Alex Kwong

Wiring Description

The goal of this problem was to return the cost of a minimum spanning tree of J vertices and C edges using Kruskal's algorithm given certain restraints on which edges can be used. I created an Edge class with fields for two vertices and the edge weight and a set class with fields for parent and rank. The find-union data structure is needed for Kruskal's algorithm. In order to implement the find-union data structure, the functions make set, find, and union were implemented where union by rank and path compression were utilized.

While reading in inputs, I kept track of which junction type corresponded with each name, which switch lights depended on, and assigned IDs to names using 3 separate maps. I approached the problem by splitting the edges into 3 groups. A left group which consisted of edges between breakers, junction boxes, and outlets, a right group of edges between lights and switches, and a middle group of edges between the left and right groups. When adding edges to each group, I filtered out the bad edges using conditionals such as if lights were in the correct switch group or if an edge that connected a vertex that belonged to the left group to a vertex that belonged to the right group. This resulted in 3 vectors which contained valid edges for each group. The vectors were then sorted in order so that the lowest weighted edges were at the beginning. Reading in the J inputs took $\Theta(J)$ time and reading in the C inputs took $\Theta(C)$ time. While there were 3 different vectors, the total amount of edges within all of them totaled to less than or equal to C edges so sorting the vectors took $\Theta(C\log(C))$ time.

Creating the set took $\Theta(J)$ time. Then, Kruskal's algorithm was run on the left and right groups of edges, resulting in two minimum spanning trees. After, Kruskal's algorithm was run on the middle group of edges, combining everything into one minimum spanning tree. The final cost results from the costs from the first two Kruskal's algorithm runs and adds that to the cost of the final Kruskal's algorithm run. Similar to the 3 vectors, although Kruskal's algorithm was run 3 times, the edges and vertices of each group sum up to the total J vertices and C edges or less if edges were filtered out.

Vectors were used instead of a priority queue in this implementation of Kruskal's algorithm. The edge was set to the first element in the vector each iteration which runs in $\Theta(1)$ time; however, resizing the vector runs in $\Theta(C)$ time and the vector has to be resized J times which results in $\Theta(C*J)$ time. Find and union using path compression and union by rank run in constant amortized time. This results in $\Theta(C*(C*J)+C*1)$ time which comes out to $\Theta(C^2*J)$ time. Combining everything from reading in inputs to running Kruskal's algorithm results in $\Theta(J+C+C\log(C)+J+ C^2*J)$ where C^2*J is the fastest growing term, so over all this runs in $\Theta(C^2*J)$.