3.5 Depth-First Search and Breadth-First Search

The term "exhaustive search" can also be applied to two very important algorithms that systematically process all vertices and edges of a graph. These two traversal algorithms are *depth-first search* (*DFS*) and *breadth-first search* (*BFS*). These algorithms have proved to be very useful for many applications involving graphs in artificial intelligence and operations research. In addition, they are indispensable for efficient investigation of fundamental properties of graphs such as connectivity and cycle presence.

Depth-First Search

It is also very useful to accompany a depth-first search traversal by constructing the so-called *depth-first search forest*. The starting vertex of the traversal serves as the root of the first tree in such a forest.Whenever a new unvisited vertex is reached for the first time, it is attached as a child to the vertex from which it is being reached. Such an edge is called a *tree edge* because the set of all such edges forms a forest. The algorithm may also encounter an edge leading to a previously visited vertex other than its immediate predecessor (i.e., its parent in the tree).

Such an edge is called a *back edge* because it connects a vertex to its ancestor, other than the parent, in the depth-first search forest. Figure 3.10 provides an example of a depth-first search traversal, with the traversal stack and corresponding depth-first search forest shown as well.

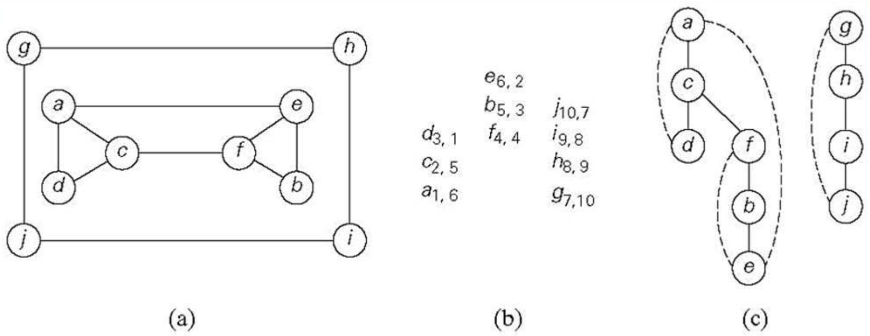

FIGURE 3.10 Example of a DFS traversal. (a) Graph. (b) Traversal's stack (the first subscript number indicates the order in which a vertex is visited, i.e., pushed onto the stack; the second one indicates the order in which it becomes a dead-end, i.e., popped off the stack). (c) DFS forest with the tree and back edges shown with solid and dashed lines, respectively.

ALGORITHM *DFS*(*G*)

//Implements a depth-first search traversal of a given graph

//Input: Graph *G* = *V, E*

//Output: Graph *G* with its vertices marked with consecutive integers

// in the order they are first encountered by the DFS traversal

mark each vertex in *V* with 0 as a mark of being "unvisited"

count ←0

for each vertex *v* in *V* do

if *v* is marked with 0

dfs(*v*)

dfs(*v*)

//visits recursively all the unvisited vertices connected to vertex *v*

//by a path and numbers them in the order they are encountered

//via global variable *count*

count ←*count* + 1; mark *v* with *count*

for each vertex *w* in *V* adjacent to *v* do

if *w* is marked with 0

dfs(*w*)

Breadth-First Search

Similarly to a DFS traversal, it is useful to accompany a BFS traversal by constructing the so-called *breadth-first search forest*. The traversal's starting vertex serves as the root of the first tree in such a forest.Whenever a new unvisited vertex is reached for the first time, the vertex is attached as a child to the vertex it is being reached from with an edge called a *tree edge*. If an edge leading to a previously visited vertex other than its immediate predecessor (i.e., its parent in the tree) is encountered, the edge is noted as a *cross edge*. Figure 3.11 provides an example of a breadth-first search traversal, with the traversal queue and Corresponding breadth-first search forest shown.

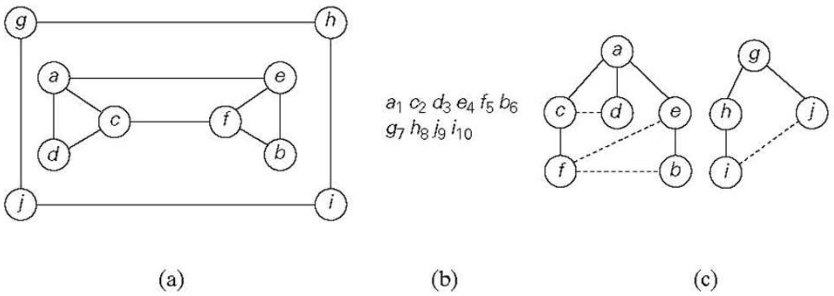

FIGURE 3.11 Example of a BFS traversal. (a) Graph. (b) Traversal queue, with the numbers indicating the order in which the vertices are visited, i.e., added to (and removed from) the queue. (c) BFS forest with the tree and cross edges shown with solid and dotted lines, respectively.

ALGORITHM *BFS*(*G*)

//Implements a breadth-first search traversal of a given graph

//Input: Graph $G = V, E$

//Output: Graph G with its vertices marked with consecutive integers

// in the order they are visited by the BFS traversal

mark each vertex in V with 0 as a mark of being "unvisited"

count ←0

for each vertex v in V do

    if v is marked with 0

        bfs(v)

bfs(v)

//visits all the unvisited vertices connected to vertex v

//by a path and numbers them in the order they are visited

//via global variable *count*

count ←*count* + 1; mark v with *count* and initialize a queue with v

while the queue is not empty do

    for each vertex w in V adjacent to the front vertex do

        if w is marked with 0

            count ←*count* + 1; mark w with *count*

            add w to the queue

    remove the front vertex from the queue

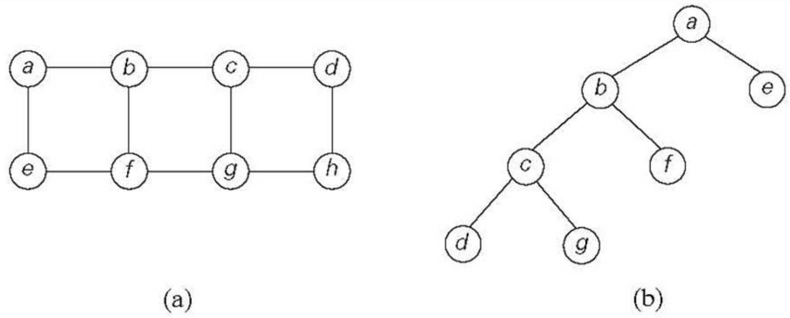

FIGURE 3.12 Illustration of the BFS-based algorithm for finding a minimum-edge path. (a) Graph. (b) Part of its BFS tree that identifies the minimum-edge path from a to g.

TABLE 3.1 Main facts about depth-first search (DFS) and breadth-first search (BFS)

	DFS	**BFS**
Data structure	a stack	a queue
Number of vertex orderings	two orderings	one ordering
Edge types (undirected graphs)	tree and back edges	tree and cross edges
Applications	connectivity, acyclicity, articulation points	connectivity, acyclicity, minimum-edge paths
Efficiency for adjacency matrix	$\Theta(\|V^2\|)$	$\Theta(\|V^2\|)$
Efficiency for adjacency lists	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\| + \|E\|)$