18MCA34C

DESIGN AND ANALYSIS OF ALGORITHM

UNIT II

Elementary Data Structures

FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

Assistant Professor,

Postgraduate Department of Computer Applications,

Government Arts College (Autonomous),

Coimbatore-641018.

STACKS AND QUEUES

One of the most common forms of data organization in computer programs is the ordered or linear list, which is often written as a = (ai,a,2,... ,an)- called as atoms and they are chosen from some set.

The null or empty list has n = 0 elements.

A stack is an ordered list in which all insertions and deletions are made at one end, called the top.

A queue is an ordered list in which all insertions take place at one end, the rear,

all deletions take place at the other end, the front.

The operations of a stack imply that if the elements A, B, C, D, and E are inserted into a

stack, in that order, then the first element to be removed (delet.ed) must be E.

the last element to be inserted into the stack is the first to be removed.

stacks are referred to as Last In First Out (LIFO) lists.

The operations of a queue require that the first element that is inserted into the queue is the first one to be removed.

Thus queues are known as First In First Out (FIFO) lists

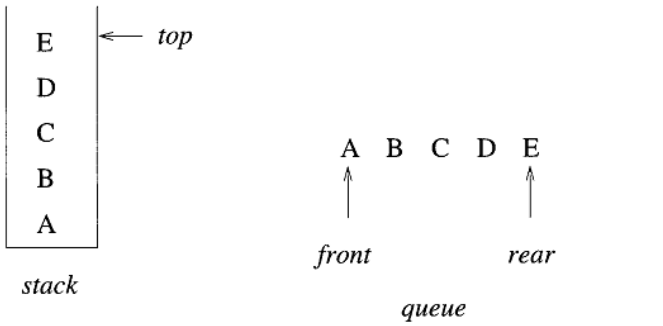


Figure 2.1 Example of a stack and a queue

The simplest way to represent a stack is by using a one-dimensional array, stack[0:n-1] where n is the maximum number of allowable entries

The first or bottom element in the stack is stored at stack[1], the second at stack[2],and the ith at

stack[i-1]

variable top which points to the top element in the stack.

Another way to represent a stack is by using links (or pointers).

A node is a collection of data and link information.

A stack can be represented by using nodes with two fields, possibly called data and link.

The data field of each node contains an item in the stack and the corresponding link field

points to the node containing the next item in the stack.

The link field of the last node is zero, for we assume that all nodes have an address greater than zero.

For example, a stack with the items A, B, C, D, and E inserted in that order, looks as in Figure 2.2

```
1   Algorithm Add(item)
2   // Push an element onto the stack. Return true if successful;
3   // else return false. item is used as an input.
4   {
5       if (top ≥ n − 1) then
6       {
7           write ("Stack is full!"); return false;
8       }
9       else
10      {
11          top := top + 1; stack[top] := item; return true;
12      }
13  }
```

```
1   Algorithm Delete(item)
2   // Pop the top element from the stack. Return true if successful
3   // else return false. item is used as an output.
4   {
5       if (top < 0) then
6       {
7           write ("Stack is empty!"); return false;
8       }
9       else
10      {
11          item := stack[top]; top := top − 1; return true;
12      }
13  }
```

Algorithm 2.1 Operations on a stack

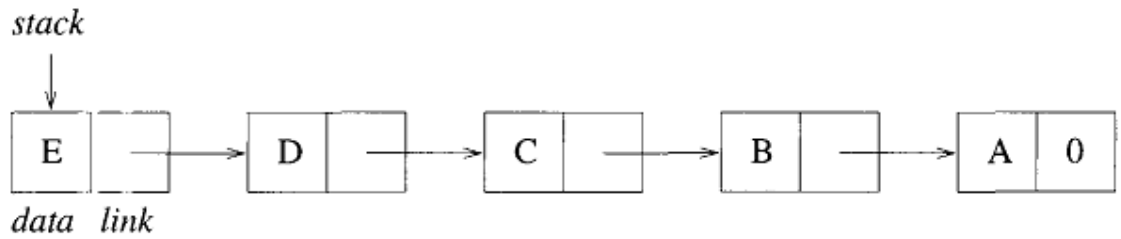


Figure 2.2 Example of a five-element, linked stack

TREES

A tree is a finite set of one or more nodes such that there is a specially designated node called the root and the remaining nodes are partitioned into n ≥ 0 disjoint sets Ti,... ,Tn, where each of these sets is a tree.

The sets Ti,... ,Tn are called the sub trees of the root.

2.2.1 Terminology

There are many terms that are often used when referring to trees. Consider the tree in Figure 2.5. This tree has 13 nodes, each data item of a node being a single letter for convenience.

The root contains A and draw trees with their roots at the top.

The number of subtrees of a node is called its degree.

The degree of A is 3,of C is 1, and of F is 0.

Nodes that have degree zero are called leaf or terminal nodes.

The set {K, L, F, G,M, I, J} is the set of leaf nodes of Figure 2.5.

The other nodes are referred to as nonterminals. The roots of the subtrees of a node X art; the children of X.

The node X is the parent of its children. Thus the children of D are H, I, and J, and the parent

of D is A.

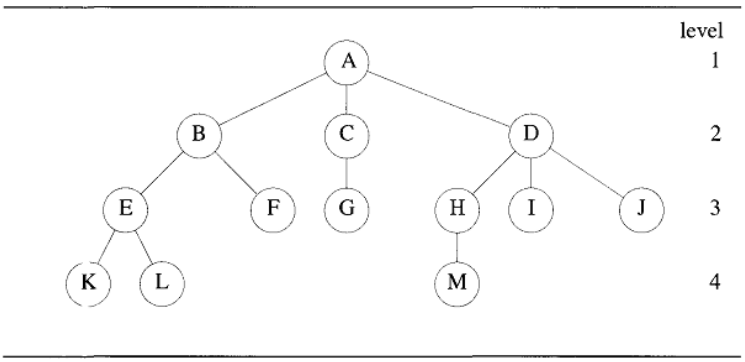


Figure 2.5 A sample tree

Children of the same parent are said to be siblings. For example H, I, and J are siblings.

The degree of a tree is the maximum degree of the nodes in the tree. The tree in Figure 2.5 has

Degree 3.

The ancestors of a node are all the nodes along the path from the root to that node.

The ancestors of M are A, D, and H.

The level of a node is defined by initially letting the root be at level one.

If a nodeis at level p, then its children are at level p + 1.Figure 2.5 shows the levels of all nodes in that tree.

The height or depth of a tree is defined to be the maximum level of any node in the tree.

A forest is a set of n $\geq$ 0 disjoint trees. The notion of a forest is very close to that of a tree because if we remove the root of a tree, we get a forest.

Binary Trees

A binary tree is an important type of tree structure that occurs very often.

It is characterized by the fact that any node can have at most two children; that is, there is no node with degree greater than two.

A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left and right subtree

Figure 2.7shows two sample binary trees. These two trees are special kinds of binary trees.

Figure2.7(a) is a skewed tree, skewed to the left.

The tree in Figure 2.7(b) is called a complete binary tree.

. The terms used in trees, such as degree,level, height, leaf, parent, and child, all apply to binary trees in the same way.

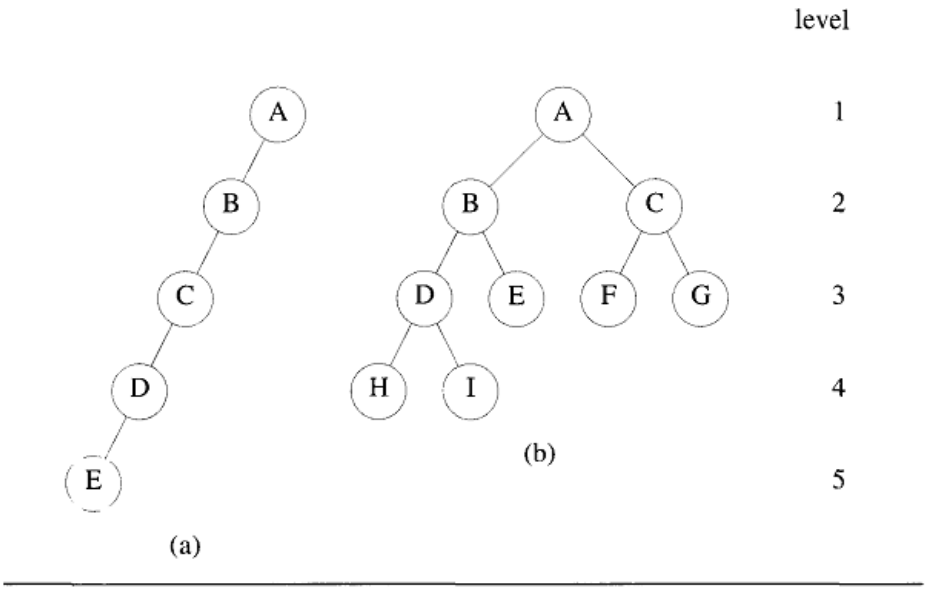


Figure 2.7 Two sample binary trees

Binary Search Trees

A binary search tree is a binary tree. It may be empty.

If it is not empty, then it satisfies the following properties:

1.Every element has a key and no two elements have the same key (i.e., the keys are distinct).

2. The keys (if any) in the left subtree are smaller than the key in the root.

3. The keys (if any) in the right subtree are larger than the key in the root.

4. The left and right subtrees are also binary search trees

A binary search tree can support the operations search, insert,and delete among others.

A binary search tree, we can search for a data element both by key value and by rank.

Searching a Binary Search Tree

Since the definition of a binary search tree is recursive, it is easiest to describe a recursive search method.

Suppose we wish to search for an element with key x.

An element could in general be an arbitrary structure that has as one of its fields a key.

If the root is 0,then the search tree contains no elements and the search is unsuccessful.

Otherwise, we compare x with the key in the root. If x equals this key, then the search terminates successfully.

If x is less than the key in the root, then no element in the right subtree can have key value x, and

Only the left subtree is to be searched.

If x is larger than the keyin the root, only the right subtree needs to be searched.

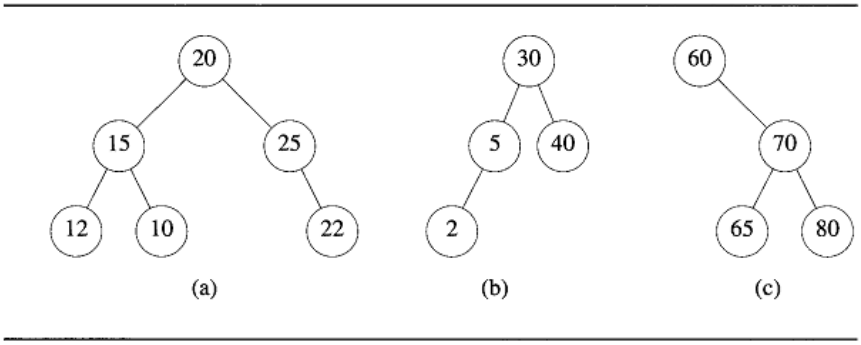


Figure 2.11 Binary trees

Insertion into a Binary Search Tree

To insert a new element x, we must first verify that its key is different from those of existing elements.

To do this, a search is carried out. If the search is unsuccessful, then the element is inserted at the Point the search terminated.

For instance, to insert an element with key 80 into the tree of Figure2.12(a),we first search for 80.

This search terminates unsuccessfully, and the last node examined is the one with key 40.

The new element is inserted as the right childof this node.

The resulting search tree is shown in Figure 2.12(b).

Figure 2.12(c) shows the result of inserting the key 35 into the search tree of Figure 2.12(b).

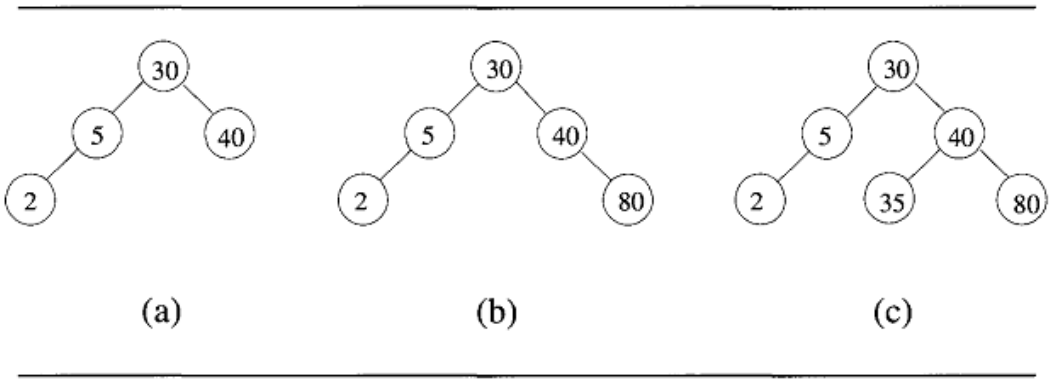


Figure 2.12 Inserting into a binary search tree

Deletion from a Binary Search Tree

Deletion of a leaf element is quite easy. For example, to delete 35 from the tree of Figure 2.12(c), the left-child field of its parent is set to 0 and the node disposed.

This gives us the tree of Figure 2.12(b). To delete the 80 from this tree, the right-child field of 40 is set to 0; this gives the tree of Figure 2.12(a).

Then the node containing 80 is disposed.

Recursive and Iterative search of BST

Eachnode has the three fields lchild, rchild, and data.

```
1   Algorithm Search(t, x)
2   {
3       if (t = 0) then return 0;
4       else if (x = t → data) then return t;
5            else if (x < t → data) then
6                    return Search(t → lchild, x);
7                else return Search(t → rchild, x);
8   }
```

Algorithm 2.4 Recursive search of a binary search tree

```
1   Algorithm ISearch(x)
2   {
3       found := false;
4       t := tree;
5       while ((t ≠ 0) and not found) do
6       {
7           if (x = (t → data)) then found := true;
8           else if (x < (t → data)) then t := (t → lchild);
9                else t := (t → rchild);
10      }
11      if (not found) then return 0;
12      else return t;
13  }
```

Algorithm 2.5 Iterative search of a binary search tree

GRAPHS

The first recorded evidence of the use of graphs datesback to 1736, when Leonhard Eulerusedthem to solve the now classical Konigsberg bridge problem.

In the town of Konigsberg (now Kaliningrad) the river Pregel (Pre-golya) flows around the island Kneiphof and then divides into two.

There are, therefore, four land areas that have this river on their borders (see Figure 2.24(a)).

These land areas are interconnected by seven bridges, labeled a to g.

Theland areas themselves are labeled A to D.

The Konigsberg bridge problem is to determine whether, starting at one land area, it is possible to walk across all the bridges exactly once in returning to the starting land area.

One possible walk: Starting from land area B,walk across bridge a to island A,take bridge e to area D, take bridge g to C, take bridge d to A, take bridge b to B, and take bridge f to D.

Solution to Konigsberg bridge problem:

He solved the problem by representing the land areas as vertices and the bridges as edges in a graph (actually a multigraph) as in Figure 2.24(b).

His solution is elegant and applies to all graphs.

Defining the degree of a vertex to be the number of edges incident to it, Euler showed that there is a walk starting at any vertex, going through each edge exactly once and terminating at the start vertex if and only if the degree of each vertex is even.

A walk that does this is called Eulerian. There is no Eulerian walk for the Konigsberg bridge problem,as all four vertices are of odd degree.

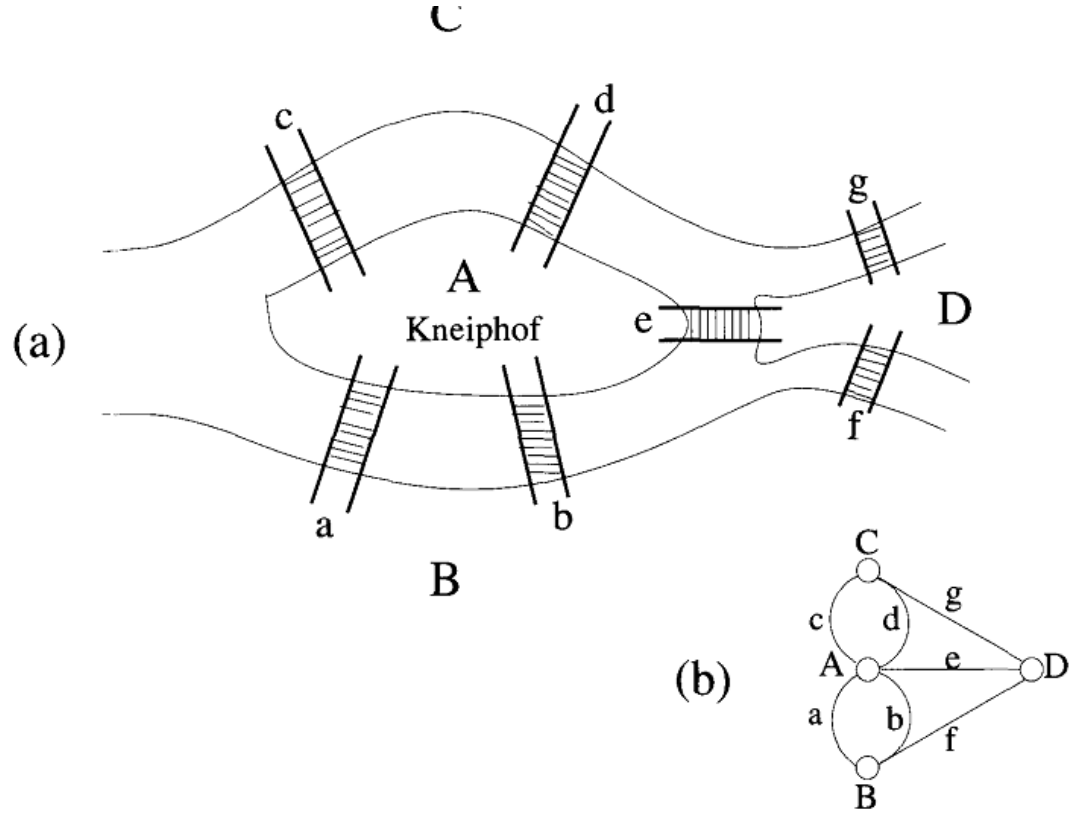


Figure 2.24 Section of the river Pregel in Königsberg and Euler's graph

Applications of graphs

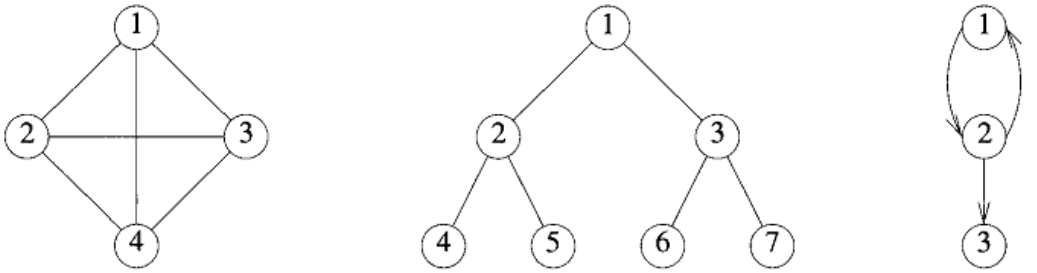
Graphs have been used in a wide variety of applications.

Some of these applications are

- analysis of electric circuits,
- finding shortest routes,
- project planning,
- identification of chemical compounds,
- statistical mechanics,
- genetics, cybernetics, linguistics, social sciences, and so on.

Definitions

- A graph G consists of two sets V and E. The set V is a finite, nonempty set of vertices. The set E is a set of pairs of vertices; these pairs are called edges.
- The notations V(G) and E{G) represent the sets of vertices and edges, respectively, of
- Graph G.
- G = (V, E) to represent a graph.
- In an undirected graph the pair of vertices representing any edge is unordered.
- Thus, the pairs(u,v) and (v, u) represent the same edge.
- In a directed graph each edge is represented by a directed pair (u,v); u is the tail and v the head of the edge.
- Therefore, (v, u) and (u,v) represent two different edges.
- Figure 2.25 shows three graphs: G1, G2, and G3. The graphs G1 and G2 are undirected and G3 is directed.



(a) G_1 (b) G_2 (c) G_3

Figure 2.25 Three sample graphs

The set representations of these graphs are

$$V(G_1) = \{1,2,3,4\} \qquad E(G_1) = \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$$
$$V(G_2) = \{1,2,3,4,5,6,7\} \qquad E(G_2) = \{(1,2),(1,3),(2,4),(2,5),(3,6),(3,7)\}$$
$$V(G_3) = \{1,2,3\} \qquad E(G_3) = \{\langle 1,2\rangle,\langle 2,1\rangle,\langle 2,3\rangle\}$$

Notice that the edges of a directed graph are drawn with an arrow from the tail to the head.

The graph G2 is a tree; the graphs G1 and G3 are not.

Restrictions on graphs:

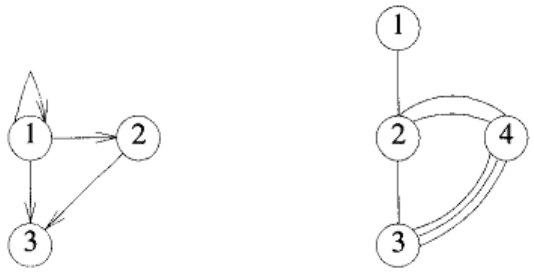
1. A graph may not have an edge from a vertex v back to itself. That is, edges of the form (v, v) and (v,v) are not legal. Such edges are known as self-edges or self-loops.

   If we permit self-edges, we obtain a data object referred to as a graph with self-edges.

   An example is shown in Figure2.26(a).

   2. A graph may not have multiple occurrences of the same edge.

If we remove this restriction, we obtain a data object referred to as a multi-graph (see Figure 2.26(b)



(a) Graph with a self edge (b) Multigraph

Figure 2.26 Examples of graphlike structures

The number of distinct unordered pairs(u, v)with $u \neq v$ u v in a graph with n vertices is $\frac{n(n-1)}{2}$

This is the maximum number of edges in any n-vertex, undirected graph.

An n-vertex, undirected graph with exactly $\frac{n(n-1)}{2}$ edges is said to be complete.

Subgraph

A *subgraph* of G is a graph G' such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. Figure 2.27 shows some of the subgraphs of G_1 and G_3.

A *path* from vertex u to vertex v in graph G is a sequence of vertices $u, i_1, i_2, \ldots, i_k, v$, such that $(u, i_1), (i_1, i_2), \ldots, (i_k, v)$ are edges in $E(G)$. If

. The length of a path is the number of edges on it.

A simple path is a path in which all vertices except possibly the first and last are distinct.

A cycle is a simple path in which the first and last vertices are the same.

The path 1, 2,3,1is a cycle in G1 and 1, 2, 1 is a cycle in G3.

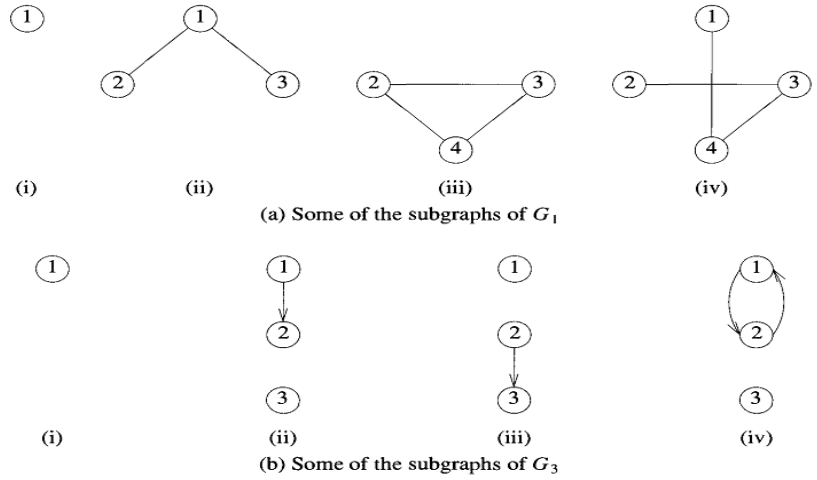


Figure 2.27 Some subgraphs

A connected component (or simply a component) H of an undirected graph is a maximal connected subgraph. By "maximal," we mean that G contains no other subgraph that is both connected and properly contains H.

G4 has two components, H1 and H2 (see Figure2.28).

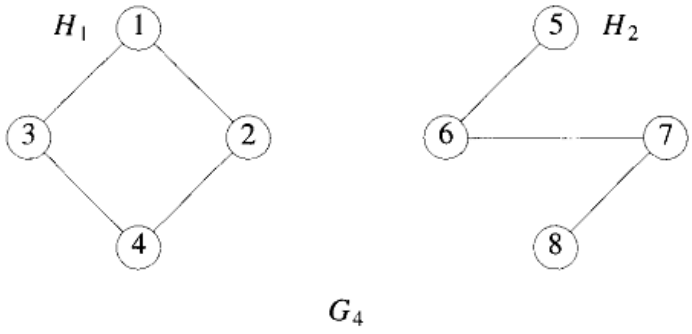


Figure 2.28 A graph with two connected components

A tree is a connected acyclic (i.e., has no cycles) graph.

A directed graph G is said to be strongly connected iff for every pair of distinct vertices u and v in V(G), there is a directed path from u to v and also from v to u.

The graph G3 (repeated in Figure 2.29(a)) is not strongly connected, as there is no path from vertex 3 to 2.

A strongly connected component is a maximal subgraph that is strongly connected.

The graph G3 has two strongly connected components (see Figure 2.29(b))

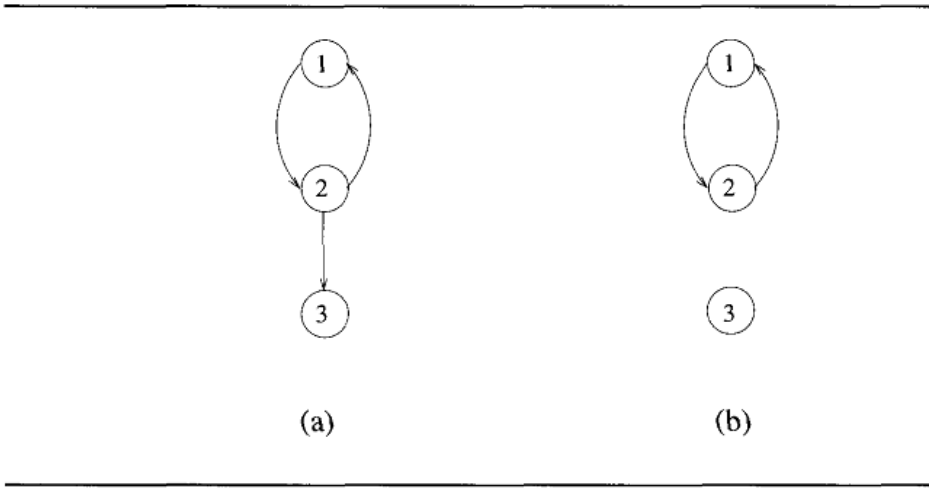


Figure 2.29 A graph and its strongly connected components

Graph Representations

Representations for graphs that are most commonly used:

- Adjacency matrices, adjacency lists, and adjacency multilists.

Adjacency Matrix

Let G = (V,E) be a graph with n vertices, n $\geq$ 1.

- The adjacency matrix of G is a two-dimensional n x n array, say a, with the property that a[i, j] = 1 iff the edge(i,j) [(i,j) for a directed graph) is in E(G).

.

- The element a[i,j] = 0 if there is no such edge in G. The adjacency matrices for the graphs Gi,G3, and G4 are shown in Figure 2.30.
- The adjacency matrix for an undirected graph is symmetric, as the edge(i,j) is in E(G) iff the
- Edge (j, i) is also in E(G).
- The adjacency matrix for a directed graph may not be symmetric (as is the case for G3).
- The space needed to represent a graph using its adjacency matrix is n^2 bits.

Adjacency Lists

- In this representation of graphs, the n rows of the adjacency matrix are represented as n linked lists.
- There is one list for each vertex in G.
- The nodes in list i represent the vertices that are adjacent from vertex i.
- Each node has at least two fields: vertex and link.
- The vertex field contains the indices of the vertices adjacent to vertex i.
- The adjacency lists for G1, G3 and G4 are shown in Figure 2.31.

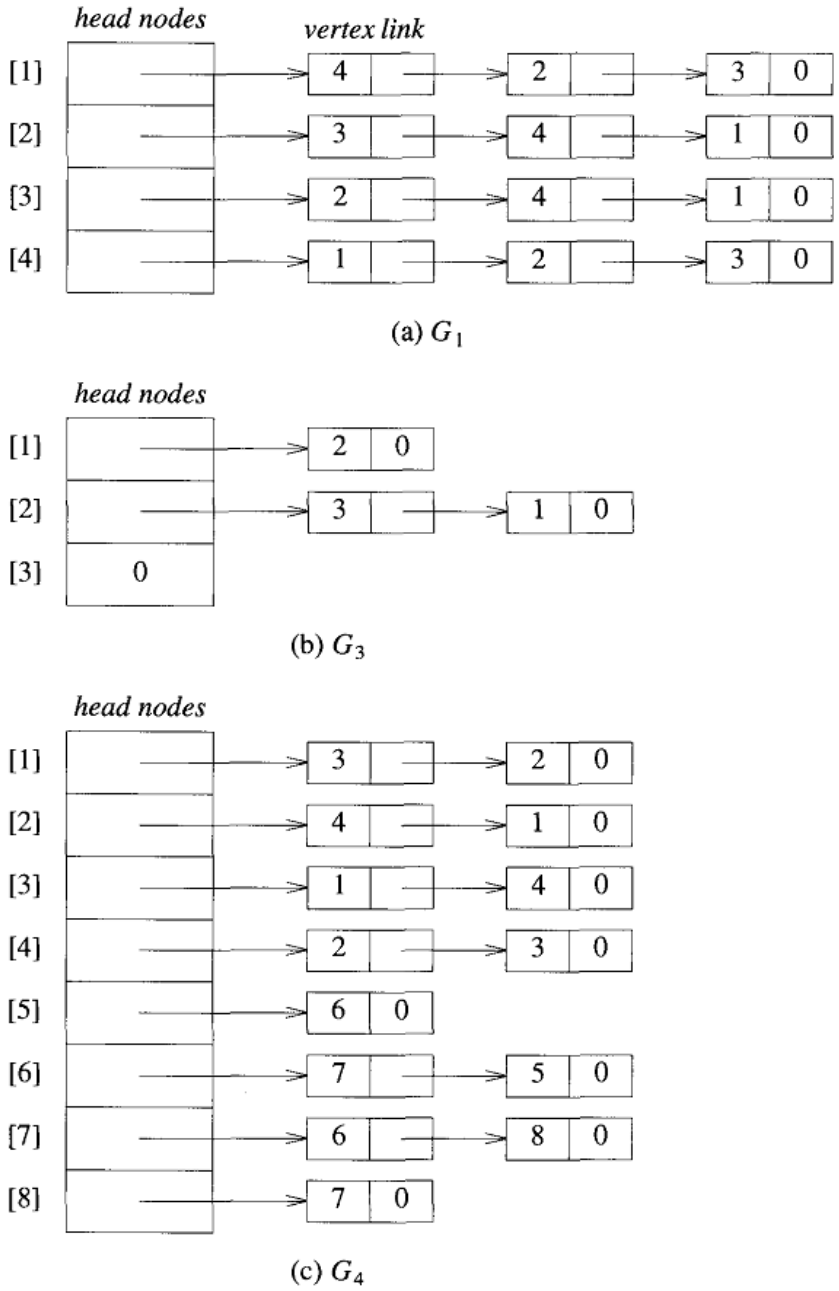


Figure 2.31 Adjacency lists

Adjacency Multilists

In the adjacency-list representation of an undirected graph, each edge (u,v) is represented by two entries, one on the list for u and the other on the list for v.

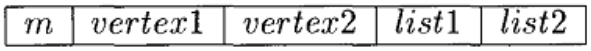
In some applications it is necessary to be able to determine the second entry for a particular edge and mark that edge as having been examined.

This can be accomplished easily if the adjacency lists are maintained as multilists (i.e.,

lists in which nodes can be shared among several lists).

For each edge there is exactly one node, but this node is in two lists (i.e., the adjacency lists for each of the two nodes to which it is incident).

The new node structure is



where m is a one-bit mark field that can be used to indicate whether the edge has been examined. The storage requirements are the same as for normal adjacency lists, except for the addition of the mark bit m.

Figure2.35 shows the adjacency multilists for G1of Figure2.25(a).

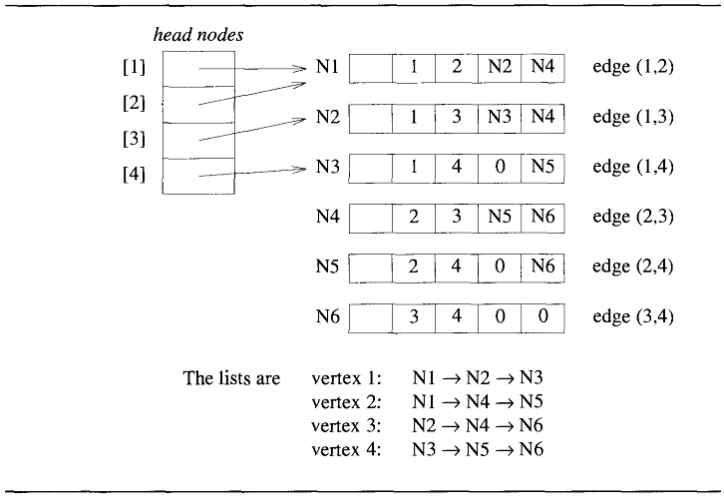


Figure 2.35 Adjacency multilists for G_1 of Figure 2.25(a)

Weighted Edges

- In many applications, the edges of a graph have weights assigned to them.
- These weights may represent the distance from one vertex to another or the cost of going from one vertex to an adjacent vertex.
- When adjacency lists are used, the weight information can be kept in the list nodes by including an additional field, weight.
- A graph with weighted edges is called a network.

THANK YOU

This content is taken from the text books and reference books prescribed in the syllabus.