

Wikipedia Maps

Rylan Mahany



WIKIPEDIA**Maps**

Defining the problem

I first got the idea of this project when playing “The Wikipedia Game”. For those who haven’t played, you start on one Wikipedia page and think of another, by only clicking the links on each page, try to get to the target page. (Project Python file in GitHub)

An example of a simple route on WikipediaMaps is Civil War to Abraham Lincoln. Since Abraham Lincoln was the leader of the Union Army, he is listed on the page for Civil War. This gives the route a length of 1. For basketball to baseball, the route could be Basketball to Michael Jordan to Michael Jordan’s short-lived baseball career to baseball. This gives the route a length of 3. The goal is to find the route with the smallest length.

Approach to the Problem

The solution I came to was to use a WebCrawler and a Breadth-First-Search (BFS). The WebCrawler looks at the HTML code of the Wikipedia pages and looks for any URLs. These URLs are then looked through in BFS. The first page will have all its’ links checked to see if it’s the target. If they are not, the first link will be explored for all its’ links. If that isn’t it, we check again and again. If we hit a dead end, we go all the way back to the top and keep looking.

Example BFS Procedure

Basketball -> Michael Jordan, Kobe Bryant, Spalding

The WebCrawler explores the Basketball page to find Baseball. Michael Jordan, the first link, is the next place to look.

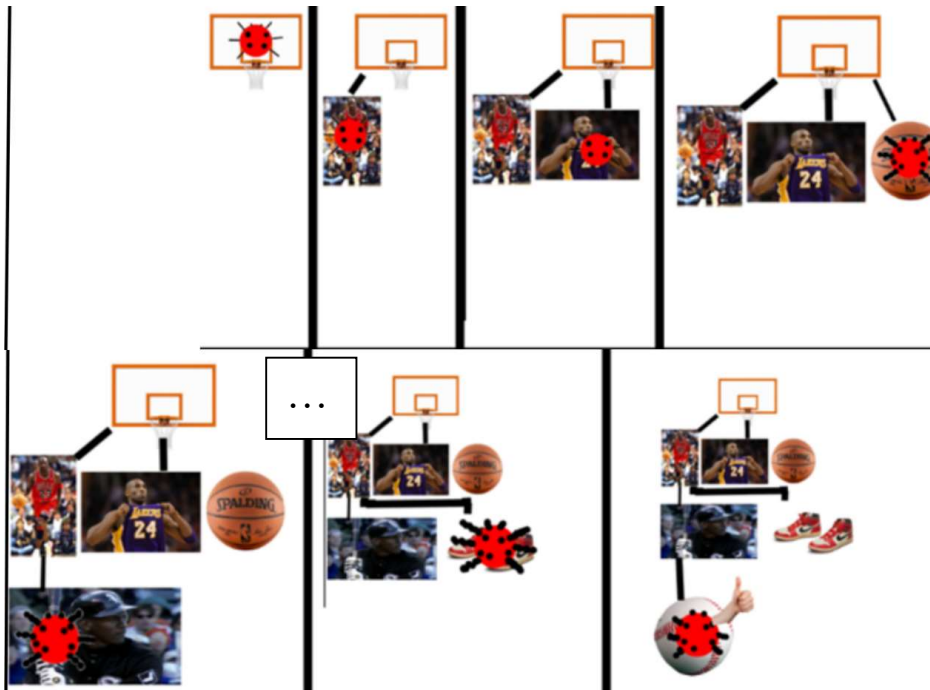
Michael Jordan-> Michael Jordan Baseball Career, Michael Jordan Shoes

After not finding baseball in Michael Jordan, Kobe Bryant's page is checked, then Spalding's page. Finally, Michael Jordan Baseball Career is checked.

Michael Jordan Baseball Career -> Baseball

Now that the target is found, the path can be returned.

Basketball -> Michael Jordan -> Michael Jordan
Baseball Career -> Baseball



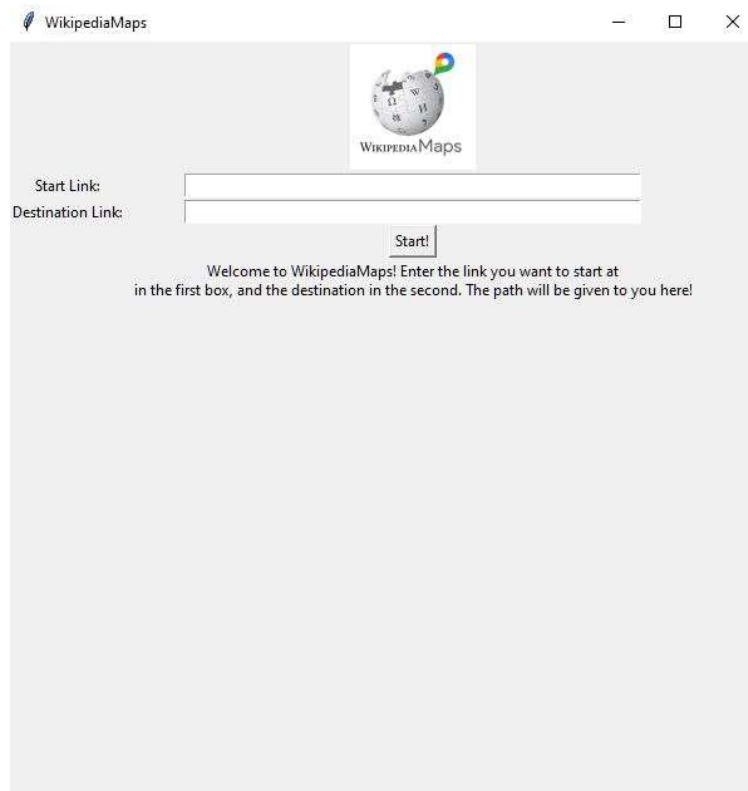
Problems and Limitations

This algorithm will absolutely work for “easier” examples of the problem. BFS is a useful search algorithm because it can find a page in $O(b^d)$ Time where b is the maximum links per page, and d is the depth of the target page. If each page has 5 links and we go for 3 pages, the maximum number of operations to find the target is $5^3=125$ operations. This is on the lower end of other search algorithms. Another great perk of BFS is that it is safe to use in trees with cycles. With other algorithms its very likely to get stuck in a loop (Michael Jordan -> Jordan Nike Shoes -> Nike -> Basketball -> Michael Jordan -> ...). Given the size of Wikipedia, there are probably gigantic cycles that could overflow the RAM of a normal computer without it knowing it's in a cycle.

The issues lie within two things, the first is the size of Wikipedia. b can be in the hundreds, and d could be far.

The other lies in that we need to recreate our path, this makes the Space complexity $O(b^m)$. Where m is the maximum depth, this is the page that is the furthest away from the starting. This could be The French Revolution and the Apple II Computer, literally the furthest away. To take this past a few page jumps, I would need either a lot stronger of a computer or a more clever algorithm.

UI/UX Design



To create the GUI for this program, I used the Tkinter and Image Pillow library. I first played with the window to get a good size, 600x600 felt good. I gave the window its title and moved into making a grid. The grid is a 2 x 5 that looks like the following:

[pillow image object(centers)]]
[Start Link: text box]
[Destination Link: text box]
[Start button(centers)]]
[Welcome message/output]]