Lab 1 and 2

Contents

1 Dijkstra's algorithm

Assignment: write a program which for a given graph $G = (V, E)$ finds the shortest paths 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 details in the following pseudocode:

Algorithm 1: Shortest Path (Dijkstra's algorithm)

Input : Weighted adjacency matrix A for graph $G = (V, E)$;
// Steps 1 and 2

Output: Vector *labels* (the shortest paths from vertex 1 to all vertices of G)

```
1  starting_vertex = 1; ;                                   // Step 3
2  labels = ∞, ∀v ∈ V ;                                     // Step 4
3  labels(starting_vertex) = 0 ;                            // Step 4
4  unvisited_vertices(v) = v, ∀v ∈ V ;                      // Steps 5-6
5  for i = 1 TO length(V) do
6  |  current_label = min label among unvisited vertices ;  // Steps
   |   7-8
7  |  current_vertex = vertex with label current_label ;    // Step 9
   |  /* Routine to update labels of vertices adjacent to
   |     current_vertex                                     */
8  |  for j = 1 TO length(V) do
9  |  |  weight = weight of edge current_vertex, j;
10 |  |  if weight > 0 then
11 |  |  |  new_label = current_label + weight;
12 |  |  |  if new_label < labels(j) then
13 |  |  |  |  labels(j) = new_label;
14 |  |  |  end
15 |  |  end
16 |  end
17 |  unvisited_vertices = unvisited_vertices \{current_vertex} ;
   |   // Step 10
18 end
```

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 minimum 6 vertices using adjacency matrix A (instead of '1' if the edge exists, use a weight value for the edge). If there is no edge, use 0[1]. Use positive integer values for weights. Plot the graph.

Step 2

Create vector *vertices* to keep the graph vertices. Initialize it with numbers from 1 to n, where n is the number of vertices.

Step 3

Create variable *starting_vertex*, which fixes a single vertex as 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 4

Create vector *labels* to keep labels of Dijkstra's algorithm for the given graph. The elements of this vector will keep the current labels, the indices are the numbers of vertices. Initialize it with infinite values for each vertex (Hint: infinite value is 'Inf' in MATLAB, initialize the vector with ones and then multiply by Inf). Change the value for the starting vertex to be 0.

Step 5

Create a logical vector *mask_unvisited*, which indicates what vertices have not been visited yet (the number of elements is equal to the number of vertices and initial value for each vertex is *true*).

[1]Please be aware that, theoretically speaking, it is better to use ∞ 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 only if there are edges with negative weights in the graph). However, to simplify our implementation we suggest to stick to the assumption that our program does not support edges with weight 0 and has the adjacency matrix where 0 represents the lack of an edge.

Step 6

Create vector *unvisited_vertices*, which contains unvisited vertices (numbers). Initialize it with all vertices (using the logical vector from the previous step).

Step 7

Create vector *unvisited_vertices_labels*, which contains labels for unvisited vertices. Initialize it using vectors from Step 4 and Step 6.

Step 8

Find the minimum current label, that is the minimum element of the vector from Step 7 and save it to variable *current_label* (you can use MATLAB built-in function to find the minimum element of the vector).

Step 9

Find the vertex, which currently has the minimum label and save it to variable *current_vertex*. Hint: use vector from Step 7 and value from Step 8 to create a mask, then use function 'find' to find the index (use 1 as a second argument to get the first index if there are several, i.e., find(mask, 1)), finally apply the resulting index to vector from Step 6.

Step 10

Change the mask for unvisited vertices (vector from Step 5) so, that the found vertex is not considered in the next iteration of the outer loop.

Finally, implement the routine to update labels of vertices that are adjacent to the currently visited vertex (it is between steps 9 and 10 in the pseudocode).

Add the code which displays the result to the end of the script.