# Graph Algorithms Sample Problem

## 1. Judging books by their covers

You have a large collection of books, and just got a new bookshelf. For aesthetic reasons, you're going to arrange your books by the color of their covers (not by author or subject). You wish to put only books of a single color of any given shelf. You have a list of pairs of books which you know to be of the same color. However, this list might only be partial (it's possible that $u, v$ and $w$ are all the same color, but the list only contains that $u$ and $v$ are the same color and that $v$ and $w$ are the same color). You should assume that the "same color" relation is transitive. Transitive means that if $u$ and $v$ are the same color, and $v$ and $w$ are the same color, then $u$ and $w$ are also the same color.

Given your list, your job is to give an upper bound on the number of shelves you will need so that no shelf has more than one color of a given book. Describe an algorithm to give the best bound you can on the number of shelves needed.

(a) Describe a graph you can use in your problem.

**Solution:**

Let $\mathcal{P}$ denote the input list of pairs, i.e. for each pair $(u, v) \in \mathcal{P}$, $u$ and $v$ are of the same color. As with graph modeling problems, the name of the game here is creating the right kind of auxiliary graph. For each pair $(u, v) \in \mathcal{P}$, we add $u, v$ to the graph if they are not already present and add an edge between the aforementioned books. More formally,

$$G = (V, E) \text{ where } V = \{u \in \mathcal{P} \mid u \text{ is a book}\} \text{ and } E = \{(u, v) \in \mathcal{P} \mid u \text{ and } v \text{ are of the same color}\}$$

Intuitively all this is doing is creating a network of books where books are connected if and only if they are of the same color. Since we mentioned that the "same color" relation is transitive, we can assert that given a "walk" between two books $x, y \in V$, they are of the same color. Therefore the number of connected components in the graph $G$ is the total number of colors of books, and thereby an upper bound on the number of shelves. In this representation; each connected component corresponds to a shelf, and any subset of books in a connected component can be on its shelf.

(b) Describe an algorithm to solve this problem. For this part, you should assume that the graph that you described in part (a) is already built; **do not rebuild it here**
We provide two solutions; one where we slightly alter one of the graph algorithms that we've covered in lecture, and one where we look at things more abstractly.

**Solution:**

Solution option 1: Using library functions:
Given this construction, the problem is now reduced to simply finding the number of connected components in $G$. Choose your favorite graph traversal algorithm; [(B/D)FS] and run it on $G$ to return the number of shelves required.

**Solution:**

Solution option 2: modifying code:
Below is some pseudocode that directly alters the BFS-calls made to find the number of connected components

```
function breadthFirstSearch(adjList, nodeQueue, seenNodes)
    while nodeQueue ≠ ∅ do
        u ← dequeue(nodeQueue)
        for v ∈ adjList[u] do
            if v ∉ seenNodes then
                enqueue(nodeQueue, v)
                seenNodes ← seenNodes ∪ {v}

function countNumShelves(bookPairs)
    adjList ← ∅                                   ▷ initializing the adjacency list of the graph
    for u, v ∈ bookPairs do                       ▷ iterating through book pairs
        if u ∉ adjList then
            adjList[u] ← ∅
        if v ∉ adjList then
            adjList[v] ← ∅
        adjList[u] ← adjList[u] ∪ {v}             ▷ adding edge between u and v; u, v are of the same color
        adjList[v] ← adjList[v] ∪ {u}

    numShelves ← 0
    seenNodes ← ∅                                 ▷ keeping track of visited nodes
    nodeQueue ← ∅

    for u ∈ adjList do                            ▷ iterating through all nodes in the graph
        if u ∉ seenNodes then
            numShelves ← numShelves + 1
            seenNodes ← seenNodes ∪ {u}
            enqueue(nodeQueue, u)
            breadthFirstSearch(adjList, nodeQueue, seenNodes)
```

(c) Explain why your algorithm returns the correct number of shelves required to house books of the same color.

**Solution:**

Solution option 1: An informal explanation, connecting the graph to the word problem
Note that any sequence of connections ($u$ is the same color as $v_1$ is the same color as $v_2$ ,... is the same color as $w$) guarantees that all of $u$ and $w$ can be on the same shelf. Thus any vertices in our graphs connected by paths are allowed on the same shelf. The full set of things connected by some path is a connected component, thus everything in the same connected component can go on one shelf, and things in different components have no such path and cannot be guaranteed to go on the same shelf. Having one shelf for each component (since components are all connected vertices) minizes the number of components.

**Solution:**

Solution option 2: If you are familiar with induction proofs from prior classes, you can use them to formalize that intuition:

The correctness of this algorithm rests on proving the correctness of our algorithm to find the number of connected components in a graph. With this in mind, observe that when searching from a particular vertex $v$, we will never reach vertices outside the connected component of $v$. Consequently, we need to prove that we will eventually reach all vertices in the connected component of $v$. We argue via induction. For some notation, let us denote the vertices at a distance $k$ from $v$ as $V_k$.

**Base Case:** $k = 0$: Observe that BFS will clearly reach all nodes in $V_0$ since $V_0 = \{v\}$; the nodes at a distance $0$ from $v$ is simply $v$ itself.

**Inductive Hypothesis:** Assume that BFS will reach all nodes in $V_k$ for some $k \geq 0$.

**Inductive Step:** Indeed consider the vertices in $V_{k+1}$. Note that all vertices in $V_{k+1}$ are adjacent to some vertex in $V_k$. By the inductive hypothesis, we know that BFS will reach all vertices in $V_k$. Therefore, BFS will reach all vertices in $V_{k+1}$ since it will reach all vertices adjacent to vertices in $V_k$.

This completes the proof.

Since our algorithm correctly finds the number of connected components in a graph, it will correctly find the number of shelves required (in direct correspondence to the number of connected components) to store the books.

(d) Describe the running time of your algorithm in terms of $b$ (the number of books); $p$ (the number of pairs of books of the same color); and $s$ (the number of shelves needed).
**Solution:**

Note that the search in the above algorithm takes a total time of $\mathcal{O}(|E| + |V|)$ since BFS takes linear time in the number of vertices and edges for each component in the graph and each component is only visited once. There's also a linear time preprocessing step to construct the adjacency list of the graph, which takes time $\mathcal{O}(|V| + |E|)$ since we're simply iterating through all pairs in the input list and adding them to the adjacency list. Since $|E| = p$ and $|V| = b$, the total running time of the algorithm is $\mathcal{O}(p + b)$.