# Labs 1 and 2

## Contents



## 1 Dijkstra's algorithm

Assignment: write a program which for a given graph $G = (V, E)$ finds the shortest-path distances between the starting vertex and all other vertices of the graph using Dijkstra's algorithm.

For your convenience, we have divided the task into multiple small steps. Download the file for this lab from the course labs website. Try to implement one step at a time. Do not try to overcomplicate things from the beginning, just solve one problem at a time and go step by step while making sure every step works well. Make sure that the code you are writing is as general as possible and that it will work with any adjacency matrix (remember that the final Dijkstra's algorithm should work on any given adjacency matrix).

### 1.1 Dijkstra's algorithm pseudocode

The algorithm is described in more detail in the following pseudocode:

**Algorithm 1:** Shortest-Path Distances (Dijkstra's algorithm)

```
   Input  : Weighted adjacency matrix A of G = (V, E) ;   // Step 1
   Output: Vector labels containing the shortest-path distances from
            vertex 1 to all vertices of G
 1 starting_vertex = 1 ;                                  // Step 2
 2 n = length(V) ;                                        // Step 3
 3 labels = ∞, ∀v ∈ V ;                                   // Step 4
 4 labels(starting_vertex) = 0 ;                          // Step 4
 5 unvisited_vertices(v) = v, ∀v ∈ V ;                    // Step 5
 6 for i = 1 TO n do
 7 |   current_label = min_{v∈unvisited_vertices} labels(v);   // Steps 6-7
 8 |   current_vertex = vertex with label current_label ;  // Step 8
   |   /* Routine to update labels of vertices adjacent to
   |      current_vertex                                   */
 9 |   for each j in unvisited_vertices do
10 |   |   weight = weight of edge current_vertex, j;
11 |   |   if weight > 0 then
12 |   |   |   new_label = current_label + weight;
13 |   |   |   if new_label < labels(j) then
14 |   |   |   |   labels(j) = new_label;
15 |   |   |   end
16 |   |   end
17 |   end
18 |   Remove current_vertex from unvisited_vertices;     // Step 9
19 end
   // Step 10:  display the results
```

This pseudocode is a bit different from the one presented in the lectures. It represents the same algorithm, but it is modified to guide you and help you with the implementation.

## 1.2 Dijkstra's algorithm steps hints

### Step 1

Create a directed weighted graph $G$ with at least six vertices using an adjacency matrix $A$ (use the edge weight as the corresponding matrix element instead of using 1 merely to indicate that an edge exists). If there is no edge, use 0[1]. Use positive integer values for weights. Plot the graph.

### Step 2

Create a variable *starting_vertex* that specifies the source vertex (Dijkstra's algorithm will find shortest paths from the source to all other vertices in the graph). Initialize this variable with 1.

### Step 3

Determine the number of vertices in the graph and store it in the variable $n$. Your code should work for any adjacency matrix.

### Step 4

Create a vector *labels* to keep labels of Dijkstra's algorithm for the given graph. The elements of this vector store the current labels, and each index corresponds to a vertex number. Initialize it with infinite values for each vertex (Hint: the infinite value in MATLAB is 'Inf'; you can initialize the vector using the function inf(n, 1)). Change the value for the starting vertex to be 0.

### Step 5

Create a vector *unvisited_vertices* that contains the numbers of the unvisited vertices. Initialize it with all vertices, that is with numbers from 1 to $n$.

[1]Please be aware that, theoretically speaking, it is better to use $\infty$ in the adjacency matrix if there is no edge. Or, even better, have weights stored in a separate vector and use just the ordinary 0/1 adjacency matrix. This way we ensure that our algorithm works with edges that have weight 0, which are supported by Dijkstra's algorithm (Dijkstra's algorithm is not guaranteed to produce an optimal solution when the graph contains edges with negative weights). However, to simplify our implementation we suggest sticking to the assumption that our program does not support edges with weight 0 and use an adjacency matrix in which 0 represents the lack of an edge.

**Step 6**

At the beginning of each iteration, create the vector *unvisited_vertices_labels*, which contains the current labels of the unvisited vertices. Obtain these values from *labels* using the indices stored in *unvisited_vertices*.

**Step 7**

Find the minimum current label, that is the minimum element of the vector *unvisited_vertices_labels* and save it to the variable *current_label* and its index to the variable *current_index* (you can use a MATLAB built-in function to find the minimum element of the vector and its index).

**Step 8**

Find the vertex that currently has the minimum label and save it to the variable *current_vertex*. Hint: use *unvisited_vertices* and *current_index*.

**Step 9**

Remove the found vertex from *unvisited_vertices*. Hint: *current_index* is still useful.

Implement the routine to update labels of vertices that are adjacent to the currently visited vertex (it is between steps 8 and 9 in the pseudocode).

**Step 10**

Add code to display the output of the algorithm.

*** Optional task**

Modify your program so that, in addition to the shortest-path distances, it displays the corresponding shortest paths as sequences of vertices from the starting vertex to each of the other vertices.