Chapter 06

Graph

01 Basic concept of graph theory:

A graph is an abstract data structure that is used to implement the mathematical concept of graphs. It is basically a collection of vertices (also called nodes) and edges that connect these vertices. For example, the following information can be represented by graphs:

- **Family trees** in which the member nodes have an edge from parent to each of their children.
- **Transportation networks** in which nodes are airports, intersections, ports, etc. The edges can be airline flights, one-way roads, shipping routes, etc.

Definition:

A graph G is defined as an ordered set (V, E), where V(G) represents the set of vertices and E(G) represents the edges that connect these vertices.

A graph can be directed or undirected. In an undirected graph, edges do not have any direction associated with them.

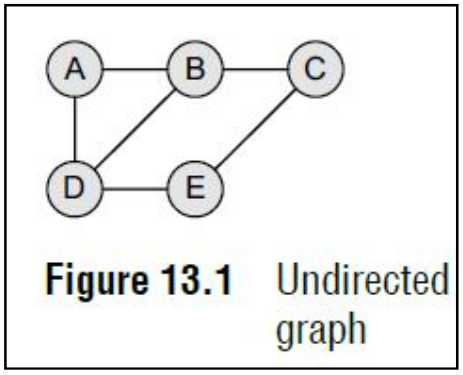


Figure 13.1 Undirected graph

Figure 13.1 shows a graph with V(G) = {A, B, C, D and E} and E(G) = {(A, B), (B, C), (A, D), (B, D), (D, E), (C, E)}.

Note that there are five vertices or nodes and six edges in the graph. if an edge is drawn between nodes A and B, then the nodes can be traversed from A to B as well as from B to A. Figure 13.1 shows an undirected graph because it does not give any information about the direction of the edges.

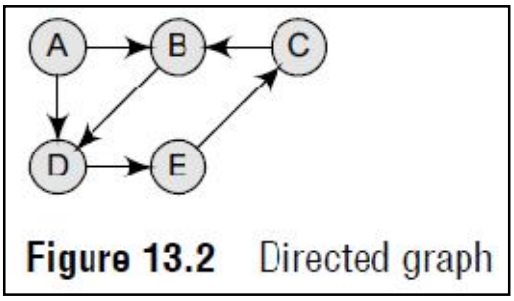
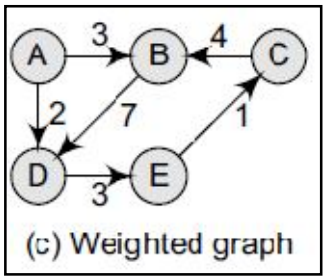


Figure 13.2 Directed graph

Look at Fig. 13.2 which shows a directed graph. In a directed graph, edges form an ordered pair. If there is an edge from A to B, then there is a path from A to B but not from B to A. The edge (A, B) is said to initiate from node A (also known as initial node) and terminate at node B (terminal node).

Graph Terminology:

- **Adjacent nodes or neighbours** For every edge, e = (u, v) that connects nodes u and v, the nodes u and v are the end-points and are said to be the adjacent nodes or neighbours.
- **Degree of a node** Degree of a node u, deg(u), is the total number of edges containing the node u. If deg(u) = 0, it means that u does not belong to any edge and such a node is known as an isolated node.
- **Path** A path P written as P = {v0, v1, v2, ..., vn), of length n from a node u to v is defined as a sequence of (n+1) nodes. Here, u = v0, v = vn and vi–1 is adjacent to vi for i = 1, 2, 3, ..., n.
- **Cycle** A path in which the first and the last vertices are same.
- **Labelled graph or weighted graph** A graph is said to be labelled if every edge in the graph is assigned some data. In a weighted graph, the edges of the graph are assigned some weight or length. The weight of an edge denoted by w(e) is a positive value which indicates the cost of traversing the edge.



(c) Weighted graph

- **Size of a graph** The size of a graph is the total number of edges in it.

02 Storage representations:

There are two common ways of storing graphs in the computer's memory. They are:

- Sequential representation by using an adjacency matrix.
- Linked representation by using an adjacency list that stores the neighbours of a node using a linked list.

1 Adjacency Matrix Representation:

An adjacency matrix is used to represent which nodes are adjacent to one another. By definition, two nodes are said to be adjacent if there is an edge connecting them.

In a directed graph G, if node v is adjacent to node u, then there is definitely an edge from u to v. That is, if v is adjacent to u, we can get from u to v by traversing one edge. For any graph G having n nodes, the adjacency matrix will have the dimension of n X n.

$$a_{ij} \begin{cases} 1 & [\text{if } v_i \text{ is adjacent to } v_j, \text{ that is there is an edge } (v_i, v_j)]A \\ 0 & [\text{otherwise}] \end{cases}$$

In an adjacency matrix, the rows and columns are labelled by graph vertices. An entry aij in the adjacency matrix will contain 1, if vertices vi and vj are adjacent to each other. However, if the nodes are not adjacent, aij will be set to zero.

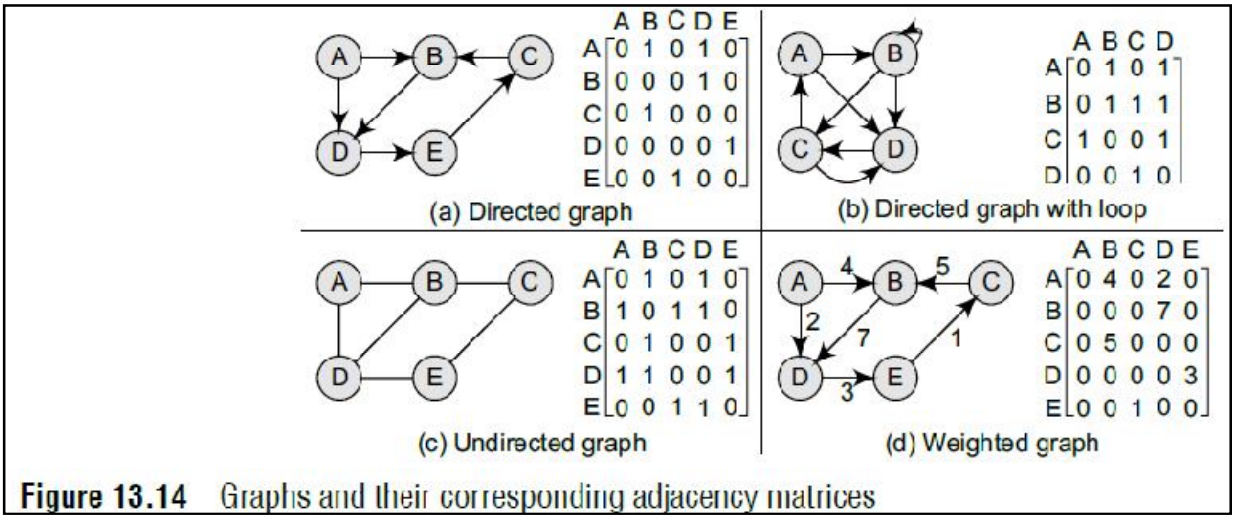


Figure 13.14 Graphs and their corresponding adjacency matrices

2 Adjacency List Representation

An adjacency list is another way in which graphs can be represented in the computer's memory. This structure consists of a list of all nodes in G. Furthermore, every node is in turn linked to its own list that contains the names of all other nodes that are adjacent to it.

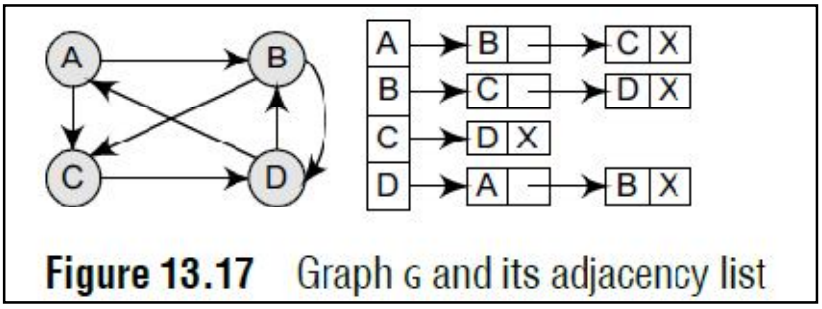


Figure 13.17 Graph G and its adjacency list

Consider the graph given in Fig. 13.17 and see how its adjacency list is stored in the memory.

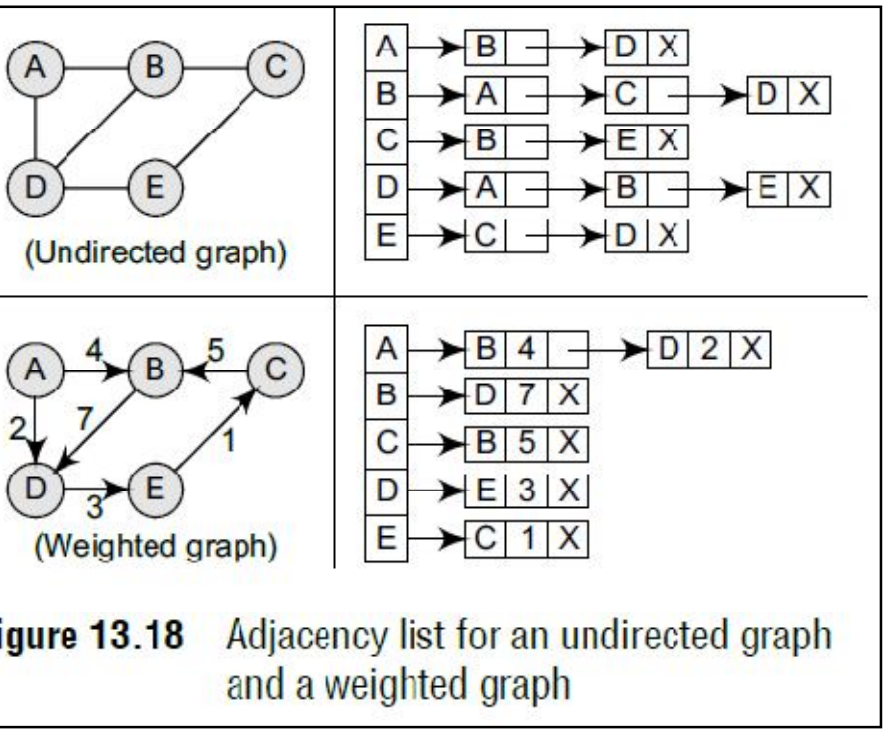


Figure 13.18 Adjacency list for an undirected graph and a weighted graph

For a directed graph, the sum of the lengths of all adjacency lists is equal to the number of edges in G. However, for an undirected graph, the sum of the lengths of all adjacency lists is equal to twice the number of edges in G because an edge (u, v) means an edge from node u to v as well as an edge from v to u. Adjacency lists can also be modified to store weighted graphs. Let us now see an adjacency list for an undirected graph as well as a weighted graph. This is shown in Fig. 13.18.

03 Graph traversal techniques:

By traversing a graph, we mean the method of examining the nodes and edges of the graph. There are two standard methods of graph traversal which are:

1. Breadth-first search

2. Depth-first search

While breadth-first search uses a queue as an auxiliary data structure to store nodes for further processing, the depth-first search scheme uses a stack. But both these algorithms make use of a variable STATUS. During the execution of the algorithm, every node in the graph will have the variable STATUS set to 1 or 2, depending on its current state. Table 13.1 shows the value of STATUS and its significance.

Table 13.1 Value of status and its significance

Status	State of the node	Description
1	Ready	The initial state of the node N
2	Waiting	Node N is placed on the queue or stack and waiting to be processed
3	Processed	Node N has been completely processed

13.6.1 Breadth-First Search Algorithm:

Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighbouring nodes. Then for each of those nearest nodes, the algorithm explores their unexplored neighbour nodes, and so on, until it finds the goal.

That is, we start examining the node A and then all the neighbours of A are examined. In the next step, we examine the neighbours of neighbours of A, so on and so forth. This means that we need to track the neighbours of the node and guarantee that every node in the graph is processed and no node is processed more than once. This is accomplished by using a queue that will hold the nodes that are waiting for further processing and a variable STATUS to represent the current state of the node.

Algorithm: BFS
Input: source & destination node
Output: Node visited list

```
Step 1: SET STATUS = 1 (ready state)
        for each node in G
Step 2: Enqueue the starting node A
        and set its STATUS = 2
        (waiting state)
Step 3: Repeat Steps 4 and 5 until
        QUEUE is empty
Step 4: Dequeue a node N. Process it
        and set its STATUS = 3
        (processed state).
Step 5: Enqueue all the neighbours of
        N that are in the ready state
        (whose STATUS = 1) and set
        their STATUS = 2
        (waiting state)
        [END OF LOOP]
Step 6: EXIT
```

Example:

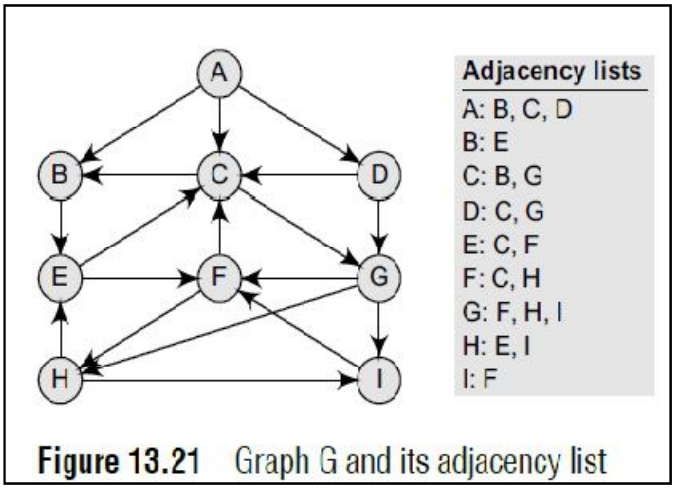


Figure 13.21 Graph G and its adjacency list

[Initial State]

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Not Defined
B	Not Defined
C	Not Defined
D	Not Defined
E	Not Defined
F	Not Defined
G	Not Defined
H	Not Defined
I	Not Defined

BFS Result:

Queue:

[-1]

Empty

1: Step: All Node are ready state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Ready
B	Ready
C	Ready
D	Ready
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

BFS Result:

Queue:

[-1]

Empty

2: Step: Enqueue the starting node A and set its STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Waiting
B	Ready
C	Ready
D	Ready
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

BFS Result:

Queue:

[-1]	**[0]**
	A

3: Step: Dequeue a node A Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Ready
C	Ready
D	Ready
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

BFS Result:

A

Queue:

[-1]	**[0]**

4: Step: Enqueue all the neighbours of A whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Waiting
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

BFS Result:

A

Queue:

[-1]	**[0]**	[1]	[2]
	B	C	D

5: Step: Dequeue a node B Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Waiting
D	Waiting
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

BFS Result:

A
B

Queue:

[-1]	**[0]**	[1]	[2]
		C	D

7: Step: Enqueue all the neighbours of B whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Waiting
D	Waiting
E	Waiting
F	Ready
G	Ready
H	Ready
I	Ready

BFS Result:

A
B

Queue:

[-1]	**[0]**	[1]	[2]	[3]
		C	D	E

8: Step: Dequeue a node C Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Waiting
E	Waiting
F	Ready
G	Ready
H	Ready
I	Ready

BFS Result:

A
B
C

Queue:

[-1]	**[0]**	[1]	[2]	[3]
			D	E

9: Step: Enqueue all the neighbours of C whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Waiting
E	Waiting
F	Ready
G	Waiting
H	Ready
I	Ready

BFS Result:

A
B
C

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]
			D	E	G

10: Step: Dequeue a node D Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Waiting
F	Ready
G	Waiting
H	Ready
I	Ready

BFS Result:

A
B
C
D

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]
				E	G

11: Step: Enqueue all the neighbours of D whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Waiting
F	Ready
G	Waiting
H	Ready
I	Ready

BFS Result:

A
B
C
D

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]
				E	G

12: Step: Dequeue a node E Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Ready
G	Waiting
H	Ready
I	Ready

BFS Result:

A
B
C
D
E

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]
					G

13: Step: Enqueue all the neighbours of E whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Waiting
G	Waiting
H	Ready
I	Ready

BFS Result:

A
B
C
D
E

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]	[5]
					G	F

14: Step: Dequeue a node G Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Waiting
G	Processed
H	Ready
I	Ready

BFS Result:

A
B
C
D
E
G

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]	[5]
						F

15: Step: Enqueue all the neighbours of G whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Waiting
G	Processed
H	Waiting
I	Waiting

BFS Result:

A
B
C
D
E
G

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]	[5]	[6]	[7]
						F	H	I

16: Step: Dequeue a node F Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Processed
G	Processed
H	Waiting
I	Waiting

BFS Result:

A
B
C
D
E
G
F

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]	[5]	[6]	[7]
							H	I

17: Step: Enqueue all the neighbours of F whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Processed
G	Processed
H	Waiting
I	Waiting

BFS Result:

A
B
C
D
E
G
F

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]	[5]	[6]	[7]
							H	I

18: Step: Dequeue a node H Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Processed
G	Processed
H	Processed
I	Waiting

BFS Result:

A
B
C
D
E
G
F
H

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]	[5]	[6]	[7]
								I

19: Step: Enqueue all the neighbours of H whoose status ready and set STATUS waiting state

Adjanacency List	
Node	**Neighours Node**
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status	
Node	**Status**
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Processed
G	Processed
H	Processed
I	Waiting

BFS Result:
A
B
C
D
E
G
F
H

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]	[5]	[6]	[7]
								I

20: Step: Dequeue a node I Process it and set its STATUS processed state

Adjanacency List	
Node	**Neighours Node**
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status	
Node	**Status**
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Processed
G	Processed
H	Processed
I	Processed

BFS Result:
A
B
C
D
E
G
F
H
I

Queue:

[-1]	**[0]**	[1]	[2]	[3]	[4]	[5]	[6]	[7]

13.6.2 Depth-first Search Algorithm

Depth-first search begins at a starting node A which becomes the current node. Then, it examines each node N along a path P which begins at A. That is, we process a neighbor of A, then a neighbour of neighbour of A, and so on. During the execution of the algorithm, if we reach a path that has a node N that has already been processed, then we backtrack to the current node.

Algorithm: DFS
Input: source & destination node
Output: Node visited list

```
Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Push the starting node A on the stack and set
        its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until STACK is empty
Step 4:   Pop the top node N. Process it and set its
          STATUS = 3 (processed state)
Step 5:   Push on the stack all the neighbours of N that
          are in the ready state (whose STATUS = 1) and
          set their STATUS = 2 (waiting state)
        [END OF LOOP]
Step 6: EXIT
```

Example:

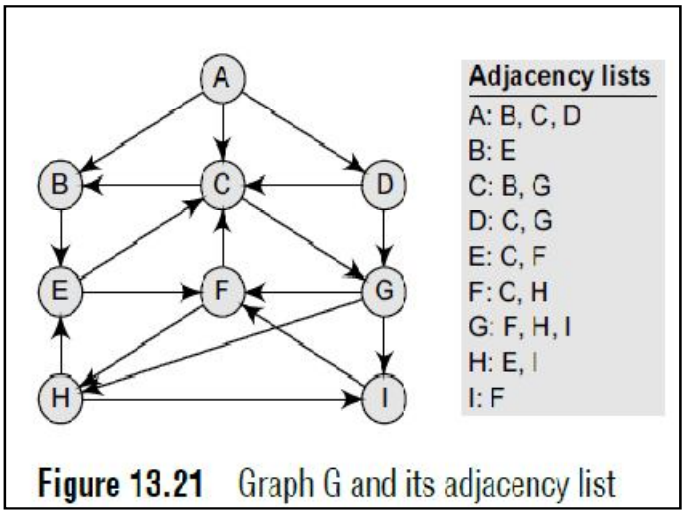


Figure 13.21 Graph G and its adjacency list

[Initial State]

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Not Defined
B	Not Defined
C	Not Defined
D	Not Defined
E	Not Defined
F	Not Defined
G	Not Defined
H	Not Defined
I	Not Defined

DFS Result:

Stack:

[-1]

Empty

1: Step: All Node are ready state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Ready
B	Ready
C	Ready
D	Ready
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

DFS Result:

Stack:

[-1]

Empty

2: Step: Push the starting node A and set its STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Waiting
B	Ready
C	Ready
D	Ready
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

DFS Result:

Stack:

[-1]	**[0]**
	A

3: Step: Pop a node A Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Ready
C	Ready
D	Ready
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

DFS Result:

A

Stack:

[-1]	**[0]**

4: Step: Push all the neighbours of A whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Waiting
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

DFS Result:

A

Stack:

[-1]	**[0]**	[1]	[2]
	B	C	D

5: Step: Pop a node D Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Ready
F	Ready
G	Ready
H	Ready
I	Ready

DFS Result:

A
D

Stack:

[-1]	**[0]**	[1]	[2]
	B	C	

6: Step: Push all the neighbours of D whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Ready
F	Ready
G	Waiting
H	Ready
I	Ready

DFS Result:

A
D

Stack:

[-1]	**[0]**	[1]	[2]
	B	C	G

7: Step: Pop a node G Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Ready
F	Ready
G	Processed
H	Ready
I	Ready

DFS Result:

A
D
G

Stack:

[-1]	**[0]**	[1]	[2]
	B	C	

8: Step: Push all the neighbours of G whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Ready
F	Waiting
G	Processed
H	Waiting
I	Waiting

DFS Result:

A
D
G

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C	F	H	I

9: Step: Pop a node I Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Ready
F	Waiting
G	Processed
H	Waiting
I	Processed

DFS Result:

A
D
G
I

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C	F	H	

10: Step: Push all the neighbours of I whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Ready
F	Waiting
G	Processed
H	Waiting
I	Processed

DFS Result:

A
D
G
I

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C	F	H	

11: Step: Pop a node H Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Ready
F	Waiting
G	Processed
H	Processed
I	Processed

DFS Result:

A
D
G
I
H

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C	F		

12: Step: Push all the neighbours of H whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Waiting
F	Waiting
G	Processed
H	Processed
I	Processed

DFS Result:

A
D
G
I
H

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C	F	E	

13: Step: Pop a node E Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Processed
F	Waiting
G	Processed
H	Processed
I	Processed

DFS Result:

A
D
G
I
H
E

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C	F		

14: Step: Push all the neighbours of E whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Processed
F	Waiting
G	Processed
H	Processed
I	Processed

DFS Result:

A
D
G
I
H
E

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C	F		

15: Step: Pop a node F Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Processed
F	Processed
G	Processed
H	Processed
I	Processed

DFS Result:

A
D
G
I
H
E
F

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C			

16: Step: Push all the neighbours of F whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Waiting
D	Processed
E	Processed
F	Processed
G	Processed
H	Processed
I	Processed

DFS Result:

A
D
G
I
H
E
F

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B	C			

17: Step: Pop a node C Process it and set its STATUS processed state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Processed
D	Processed
E	Processed
F	Processed
G	Processed
H	Processed
I	Processed

DFS Result:

A
D
G
I
H
E
F
C

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B				

18: Step: Push all the neighbours of C whoose status ready and set STATUS waiting state

Adjanacency List

Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node Status

Node	Status
A	Processed
B	Waiting
C	Processed
D	Processed
E	Processed
F	Processed
G	Processed
H	Processed
I	Processed

DFS Result:

A
D
G
I
H
E
F
C

Stack:

[-1]	**[0]**	[1]	[2]	[3]	[4]
	B				

19: Step: Pop a node B Process it and set its STATUS processed state

Adjanacency List	
Node	Neighours Node
A	B, C, D
B	E
C	B, G
D	C, G
E	C, F
F	C, H
G	F, H, I
H	E, I
I	F

Node	Status
A	Processed
B	Processed
C	Processed
D	Processed
E	Processed
F	Processed
G	Processed
H	Processed
I	Processed

DFS Result:
A
D
G
I
H
E
F
C
B

Stack:

[-1]	[0]	[1]	[2]	[3]	[4]

04 Graph representation using sparse matrix:

Sparse matrix is a matrix that has large number of elements with a zero value. In order to efficiently utilize the memory, specialized algorithms and data structures that take advantage of the sparse structure should be used. If we apply the operations using standard matrix structures and algorithms to sparse matrices, then the execution will slow down and the matrix will consume large amount of memory. Sparse data can be easily compressed, which in turn can significantly reduce memory usage.

	0	1	2	3	4	5
0	0	8	0	0	0	0
1	0	0	0	3	0	0
2	0	0	0	0	0	5
3	0	6	9	2	0	0
4	0	0	0	0	0	0

	0	1	2
	Rows	Columns	Value
0	0	1	8
1	1	3	3
2	2	5	5
3	3	1	6
4	3	2	9
5	3	3	2