Component (Graph Theory)

Definitions

Definition **Connectivity:** Two vertices u and v are called **connected** if there exists a **path** in graph G from u to v (denoted as: $u \rightsquigarrow v$).

Equivalence relation:

- **Reflexivity:** $\forall a \in V, a \rightsquigarrow a$
- **Symmetry:** $a \rightsquigarrow b \Rightarrow b \rightsquigarrow a$
- **Transitivity:** $a \rightsquigarrow b \wedge b \rightsquigarrow c \Rightarrow a \rightsquigarrow c$

Definition **Component:** A **connected component** is defined as an equivalence relation with the connectivity relation.

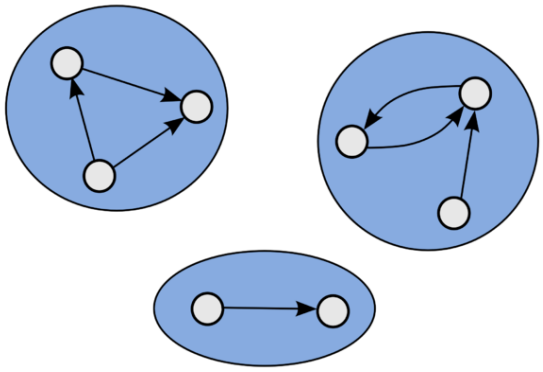

Figure 1: Example of connected components in a graph.

C++ 11 Code

```cpp
#include <iostream>
#include <vector>

const int SIZE = 1e5 + 1;

std::vector<int> adj[SIZE];
int cmp[SIZE];
bool vis[SIZE];
```

```
void dfs(int v) {
    vis[v] = true;
    for (int i = 0; i < adj[v].size(); i++) {
        int next_v = adj[v][i];
        if (!vis[next_v]) {
            cmp[next_v] = cmp[v];
            dfs(next_v);
        }
    }
}

int main() {
    // For this example, we'll hardcode a small graph.
    // In a real-world application, you would read these values from an input source.
    int n = 7; // Number of vertices
    int m = 5; // Number of edges

    // Manually adding edges to the adjacency list
    adj[0].push_back(1);
    adj[1].push_back(0);

    adj[1].push_back(2);
    adj[2].push_back(1);

    adj[3].push_back(4);
    adj[4].push_back(3);

    adj[5].push_back(6);
    adj[6].push_back(5);

    int cmp_num = 0;
    for (int i = 0; i < n; i++) {
        if (!vis[i]) {
            cmp_num++;
            cmp[i] = cmp_num;
            dfs(i);
        }
    }

    std::cout << "Number of components: " << cmp_num << std::endl;
    std::cout << "Component IDs for each vertex:" << std::endl;
    for (int i = 0; i < n; ++i) {
        std::cout << "Vertex " << i << ": " << cmp[i] << std::endl;
    }
```

```
    return 0;
}
```

Algorithm Step-by-Step Explanation

For each vertex in the graph:

- If it has not been visited:
  - Increment the component number.
  - Start DFS from this vertex.
  - Mark all reachable vertices as visited and assign them the same component number.
- Else print the component number.