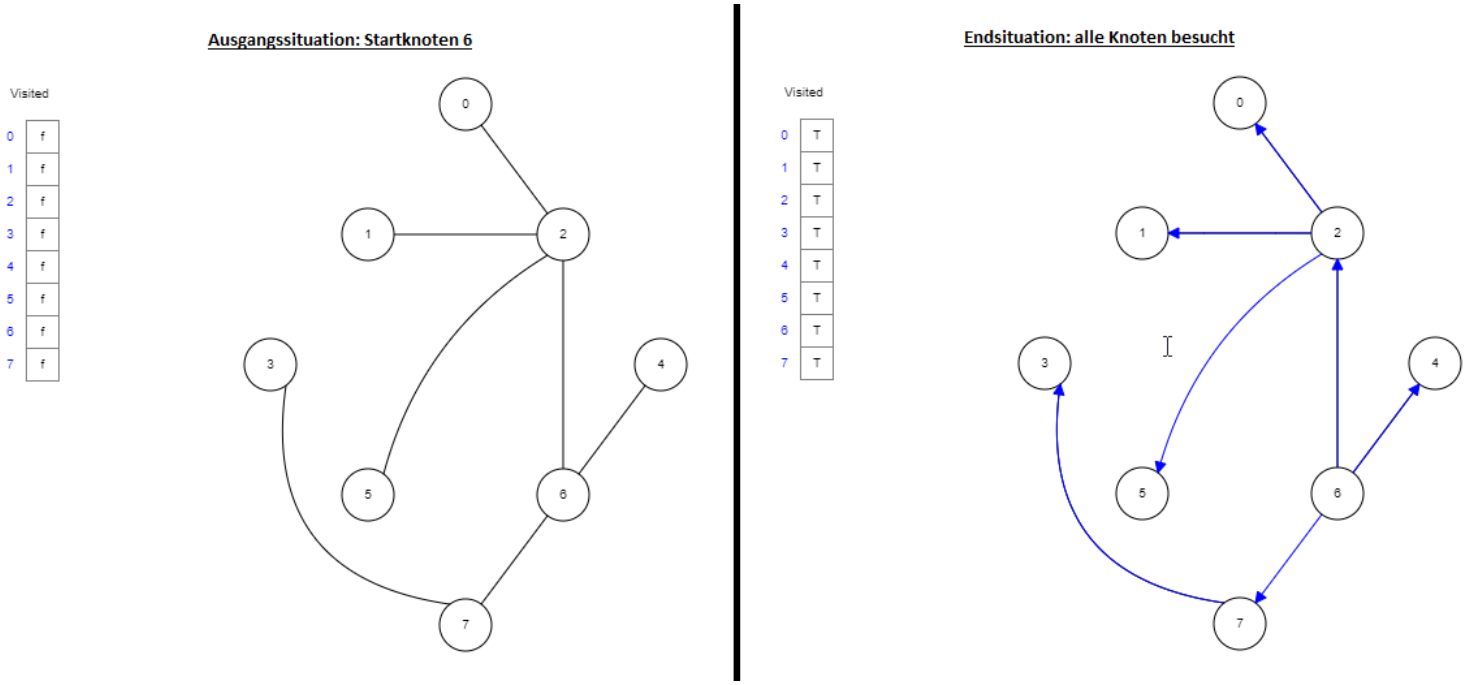
[A09] Graph Zusammen

Grafische Beschreibung



Textuelle Beschreibung

Bei diesem Beispiel handelt es sich um einen beliebigen Graphen. Dieser soll mittels eines Tiefensuche Algorithmus überprüft werden und dabei soll festgestellt werden, ob der Graph zusammenhängend ist. Von einem zusammenhängenden Graphen spricht man, wenn jeder Knoten von jedem anderen Knoten aus über Kanten erreichbar ist. Mit Hilfe der Methode „getNumberofComponents“ wird die Anzahl der zusammenhängenden Komponenten von einem beliebigen Graphen retourniert. Wichtig dabei ist, dass die bereits besuchten Knoten mit getrackt werden. Der Wert 1 wird retourniert, wenn der Graph vollständig zusammenhängend ist.

Laufzeit

Durchschnittsfall: O(|V| + |E|)
Wobei O(|E|) zwischen $O(V^2)$ und O(1) wandert

Wichtige Elemente

```java
public int getNumberOfComponents(Graph graph) {

        // Graph (V,E) -> (Vertices (Nodes) , Edges)
        // vertex = node

        if (graph.numVertices() == 0)
// initial check for zero vertices in graph
            return 0;
        if (graph.numVertices() == 1)
// initial check if only one vertex in graph
            return 1;

        int components = 0;
// initialize number of components with 0
        boolean[] visited = new boolean[graph.numVertices()];
// create new array for all nodes - per default all values are set to false
        Stack<Integer> stack = new Stack<>();              // create stack

        stack.push(0);
// add vertex 0 as initial element on stack / starting point
        //stack.push(new Random().nextInt(graph.numVertices()));   // could
be any :)

        while (!stack.isEmpty()) {
// run loop as long as elements are on the stack

            int vertex = stack.pop();
// get element from stack for processing

            for (int i = 0; i < graph.numVertices(); i++) {
// run through all vertices of graph
                if (graph.hasEdge(i, vertex) && !visited[i]) { // check if
edge exists between popped vertex and vertex from current loop iteration
                    // check also if vertex has not been visited yet
                    stack.push(i);
// if true -> edge exists and vertex hasn't been visited yet add to stack
                }
            }

            visited[vertex] = true;
// mark vertex as visited with true in array

            if (stack.isEmpty()) {    // check if stack is empty -> graph
has been found and interconnected vertices are marked
                components++;        // increase component count by 1
                for (int i = 0; i < graph.numVertices(); i++) {
// loop through all vertices and check if they have been visited
                    if (!visited[i]) { // if vertex has not been visited
                        stack.push(i); // add vertex to stack
                        break;
// stop if one unvisited vertex has been found
                    }
                }
            }
        }
        return components;
    }
}
```