Algorithms 2023-24

Degree in Computer Science Engineering

Practical 4

Graphs: shortest path problem

Submission deadline: Saturday, 25th November

A pseudocode follows to calculate the shortest path from each vertex to the rest of the vertices in a weighted graph, following *Dijkstra*'s algorithm. The argument is the graph's adjacency matrix, and the result is a table with the minimum distances from each vertex to the others.

```
procedure dijkstra( M[1..n,1..n], Distances[1..n,1..n] );
  for m := 1 to n do
    unvisited := { 1, 2, ..., m-1, m+1, ..., n };
    for i := 1 to n do
      Distances[m, i] := M[m, i]
    end for
    repeat n-2 times:
      v := node from unvisited that minimizes Distances[m, v];
      unvisited := unvisited - { v };
      for each w in unvisited do
        if Distances[m, w] > Distances[m, v] + M[v, w]
        then Distances[m, w] := Distances[m, v] + M[v, w]
        end if
      end for
    end repeat
  end for
end procedure
```

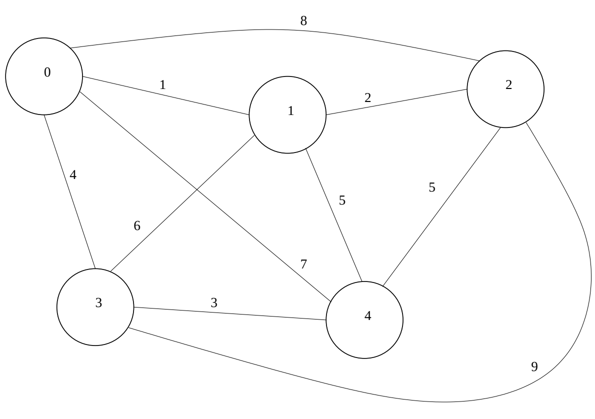
You must:

1. Implement the presented algorithm in C (Figure 1).
2. Validate the correct functioning of the implementation. In Figures 2 and 3 we propose two test cases.
3. Using the functions in Figure 4 to randomly generate undirected graphs, calculate empirically the complexity of the algorithm for the calculation of minimum distances.
4. Submit the C code files and the `.txt` file with the report using the task *Practical 4 Submission* at the Algorithms page in `https://campusvirtual.udc.gal`. We remind you that the deadline to complete the task is on Saturday, 25 November, at 23:59, and once submitted, files cannot be changed. **All the students in a team must submit the work.**

```
typedef int ** matrix;

void dijkstra(matrix graph, matrix distances, int sz) {
  int n, i, j, min, v=0;
  int *unvisited = malloc(sz*sizeof(int));
  for (n=0; n<sz; n++) {
    for (i=0; i<sz; i++) {
      unvisited[i] = 1;
      distances[n][i] = graph[n][i];
    }
    unvisited[n] = 0;
   /*
    ...
    */
  }
  free(unvisited);
}
```

Figure 1: Part of the procedure `dijkstra`



(a) Graph

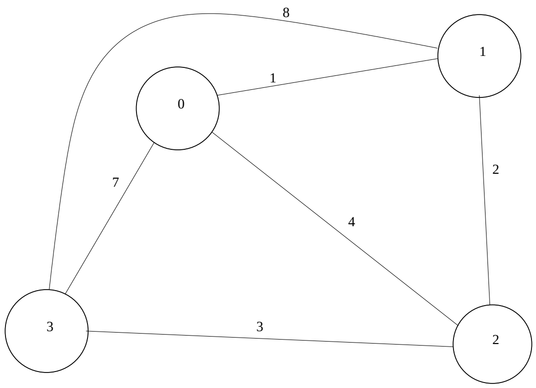
$$\begin{pmatrix} 0 & 1 & 8 & 4 & 7 \\ 1 & 0 & 2 & 6 & 5 \\ 8 & 2 & 0 & 9 & 5 \\ 4 & 6 & 9 & 0 & 3 \\ 7 & 5 & 5 & 3 & 0 \end{pmatrix}$$

(b) Adjacency matrix

	0	1	2	3	4
0	0	1	3	4	6
1	1	0	2	5	5
2	3	2	0	7	5
3	4	5	7	0	3
4	6	5	5	3	0

(c) Minimum distances

Figure 2: First example



(a) Graph

$$\begin{pmatrix} 0 & 1 & 4 & 7 \\ 1 & 0 & 2 & 8 \\ 4 & 2 & 0 & 3 \\ 7 & 8 & 3 & 0 \end{pmatrix}$$

(b) Adjacency matrix

	0	1	2	3
0	0	1	3	6
1	1	0	2	5
2	3	2	0	3
3	6	5	3	0

(c) Minimum distances

Figure 3: Second example

```
#define MAX_SIZE 1000
matriz createMatrix(int n) {
  int i;
  matrix aux;
  if ((aux = malloc(n*sizeof(int *))) == NULL)
    return NULL;
  for (i=0; i<n; i++)
    if ((aux[i] = malloc(n*sizeof(int))) == NULL)
      return NULL;
  return aux;
}
/* Pseudorandom initialization [1..MAX_SIZE] of a complete undirected graph
   with n nodes, represented by its adjacency matrix  */
void initMatrix(matrix m, int n) {
  int i, j;
  for (i=0; i<n; i++)
    for (j=i+1; j<n; j++)
      m[i][j] = rand() % MAX_SIZE + 1;
  for (i=0; i<n; i++)
    for (j=0; j<=i; j++)
      if (i==j)
        m[i][j] = 0;
      else
        m[i][j] = m[j][i];
}
void freeMatrix(matrix m, int n) {
  int i;
  for (i=0; i<n; i++)
    free(m[i]);
  free(m);
}
```

Figure 4: The functions `createMatrix`, `initMatrix`, and `freeMatrix`