Graphs :

Lightbox

The following two are the most commonly used representations of a graph.

1. Adjacency Matrix

Lightbox

Algorithm :

1. Take input as number of vertice ;
2. Initializa 2d matrix;
3. Initially all values should be 0
4. Take two inputs as node1 and node2
5. Put adj\_matrix[node1][node2] = adj\_matrix[node1][node2] = 1;
6. Print 2d matrix

Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time.

Cons: Consumes more space O(V^2).

2. Adjacency List

An array of lists is used. The size of the array is equal to the number of vertices.

Lightbox

Algorithm :

1. Take input as number of vertices make it global;
2. Struct ionization of node
3. struct node \*adj[no\_vertices];
4. Put all adj[i] = NULL;
5. Call readgraph();
6. Call printgraph();

Readgraph();

1. For loop for number of vertices
2. Take input for no of vertices as enter number of neighbors
3. using for loop
4. initailze last\* = NULL
5. take input of all neighbors
6. creade newnode
7. Check if its first element or not using adj[i] = NULL if yes then adj[i] = newnode;
8. Else last->next = newnode;
9. Last = newnode

Printgraph()

1. Temp\* = NULL;
2. Using for loop
3. Temp = adj[i];
4. Print i(I is nothing but the element for which we are printing adjuest
5. While(temp 1= NULL)
6. Print temp->data
7. Temp -= temp->next

Breadth First Search : (Queue)

Time Complexity: O(V+E), where V is the number of nodes and E is the number of edges.

Auxiliary Space: O(V)

Enque all adjenst elemnt to the queue

Algorithm :

1. Strcut queue with size , front , rear , arr
2. Create isempy() isfull() enque() deque() function
3. Main function
4. Initialize q , size , front , rear and arr
5. Create visted\_array with initial value is 0
6. Int node =0 /// root element visted first
7. Print root element
8. And visted[node] = 1 and a[7][7]
9. Enque(node)
10. While(q is not empty )
11. Node = deque
12. For(int i<7)
13. if(a[node][i]==1 && visted[i]==0)
14. Print node
15. Visted = 1
16. Enque(i)

Deapth First Search : (Stack)

Push only one adjust elemnt to the stack

1. Push root
2. Print root
3. Push only one adjent to stack
4. All dfs for this push their one adj to the stack
5. Print pushed element
6. If all adjent is visted
7. Pop and check if any adj unvisted node is there

Algorithm :

1. Global matix and visted array
2. Dfs(i)
3. Print i
4. For(j<7)
5. if(A[i][j]==1 && visted[j]==0) dfs(j);
6. Dfs(j)
7. Int main function call dfs

The time complexity of the DFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges.

The space complexity of the algorithm is O(V).

