Analysing and Exploring Graphs Data Structure and Searching Algorithms

SREESHANTH	KAMAL DAS	NAVADEEP	SUHAAS
22BDS016	22BDS035	22BDS040	22BDS056

Abstract : The report showcases efficient C programs with mastery of data structures and algorithmic techniques. They demonstrate ingenuity, problem-solving, advanced graph traversal, and optimisation strategies, including Bellman-Ford and Prim's MST algorithms.

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 :

- The program starts by asking the user to provide the number of elements in the linked list Following this, the user is asked to enter n integers as the elements of the linked list.
- Enter the number of elements in the linked list : 5
- Enter the elements : 1 2 3 2 1

OUTPUT :

- Once the linked list is constructed and displayed, the program determines whether it forms a palindrome or not. If the linked list forms a palindrome, the program will print "The linked list is a palindrome." However, when the linked list is not a palindrome, it will display "The linked list is not a palindrome."
- Linked List : 1 -> 2 -> 3 -> 2 -> 1 -> NULL
  The linked list is a palindrome.

STACKS

The program is to find the maximum element in a stack. It uses two stacks: a main stack and an auxiliary stack.

INPUT :

- The program starts by creating two stack instances: **mainStack** and **auxStack**, using the **createStack()** function. The **mainStack** is utilised to hold the elements, while the **auxStack** is not used
- Set 1: 5, 10, 15, 20
  Set 2: 25, 18, 30

OUTPUT :

- Maximum element in the stack: 20
  Maximum element in the stack: 30

QUEUE

The provided C program is an implementation of a queue data structure with operations such as enqueue, dequeue, check if the queue is empty, and display the elements in the queue.

INPUT :

The program starts by creating an empty queue using the **createQueue** function, which allocates memory for the queue structure. The user is then presented with a menu of queue operations:

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

OUTPUT :

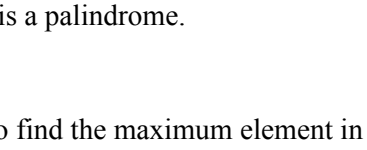
The program outputs messages to inform the user about the result of their chosen operation. The output is displayed after each operation is performed.

BREADTH-FIRST SEARCH (BFS)

The provided C program is designed to detect cycles in an undirected graph using Breadth-First Search (BFS).The program then checks if the graph contains any cycles using BFS.

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



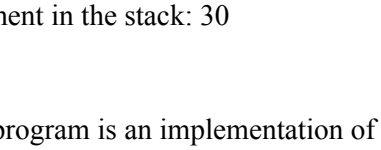
OUTPUT : The graph contains a cycle.

DEPTH-FIRST SEARCH (DFS)

The provided C program is designed to detect cycles in an undirected graph using Depth-First Search (DFS).The program then checks if the graph contains any cycles using DFS.

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

The provided C program implements Kruskal's algorithm to find the Minimum Spanning Tree (MST) of a given graph.

INPUT :

- The user is prompted to input the number of vertices and edges in the graph, and then they can input the edges, including their source, destination, and weight. The program then calculates the Minimum Spanning Tree and outputs the edges included in the MST along with their weights and the total weight of the MST
- 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

DIJKSTRA's ALGORITHM

The provided C program implements Dijkstra's algorithm to find the shortest paths from a given starting node to all other nodes in a graph represented by an adjacency matrix.

INPUT :

Enter the number of vertices: 5
Enter the adjacency matrix:
0 3 0 0 7
3 0 1 0 0
0 1 0 2 0
0 0 2 0 5
7 0 0 5 0
Enter the starting node: 0

OUTPUT :

- Distance of 1 = 3
  Path = 1 <- 0
- Distance of 2 = 4
  Path = 2 <- 1 <- 0
- Distance of 3 = 6
  Path = 3 <- 2 <- 1 <- 0
- Distance of 4 = 7
  Path = 4 <- 0

TREES

The provided C code demonstrates the implementation of a binary tree data structure along with three types of tree traversals: in-order, pre-order, and post-order traversals.

INPUT :

```
     10
   /   \
  7     15
 / \   / \
5   8 12  18
```

OUTPUT :

In-order Traversal: 5 7 8 10 12 15 18
pre-order Traversal: 10 7 5 8 15 12 18
post-order Traversal: 5 8 7 12 18 15 10

BELLMAN FORD

The provided C code implements Bellman-Ford algorithm to find minimum cost to reach each vertex from given source vertex in weighted graph.

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 3: 10
Vertex 1: 4	Vertex 4: 12
Vertex 2: 5	

PRIM's ALGORITHM

The provided C code implements Prim's algorithm to find the Minimum Product Spanning Tree (MPST) of a weighted graph.

INPUT :

Enter the number of vertices in the graph: 5
Enter the adjacency matrix of the graph:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0

OUTPUT :

Product of Minimum Product Spanning Tree: 180

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	5
1 - 4	6

ASSIGNMENT (LAB 11)

QUESTION NO:	INPUT	OUTPUT
1)	1) Suhaas Marks: 89.10 2) Sreeshanth Marks: 77.50 3) Navdeep Marks: 96.00	After deletion 1) Suhaas Marks: 89.10 2) Navdeep Marks: 96.00
2a)	First linked list: 11 ,22 ,33 Second linked list:10 ,50 ,60	Concatenated Linked List: 11 ,22, 33, 10, 50, 60
2b)	Elements: {23, 45, 10, 20, 35}	Linked List after inserting elements in sorted order: {10 ,20, 23, 35, 45}
2c)	Elements: {7, 8, 10, 14, 17}	Deque after insertion at the front: {17, 14, 10, 8, 7}. Deleted element from the rear: {7}. Deque after deletion from the rear: {17, 14, 10, 8}.