EXPERIMENT 1

Objective: Implementation of Linked Representation of Graph.

Brief Theory:

The adjacency matrix, though commonly used, can be inefficient due to its rigid structure and space requirements, particularly for sparse graphs or dynamically changing graphs. A more flexible approach is the **linked representation**, which uses a combination of nodes and lists:

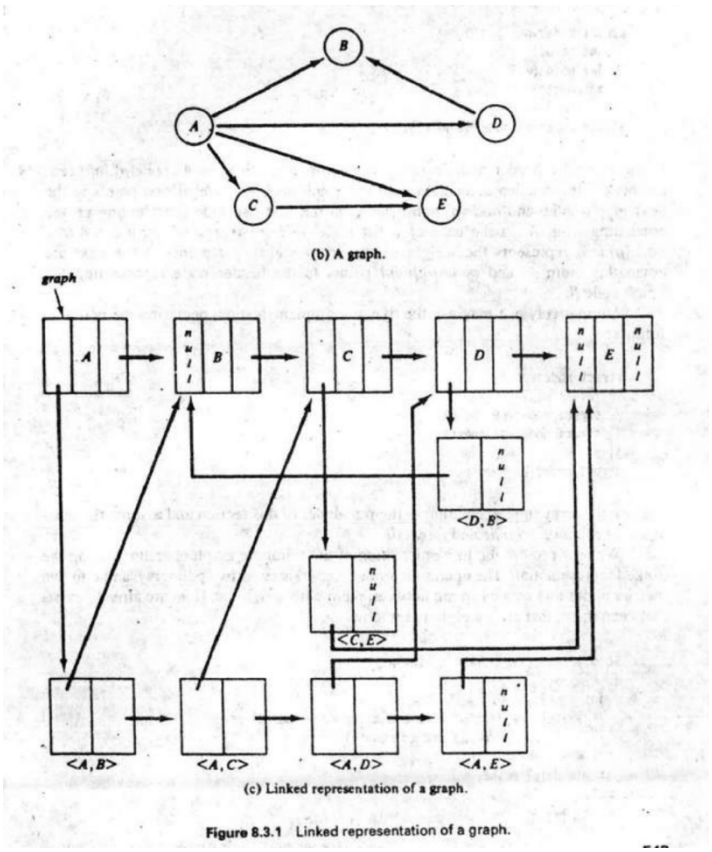

Figure 8.3.1 Linked representation of a graph.

1. **Graph Nodes**: Represented as header nodes containing:
   - info: Data associated with the graph node.
   - nextnode: Pointer to the next graph node in the linked list.
   - arcptr: Pointer to the adjacency list (list of arcs) of the graph node.
2. **Adjacency Lists**: Represent arcs emanating from a node, stored as a separate linked list. Each node in this list (called an **arc node**) contains:
   - ndptr: Pointer to the destination graph node.
   - nextarc: Pointer to the next arc in the adjacency list.

This structure allows dynamic memory allocation, making it suitable for graphs with frequently changing nodes and edges. It is efficient in terms of both space and flexibility compared to adjacency matrices, especially for sparse graphs.

Tasks:
1) Implement a graph using linked representation and display the adjacency list.
2) Perform BFS and DFS traversal on the graph.
3) Add functionality to dynamically add/delete nodes and edges in the graph.

Apparatus and components required: Computer with C or C++ Compiler and Linux/Windows platform.

Experimental/numerical procedure: Coding, compilation, editing, run and debugging.