Daily Coding Problem #280

Problem

This problem was asked by Pandora.

Given an undirected graph, determine if it contains a cycle.

Solution

One way to think about this problem is as follows: suppose we are traversing the graph's edges, starting from a given vertex. If, for some vertex, we find that one of its neighbors has already been marked, then we know that there are two ways to reach that neighbor from the same starting point, which indicates a cycle.

We can implement this solution using depth-first search. For each vertex in the graph, if it has not already been visited, we call our search function on it. This function will recursively traverse unvisited neighbors of the vertex, and return `True` if we come across the situation described above.

If we are able to visit all vertices without finding duplicate paths, we return `False`.

```
def search(graph, vertex, visited, parent):
    visited[vertex] = True

    for neighbor in graph[vertex]:
        if not visited[neighbor]:
            if search(graph, neighbor, visited, vertex):
                return True
```

```
            elif parent != neighbor:
                return True

        return False

    def has_cycle(graph):
        visited = {v: False for v in graph.keys()}

        for vertex in graph.keys():
            if not visited[vertex]:
                if search(graph, vertex, visited, None):
                    return True

        return False
```

The time complexity of this solution will be $O(V + E)$, since in the worst case we will have to traverse all edges of the graph. Our search will take $O(V)$ space in the worst case to store the vertices in a given traversal on the stack.

© Daily Coding Problem 2019

Privacy Policy

Terms of Service

Press