Unit 5 Graphs

Structure:

5.1 Introduction

In the previous units we have seen the different ways of representing the Data structure. In this unit, we will discuss about the other form known as Graph. Graphs are the fundamental data structure in the world of programming and are used to model a large variety of data. Graph algorithms are useful in a huge number of applications in computer science. There are many problems which can be formulated in terms of a set of entities and relationships between them. There can be objects as entities which are related or linked to each other in some way and some type of planning or analysis has to be done between them. For example towns are entities which are related (or linked) by roads (or railways) and one has to find the shortest route between them. Computer nodes can be entities which are connected and related through the networks and this picture can make easier in solving any type of routing problems.

By the use of Graphs and several algorithms associated to it, so many applications in the computer science and mathematics have been written. Several algorithms in graphs are used to perform the standard graph operations such as searching the graph and finding the shortest path between nodes of a graph.

Objectives:

After studying this unit, you should be able to:

- describe basic concepts about graphs as data structures
- represent graph in matrix form and in list structures
- write algorithms for finding paths between nodes
- explain the spanning tree

5.2 Basic Concepts about Graphs

In a graph data structure there are certain terms and terminologies that you should know about. A graph data structure consists mainly of

a) A set of entities called nodes (or vertices or points).
b) A set of edges (or arcs or links) connecting the vertices.

A graph structure G is defined as G = (V, E) where V is a set of vertices, E is a set of edges, and each edge is formed from pair of distinct vertices in V.

A graph data structure may also relate to each edge with some edge value, such as a symbolic label or a numeric attributes like cost, capacity, length, etc.

Unlike tree data structure graph does not have any root node like structure. Any node can be connected with any other node in the graph. Nodes do not have any clear parent-child relationship like in the tree. Instead nodes are called as neighbors if they are connected by an edge.

As in mathematics, an edge E=(v_1,v_2) is said to point or traverse from v_1 to v_2. The nodes may be part of the graph structure, or may be external entities represented by integer indices or references.

Basic Terms and Terminologies

- **Arc:** Directed link between two nodes is known as Arc. For example, the arc (v_1,v_2) traverse from *tail* node v_1 to *head* node v_2.
- **Edge:** Undirected link between the nodes is known as Edge.

- **Path:** A path is a sequence of consecutive edges in a graph and the length of the path is the number of edges traversed.

Operations

The basic operations provided by a graph data structure *G* usually include:

- Adjacent (*G*, v_1, v_2): tests whether there is an edge from node v_1 to node v_2 .
- Neighbors (*G*, v_1): lists all nodes *y* such that there is an edge from v_1 to v_2.
- Add (*G*, v_1, v_2): adds to *G* the edge from v_1 to v_2, if it is not there.
- Delete (*G*, v_1, v_2): removes the edge from v_1 to v_2, if it is there.
- get_node_value (*G*, v_1): returns the value associated with the node v_1.
- set_node_value (*G*, v_1, *a*): sets the value associated with the node v_1 to *a*.

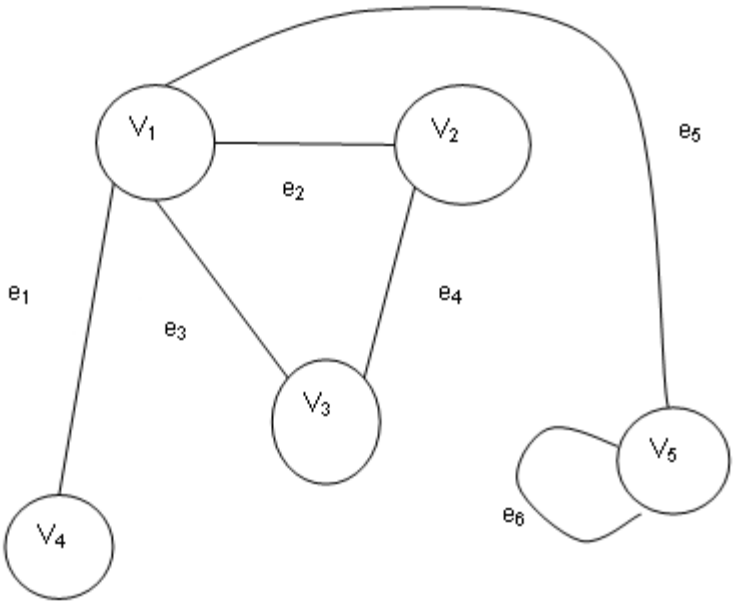
  Structures that associate values to the edges usually provide also

- get_edge_value (*G*, v_1, v_2): returns the value associated to the edge (v_1, v_2).
- set_edge_value (*G*, v_1, v_2, *v*): sets the value associated to the edge (v_1,v_2) to *v*.

Types of Graphs

Depending upon the vertices and edges and the weight associated to it, graphs can be classified as:

1) **Undirected graph:** In Undirected Graph, the directions of edges are not assigned. Edges in the Undirected graph only connect to each other. In an undirected graph, edge (v_1,v_2) is equivalent to edge (v_2,v_1) since they are unassigned. An example of Undirected graph is given in Figure 5.1.



Here $e_1 = (v_1, v_4)$ and $e_6 = (v_5, v_5)$. e_6 is a self-loop.

Figure 5.1: Example of undirected graph

2) **Directed graph:** A directed graph **G** is defined as a graph where each edge **e** in **G** is assigned a direction. In directed graph each edge **e** is identified with an ordered pair (v_1,v_2) of nodes in G rather than an unordered pair (v_1,v_2). Or we can also say that in directed graph (v_1,v_2) and (v_2,v_1) are two edges where the former edge leaves v_1 and ends at v_2, and the opposite for the latter.

   In *directed graph* since each edge has a capacity and each edge receives a flow, it can be seen as a *flow network.* The *Ford-Fulkerson algorithm* is used to find out the maximum flow from a source to a sink in a graph. An example of Directed graph is shown in figure 5.2.

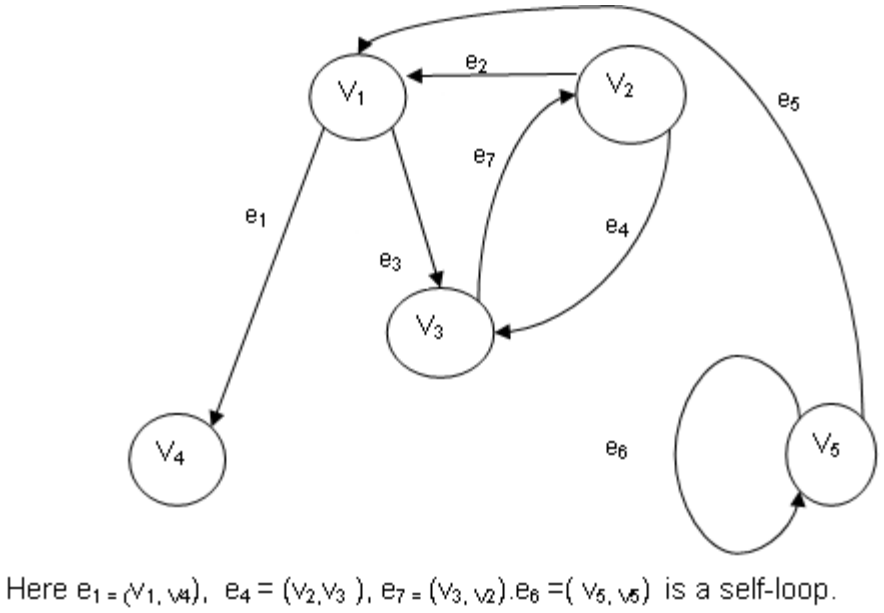


Figure 5.2: Example of directed graph.

3) **Weighted graphs:** The graph is called *Weighted Graph***,** if information like cost or weight is associated to the traversal of an edge. Directed and undirected graphs may both be weighted. The operations on a weighted graph are the same with addition of a weight parameter during edge creation. Weighted Graph operation is the extension of Undirected or Directed graph operation.

4) **Multigraph:** A multigraph is a graph which has more than one edge between the same two vertices. For example, if one were modeling airline flights, there might be multiple flights between two cities, occurring at different times of the day.

   When we want to distinguish between different nodes and edges then we can associate labels with each nodes and edges. If a label is associated with each nodes and edges in a graph then such a graph is said to be **labeled**.

5) **Sparse graph:** Graphs are said to be ***sparse*** if the number of edges is far less than $|V|^2$ i.e. $|E| << |V|^2$.

6) **Acyclic graph:** It is defined as a *graph* with no *path* and starts and ends at the same *vertex*. *An acyclic undirected graphic* is like a tree*.*

Directed Acyclic graph (**DAG**) is a *directed graph* with no *path* that starts and ends at the same *vertex*. It is also known as *oriented acyclic graph.*

7) **Graph with isolated vertices:** If a vertex is not adjacent to any other vertices in a graph, then such type of vertex is known as Isolated Vertex. Figure 5.3 shows graph with Isolated Vertices.

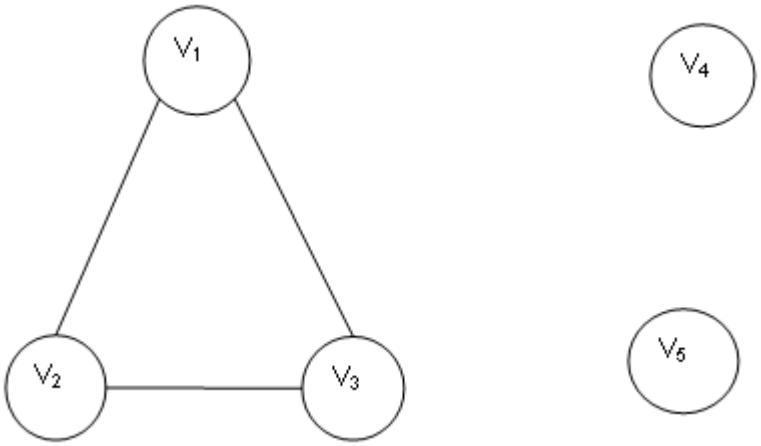


Figure 5.3: Graph with Isolated Vertex. Here v_4 and v_5 are isolated vertex

8) **Null graph:** A graph containing only isolated vertices is called Null Graph. Figure 5.4 shows Null Graph.

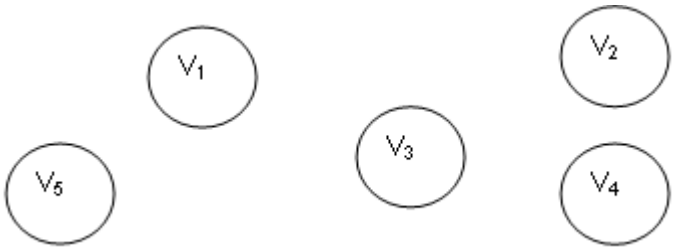


Figure 5.4: Example of a Null Graph

9) **Strongly connected graph:** A directed graph G is said to be strongly connected if for each pair (v_1, v_2) of nodes in G there is a path from v_1 to v_2 and there is also a path from v_2 to v_1. Whereas G is said to be unilaterally connected graph if for any pair (u, v) of nodes in G, there is a path from u to v or a path v to u.

10) **Unilaterally connected graph:** A graph is said to be Unilaterally connected graph if for any pair (v_1,v_2) of nodes in G, there is a path from v_1 to v_2 or a path from v_2 to v_1.

Degree

In an undirected graph, the ***degree*** of a vertex v is the number of edges connected to v.

In a directed graph, the ***out-degree*** of a vertex v is the number of edges leaving v, and its ***in-degree*** is the number of edges ending at v.

In the figure 5.1, the degree of V_5 is 2. In figure 5.2, the out-degree of V_5 is 2 whereas the in-degree of V_5 is 1.

Number of Edges

If an edge e connects vertex v_1 and v_2, we denote it as e= (v_1, v_2). And if $v_1 = v_2$, e is called a *self-loop*. In the figure 5.1, number of edge is 6 and e_6 is a self-loop.

If there is an edge between each pair of vertices in V, an undirected or directed graph can have O ($|V|^2$) edges, namely, $|E| = O(|V|^2)$.

In many applications, the number of edges is far less than $|V|^2$ i.e. $|E| << |V|^2$. Such graphs are said to be ***sparse.***

In Figure 5.2, if we consider the elements of edge E, we find that E is a set of 7 tuples, i.e

E= {(v_1, v_4), (v_2, v_1), (v_1, v_3), (v_3, v_2), (v_5, v_1), (v_5, v_5), (v_3, v_2) }

Where e_1 = (v_1, v_4), e_2 = (v_2, v_1), e_3 = (v_1, v_3), e_4 = (v_2, v_3), e_5 =(v_5, v_1), e_6= (v_5, v_5) and e_7 =(v_3, v_2)

From e_3 and e_7, we find that v_3 is common in both. In the e_3, v_3 is the terminal vertex and it is initial vertex of e_7. Similarly, in e_7 and e_2, v_2 is common and so on. We can see that there is a cycle in the graph between vertices v_1, v_3 and v_2 and there exists a *circular path*.

Self Assessment Questions

1. Nodes do not have any clear parent-child relationship like in the tree. (True/False)
2. In an undirected graph, edge (v1,v_2) is equivalent to edge _____ since they are unassigned.

3. A graph containing only isolated vertices is called _____. (Pick the right option)
   a) isolated graph
   b) null graph
   c) directed graph
   d) multigraph
4. In a directed graph, the _____ of a vertex v is the number edges of leaving v, and its _____ is the number of edges ending at v. (Pick the right option)
   a) in-degree, out-degree
   b) out-degree, in-degree
   c) degree, edges
   d) edges, degree

5.3 Matrix Representation of Graphs

Till now you already have a basic idea of the logical representation of the graphs. Let's take a look at Matrix representation of a graph that is commonly represented in computers. The representation is commonly in two forms. They are **Adjacency matrix** and **Incidence Matrix.**

5.3.1 Adjacency matrix

It is a two-dimensional Boolean matrix to store the information about the graph nodes. Here the rows and columns represent source and destination vertices and entries in the matrix indicate whether an edge exists between the vertices associated with that row and column. The values of matrix are either 0 or 1.

Suppose if a graph G consists of v_1 , v_2 , v_3 , v_4 , v_5 vertices then the adjacency matrix A = $[a_{ij}]$ of the graph G is the n x n matrix and can be defined as:

a_{ij} =1 if v_i is adjacent to v_j (i.e if there is an edge between v_i and v_j) whereas a_{ij} =0 if there is no edge between v_i and v_j

In figure 5.5, there are 5 vertices and 7 edges. The set of vertices from the graph shown in figure 5.5 is

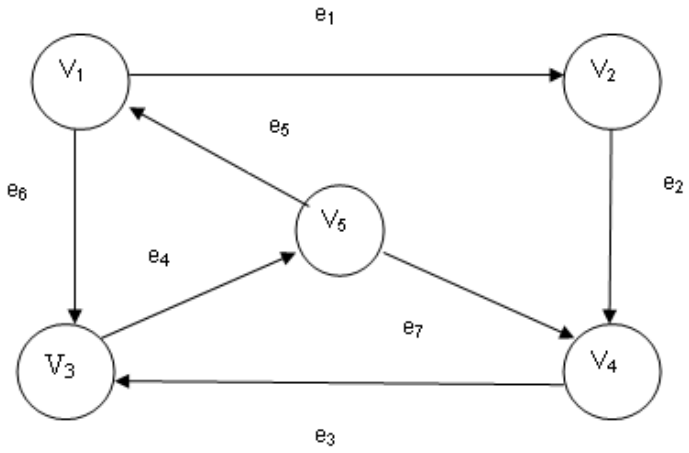


Figure 5.5: Example to demonstrate a matrix for directed graph

V= {v_1, v_2, v_3, v_4, v_5}

And the set of edges is

E= {(v_1, v_2), (v_2, v_4), (v_4, v_3), (v_3, v_5), (v_5, v_1), (v_1, v_3), (v_5, v_4) }

Where e_1 = (v_1, v_2), e_2 = (v_2, v_4), e_3 = (v_4, v_3), e_4 =(v_3, v_5), e_5 =(v_5, v_1), e_6= (v_1, v_3) and e_7 =(v_5, v_4)

The adjacency matrix for G for the graph of figure 5.5 is as follows:

	V_1	V_2	V_3	V_4	V_5
V_1	0	1	1	0	0
V_2	0	0	0	1	0
V_3	0	0	0	0	1
V_4	0	0	1	0	0
V_5	1	0	0	1	0

We see that '1' is marked in a cell if there exists an edge between two nodes that index that cell. For example, since we have edge e_1 from v_1 to v_2, we mark a '1' in the cells indexed by v_1 and v_2 otherwise it is 0. Similarly, the edge e_5 is there between v_5 and v_1, so we mark a '1' in the cells indexed by v_5 and v_1.

For a null graph, which consists of V vertices but no edges, the adjacency matrix has all of its entire elements as zero.

5.3.2 Incidence matrix

It is a two-dimensional matrix, in which the rows represent the vertices and columns represent the edges. The entries in an array indicate if both are related to each other through edges. The values of the matrix are given as -1, 0, or 1.

If the k^{th} edge is $(v_{i,} v_j)$ then the k^{th} column has a value 1 in the i^{th} row, -1 in the j^{th} row and 0 elsewhere.

For example, the Incidence matrix for the graph of figure 5.5 is given as:

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
V_1	1	0	0	0	-1	1	0
V_2	-1	1	0	0	0	0	0
V_3	0	0	-1	1	0	-1	0
V_4	0	-1	1	0	0	0	-1
V_5	0	0	0	-1	1	0	1

From the figure 5.5, we observe that, $e_1 = (v_{1,} v_2)$, $e_2 = (v_{2,} v_4)$, $e_3 = (v_{4,} v_3)$, $e_4 = (v_{3,} v_5)$, $e_5 = (v_{5,} v_1)$, $e_6 = (v_{1,} v_3)$ and $e_7 = (v_{5,} v_4)$.

So , 1^{st} edge (e_1) has vertices v_1 and v_2 and hence the first column has the value 1 in the 1^{st} row , -1 in the 2^{nd} row. Similarly 3^{rd} edge (e_3) has vertices v_4 and v_3 and hence the third column has the value 1 in the 4^{th} row , -1 in the 3^{rd} row. We see that zero is elsewhere.

If the vertex is having a self-loop, in such case the k^{th} edge is $(v_{i,} v_i)$, then this will be represented in a matrix with 1.

Self Assessment Questions

5. In adjacency matrix, '1' is marked in a cell if there exists an edge between two nodes that index that cell. (True/false)
6. A null graph consists of V vertices but no edges, then the adjacency matrix has all of its entire elements as _____.
7. In incidence matrix, If the k^{th} edge is (v_i, v_j) then the k^{th} column has a value _____ in the i^{th} row, _____ in the j^{th} row and _____ elsewhere.
   a) 1,-1,0
   b) -1,1,0
   c) 0,1,-1
   d) 0,-1,1

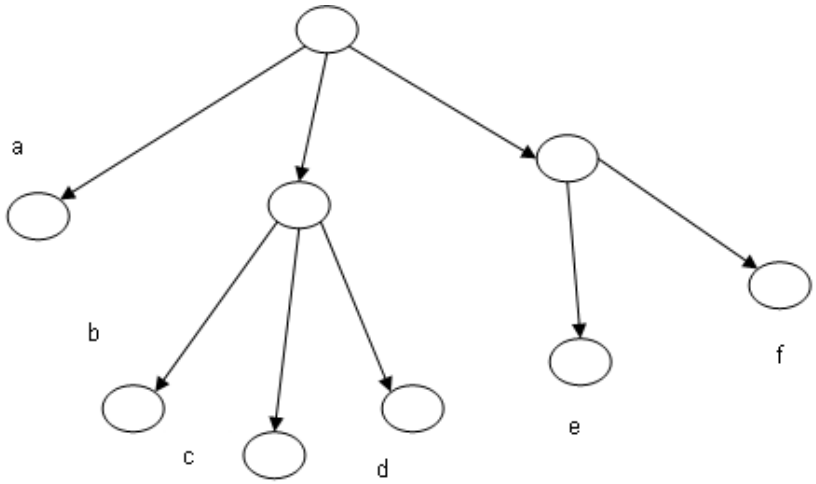
5.4 List Structures

Graphs can also be represented as List structures and can be used to represent a directed graph. Manipulation of such a list structure is known as *List processing*. List structures are used to solve the problem related to the area of artificial intelligence, algebraic manipulation, text processing, graphics etc. When the programs are written for these areas, then (a) the exact amount of data storage required cannot be determined easily and (b) operations like insertions and deletions are performed frequently on the data structures used. Hence a data structure was required which would utilize the available storage space to provide maximum problem solving capacity and would support efficient manipulation algorithm in terms of time. The list structures satisfy both the conditions.

In the context of list processing, we define a list to be any finite sequence of zero or more atoms or lists, where an atom is taken to be any object (e.g., a string of symbols) that is distinguishable from a list by treating the atom as structurally indivisible. If we enclose lists within parentheses and separate elements of lists by commas, then the following can be considered to be lists:

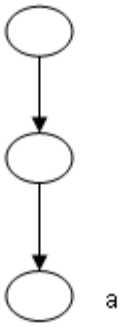
i) (a, (b,c, d), (e,f))
ii) ()
iii) ((a))

The first list contains 3 elements, namely, the atom a, the list (b, c, d) which contains the atoms b, c and d and the list (e, f) containing the atoms e and f. The second list has no elements and hence it is a null list which is too a valid list. The third list has one element, the list (a) which in turn contains the atom a. This list structures can be represented graphically and hence the graphic representation of the above examples is given in Figure 5.6.



(i) For list (a, (b,c,d) , (e,f))

(ii) For list ( )



(iii) For list ((a))

Figure 5.6: Graphic representations of list structures

Another notation that is often used to illustrate lists is similar to that used in the linked list representation of trees. This type of representation can be shown in the form of Storage representation. The example of such type of representation is given in Figure 5.7 in sub-section 5.4.1.

The following three properties are associated with list structures:

i) **Order:** A transitive relation defined on the elements of the list and specified by the sequence in which the elements appear within the list. In the list (a, b, c), a precedes b and b precedes c implies that a precedes c. This list is not equal to the list (c, b, a).

ii) **Depth:** The depth of a list is the maximum level attributed to any element within the list or within any sublist in the list. The level of an element is indicated by the nesting of lists within lists. i.e., by the number of pairs of parentheses surrounding the element. In the list of Figure 5.6(i), the element a is in a level 1 whereas remaining elements b,c,d and e,f are in a level of 2. The depth of entire list is 2.

iii) **Length:** This is the number of elements at level 1 in a list. For example, the length of list (a, (b, c, d), (e, f)) is 3.

5.4.1 Adjacency list

An adjacency list is implemented as an array of lists with one list of destination nodes for each source node. The adjacency list structure extends the edge list structure by adding incidence containers to each vertex.

In undirected graph, every entry is a set (or multiset) of two nodes containing the two ends of the corresponding edge.

In directed graph, every entry is a tuple of two nodes, one denoting the source node and the other denoting the destination node of the corresponding arc.

Typically, adjacency lists are unordered. Adjacency lists are preferred for sparse graphs.

Let G = (V, E) is a graph with **n** vertices. This graph can be represented **by n x n** adjacency matrix. The **n** linked lists represent the **n** rows of the adjacency matrix **A** and there is one linked list for each vertex in **G**. The vertices in the list **L** represent the vertices that are adjacent to the vertex **L.** Here each vertex is represented by two field i.e Vertex and Link. The Vertex fields contain the indices of the vertices adjacent to vertex **L**. The adjacency list for the graphs in figure 5.5 is given in figure 5.7. The figure is shown as storage representations.

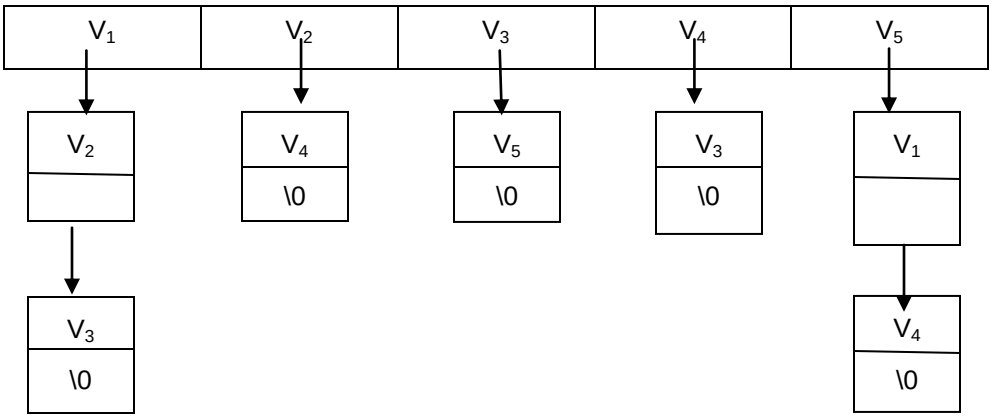


Figure 5.7: Storage representations for adjacency list for the graph of figure 5.5

The number of distinct unordered pairs **(v_i, v_j)** with **$v_i \neq v_j$** in a graph with **n** vertices is **n(n-1)/2**. This is the maximum number of edges in any **n** vertex undirected graph. An **n** vertex undirected graph with exactly **n(n-1)/2** edges is said to be complete. In the case of a directed graph having **n** vertices the maximum number of edges is **n(n-1).**

5.4.2 Incidence list

In Incidence List, each vertex has list of pointers to edges which it is incident to. Advantage of this is that two vertices that are adjacent to this edge share the same instance of edge, so when you want to manipulate with edge data (for example flow or cost), you only change data in one object. You also modify neighbor function to make your graph directed, or mixed.

Self Assessment Questions

8. For adjacency list the number of distinct unordered pairs (v_i, v_j) with $v_i \neq v_j$ in a graph with n vertices is _____.
9. *Pick the right option*
   The three properties associated with list structures are:
   a) Order, depth, length
   b) Degree, vertices, edges
   c) Atom, element, list
   d) Cost, weight, data

5.5 Other Representations of Graphs

A graph can be represented in many different ways depending on the application whichever is appropriate and it can be stored in variety of ways depending upon its way of representation. We have already seen few way of data representation in the above subunits. We will briefly introduce about few of the other ways of representing and storing the graph.

Storage representation for adjacency list: This form of storing a graph is better if there are a number of edges between a pair of nodes and a considerable number of nodes that are connected to only a few other nodes. Undirected graphs can also be stored using this data structure but each edge is represented twice, once in each direction. Such a representation becomes important in applications where grammars must be examined and manipulated in top-down parsing as in case of Trees. The storage representation for adjacency list is given in figure 5.7 above.

Edge list: It is one of the simplest forms of representing a graph. In this method, each pair of nodes (v_1, v_2), connected by an edge are included in the listing. If the graph is a digraph, then v_1 is the initiating node and v_2 is the terminating node. If the graph is undirected, ordering is not imposed on the node. A simple data structure to represent this listing would be a linked list. Fields could be added to this structure to store information about the edges but an additional structure is needed if we want to add information about the nodes.

5.6 Algorithms for Graph Traversal

Graph algorithms are generally used in computer science to find and examine a path between the nodes using algorithm like *depth-first search* and *breadth-first search*. These algorithms are used generally for the examination of the graph so that further manipulation of data structure becomes easier and more appropriate. We will deal with Depth first search and breadth first search in this topic.

5.6.1 Depth first search

The depth first search algorithm is used to visit all of the nodes in the graph for its examination and not always necessary to find the shortest path.

Depth first search is done by taking a node, checking its neighbors, expanding the first node it finds among the neighbors, checking if that

expanded node is our destination, and if not, continue exploring more nodes. In Depth First search it should be noted that same node is not visited more than once, otherwise possibility of infinite recursion may occur.

Algorithms for Depth first search is similar to the one that is mentioned in the previous unit. The concept of algorithm is same either it be in trees or in graph.

5.6.2 Breadth first search

This algorithm uses a queue data structure to perform the search. The effect of this is to process all nodes adjacent to the start node before we process the nodes adjacent to those nodes.

If all of the edges in a graph are unweighted (or the same weight) then the first time a node is visited is the shortest path to that node from the source node.

Algorithms for Breadth first search is also similar to the one that is mentioned in the previous unit. The concept of algorithm is same either it be in trees or in graph.

There are other algorithms used to find the shortest path from one node to another like Dijkstra's algorithm . Dijkstra's algorithm is described in unit 7. The shortest path from each node to every other node is also solved using the Floyd algorithm, Warshall algorithm and modified Warshall algorithms.

Self Assessment Questions

10. In Depth First search the same node is visited more than once. (True/False)
11. Breadth first search algorithm uses a _____ data structure to perform the search. (Pick the right option)
    a) Tree
    b) Linked-list
    c) Queue
    d) Hashes

5.7 Spanning Tree

A spanning tree T is defined as an undirected tree of a connected graph G which is composed of all the vertices and the edges necessary to connect all the nodes of graph *G*. It has the property that every vertex lies in the tree,

but no cycles (or loops) are formed. That is, for any pair of nodes there exits only one path between them and the insertion of any edge to a spanning tree forms a unique cycle. Those edges left out of the spanning tree that were present in the original graph connect paths together in the tree. If the Graph G has n vertices then it has n-1 edges without any cycles. If the graph is not connected, it forms a ***spanning forest.***

A single graph can have many spanning trees. The particular spanning tree for a graph depends on the criteria used to generate it. If a depth first search is used, those edges traversed by the algorithm form the edges of the tree, referred to as a depth first spanning tree. If a breadth first search is used, the spanning tree is formed from those edges traversed during the search, producing a breadth first spanning tree.

For example, consider the connected and undirected graph G in figure 5.8.

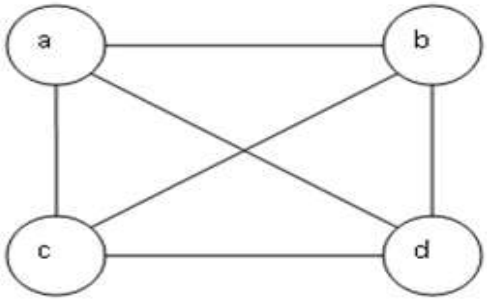


Figure 5.8: Connected undirected graph G

A spanning tree is a subgraph of G that includes all the vertices of G and is also a tree. The edges of the trees are also called branches.. As we have already said that depending upon the criteria or algorithm used to generate a tree, we can have different forms of trees. The four spanning trees of graph G of figure 5.8 are shown in the figure 5.9.

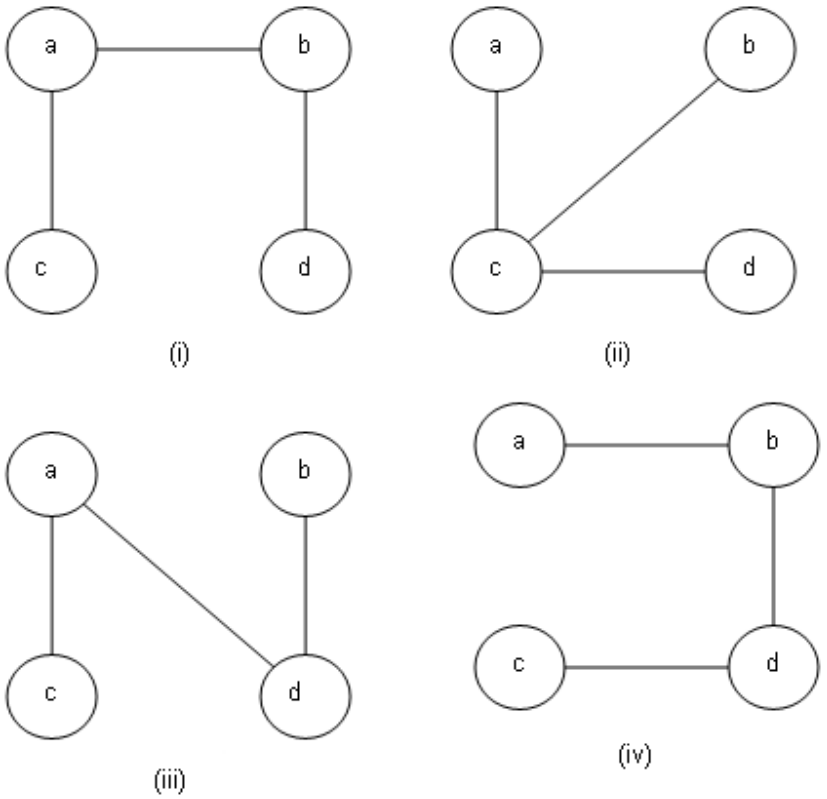


Figure 5.9: Four of the spanning trees of the graph G of figure 5.8

Generally, to find a spanning tree of a graph, we can pick an initial node and call it part of the spanning tree and do a search from the initial node. While doing a search each time you find a node that is not in the spanning tree, add to the spanning tree both the new node *and* the edge you followed to get to it.

Sometimes we need to find out the cost of a spanning tree of a weighted graph. The cost is given by the sum of the weights of the tree's edge. We always need to minimize the total cost and hence a minimal cost spanning tree is formed.

Description about the minimum spanning tree (MST) is given in the next unit. It is a tree with the lowest total cost in the Spanning Tree.

Self Assessment Questions:

12. A spanning tree is known as spanning forest if the graph is undirected and connected. (True/False)
13. If the graph is not connected in the spanning tree, then it is known as _____.

5.8 Summary

This unit has given you a brief overview of various types of graphs and various ways of representing and storing a graph depending upon the computer applications. It has also described different types of algorithms to find out the possible paths to be traversed from one node to another node. Let us summarize the concepts:

- *Various forms of Graphs:*
  The graphs can be undirected graph, Directed graph, Weighted Graph, Multigraph, sparse Graph etc.
- *Structures of Graphs:*
  Graphs can be represented in the form of lists or in the matrix form and each form has its own significance with it.
- *Algorithms:*
  Various types of algorithms are there to find and examine paths between the nodes. They are depth first search, breadth first search etc.

Along with these you also came to know about spanning tree.A spanning tree of a graph is an undirected tree composed of all the vertices and the edges necessary to connect all the nodes of graph. We find minimum spanning tree in order to get the least cost in terms of weight of the edges of tree. A *minimum spanning tree* would be one with the lowest total cost.

5.9 Terminal Questions

1. Explain different types of Graphs.
2. What do you understand by in-degree and out-degree?
3. Explain Adjacency Matrix and Incidence matrix form of representing a graph.
4. Explain Breadth-first search algorithm.
5. Explain Depth-first search algorithm.
6. Explain spanning tree.

5.10 Answers

Self Assessment Questions

1. True
2. (v_2,v_1)
3. b) - null graph
4. b) Out-degree, In-degree.
5. True
6. zero)
7. a) 1,-1,0
8. n (n-1)/2
9. a) order, depth, length
10. False
11. c) Queue.
12. False
13. spanning forest

Terminal Questions

1. Depending upon the direction and weight associated to edges, Graph can be Directed, Undirected, Null and so on. (Refer section 5.2 for detail)
2. In an undirected graph, the ***degree*** of a vertex v is the number of edges connected to v. In a directed graph, the ***out-degree*** of a vertex v is the number of edges leaving v, and its ***in-degree*** is the number of edges ending at v. (Refer section 5.2)
3. Adjacency Matrix is Boolean matrix to store the information about the graph nodes where the rows and columns represent source and destination vertices and entries in the matrix indicate whether an edge exists between the vertices associated with that row and column. Where as in Incidence matrix the rows represent the vertices and columns represent the edges. And the entries in an array indicate if both are related to each other through edges. The values of the matrix are given as -1, 0, or 1. (Refer section 5.3)
4. Breadth-first search algorithm uses a queue data structure to perform the search. (Refer section 5.3)

5. The depth first search algorithm is used to visit all of the nodes in the graph for its examination and not always necessary to find the shortest path. (Refer section 5.6)
6. A spanning tree T is defined as an undirected tree of a connected graph G which is composed of all the vertices and the edges necessary to connect all the nodes of graph *G*. Every vertex lies in a tree. (Refer section 5.6)