Project Proposal

Team Name: EmbeddedSpaces
Team Members: Tony Cannistra, Brett Fischler, Dan Griffin

Project Idea:

We intend to create an interactive visualization of Wikipedia, the implementation of which is divided into two conceptual modules: a concurrent back end which handles the Wikipedia database, and a Javascript front end which builds and displays the visualization. The back end of our project will expose a service to the world which, given two Wikipedia article names and assorted tuning parameters, will return a representation of a force-directed graph which describes the n shortest paths between the two articles given. This information will then be received by the Javascript web front end, which will display it using some framework. The web front end will also produce a user interface for interacting with the back end, with features such as article title autocompletion and parameter choice. The back end of the project will be where our knowledge of concurrency enters: using Erlang threads, we hope to increase the speed of wikipedia graph access by spawning worker threads as we deepen the search throughout the network.

Deliverables:

At a minimum, we'd like to create a back end likely in Erlang which concurrently accesses some representation of the Wikipedia graph and returns the n shortest paths between two articles in the network. In addition, we will build the visualization to display this information. Our maximum deliverable is a more highly parameterized version of this project, which allows potentially for the comparison of more than two articles at once, and a more robust Wikipedia access mechanism which depends less on the representation of the encyclopedia.

Our First Step:

Our first task is to explore the available representations of Wikipedia. Currently Wikipedia is available via web crawling and database downloads of various formats, though for our sanity and that of the WikiMedia foundation the most favored approach is that of a downloaded database. Pre-processing this database by pruning away article contents and only maintaining the network of connected articles (names/IDs and edges) is likely to be our first implementation challenge. We will then move on to designing an algorithm for the concurrent exploration of this data using Erlang threads once we have a

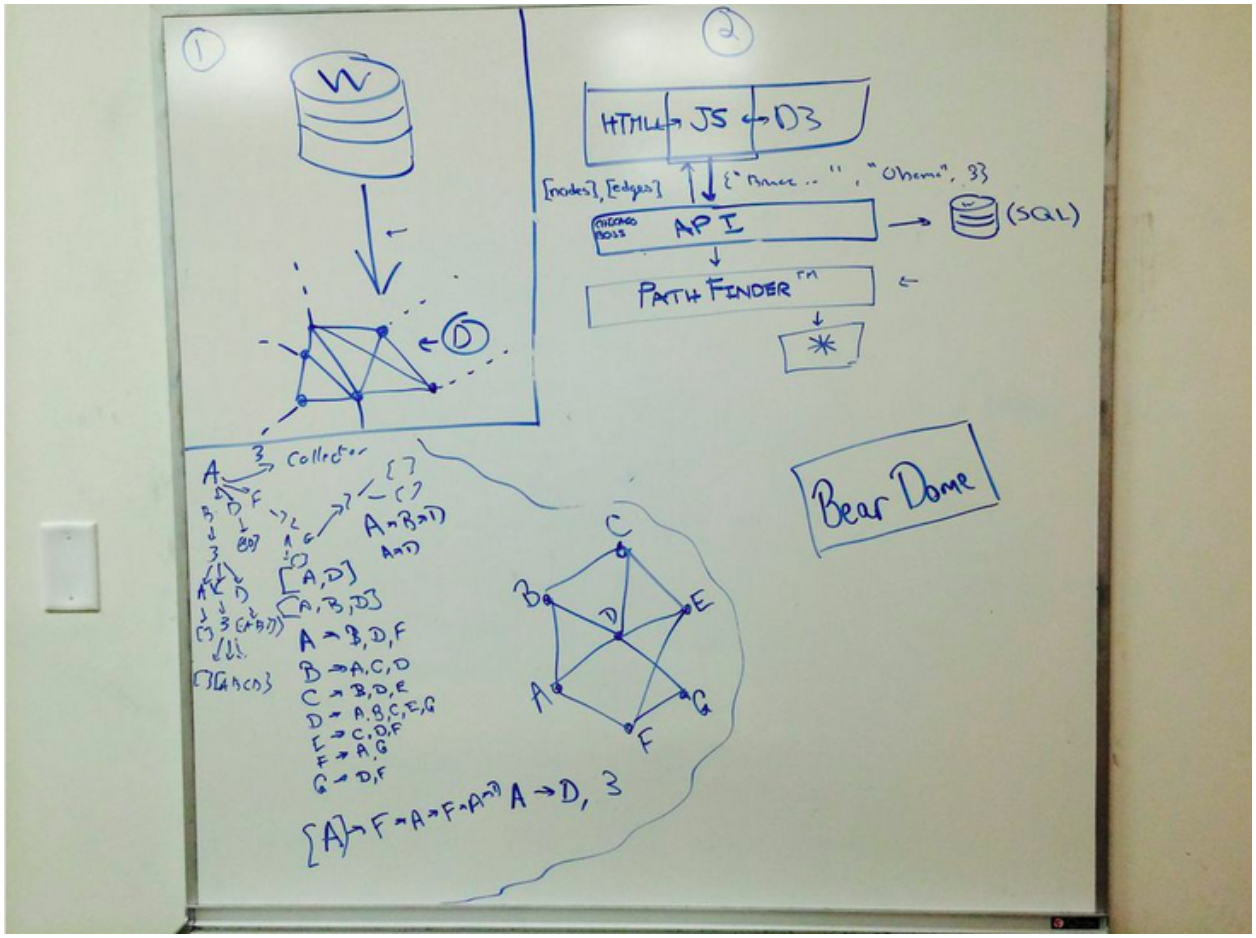
representation we're comfortable with.

Biggest Problem we Foresee:

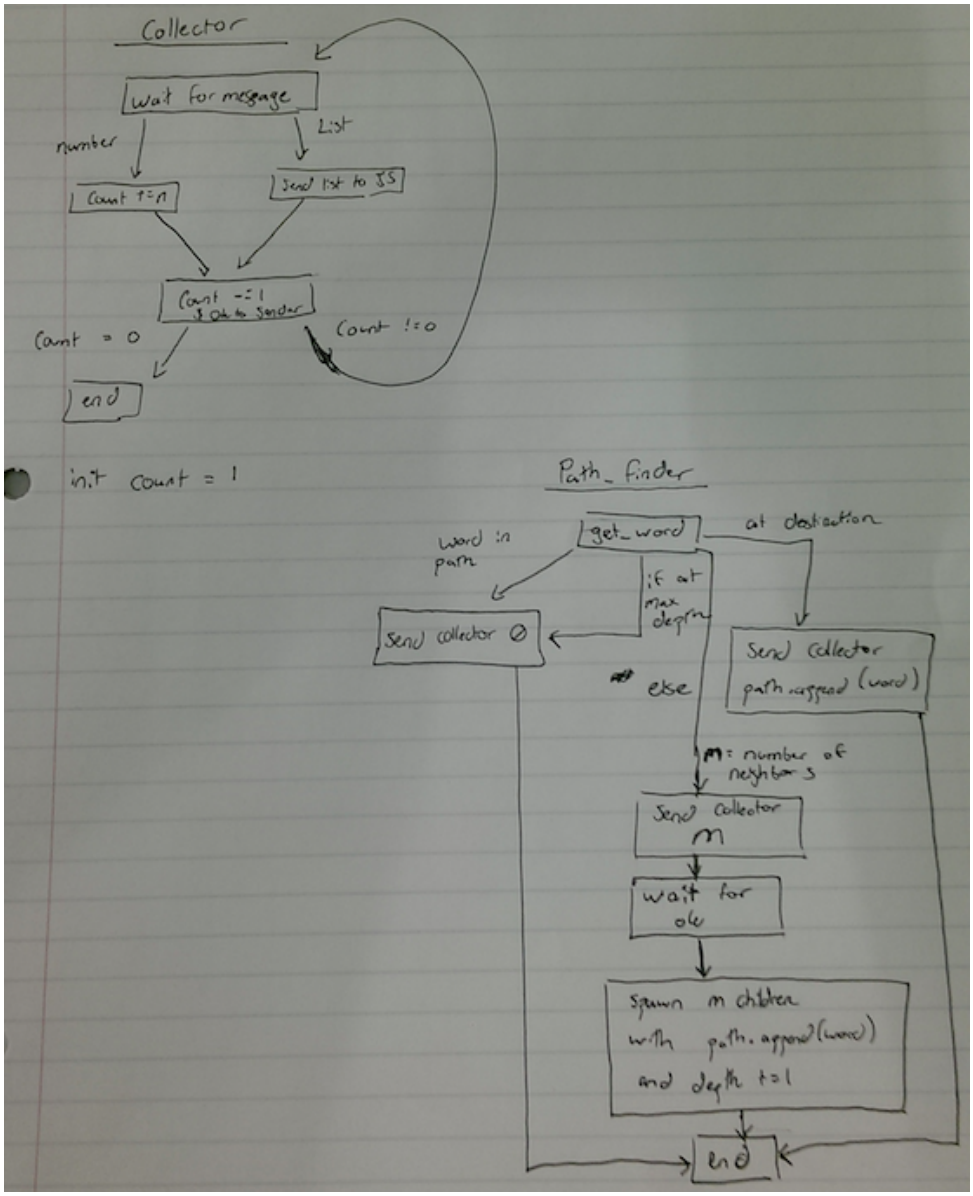
We believe the most difficult challenge will be in the design of the threaded system for accessing the wikipedia graph. While we feel pretty confident in our ability to represent the network in a way that facilitates easy access to the information we need, we're slightly concerned about thread communication (how do we know if two paths, therefore two threads, converge upon the same node on their path to the final destination?). We're also anticipating some challenges in using Erlang, as it is a new language for all of us, and the unfamiliar concurrency aspects of the language will be interesting to learn.

Initial Design Proposal

Module Diagram (top right!):



Interesting State/Process Diagram:



For this project, a critical portion is the development of an algorithm which finds all possible paths under length *n* between two Wikipedia articles in a fast, scalable, and concurrent fashion. Working with the assumption that we have a well-formed Wikipedia network stored somewhere as a graph (in adjacency list representation, for example), we developed several initial attempts at an algorithm before settling on our final version.

Our first attempt took cues from the MergeSort homework problem, in which we begin at the node that represents the “source” or “start” article from which we’re looking for a path, and at each step spawn threads that would search for the “destination” or “stop” article in the network, one thread for each of the source node’s neighbors. When a process finds the destination article, it sends a message to the process above it with that information. We disliked this plan

because it doesn't capture the power of Erlang processes, and would require complex message passing and PID management that we weren't sure we could handle.

Our next approach, which turned out to be the one we chose with some refinement, was one in which each spawned thread (as above, one for each "source" node as we "recurse" through the network) sends a message to a *single* "collector" function which then keeps track of the paths through the network that satisfy the input criteria. Our original plan was to have each thread that discovered the destination article in the network send a message to the collector which contained the path of nodes through the network necessary to reach the destination node. We realized, however, that this approach does not allow for any sort of termination condition on the "receive" block in the collector function, as we have no way to determine whether or not all traversals through neighbors in the network have completed. Therefore, our final design has each process, when spawned on a node in the network, send a message to the collector function with the *number of neighbors* the current source node has. Since each process has a contract with the collector function that it will return *at least one* message (either *num_neighbors*, *failure,* or *success*), we can keep track of the number of messages we've received with respect to the total number of nodes that will be traversed in the search process. This gives us a method for finding paths under length *n* from node A to B in a network, which was our desired goal.

Data and Algorithms

We plan to represent the Wikipedia data first using the SQL schema provided by the WikiMedia foundation for the raw Wikipedia database, and next as an adjacency list via a cleverly-designed SQL schema that we're still working out the details for. We envision a many-one relational system, but will have to spend a small amount of time studying how graphs are typically represented using SQL database. This isn't the focus of our project and thus we're likely going to use a standard solution for such things.

The algorithm for actually solving the problem we've set out to solve in this project is given in some detail above, though it misses a couple trivial implementation details. It is essentially complete and is at the implementation stage.

External Packages

We plan to use a web/API framework for Erlang called ChicagoBoss, which is hopefully going to alleviate our concerns about having to deal with network connections and handling requests. It appears to be a robust framework that will allow us much flexibility in the other parts of our algorithm.

Development Plan

We plan to parallelize the work on this project, giving a clearly defined “module” of work to each individual which then will come together into the final project with checkpoints along the way. The tasks which are of a dividable size are:

By Nov. 9:

- Download and store Wikipedia database
- Research methods for extracting network from Wikipedia SQL database download
- Research and implement SQL schema for representing network in SQL

By Nov. 16

- Implement (Python) the algorithm for moving from wikipedia representation to our network representation
- Download and understand ChicagoBoss web framework
- Design initial endpoint specifications for ChicagoBoss setup
- Research hosting options for web front end

By Nov. 23

- Design simplest form of web front end interface that describes results (non-interactive)
- Implement in JavaScript the querying of the Erlang back end from front-end clients

By Nov. 30

- Define modules for collector and path_finder algorithms as designed above
- Clarify connecting modules between database, path_finder algorithm, and web front-end.

Extra Tasks

- Make a force directed graph display for web front end.