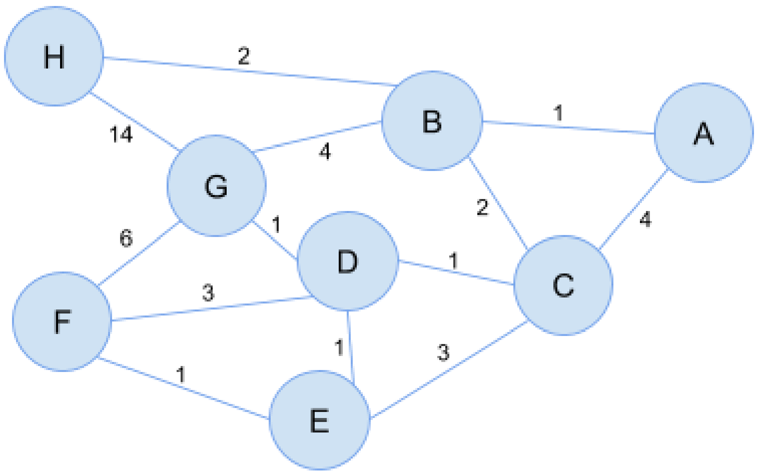
Exercise to deliver 2

Dijkstra’s algorithm

Given a graph G and an initial node N, Dijkstra’s algorithm finds the path with minimum cost from N to every other node in G.

In the solution of the Dijkstra’s algorithm given in the virtual campus, the following graph is represented with an adjacency matrix:



If we compile and execute the source file *“Dijkstra.cpp”* found in the virtual campus, we will see the following output in the console:

```
A
B <- A
C <- B <- A
D <- C <- B <- A
E <- D <- C <- B <- A
F <- E <- D <- C <- B <- A
G <- B <- A
H <- B <- A
```

Each one of the previous lines shows the minimum path to reach every node in the graph, starting from node A. Notice that the traversal of nodes goes from right to left, being A the rightmost node, and being the destination node the leftmost one.

In the presented code, the results of the algorithm are stored in the two arrays *prev* and *dist*. *Prev* is an array that tells, for each node N in the graph, which is the previous adjacent node with minimum cost to reach N. On the other hand, the array *dist* stores which is the cost of the best path to each node N in the graph.

Parallel Dijkstra

Adapt the sequential Dijkstra's algorithm so that it is multithreaded. The algorithm will have to execute a thread for each node in the graph concurrently.

You will have to create a project and complete some parts in the body of the source file "*Parallel_Dijkstra.cpp*" with the following steps. **You will identify the parts to complete that have the commented lines with the keyword *TODO***.

More concretely, the main program will be in charge of:

1. Initializing the arrays with the minimum distance and previous node (dist and prev).
2. Create a thread for each node.
3. Notify the initial node (with an event) that it needs to start dijkstra's algorithm.
4. Wait for a while until the algorithm has propagated completely.
    a. Of course, this is a simple end condition that could be improved. We don't know exactly how long the algorithm will take, so there are better ways to set the ending condition.
5. Notify all nodes (with an event) to finish their execution.
6. Wait for all nodes to end.
7. Print the solution for all nodes.

Furthermore, each one of the node's threads will have to:

1. Wait for an update notification (an event) from another node thread (or from the main program thread, in the case of the initial node).
2. If the notified event provides a path with more optimum cost to the current node, then, notify a new update event to to all adjacent nodes.

This way, the algorithm will propagate the shortest path in a few steps. The program's output should be the same as the solution provided by the sequential algorithm:

```
A
B <- A
C <- B <- A
D <- C <- B <- A
E <- D <- C <- B <- A
F <- E <- D <- C <- B <- A
G <- B <- A
H <- B <- A
```

NOTE: Use the threading functions and objects given by the the C++ standard library, as explained in the document of this session. They mainly provide mechanisms to create and destroy *threads*, *mutex* and *unique_lock* objects to protect the concurrent access to critical sections, and *codition_variable* objects for the communication (events) among threads.