Output is this:
Top 5 PageRank scores:
Node 603 has PageRank 0.0023
Node 299 has PageRank 0.002
Node 519 has PageRank 0.0019
Node 867 has PageRank 0.0019
Node 458 has PageRank 0.0019

A lot of explanation about each function specifically can be found in the code comments, but I will go over how I planned the structure of the program here. I wanted to write the main function in one file, but have all my function definitions in a separate mod file; I named this file prgraph.rs. I declared a few local types, that are mainly there for readability. These can be seen at the top of prgraph.rs. Then, I created a struct to represent the graph, which would contain the number of vertices in the graph, an edge list, and an adjacency list. I wrote one function that would take the file that professor gave in the question description and return both the first line, which was the number of vertices, and the rest of the lines in a list that represented the edge list for the graph. The next function was written to turn that edge list into an adjacency matrix, then a creator function was written that returns instances of the graph struct I created. I then created a new tuple type that I use to store the count of times a random walk ended at a certain node, and the identity of that node. This was done so that when I sort the list of counted endings later, I do not disassociate each score with its node ID. There was also a function written that will take instances of this tuple, check my current endings list for the presence of whatever node the tuple contained, then either add one to the count associated with it, or add that node into the ending list with a count of 1.

I then put the rest of the functions in a submodule page_rank. This submodule imported all the types and functions defined in the graph module. I wrote the walking function here, which follows the question description. It uses a nested loop over the range 0..100, where the outer loop picks a new node for the inner loop to start walking from each time it runs. At the end of the inner loop, the function to update a nodes ending count is called to update the count for whatever node the walk ended on. A function is written here also to pick new node from anywhere on the graph, for the main purpose of having this run on a different rng thread than the walking function. The final function is written to sort my ending list, get the top 5 values, get the page rank score for each by dividing by 100 times the number of nodes in the graph, and print the top 5 values. This function uses a closure to sort the ending list by the second value of the custom tuple I implemented, since the second value contains the count of how many random walks ended at the given node.