Elementary Graph Algorithms

Graphs

Definition

A graph \( G(V, E) \) is a collection \( V \) of vertices and \( E \) of edges.

Vertices are simple objects that can have names and other properties; an edge is a connection between two vertices.
Graph Terminology

• A graph may have many representations

![Figure 1. A graph](image1)

![Figure 2. The same graph?](image2)

• Graph terminology: path, simple path, cycle, connected, connected component, tree, forest, spanning tree.

An Old Friend

Definition

A **tree** is a graph in which there is exactly one path connecting any two nodes.

![Tree](image3)

Defining Property

A **tree** is a graph with n nodes and n-1 edges.
### Adjacency-Matrix of $G(V,E)$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>M</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Advantages and Disadvantages

- Adjacency-matrices require $O(1)$ steps to determine whether $(u,v) \in E$.
- However, the representation requires $O(V^2)$ bits of storage and $O(V^2)$ steps to initialize. This can be a problem if the graph is not dense.
Adjacency List of $G(V,E)$

- Some simple operations are not easily supported by the adjacency-list representation. For example, how do we delete a node and all the edges connected to it?
Breadth-First Search

BFS(G, s)

for u in G.V
    u.color = WHITE
    u.d = ∞
    u.parent = NIL
s.color = GRAY
s.d = 0
s.parent = NIL
Q = Enqueue(Empty-Queue, s)
while Q ≠ Ø
    u = Dequeue(Q)
    for v in G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.parent = u
            Enqueue(Q, v)
    u.color = BLACK

Breadth-First Search in Action: BFS(Gr, A)

Graph

Contents of queue during search
Your Turn: BFS(Gr, E)

Graph

Contents of queue during search

Picking Up a Graph to Explore It
**Shortest Path**

**Definition**

The shortest-path distance \( \delta(s,v) \) from \( s \) to \( v \) is defined to be the minimum number of edges in any path from vertex \( s \) to vertex \( v \).

**Correctness of Breadth-First Search**

**Theorem 22.5**

BFS discovers every vertex \( v \in V \) that is reachable from the source \( s \), and upon termination, \( v.d = \delta(s,v) \).
Depth-First Search

- Using a stack rather than a queue as the data structure to hold vertices yields a much different search order.

G(V, E)

Contents of stack during search

Depth-First Forest

Depth-First Forest of G
Ancient Arabic Text*

“To find the way out of a labyrinth there is only one means. At every new junction, never seen before, the path we have taken will be marked with three signs. If … you see that the junction has already been visited, you will make only one mark on the path you have taken. If all the apertures have already been marked, then you must retrace your steps. But if one or two apertures of the junction are still without signs, you will choose any one, making two signs on it. Proceeding through an aperture that bears only one sign, you will make two more, so that now the aperture bears three. All the parts of the labyrinth must have been visited if, arriving at a junction, you never take a passage with three signs, unless none of the other passages is now without signs.”

*Depth-first search was first stated formally hundreds of years ago as a method for traversing mazes.