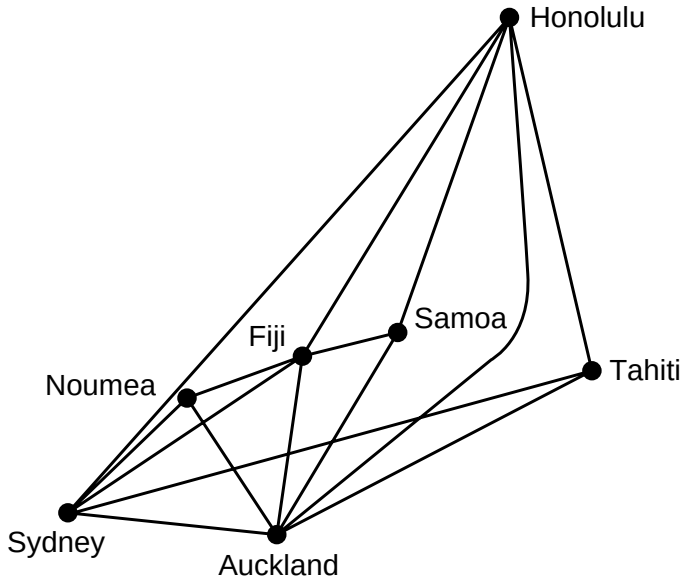
Chapter 11

GRAPHS

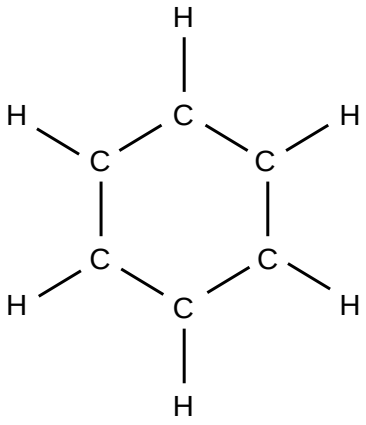
1. Mathematical Background
2. Computer Representation
3. Graph Traversal
4. Topological Sorting
5. A Greedy Algorithm: Shortest Paths
6. Graphs as Data Structures

Graphs: Definitions

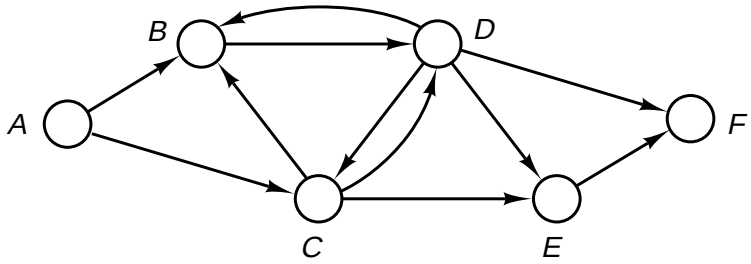
1. A ***graph*** G consists of a set V, whose members are called the ***vertices*** of G, together with a set E of pairs of distinct vertices from V.
2. The pairs in E are called the ***edges*** of G.
3. If $e = (v, w)$ is an edge with vertices v and w, then v and w are said to ***lie on*** e, and e is said to be ***incident*** with v and w.
4. If the pairs are unordered, G is called an ***undirected graph***.
5. If the pairs are ordered, G is called a ***directed graph***. The term *directed graph* is often shortened to ***digraph***, and the unqualified term *graph* usually means *undirected graph*.
6. Two vertices in an undirected graph are called ***adjacent*** if there is an edge from the first to the second.
7. A ***path*** is a sequence of distinct vertices, each adjacent to the next.
8. A ***cycle*** is a path containing at least three vertices such that the last vertex on the path is adjacent to the first.
9. A graph is called ***connected*** if there is a path from any vertex to any other vertex.
10. A ***free tree*** is defined as a connected undirected graph with no cycles.
11. In a directed graph a path or a cycle means always moving in the direction indicated by the arrows. Such a path (cycle) is called a ***directed*** path (cycle).
12. A directed graph is called ***strongly connected*** if there is a directed path from any vertex to any other vertex. If we suppress the direction of the edges and the resulting undirected graph is connected, we call the directed graph ***weakly connected***.
13. The ***valence*** of a vertex is the number of edges on which it lies, hence also the number of vertices adjacent to it.



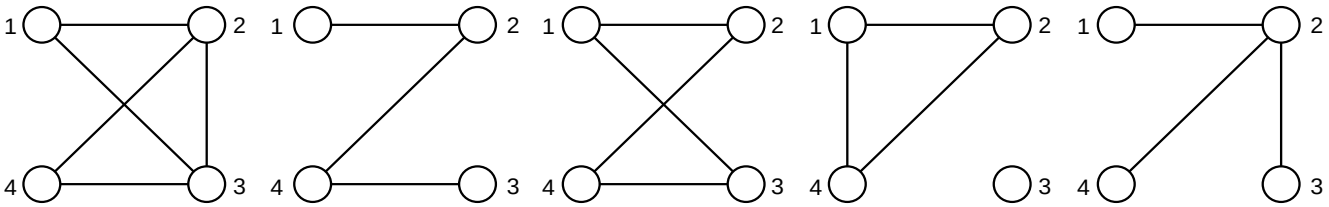
Selected South Pacific air routes



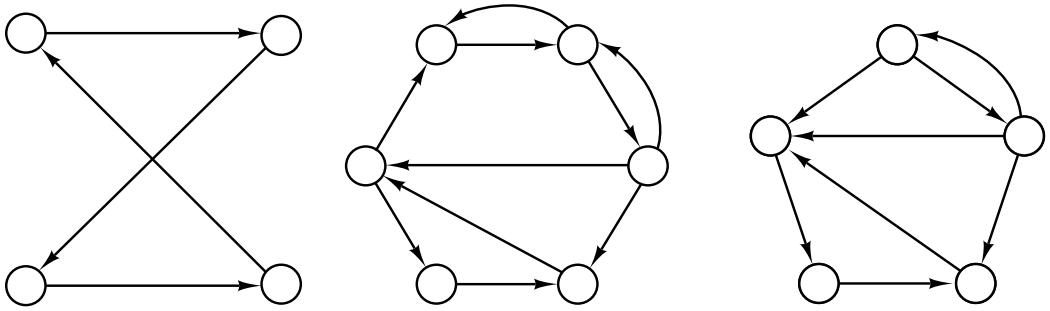
Benzene molecule



Message transmission in a network



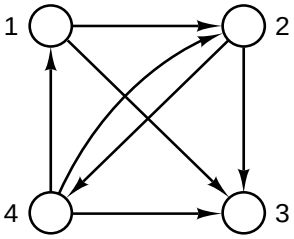
Connected (a) | Path (b) | Cycle (c) | Disconnected (d) | Tree (e)



Directed cycle (a) | Strongly connected (b) | Weakly connected (c)

DEFINITION A ***graph*** G consists of a set V, called the ***vertices*** of G, and, for all $v \in V$, a subset A_v of V, called the set of vertices ***adjacent*** to v.

Directed graph

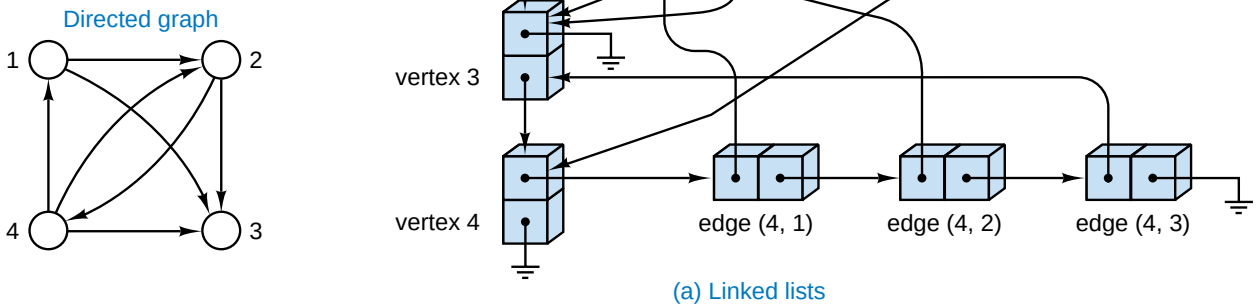


Adjacency sets

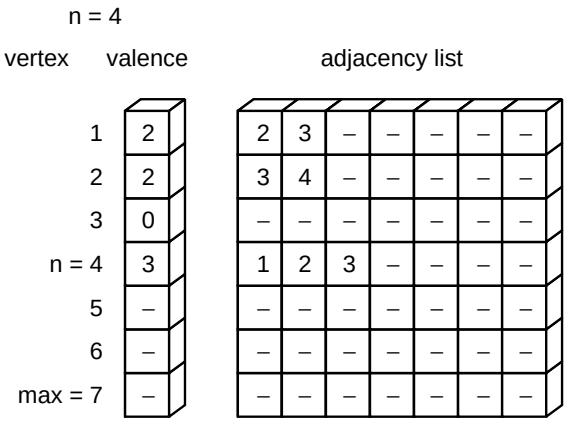
vertex	Set
1	{2, 3}
2	{3, 4}
3	ϕ
4	{1, 2, 3}

Adjacency table

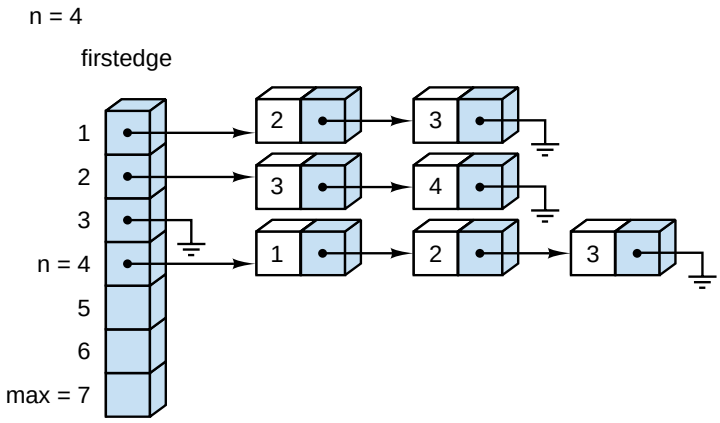
	1	2	3	4
1	F	T	T	F
2	F	F	T	T
3	F	F	F	F
4	T	T	T	F



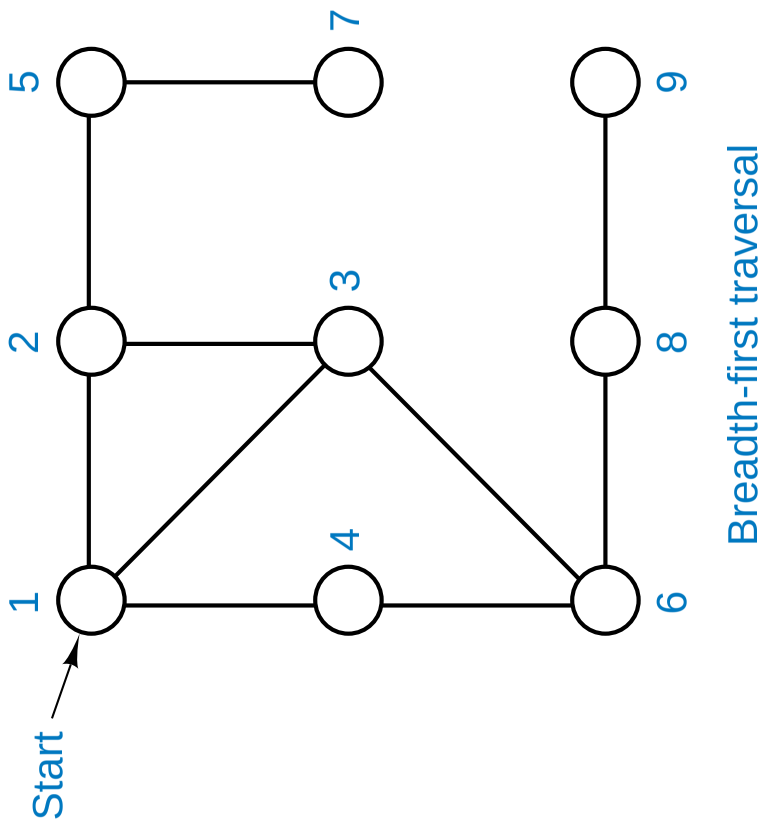
(a) Linked lists

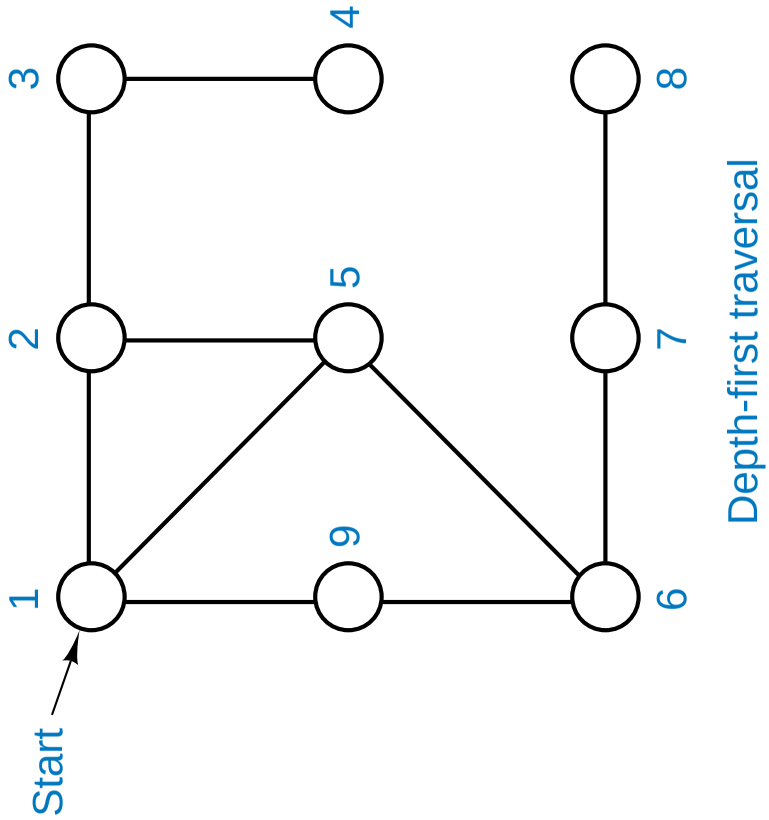


(b) Contiguous lists



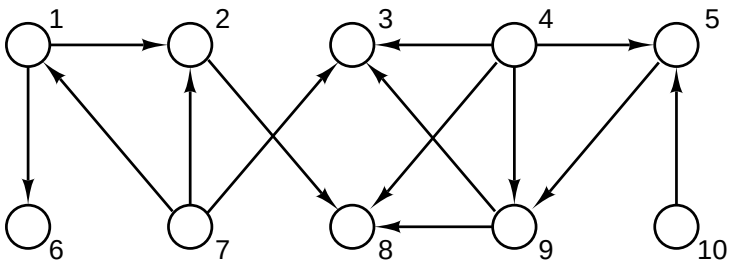
(c) Mixed



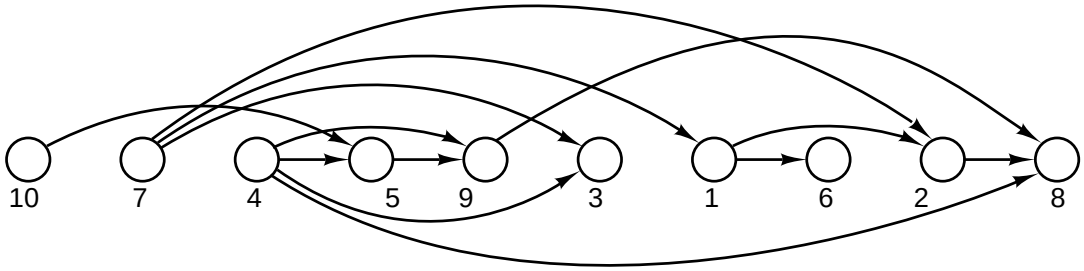


Topological Sorting

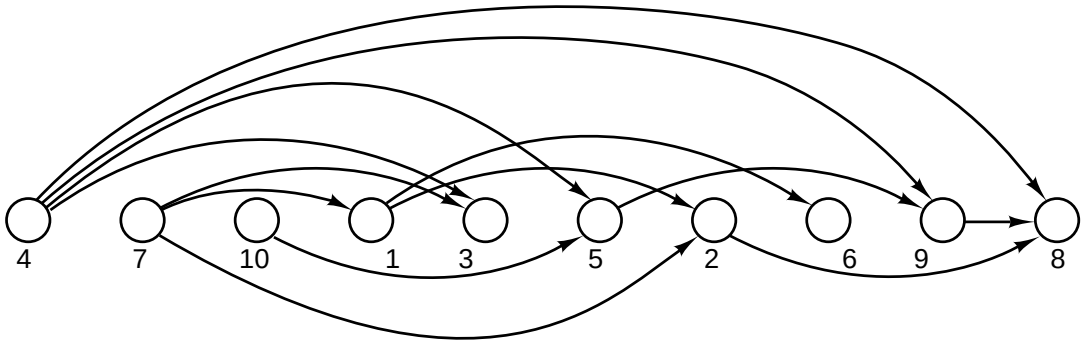
Let G be a directed graph with no cycles. A ***topological order*** for G is a sequential listing of all the vertices in G such that, for all vertices $v, w \in G$, if there is an edge from v to w, then v precedes w in the sequential listing.



Directed graph with no directed cycles



Depth-first ordering

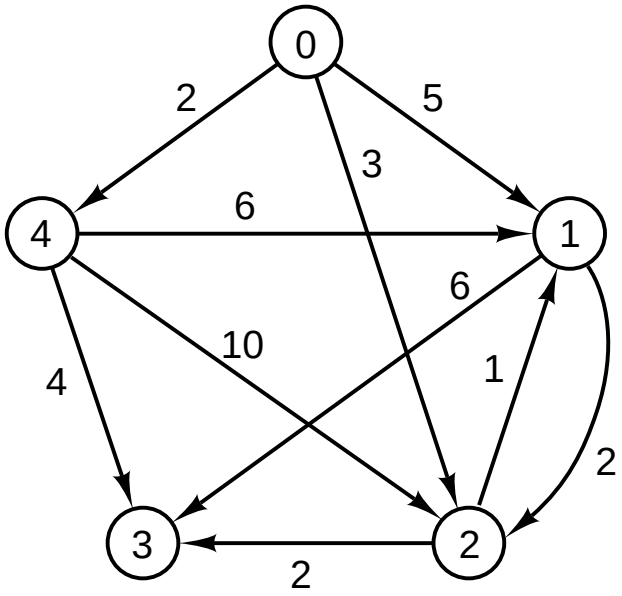


Breadth-first ordering

A Greedy Algorithm: Shortest Paths

The problem of shortest paths:

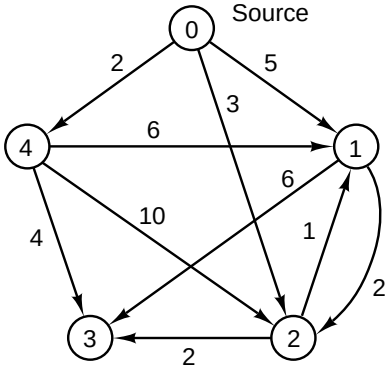
Given a directed graph in which each edge has a nonnegative ***weight*** or cost, find a path of least total weight from a given vertex, called the ***source***, to every other vertex in the graph.



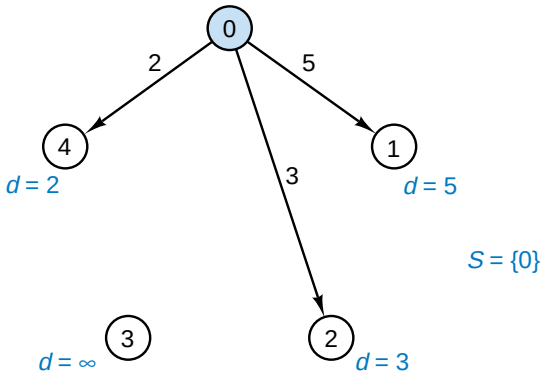
Method:

We keep a set S of vertices whose closest distances to the source, vertex 0, are known and add one vertex to S at each stage. We maintain a table D that gives, for each vertex v, the distance from 0 to v along a path all of whose vertices are in S, except possibly the last one. To determine what vertex to add to S at each step, we apply the ***greedy*** criterion of choosing the vertex v with the smallest distance recorded in the table D, such that v is not already in S.

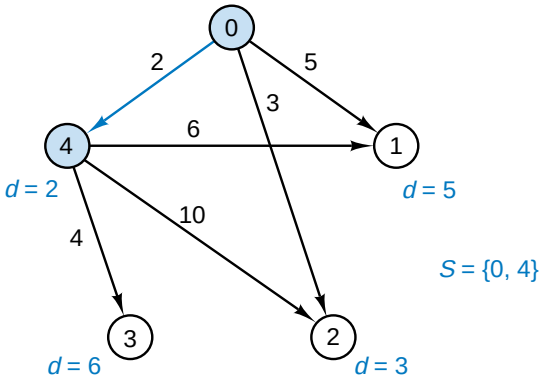
Example of Shortest Path



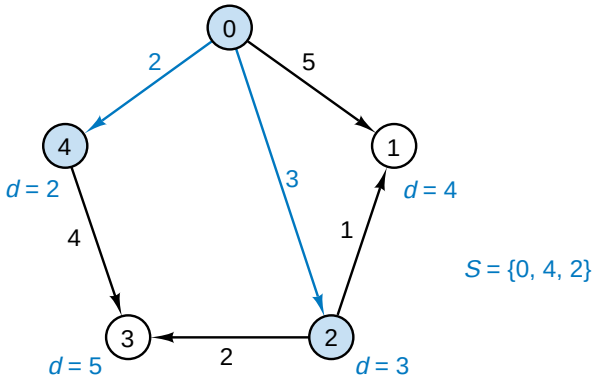
(a)



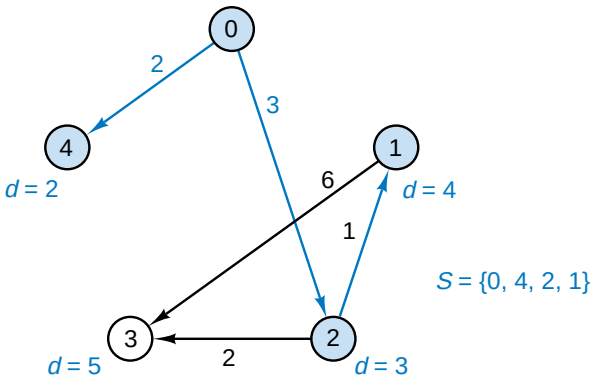
(b)



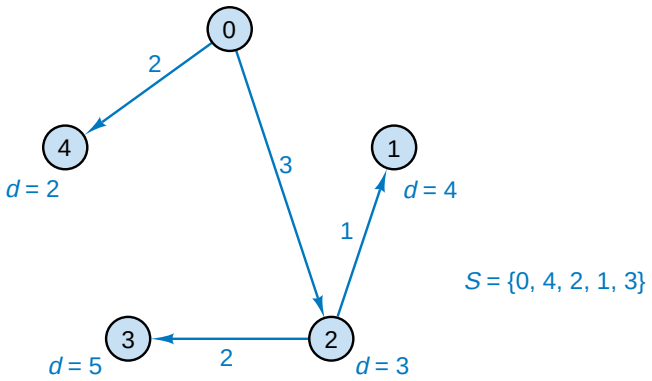
(c)



(d)



(e)



(f)

Pointers and Pitfalls

1. Graphs provide an excellent way to describe the essential features of many applications, thereby facilitating specification of the underlying problems and formulation of algorithms for their solution. Graphs sometimes appear as data structures but more often as mathematical abstractions useful for problem solving.

2. Graphs may be implemented in many ways by the use of different kinds of data structures. Postpone implementation decisions until the applications of graphs in the problem-solving and algorithm-development phases are well understood.

3. Many applications require graph traversal. Let the application determine the traversal method: depth first, breadth first, or some other order. Depth-first traversal is naturally recursive (or can use a stack). Breadth-first traversal normally uses a queue.

4. Greedy algorithms represent only a sample of the many paradigms useful in developing graph algorithms. For further methods and examples, consult the references.