COS 350

Algorithms and Data Structures

Term Project

Andrew Cramer, Katelyn Manzo, Evan Sampson

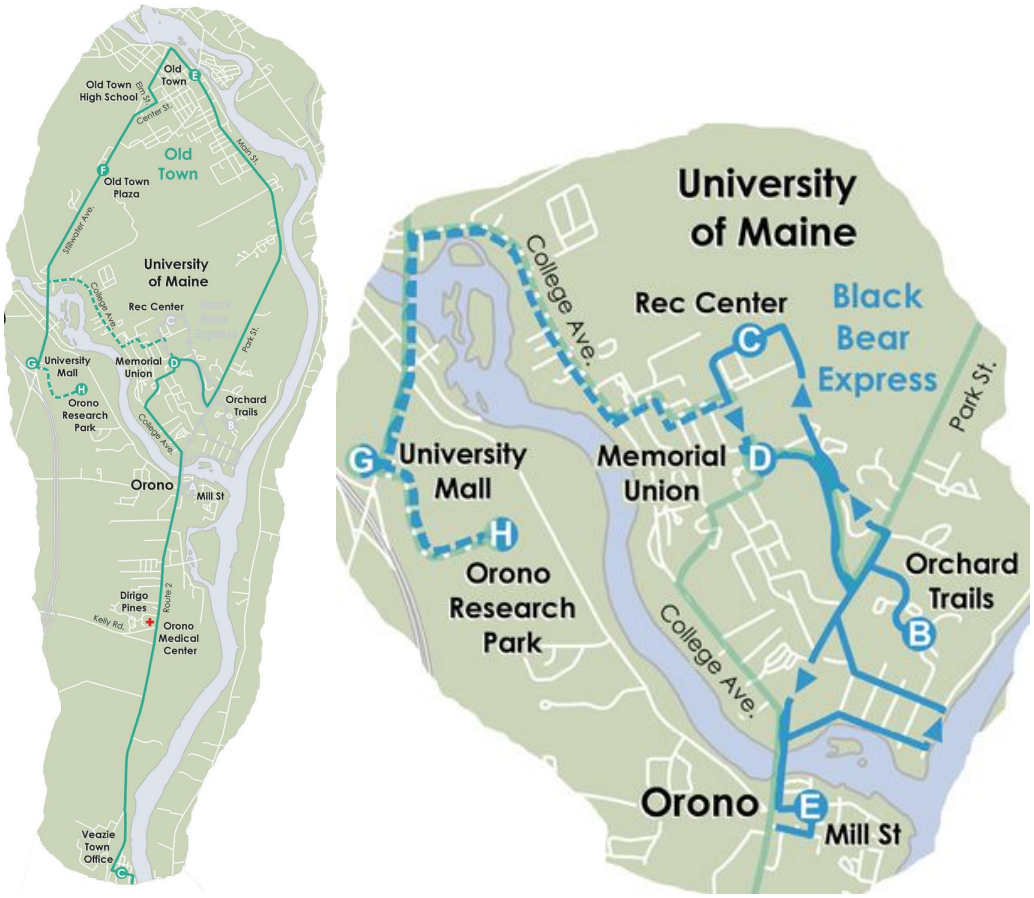
Community Connector Calculator

For our Algorithms and Data Structures term project, we wanted to create a program that would build and trace through a graph. We did not want the graph to have arbitrary data from a randomized generator, so instead we constructed one based off of the local buses in the greater Orono, ME area. The type of graph created is a directed graph (digraph) of bus stops. Bus routes have a few variables such as distance between stops, time between stops, and the cost of each stop. Since UMaine students ride for free the cost won't be factored here, and distance will be replaced by the time between each stop in minutes rounded off by ceiling. We treat the variables as such: the bus stops would be our vertices and the time between the stops would be the weighted value of the edge.

The algorithm we used to compute all pairs shortest paths of a set of bus stops was the Floyd-Warshall algorithm method. We created an adjacency matrix that would have the set of vertices (bus stops) and the set of given (already known) edge values between vertices. The Floyd-Warshall algorithm is being used to compute the shortest path for all pairs. To achieve an implementation of Floyd-Warshall, we had to design a class for adjacency matrices, as well as a way to construct a graph into an adjacency

matrix. The algorithm is then performed on the adjacency matrix, rather than the abstract graph. The expected runtime for the algorithm, all cases, is $\Theta(|V|3)$, where V is the set of vertices, and $|V|$ is the cardinality of that set.

In our local bus route we used two bus routes with different number of stops, and connect them to each other at certain stops, such as the stop at University Union. The University Union is stop D when looking at the below graphs.



The adjacency matrix would display the shortest path between all stops. A key would be outputted as well for the user to tell which bus he is taking at the start, as well

as if there was a bus change in the process. An example of that would be from the Bangor Depot to the UMaine Rec Center. The fastest route outputted would be (A,B,C,D,I,J,), using the key in the console, would then tell you which stops are needed to take in order to reach your destination in the fastest time. We feel this algorithm is the most efficient for finding the quickest path to any destination of the user's choice. A future step would also to be to calculate the fastest route as well as the fewest stops encountered on the way, if the user wasn't a fan of stopping constantly on the way.