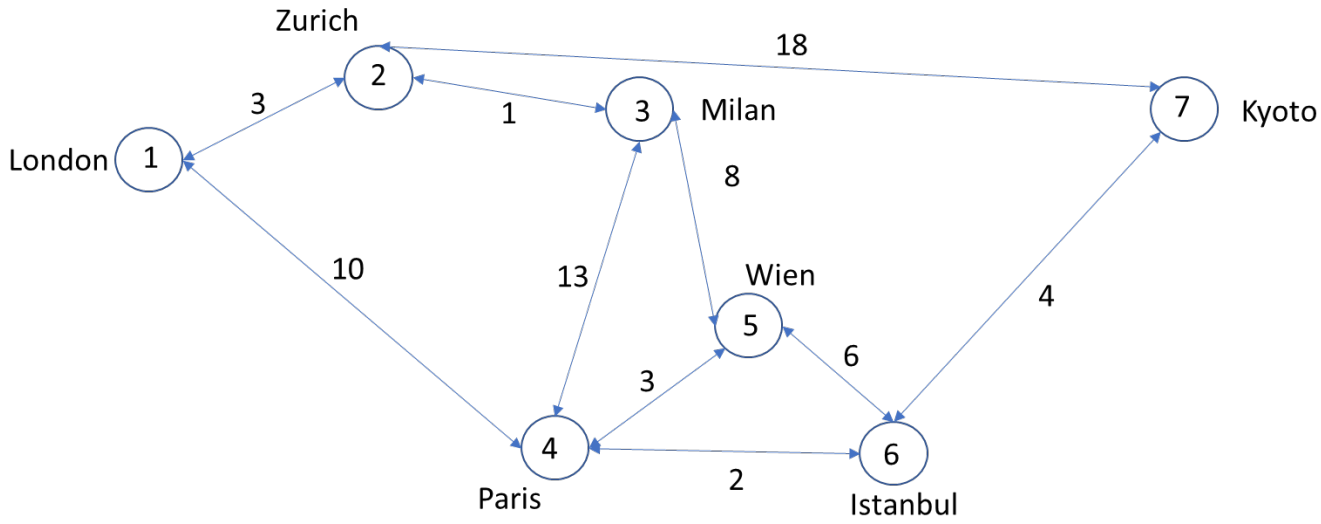
**Dijkstra path optimization algorithm**




Definitions:

An **arc** is a flight, defined by a couple of **nodes** *(i, i+1)* (airports), and a **cost** (e.g. fee to be paid);

Problem:

Try to find the cheapest way to reach Kyoto from Milan, using the provided nodes and arcs. Solve the exercise by applying Dijkstra path optimization algorithm.

Data:

- a set of **nodes**

Node 1 = London

Node 2 = Zurich

Node 3 = Milan

Node 4 = Paris

Node 5 = Wien

Node 6 = Istanbul

Node 7 = Kyoto

Initialization and meaning of variables in the given skeleton. To create the Matlab variable containing all the nodes, copy the command below in this way you will create a vector of strings containing the names of your nodes.

```
node = {'London', 'Zurich', 'Milan', 'Paris', 'Wien', 'Istanbul',
'Kyoto'};
```

- a set of **arcs**

An arc connecting two nodes is defined by 3 parameters:

Arc i: Start node, End node, Cost of the arc)

To create the Matlab variable containing all the arcs, copy the command below that creates a matrix of 3 columns containing all the parameters necessary to define the arcs of the exercise (note, in this case the univocal identifier of an arc is its row number!)

```
arc = [1 2   3; ...
       2 1   3; ...
       1 4  10; ...
       4 1  10; ...
       2 3   1; ...
       3 2   1; ...
       3 4  13; ...
       4 3  13; ...
       3 5   8; ...
       5 3   8; ...
       4 5   3; ...
       5 4   3; ...
       5 6   6; ...
       6 5   6; ...
       6 4   2; ...
       4 6   2; ...
       6 7   4; ...
       7 6   4; ...
       2 7  18; ...
       7 2  18; ];
```

Note: to simplify both the ways of the flights have the same price.

- ID of the departure node (Milan) and ID of the arrival node (Kyoto)

To create these two variables, You can set either manually or by the commands

```
first_id = find(strcmp(node,'Milan'));        % Departure node
final_id = find(strcmp(node,'Kyoto'));        % Arrival node
```

‘strcmp’ : function to compare strings; when it finds in your vector of strings the corresponding string you are searching (Milan or Kyoto) it returns TRUE and the ‘find’ function gives you the corresponding ID.

**Dijkstra algorithm**

We suggest to use at least the following variables, to be filled during the processing:

- *ttn*: vector of dimension (n_nodes, 1) containing the sum of the costs of all the arcs to be used to reach each node from the starting node

- *prev_id*: vector of dimension (n_nodes, 1) to store the predecessor of each estimated node in the minimal path

- *visited_id*: vector of visited / explored nodes (every time a new node is visited, this vector increase by 1 element)

- *id_to_visit*: vector of nodes already estimated, that can be visited / explored

1) initialize all the suggested variables

```
ttn = inf(size(node));

ttn(first_id) = 0;

prev_id = nan(size(node));

prev_id(first_id) = first_id;

visited_id = [];

id_to_visit = [ first_id ];
```

While the *final_id* is not member of the *visited_id follow guidelines given in the skeleton*

Other suggestions: read carefully basic help of the functions and use them:

- setdiff
- find
- ismember