Exercise 1: Check for Reachability Between Two Vertices in a Directed Graph

Given a directed graph represented using the `Digraph` class, write a function to check whether there is a path from a specified source vertex to a target vertex.

Input:

- A directed graph represented as an instance of the `Digraph` class.
- Two integers, `source` and `target`, which represent the vertices.

Output:

- Print `true` if there is a path from `source` to `target`, otherwise print `false`.

Algorithm: You should use Breadth-First Search (BFS), which is a systematic way of exploring the graph:

1. Start at the `source` vertex.
2. Explore all vertices directly connected to the `source`.
3. Repeat for their neighbors, level by level, until:
   - You reach the `target` vertex (path found, return `true`).
   - Or all reachable vertices have been explored without finding the `target` (return `false`).

Exercise 2: Cycle Detection in a Directed Graph

A cycle in a directed graph occurs when a vertex is revisited during a depth-first traversal before the traversal completes. The goal of this exercise is to determine whether a given directed graph contains any cycles.

Task: Implement a function `bool hasCycle(const Digraph& graph)` that takes a directed graph as input and returns:

- `true` if the graph contains at least one cycle.
- `false` if the graph is acyclic.

Algorithm:

1. Start DFS at any unvisited vertex.
2. For each vertex:
   - Mark it as visited and on stack.
   - Explore all neighbors:
     - If a neighbor is not visited, recursively perform DFS.
     - If a neighbor is already on the stack, a cycle exists.
   - Remove the vertex from the stack after all its neighbors are processed.
3. Repeat for all vertices.
4. If no cycles are found, the graph is acyclic.