Project name:
“Graph”-ing Wikipedia

Group members:
Jane Chen - janeziyanchen@college.harvard.edu
Dan Leichus - Leichus@college.harvard.edu
Jason Shen - chungweishen@college.harvard.edu

Brief Overview

What is the problem you are addressing?

We are interested in examining a directed graph of Wikipedia articles, where hyperlinks are edges between pages. Is Wikipedia even a connected graph? Given two articles, what is the shortest path from the first to the second? What is a page that is “central” on Wikipedia? Is Hitler close to the center? Is Jesus? Is Kevin Bacon? Or how closely related are Jesus and Kevin Bacon exactly? Our project will explore these questions and others.

In broad terms, how do you plan to address it?

We will obtain a dump of Wikipedia from the Wikimedia foundation. From there, we will extract the pages and links and discard all the text and multimedia. These pages and links will serve as the basis for a graph of Wikipedia which will be constructed. To solve the shortest path problem, we will use Dijkstra’s algorithm implemented using a priority queue implemented using a Fibonacci Heap.

What are your goals for the project?

Our goal for this project is to create a cool program to give advice to participants of the Wikipedia game - finding the shortest path from one page to another page. Perhaps representing Wikipedia as a image showing its connectedness. Perhaps examining disconnected regions of Wikipedia, if they exist. It will also be interesting to see what articles have the most links that link to it.

Feature List

Core Features:

- Dijkstra's Algorithm / Floyd-Warshall Algorithm
- Fibonacci Heap
- User interface for querying our graph

Middle tier:
- Website that users can access to use our tool

Cool Extensions:
- Examining and classifying disconnected subgraphs
- Common Wikipedia searches; ex. "Five Clicks to Jesus," "Clicks to Hitler"
- Finding most commonly visited articles
- Locating articles with the most links yet fewest clicks
- Finding a median article on Wikipedia
- Pictorial image of Wikipedia

Technical Specification

The front end and back end of our project can be completely separated.

On the front end of our project, we have everything that involves Wikipedia. Obtaining pages and links from a dump of Wikipedia is a clean segment of our project. Another segment is constructing a graph using these links. Finally another segment is creating an application for users to query our graph and request that Dijkstra's Algorithm is run between two articles.

The pages and links of Wikipedia can be implemented in a string list and a string list list, where the nth string in the first list is a page and the nth list in the second list is a list of all of that page's links.

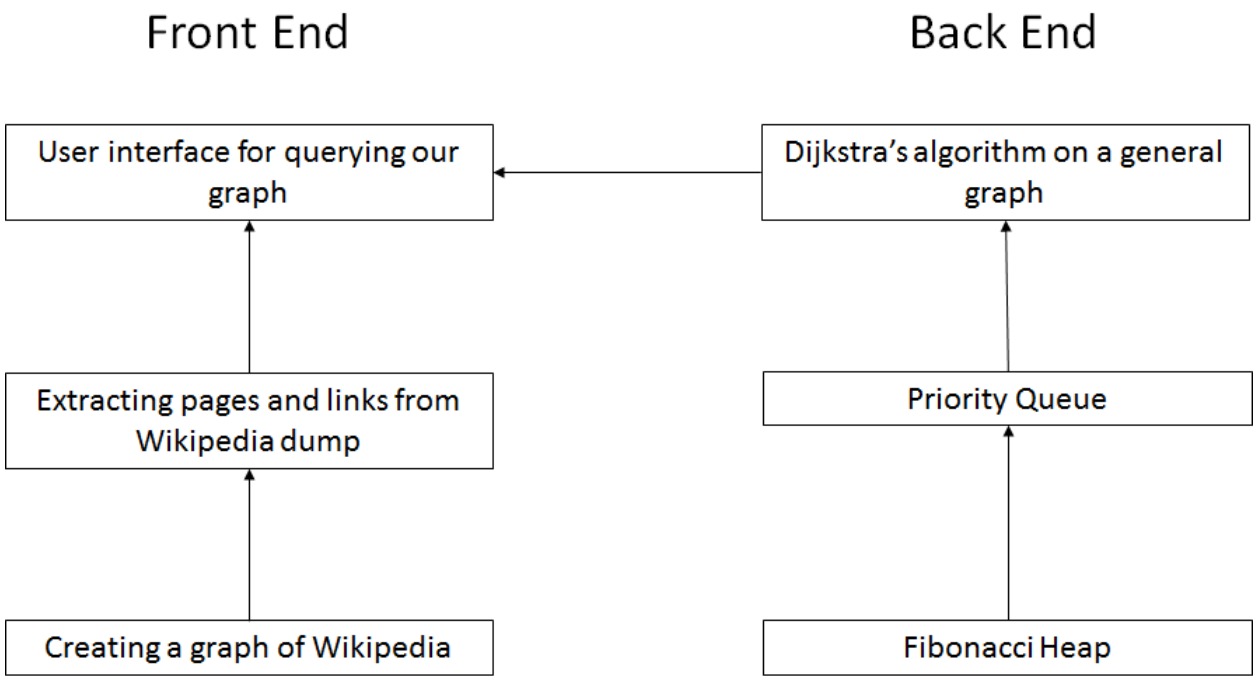
To turn these lists into a graph, we are not really sure what to do. Perhaps we could use objects that represent the nodes (pages), because then these objects could each have functions associated with it. Each object could have a list of strings that represent other nodes? Or maybe a list of refs to other nodes? And then have a graph be another object containing all of the nodes, or refs to all of the nodes. And we could have functions on the graph too.

For the application, we could use the OCaml Graphics library or some other library. It would query a function named "find_shortest_path" or something of the like and would pass in a starting article and an ending article.

If we end up creating a website, I'm not sure how that will work with OCaml. Will we have to code in all the results for each possible query or will our website dynamically query our program. We will explore this with our TF.

For the back end, we will be implementing Dijkstra's algorithm on a graph. We can abstract out the implementation of the graph and have it run on a general graph, having a graph module abstract away the graph functions.

We can also abstract out the implementation of the priority queues used in Dijkstra's algorithm. In this way, if we discover another implementation to be better, we can replace the Fibonacci heap with a better abstract data type. This also allows us to modify the Fibonacci heap module without messing up the algorithm, and will allow our team to work on separate parts of the project simultaneously.



Next Steps

Possible Languages: OCaml, Java, C
Possible Tools: Dropbox, VMWare, BlueJ, Eclipse
Possible Frameworks: CSS, Django, Ajax

Before we write our final spec, we will all decide on how to set up our environment and be familiar with the algorithms that we will implement. We will also agree on our language.