Graphs

Introduction

- *Graphs* are natural models that are used to represent arbitrary relationships among data objects. We often need to represent such arbitrary relationships among the data objects while dealing with problems in computer science, engineering, and many other disciplines. Therefore, the study of graphs as one of the basic data structures is important.

Basic Definitions and Terminology

- A graph is a structure made of two components: a set of vertices V, and a set of edges E. Therefore, a graph is G = (V,E), where G is a graph. The graph may be directed or undirected. In a *directed graph*, every edge of the graph is an ordered pair of vertices connected by the edge, whereas in an *undirected graph*, every edge is an unordered pair of vertices connected by the edge.

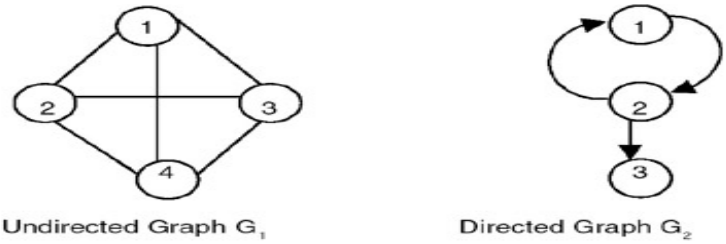

Figure: Graphs.

- **Incident edge:** (v_i,v_j) is an edge, then edge(v_i,v_j) is said to be incident to vertices v_i and v_j. For example, in graph G_1 shown in Figure, the edges incident on vertex 1 are (1,2), (1,4), and (1,3), whereas in G_2, the edges incident on vertex 1 are (1,2).
- **Degree of vertex:** The number of edges incident onto the vertex. For example, in graph G_1, the degree of vertex 1 is 3, because 3 edges are incident onto it. For a directed graph, we need to define indegree and outdegree. *Indegree* of a vertex vi is the number of edges incident onto vi, with vi as the head. *Outdegree* of vertex vi is the number of edges incident onto vi, with vi as the tail. For graph G_2, the indegree of vertex 2 is 1, whereas the outdegree of vertex 2 is 2.
- **Directed edge:** A directed edge between the vertices vi vj is an ordered pair. It is denoted by <vi,vj>.
- **Undirected edge:** An undirected edge between the vertices v_i and v_j is an unordered pair. It is denoted by (v_i,v_j).
- **Path:** A path between vertices v_p and v_q is a sequence of vertices v_p, v_{i1}, v_{i2},…, v_{in},v_q such that there exists a sequence of edges (v_p, v_{i1}), (v_{i1}, v_{i2}), …, (v_{in}, v_q). In case of a directed graph, a path between the vertices v_p and v_q is a sequence of vertices v_p, v_{i1}, v_{i2},…, v_{in}, v_q such that there exists a sequence of edges $<v_p, v_{i1}>$, $< v_{i1}, v_{i2}>$, …, $<v_{in}, v_q>$. If there exists a path from vertex v_p to v_q in an undirected graph, then there always exists a path from v_q to v_p also. But, in the case of a directed graph, if there exists a path from vertex v_p to v_q, then it does not necessarily imply that there exists a path from v_q to v_p also.
- **Simple path:** A simple path is path given by a sequence of vertices in which all vertices are distinct except the first and last vertices. If the first and last vertices are same, the path will be a cycle.
- **Maximum number of edges:** The maximum number of edges in an undirected graph with *n* vertices is $n(n-1)/2$. In a directed graph, it is $n(n-1)$.
- **Subgraph:** A *subgraph* of a graph G = (V,E) is a graph G where V(G) is a subset of V(G). E(G) consists of edges (v1,v2) in E(G), such that both v1 and v2 are in V(G). [Note: If G = (V,E) is a graph, then V(G) is a set of vertices of G and E(G) is a set of edges of G.]
- If E(G) consists of all edges (v1,v2) in E(G), such that both v1 and v2 are in V(G), then G is called an induced subgraph of G.

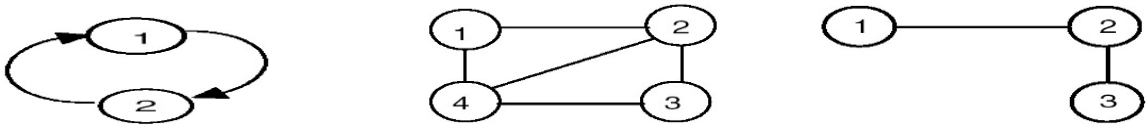

Figure: (a) The subgraph of graph G2. (b) Graph G. (c) Induced subgraph of Graph G.

- In the undirected graph G, the two vertices v_1 and v_2 are said to be connected if there exists a path in G from v_1 to v_2 (being an undirected graph, there exists a path from v_2 to v_1 also).
- **Connected graph:** A graph G is said to be connected if for every pair of distinct vertices (v_i,v_j), there is a path from v_i to v_j. A connected graph is shown in below Figure.

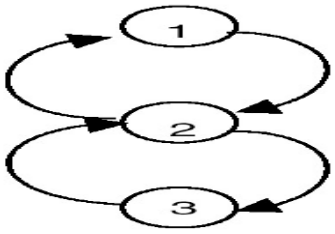

Figure: A connected graph.

- **Completely connected graph:** A graph G is completely connected if, for every pair of distinct vertices (v_i,v_j), there exists an edge. A completely connected graph is shown in below Figure.

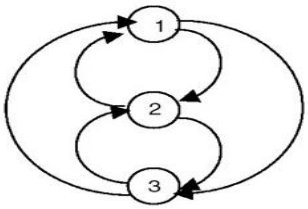

Figure: A completely connected graph.

Representations of graphs

- There are two standard ways to represent a graph $G = (V, E)$: as a collection of adjacency lists or as an adjacency matrix. Either way is applicable to both directed and undirected graphs. The adjacency-list representation is usually preferred, because it provides a compact way to represent ***sparse*** graphs-those for which $|E|$ is much less than $|V|^2$. An adjacency-matrix representation may be preferred, however, when the graph is ***dense***-$|E|$ is close to $|V|^2$-or when we need to be able to tell quickly if there is an edge connecting two given vertices.
- The ***adjacency-list representation*** of a graph $G = (V, E)$ consists of an array Adj of $|V|$ lists, one for each vertex in V . For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices v such that there is an edge $(u, v) \in E$. That is, $Adj[u]$ consists of all the vertices adjacent to u in G. (Alternatively, it may contain pointers to these vertices.) The vertices in each adjacency list are typically stored in an arbitrary order.

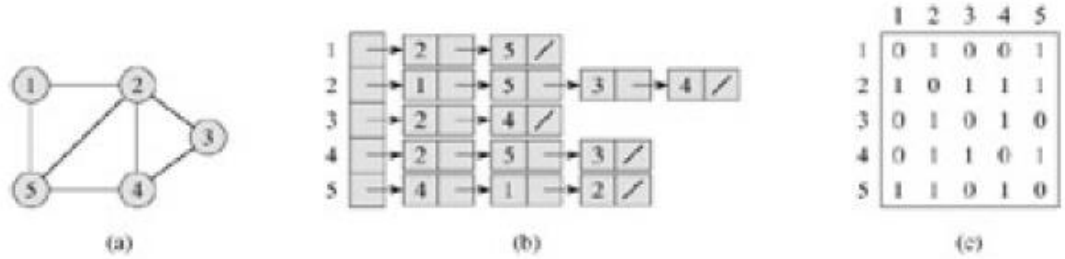

Figure: Two representations of an undirected graph. *(a)* An undirected graph G having five vertices and seven edges. *(b)* An adjacency-list representation of G. *(c)* The adjacency-matrix representation of G.

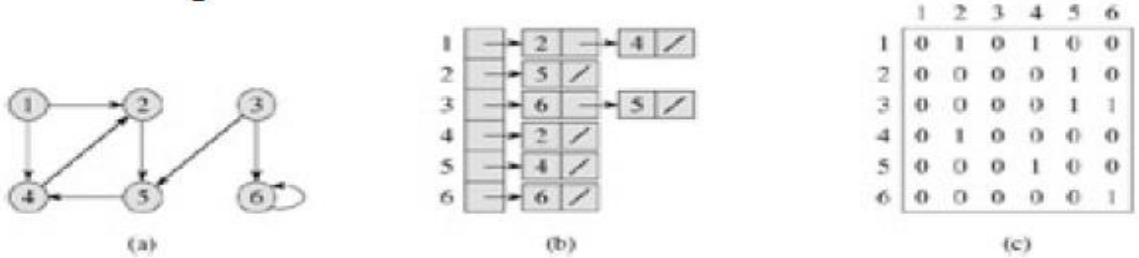

Figure: Two representations of a directed graph. *(a)* A directed graph G having six vertices and eight edges. *(b)* An adjacency-list representation of G. *(c)* The adjacency-matrix representation of G.

- If G is a directed graph, the sum of the lengths of all the adjacency lists is $|E|$, since an edge of the form (u, v) is represented by having v appear in $Adj[u]$. If G is an undirected graph, the sum of the lengths of all the adjacency lists is $2|E|$, since if (u, v) is an undirected edge, then u appears in v's adjacency list and vice versa. For both directed and undirected graphs, the adjacency-list representation has the desirable property that the amount of memory it requires is $\Theta(V + E)$.
- Adjacency lists can readily be adapted to represent ***weighted graphs***, that is, graphs for which each edge has an associated ***weight***, typically given by a ***weight function*** $w : E \rightarrow \mathbf{R}$. For example, let $G = (V, E)$ be a weighted graph with weight function w. The weight $w(u, v)$ of the edge $(u, v) \in E$ is simply stored with vertex v in u's adjacency list. The adjacency-list representation is quite robust in that it can be modified to support many other graph variants.
- A potential disadvantage of the adjacency-list representation is that there is no quicker way to determine if a given edge (u, v) is present in the graph than to search for v in the adjacency list $Adj[u]$. This disadvantage can be remedied by an adjacency-matrix representation of the graph, at the cost of using asymptotically more memory.
- For the ***adjacency-matrix representation*** of a graph $G = (V, E)$, we assume that the vertices are numbered 1, 2,..., $|V|$ in some arbitrary manner. Then the adjacency-matrix representation of a graph G consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E , \\ 0 & \text{otherwise} . \end{cases}$$

- The adjacency matrix of a graph requires $\Theta(V^2)$ memory, independent of the number of edges in the graph.
- Observe the symmetry along the main diagonal of the adjacency matrix in above Figure. We define the ***transpose*** of a matrix $A = (a_{ij})$ to be the matrix $A^T = (a_{ij}^T)$ given by $a_{ij}^T = a_{ji}$. Since in an undirected graph, (u, v) and (v, u) represent the same edge, the adjacency matrix A of an undirected graph is its own transpose: $A = A^T$. In some applications, it pays to store only the entries on and above the diagonal of the adjacency matrix, thereby cutting the memory needed to store the graph almost in half.
- Like the adjacency-list representation of a graph, the adjacency-matrix representation can be used for weighted graphs. For example, if $G = (V, E)$ is a weighted graph with edge-weight function w, the weight $w(u, v)$ of the edge $(u, v) \in E$ is simply stored as the entry in row u and column v of the adjacency matrix. If an edge does not exist, a NIL value can be stored as its corresponding matrix entry, though for many problems it is convenient to use a value such as 0 or ∞.
- Although the adjacency-list representation is asymptotically at least as efficient as the adjacency-matrix representation, the simplicity of an adjacency matrix may make it preferable when graphs are reasonably small. Moreover, if the graph is unweighted, there is an additional advantage in storage for the adjacency-matrix representation. Rather than using one word of computer memory for each matrix entry, the adjacency matrix uses only one bit per entry.