Dijkstra's algorithm implementation

C project - G3 n.3, numerical calc and programming [145725] AY 2020/2021

Cristian Merli

20/07/2021

Abstract

C-code implementation of Dijkstra's algorithm, inside a dedicated library to manage graphs. This library has also been extended so that a graph's structure could be allocated inside heap to test Dijkstra's algorithm. With the aim of getting a more user-friendly output, gnuplot takes care of plotting graphics to show the structure of the graph and the elaborated shortest path.

1 Project request

Dijkstra. Write a software which reads a graph and given two nodes, calculates the minimum path with Dijkstra's algorithm.

2 Introduction

This document has the main purpose of giving an overview of the project, deeping into theoretical aspects of Dijkstra's algorithm and how it has been implemented in C-code. While to have further details about technical aspects, there is the possibility to consult html documentation of the software (see 'Doxygen html documentation' section inside 'README.md' file).

3 Dijkstra's algorithm

Dijkstra's algorithm has been conceived in 1956, by a Dutch computer sientist called Edsger Wybe Dijkstra. The algorithm is capable of finding shortest path between two nodes (source and destination), inside a graph data structure. It has numerous applications in different fields: from gps-navigation (A* search), electrical/pipelines grids design, social networks suggestions, to AI applications as 'best-first search' (uniform cost search). This algorithm can be implemented in many different ways, adopting various data-structures and it comes in countless variants. In presented c-code library it has been chosen to take advantage of arrays, as data storing-structures. While as far as the algorithm itself is concened, it has been implemented in a variant to produce a shortest-path tree from source node, to all other nodes inside nodes collection-vector. That allows to be able to re-calculate more min-cost paths towards other nodes, without the need of running the algorithm again, if the source node remains unvaried.

4 Graph data-structure

As mentioned in the abstract, graph-library does not contain only Dijkstra's algorithm, but also a set of functions to allow graph data-structure management *(for further technical details, see doxygen html documentation)*. As briefly touched upon in previous chapter [3], for the purpose of storing **arches** and **nodes**, dynamic-memory vectors have been chosen. The major benefit of that consists of being able to resize the graph during runtime, adding or removing elements in allocated memory (**nodes and arches collection-vectors inside heap**). Regarding arches and nodes, they are structure mainly made up of pointers to memory cells of connected elements, in addition to element-name and eventual cost. Besides, nodes also have an additional pointer to an element of **dijkstra-dataset dynamic-memory vector**, which contains a set of informations on which Dijkstra's algorithm works on. This way once the algorithm has been executed, it is possible to recontruct the shortest path backwards (from destination to source node), then to be later turned into a forward-path.

5 Library testing

It is possible to test graph-library and Dijkstra's algorithm, through the developed main code (graph-test). There are two different testing options:

- **Prepared test:** calculte shortest path using Dijkstra's algorithm from pre-defined source to pre-defined destination nodes.
- **Personalized test:** calculte shortest path using Dijkstra's algorithm from specified source to specified destination nodes.

6 Gnuplot

In order to get a more user-friendly output, cdcdscds.