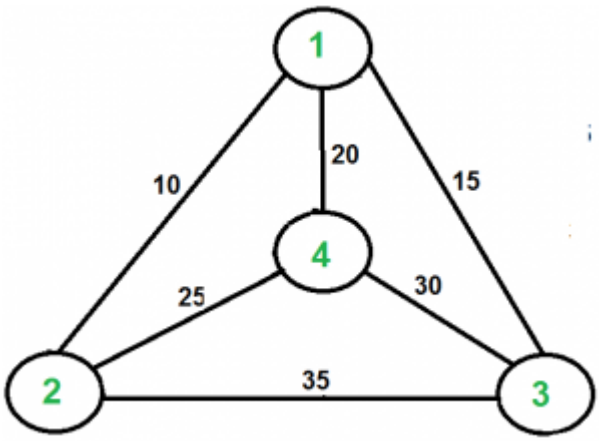
Traveling Salesman Problem

The description of the problem and its use case is well described on this Wiki page.

This application provides an interface to start with an example complete graph (shown below), computes the minimum hamiltonian route cost of the given complete graph and add/remove nodes from the graph and recompute the route cost.



To use the application, unpack the provided zip file and build and run using the instructions below.

Dependencies

The application has dependencies that will be available with out of the box Ubuntu 20.04 operating system. They are as follows,

- *Operating system*: Ubuntu 20.04 or equivalent
- *CMake* 3.8.10 or above
- *gcc/g++* 7.5.0 or above

Build

The application builds using **CMake** (minimum required version is **3.8.10**). The folder structure of the application is,

```
.:
build.sh  CMakeLists.txt  include  README.md  run.sh  src

./include:
graph.h

./src:
graph.cpp  tsp_main.cpp
```

Within the root directory of the application, the source files are in *./src* directory and the included headers are in *./include*. Two scripts *build.sh* and *run.sh* are provided for building and running the application.

Open a terminal window and follow the steps below,

- Make sure the scripts have execute permissions by running the command `ls -l`
- Run the provided build script using the command `./build.sh`

Run

Open a new terminal window if not following from the "Build" section and execute the following commands from within the root directory of the application,

- Run the provided run script using the command `./run.sh`

The application should start up and the interactive shell console will show on your terminal window as shown below,

```
The following options are available for interacting with this program:
'h': Print this help message
'p': Print current graph
'c': Compute fastest route
'a': Add new node
'r': Remove existing node
'q': Exit program

Enter command here:
```

To proceed, enter one of the available commands. FOr example, to print the pre-loaded graph and compute the fastest route enter the command `p` followed by `c` as shown below,

```
Enter command here: p
Entered command is: p

Graph:
        Nodes:  [1, 2, 3, 4, ]
        Edges:  * [v1 -> v2]: distance
                * [4  ->  3]: 30
                * [4  ->  1]: 20
                * [4  ->  2]: 25
                * [3  ->  4]: 30
                * [3  ->  1]: 15
                * [3  ->  2]: 35
                * [2  ->  4]: 25
                * [2  ->  1]: 10
                * [2  ->  3]: 35
                * [1  ->  4]: 20
                * [1  ->  2]: 10
                * [1  ->  3]: 15

Enter command here: c
Entered command is: c

Fastest route: 1 -> 2 -> 4 -> 3 -> 1
Distance: 80

Enter command here:
```

Note: The edges shown in the graph display are redundant because of the bi-directional nature of the edges in the graph.

You can add/remove nodes from the graph using the commands `a` and `r`. A demonstration of the add command is shown below,

```
Enter command here: a
Entered command is: a

Enter an integer number to add a new node: 9
Entered node number is: 9
Enter a positive integer for the distance between nodes 9 -> 1: 15
Adding edge: [9  ->  1]: 15
Enter a positive integer for the distance between nodes 9 -> 2: 25
Adding edge: [9  ->  2]: 25
Enter a positive integer for the distance between nodes 9 -> 3: 30
Adding edge: [9  ->  3]: 30
Enter a positive integer for the distance between nodes 9 -> 4: 10
Adding edge: [9  ->  4]: 10
Updated Graph:
        Nodes:  [1, 2, 3, 4, 9, ]
        Edges:  * [v1 -> v2]: distance
                * [9  ->  4]: 10
                * [9  ->  3]: 30
                * [9  ->  1]: 15
                * [9  ->  2]: 25
                * [4  ->  9]: 10
                * [4  ->  3]: 30
                * [4  ->  1]: 20
                * [4  ->  2]: 25
                * [3  ->  9]: 30
                * [3  ->  4]: 30
                * [3  ->  1]: 15
                * [3  ->  2]: 35
                * [2  ->  9]: 25
                * [2  ->  4]: 25
                * [2  ->  1]: 10
                * [2  ->  3]: 35
                * [1  ->  9]: 15
                * [1  ->  4]: 20
                * [1  ->  2]: 10
                * [1  ->  3]: 15

Enter command here: 
```

Node with ID 9 is added to the graph with the required edge weights. To recompute the fastest route after updating the graph, use the command `c`.

A demonstration of the remove command is shown below,

```
Enter command here: r
Entered command is: r

Current Graph:
        Nodes:  [1, 2, 3, 4, 9, ]
Enter node number to remove: 3
Entered node number is: 3
Updated Graph:
        Nodes:  [1, 2, 4, 9, ]
        Edges:  * [v1 -> v2]: distance
                * [9  ->  4]: 10
                * [9  ->  1]: 15
                * [9  ->  2]: 25
                * [4  ->  9]: 10
                * [4  ->  1]: 20
                * [4  ->  2]: 25
                * [2  ->  9]: 25
                * [2  ->  4]: 25
                * [2  ->  1]: 10
                * [1  ->  9]: 15
                * [1  ->  4]: 20
                * [1  ->  2]: 10

Enter command here: c
Entered command is: c

Fastest route: 1 -> 2 -> 4 -> 9 -> 1
Distance: 60

Enter command here:
```

Here Node with ID 3 is removed from the graph and the fastest route is recomputed using the command `c`. Note that there needs to be a minimum of 2 nodes in the graph at any point and the application will not allow the user to do the same.

```
Enter command here: r
Entered command is: r

Current Graph:
        Nodes:  [3, 4, 5, ]
Enter node number to remove: 5
Entered node number is: 5
Updated Graph:
        Nodes:  [3, 4, ]
        Edges:  * [v1 -> v2]: distance
                * [4  ->  3]: 30
                * [3  ->  4]: 30

Enter command here: r
Entered command is: r

Minimum 2 nodes need to be present in the graph. Cannot remove any nodes at the moment
```

To exit the application, enter `q` at the command prompt and the application will exit cleanly.