Connected Components

A connected component of an undirected graph is a maximal set of vertices such that each pair of vertices is connected by a path.
Connected Components

What are the connected components of the graph below?
Connected Components

What are the connected components of the graph below?

The graph has 3 connected components:
{0,1,2,3,6,8}, {4,5}, and {7}
Note that every vertex belongs to exactly 1 connected component
Connected Components

How to find connected components?
We can find all connected components by repeatedly applying a search algorithm

cc = []
while there are unvisited vertices:
    Let v be the first unvisited vertex
    cc.append(set of vertices visited by bfs(v))
Connected Components

How to find connected components

A more efficient algorithm uses the disjoint set forest.

The disjoint set forest allows also to keep track of the connected components in dynamic graphs where edges might be added (but not removed)
Disjoint Set Forest

A disjoint set forest is a set of trees where each non-root node contains a reference to its PARENT.
Disjoint Set Forest

A disjoint set forest can be represented by an array of integers as follows:

- `parent[i] = -1` if `i` is a root
- `parent[i] =` the parent node of `i` in the dsf

Thus the array representing the DSF forest above would be:

```
parent = [-1, 0, 4, 0, -1, 3]
```
Disjoint Set Forest

Each connected component will be represented by a tree in the disjoint set forest
Initially, every vertex is a root, so we have as many tree as we have vertices
Disjoint Set Forest and Connected Components

Each connected component will be represented by a tree in the disjoint set forest.

The for every edge \((u,v)\) in the graph, if \(u\) and \(v\) belong to different trees, we join the trees by making the root of \(v\)’s tree point to the root of \(u\)’s tree.

Forest after processing edge \((0,1)\)
class DSF:
    # Constructor
    def __init__(self, sets):
        # Creates forest with 'sets' root nodes
        self.parent = np.zeros(sets, dtype=int) - 1

    def find(self, i):
        # Returns root of tree that i belongs to
        if self.parent[i] < 0:
            return i
        return self.find(self.parent[i])

    def union(self, i, j):
        # Makes root of j's tree point to root of i's tree if they are different
        # Return 1 if a parent reference was changed, 0 otherwise
        root_i = self.find(i)
        root_j = self.find(j)
        if root_i != root_j:
            self.parent[root_j] = root_i
            return 1
        return 0
def connected_components(g):
    vertices = len(g.al)
    components = vertices
    s = dsf.DSF(vertices)
    for v in range(vertices):
        for edge in g.al[v]:
            components -= s.union(v, edge.dest)
    return components, s
After processing all edges, each tree contains the vertices of a connected component and the number of trees in the DSF is the number of connected components in the original graph.