Bărnuț Cristiana Florina Diana
groupe 911

# Breadth First Search for undirected graph

These functions implement a breadth-first search (BFS) algorithm to find the connected components of an undirected graph. Here's a concise description of how it works:

The function **accessible_nodes()** utilize BFS to find all vertices accessible from a given starting vertex. By maintaining a set to track accessible vertices and a list to manage unvisited ones it systematically identifies and adds reachable vertices to the set until all are visited.

Iterating through vertices, the function **connected_components()** identify connected components by assigning vertices to components if not already included. By employing `accessible_nodes()` to find reachable vertices from each unassigned vertex, it constructs distinct connected components until all vertices are accounted for.

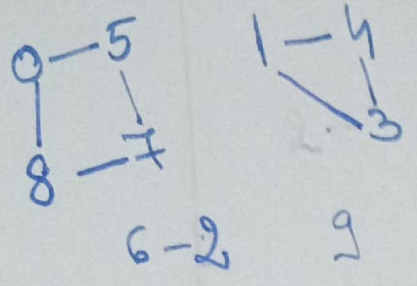
```python
def accessible_nodes(self, node):
    """
    Breadth first search for an undirected graph
    Finding all the vertices accessible from the given node
    :param node: the start vertex
    :return: the accessible vertices
    """
    # create the set and the list to save the accessible vertices and to keep track on all unvisited vertices

    acc = set()
    # we add the first vertex in the accessible list
    acc.add(node)
    list_acc = [node]
    while len(list_acc) > 0:
        # we select every vertex in the tracking list in the order is added
        vertex1 = list_acc[0]
        # eliminate the vertex from the list
        list_acc = list_acc[1:]
        # start looking for all the vertices accessible from the current one
        for vertex2 in self.parse_vertex_out(vertex1):
            # if the vertex is not already in the accessible list we add it, else we continue with the next
            # accessible vertex
            if vertex2 not in acc:
                acc.add(vertex2)
                list_acc.append(vertex2)
    # return the accessible vertices as a set
    return acc
```

```python
def connected_components(self):
    """
    Search for the connected components of the graph
    :return: A dictionary containing the connected components
    """
    # we create a dictionary containing the as a key the number of component and as value the
    # connected component
    components = {}
    count = 0
    # take the all vertices in the graph
    for vertex in self.parse_vertices():
        there = False
        # search if the vertex is already in a component
        for component in components.keys():
            if vertex in components[component]:
                there = True
        # if the vertex is not in the components dictionary
        if not there:
            # search for all the accessing components for that vertex
            acc = self.accessible_nodes(vertex)
            count += 1
            # add the new component in the dictionary
            components[count] = acc
    # return the component dictionary
    return components
```

Lab 2

# Find the connected components of an undirected graph using BFS.

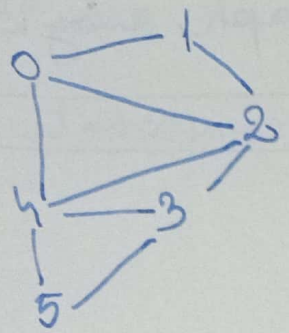



Nout dictionary:

| Key | value |
|---|---|
| 0 | [5,8] |
| 1 | [3,4] |
| 2 | [6] |
| 3 | [1,4] |
| 4 | [1,3] |
| 5 | [0,7] |
| 6 | [2] |
| 7 | [5,8] |
| 8 | [0,7] |
| 9 | [] |

connected_components()
1: 0-5-8-7
2: 1-3-4
3: 2-6
4: 9

| call | vertex1 | vertex2 | acc | list_acc | component |
|---|---|---|---|---|---|
| accessible_nodes(0) | 0 | 5 | {0} | [0] → [ ] | |
| | | 8 | {0,5} | [5] | |
| | 5 | 0 | {0,5,8} | [5,8] | |
| | | 7 | {0,5,8} | [8] | |
| | 8 | 0 | {0,5,8,7} | [8,7] | |
| | | 7 | - " - | [7] | |
| | 7 | 5 | - " - | [7] | |
| | | 8 | - " - | [ ] | |
| | | | {0,5,8,7} | [ ] | connected comp: 0→5→8→7 |
| accessible_nodes(1) | 1 | 3 | {1} | [1] → [ ] | |
| | | 4 | {1,3} | [3] | |
| | 3 | 1 | {1,3,4} | [3,4] | |
| | | 4 | {1,3,4} | [4] | |
| | 4 | 1 | {1,3,4} | [4] | |
| | | 3 | {1,3,4} | [ ] | |
| | | | {1,3,4} | [ ] | connected comp: 1→3→4 |
| accessible_nodes(2) | 2 | 6 | {2} | [2] → [ ] | |
| | 6 | 2 | {2,6} | [6] | |
| | | | {2,6} | [ ] | connected comp: 2→6 |
| accessible_nodes(9) | 9 | | {9} | [9] → [ ] | connected_comp: 9 |




Neat dictionary:

| Key | Value |
|---|---|
| 0 | - [1, 2, 4] |
| 1 | - [0, 2] |
| 2 | - [0, 1, 3, 4] |
| 3 | - [2, 4, 5] |
| 4 | - [0, 2, 3, 5] |
| 5 | - [3, 4] |

| call | vertex1 | vertex2 | acc | list_acc | component |
|---|---|---|---|---|---|
| accessible_nodes(0) | 0 | | {0} | [0] -> [] | |
| | | 1 | {0, 1} | [1] | |
| | | 2 | {0, 1, 2} | [1, 2] | |
| | | 4 | {0, 1, 2, 4} | [1, 2, 4] | |
| | 1 | 0 | {0, 1, 2, 4} | [2, 4] | |
| | | 2 | {0, 1, 2, 4} | [2, 4] | |
| | 2 | 0 | {0, 1, 2, 4} | [4] | |
| | | 1 | {0, 1, 2, 4} | [4] | |
| | | 3 | {0, 1, 2, 4, 3} | [4, 3] | |
| | | 4 | {0, 1, 2, 4, 3} | [4, 3] | |
| | 4 | 0 | {0, 1, 2, 4, 3} | [3] | |
| | | 2 | {0, 1, 2, 4, 3} | [3] | |
| | | 3 | {0, 1, 2, 4, 3} | [3] | |
| | | 5 | {0, 1, 2, 4, 3, 5} | [3, 5] | |
| | 3 | 2 | {0, 1, 2, 4, 3, 5} | [5] | |
| | | 4 | {0, 1, 2, 4, 3, 5} | [5] | |
| | | 5 | {0, 1, 2, 4, 3, 5} | [5] | |
| | 5 | 3 | {0, 1, 2, 4, 3, 5} | [] | connected-comp: |
| | | 4 | {0, 1, 2, 4, 3, 5} | [] | 0-1-2-4-3-5 |
| connected_components() | | | | | 1: 0-1-2-4-3-5 |