements a queue data structure using an array. It defines functions to create, enqueue, dequeue, check if the queue is empty or full, and display its contents. The program presents a menu to interactively perform these operations and deallocates memory properly upon exit. So know lets see what output we get in the code

INPUT :
Queue Operations:
1. Enqueue
2. Dequeue
3. Check if Queue is Empty
4. Display Queue
5. Exit

OUTPUT:
The program outputs messages to inform the user about the result of their chosen operation. The output is displayed after each operation is performed.

LINKED LIST

The program is to check whether a given linked list is a palindrome or not. A palindrome linked list is one where the elements read the same forwards and backwards.

INPUT :
Enter the number of elements in the linked list: 5
Enter the elements: 1
2
3
2
1

OUTPUT:
Linked List: 1 -> 2 -> 3 -> 2 -> 1 -> NULL
The linked list is a palindrome

TREE TRAVERSAL

The provided C code defines a binary tree data structure with functions to create nodes and perform in-order, pre-order, and post-order traversals. It creates a binary tree, displays the tree elements in in-order, pre-order, and post-order sequences.

INPUT :
as mentioned in the code
10
8
16
1
9
13
5

OUTPUT:
In-order Traversal: 1 8 9 10 13 15 16
pre-order Traversal: 10 8 1 9 16 13 15
post-order Traversal: 1 9 8 15 13 16 10

STACKS

This C code implements two stacks to find the maximum element in a stack efficiently. It uses an auxiliary stack to keep track of the maximum elements as new elements are pushed onto the main stack. The program demonstrates finding the maximum element in the stack after performing push operations.

INPUT :
Set :1
5, 10, 15, 20.

Set :2
25, 18, 30.

OUTPUT :

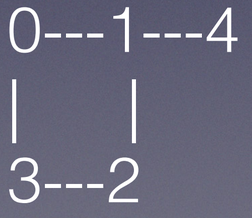
Maximum element in the first set of the stack: 20

Maximum element in the second set of the stack: 30

BREADTH-FIRST SEARCH

This BFS code implements a graph data structure using an adjacency matrix for an undirected graph. It detects cycles using BFS traversal, checking for back edges. It prompts the user for graph details, performs cycle detection, and outputs whether a cycle exists or not.

INPUT :
Enter the number of vertices: 5
Enter the number of edges: 6
Edges (source destination weight):
0 1
1 2
2 3
3 4
4 0
1 4



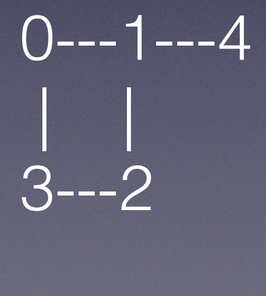
OUTPUT :
The graph contains a cycle

DEPTH-FIRST SEARCH

This DFS code implements a graph data structure using an adjacency matrix for an undirected graph. It detects cycles using DFS traversal, checking for back edges. Users input graph details, and the program outputs cycle presence.

INPUT :

Enter the number of vertices: 5
Enter the number of edges: 6
Edges (source destination weight):
0 1
1 2
2 3
3 4
4 0
1 4



OUTPUT :

The graph contains a cycle

KRUSKAL’S ALGORITHM

This code uses Kruskal's algorithm to find the Minimum Spanning Tree (MST) of a weighted undirected graph. It efficiently constructs the MST by sorting edges based on weights and unioning sets of connected vertices. The final MST is printed with total weight. Suitable for finding the minimum weight spanning tree among all possible trees.

INPUT :
Enter the number of vertices: 6
Enter the number of edges: 9
Edges (source destination weight):
0 1 5.
0 2 3
0 3 1
1 4 4
1 5 2
2 4 6
2 5 7
3 4 8
4 5 9

OUTPUT :
Edges in the Minimum Sum Spanning Tree:
0 - 3 with weight 1
1 - 5 with weight 2
0 - 2 with weight 3
0 - 1 with weight 5
1 - 4 with weight 4
Sum of Minimum Sum Spanning Tree: 15