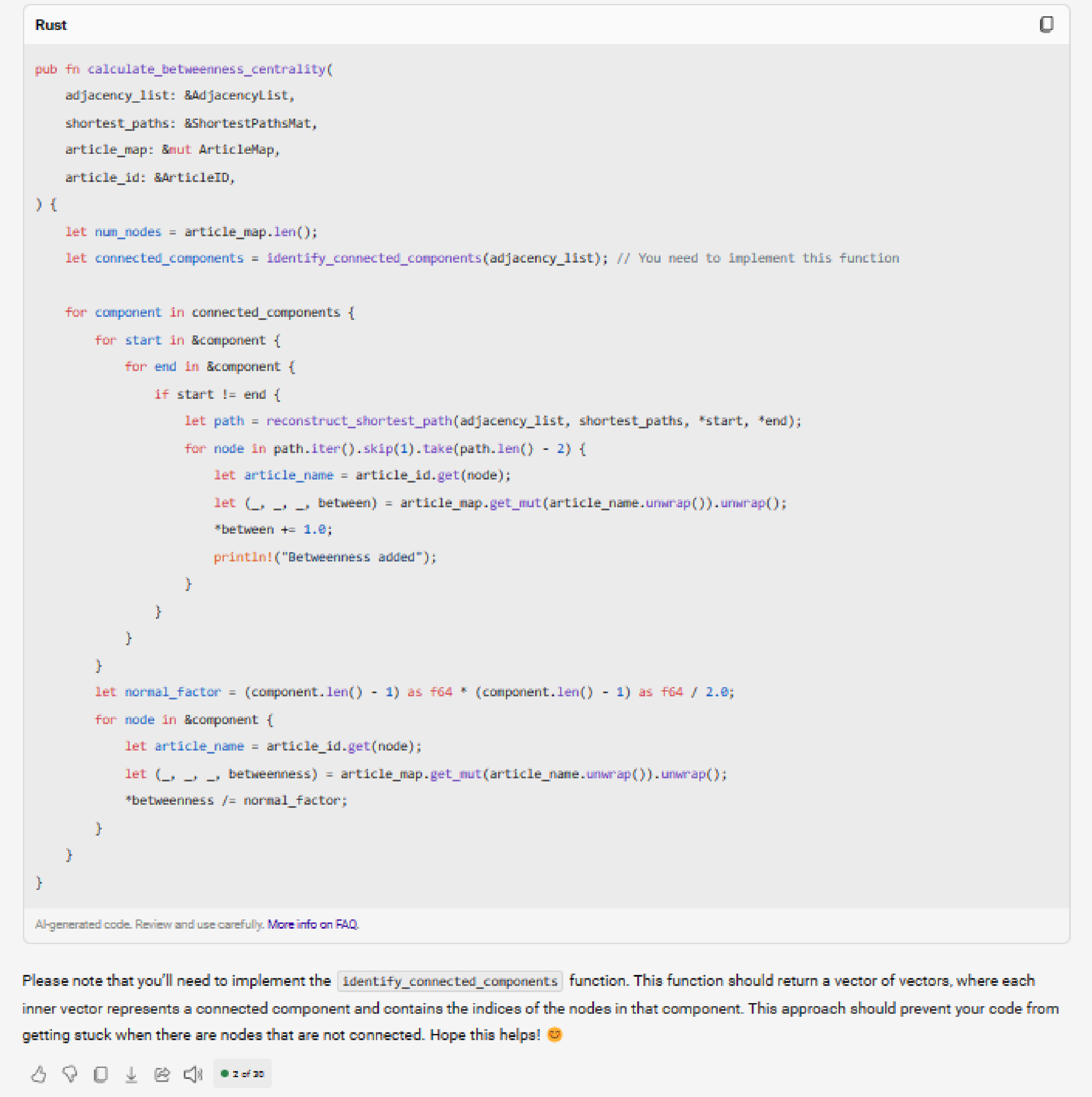
The dataset this project uses can be found here: https://snap.stanford.edu/data/wikispeedia.html
This project analyzes the centrality of Wikipedia articles. No user input is needed to run, only the links.tsv and articles.tsv files found at the link above. When run, the code will output:

```
0 starting vertices shortest path to all other vertices checked
500 starting vertices shortest path to all other vertices checked
1000 starting vertices shortest path to all other vertices checked
1500 starting vertices shortest path to all other vertices checked
2000 starting vertices shortest path to all other vertices checked
2500 starting vertices shortest path to all other vertices checked
3000 starting vertices shortest path to all other vertices checked
3500 starting vertices shortest path to all other vertices checked
4000 starting vertices shortest path to all other vertices checked
0 starting vertices shortest path to all other vertices checked
…(same output for a while)
0 starting vertices shortest path to all other vertices checked
There were 495 components found
Imagism indegree centrality: 0
Western_Front_(World_War_I) indegree centrality: 0
Bouvet_Island indegree centrality: 0
Georg_von_Boeselager indegree centrality: 0
Simca_Vedette indegree centrality: 0
Flag_of_Belarus indegree centrality: 0
Babe_Ruth indegree centrality: 0
Charlemagne indegree centrality: 0
Private_Peaceful indegree centrality: 0
Gothenburg indegree centrality: 0
Berlin indegree centrality: 0
Children's_Crusade indegree centrality: 0
Vasco_da_Gama indegree centrality: 0
Barnacle indegree centrality: 0
Bjørnøya indegree centrality: 0
Nitrogen_cycle indegree centrality: 0
Tasmanian_Devil indegree centrality: 0
Philosophy indegree centrality: 0
StarCraft indegree centrality: 0
Electromagnetic_radiation indegree centrality: 0
Imagism outdegree centrality: 0
Western_Front_(World_War_I) outdegree centrality: 0
Bouvet_Island outdegree centrality: 0
Georg_von_Boeselager outdegree centrality: 0
Simca_Vedette outdegree centrality: 0
Flag_of_Belarus outdegree centrality: 0
Babe_Ruth outdegree centrality: 0
Charlemagne outdegree centrality: 0
```

```
Private_Peaceful outdegree centrality: 0
Gothenburg outdegree centrality: 0
Berlin outdegree centrality: 0
Children's_Crusade outdegree centrality: 0
Vasco_da_Gama outdegree centrality: 0
Barnacle outdegree centrality: 0
Bjørnøya outdegree centrality: 0
Nitrogen_cycle outdegree centrality: 0
Tasmanian_Devil outdegree centrality: 0
Philosophy outdegree centrality: 0
StarCraft outdegree centrality: 0
Electromagnetic_radiation outdegree centrality: 0
thread 'main' panicked at src\filehandling.rs:124:99:
called `Option::unwrap()` on a `None` value
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

This code should calculate degree centrality and betweenness centrality for all of the nodes provided. It will then output the articles with the top 20 indegree centrality, outdegree centrality, and betweenness centrality. The use of each function is described in the code comments. Both a BFS and a DFS were implemented, the BFS used to calculate predecessors of each node while the DFS finds components. Both functions were crosschecked with the BFS and DFS implementations shown in class to ensure correctness. This code does take a long time to run, as the betweenness centrality algorithm implemented has a complexity of O(N^2). With ~4,600 vertices and ~110,000 edges, this code takes upwards of an hour to run. I ran out of time to do this project, so it is incomplete. I did not write any tests, and the code does not actually function correctly. There is a bug in the print_top_20_by_criteria and even if I fixed the bug I do not have enough time to run the program again and see if it outputs correctly, as the bug will not occur until the code has already been running for an hour and a half and it is 10:30 pm as I write this. There is also a bug in how the degree centrality is calculated, as the information is not being updated in my ArticleMap correctly. I don’t think betweenness centrality is correct either, as one of the values placed in causes the code to panic trying to unwrap a None value.

Microsoft Copilot helped write some of the code provided, namely the betweenness centrality function. The function given by Copilot can be seen in the screenshot below. The identify_connected_components and reconstruct_shortest_path both also had to get written, I wrote identify_connected_components (named find_components in my code), but had copilot help me write reconstruct_shortest_path, which can also be seen below. Reconstruct_shortest_path also required the predecessors function, which I wrote as a breadth first search where I would add each node that was connected to my starting node to the predecessor list of that node. This works since reconstruct_shortest_path works in reverse, so to see the shortest path to your starting node, you would need a BFS from that node to see which nodes could be an endpoint for a shortest path from the starting node to any ending node.

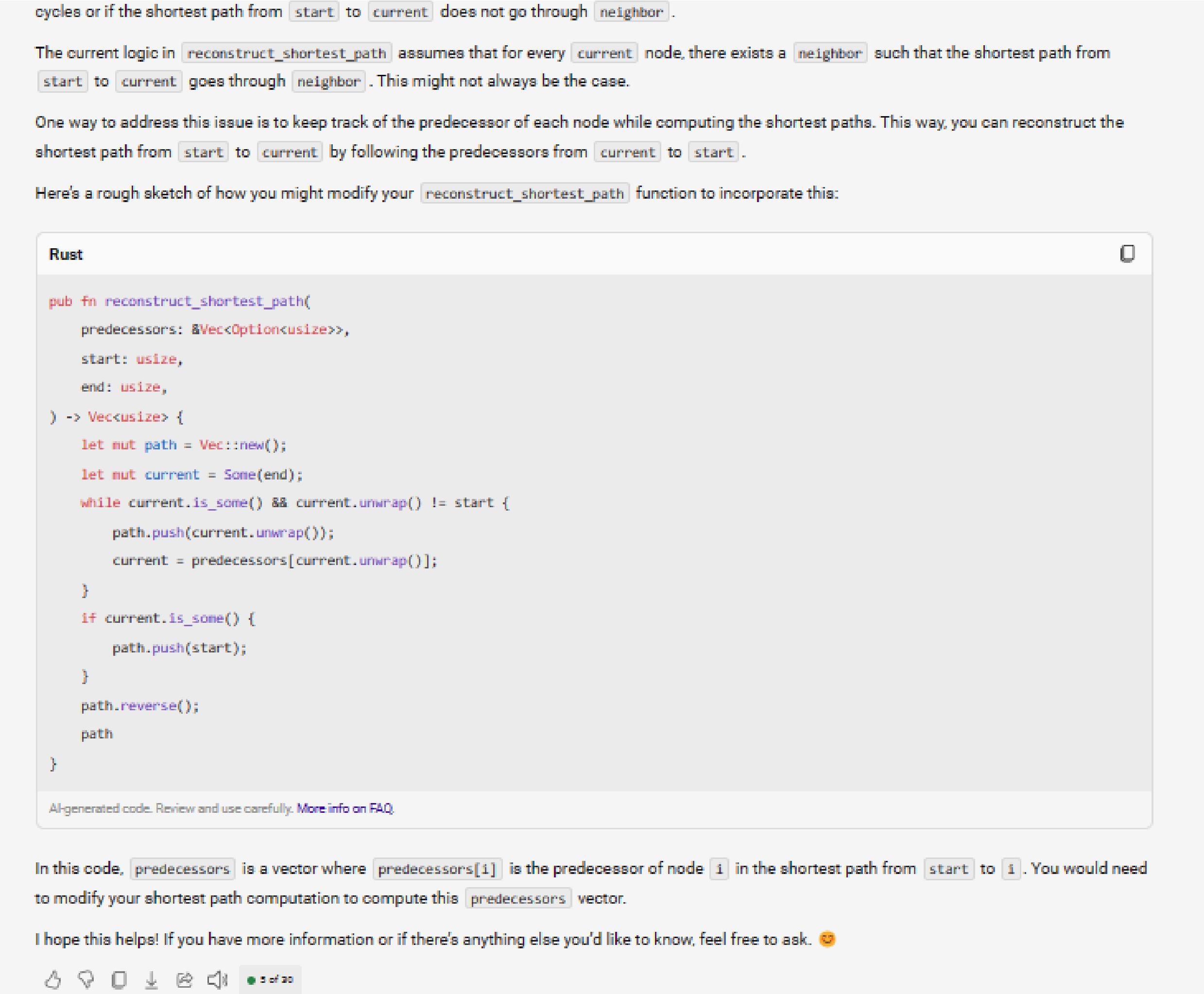


```rust
pub fn calculate_betweenness_centrality(
    adjacency_list: &AdjacencyList,
    shortest_paths: &ShortestPathsMat,
    article_map: &mut ArticleMap,
    article_id: &ArticleID,
) {
    let num_nodes = article_map.len();
    let connected_components = identify_connected_components(adjacency_list); // You need to implement this function

    for component in connected_components {
        for start in &component {
            for end in &component {
                if start != end {
                    let path = reconstruct_shortest_path(adjacency_list, shortest_paths, *start, *end);
                    for node in path.iter().skip(1).take(path.len() - 2) {
                        let article_name = article_id.get(node);
                        let (_, _, _, between) = article_map.get_mut(article_name.unwrap()).unwrap();
                        *between += 1.0;
                        println!("Betweenness added");
                    }
                }
            }
        }
        let normal_factor = (component.len() - 1) as f64 * (component.len() - 1) as f64 / 2.0;
        for node in &component {
            let article_name = article_id.get(node);
            let (_, _, _, betweenness) = article_map.get_mut(article_name.unwrap()).unwrap();
            *betweenness /= normal_factor;
        }
    }
}
```

AI-generated code. Review and use carefully. More info on FAQ.

Please note that you'll need to implement the `identify_connected_components` function. This function should return a vector of vectors, where each inner vector represents a connected component and contains the indices of the nodes in that component. This approach should prevent your code from getting stuck when there are nodes that are not connected. Hope this helps! 😊



cycles or if the shortest path from `start` to `current` does not go through `neighbor`.

The current logic in `reconstruct_shortest_path` assumes that for every `current` node, there exists a `neighbor` such that the shortest path from `start` to `current` goes through `neighbor`. This might not always be the case.

One way to address this issue is to keep track of the predecessor of each node while computing the shortest paths. This way, you can reconstruct the shortest path from `start` to `current` by following the predecessors from `current` to `start`.

Here's a rough sketch of how you might modify your `reconstruct_shortest_path` function to incorporate this:

Rust

```rust
pub fn reconstruct_shortest_path(
    predecessors: &Vec<Option<usize>>,
    start: usize,
    end: usize,
) -> Vec<usize> {
    let mut path = Vec::new();
    let mut current = Some(end);
    while current.is_some() && current.unwrap() != start {
        path.push(current.unwrap());
        current = predecessors[current.unwrap()];
    }
    if current.is_some() {
        path.push(start);
    }
    path.reverse();
    path
}
```

AI-generated code. Review and use carefully. More info on FAQ.

In this code, `predecessors` is a vector where `predecessors[i]` is the predecessor of node `i` in the shortest path from `start` to `i`. You would need to modify your shortest path computation to compute this `predecessors` vector.

I hope this helps! If you have more information or if there's anything else you'd like to know, feel free to ask. 😊