Written by: Gabriel Arellano

Function Descriptions

A. `divBy3and5` - returns a list of all the numbers between 1 and 1000 that are divisible by 3 and 5.
B. `objective1` - sums the list generated by divBy3and5.
C. `sumOfSquares x` - returns the sum of all squares from 1 to x.
D. `squareOfSums x` - returns the value of the square of the sum of numbers 1 to x.
E. `objective2 x` - calculates the difference between the sumOfSquares and the squareOfSum methods when provided the given parameter x.
F. `replace index value lst` - a method that replaces the content of the provided list at the element with the index provided with the value passed.
G. `fetch_from_index index lst` - a method that returns the element from the list at the provided index.
H. `strip_column n matrix` - this method was designed for taking the nth column from a list of lists, which is the intended representation of a matrix.
I. `fetch_val (x,y) matrix` - this method returns the entry from the provided matrix (or list of lists) at the position (x,y) which translates to the xth element of the yth row.
J. `replace_mtrx_value (x,y) value matrix` - this method replaces the value within a matrix at the location (x,y) or the xth element of the yth row.
K. `get_row_edges row_index row col_index` - this method takes a row of values from a matrix and returns a list of tuples that represent those values and their location within the matrix in the form (value, (x,y)).
L. `get_column_edges col_index column row_index` - this method takes a column of values from a matrix and returns a list of tuples that represent those values and their location within the matrix in the form (value, (x,y)).
M. `find_shortest_length node1 node2 matrix` - this method recursively solves the shortest distance problem of finding the shortest distance from node1 to node2 within an undirected weighted graph represented in the form of a symmetric adjacency matrix.
N. `get_distance_matrix matrix node_list` - this method runs the Floyd-Warshall (FW) algorithmon the given matrix until the node_list is empty (meaning that all nodes were already evaluated). Also processes the entries from the matrix into edges using the aforementioned get_ _edges methods.
O. `floyd_warshall matrix row column` - part of the FW algorithm, this method recursively evaluates each row within the matrix by updating the distance matrix for every entry within the provided row until it is empty.
P. `floyd_warshall’ matrix entry column` - part of the FW algorithm, this method checks for each row entry provided that it is not 0 (meaning the distance from the node to itself) or infinity (meaning the distance between the two nodes is undefined). If it is not either of those cases, then the value for the entry will be evaluated against the column provided until it is empty.
Q. `floyd_warshall’’ matrix row_entry col_entry` - the final piece of the FW

algorithm, this method checks if the distance of the combined edges between the three nodes i, j, and k provided by the row_entry and col_entry parameters, which share a vertex such that j is an intermediate vertex on a path from i to k, is smaller than the current entry within the distance matrix from i to k. In the case that the current entry is smaller than the path i-j-k, then the matrix is unchanged, otherwise the entry is changed to the length of i-j-k.

For more details on the Floyd-Warshall algorithm, please see either Wikipedia (http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm) or the excellent set of lecture notes I found from a private Japanese university (http://cis.k.hosei.ac.jp/~rhuang/Miccl/Algorithm3/lect24-floyd-warshall.pdf).

How to execute the methods for the assignment

Further explanation of the algorithms used are provided in the previous section, but to run the Haskell code, simply load the GArellano-project3.hs file into GHCI and run the following commands for the specified project objective:

- `objective1`
  Returns the value that satisfies the first objective.
- `objective2 x`
  Returns the value that satisfies the second objective for the project for the given parameter x, which should be of type Integer.
- `find_shortest_length node1 node2 matrix`
  Returns the value that satisfies the third objective provided:
  1. two integers representing the node numbers for the start and end nodes of the shortest path to be found. Note that I am considering matrices with indices from 1 to n, which is not the same as the habitual computer programmer perspective of indices starting with zero and ending at (n-1).
  2. a symmetric adjacency matrix in the form of a list of lists of integers (i.e. [[Int]]). Distances that are unknown are to be represented by infinity, which I computed by dividing one by zero. I provided some sample graphs within my code to reflect this design; although the matrices appear as a list of lists of integers, the inclusion of infinity (or 1/0) forces the matrices to become a list of lists of doubles. I will stress the need for the matrix to be symmetric, as the assignment was over undirected weighted graphs, which can only be represented by a symmetric adjacency matrix. If the matrix is not symmetric, then the algorithm will not work properly.