Dijkstra's algorithm implementation

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

Cristian Merli, id. 211384

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.

Contents

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 route planning (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 [Figure 1] management *(for further technical details, see doxygen html documentation)*. As briefly touched upon in previous chapter [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. In 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.

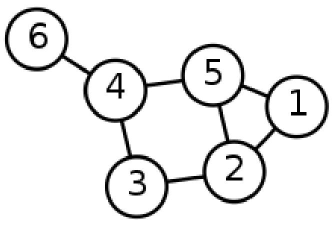

Figure 1: *source www.wikipedia.org*

5 Gnuplot

In order to get a more user-friendly output, gnuplot has been integrated in software to draw graph's graphics. More specifically, two gnuplot scripts are called and loaded from main-testing software, using system commands. The former loads data from three static (.dat) files, to **print the strcture of the graph** [Figure 2]; while the latter takes care of plotting loaded data from four dynamic (.dat) files, to **draw graph-structure and highlighting the detected shortest path** [Figure 3]. Dynamic-data files are manipulated by main-testing software, picking min-cost path plotting informations from the static files (targeting arches/nodes names of the shortest path inside files, to copy them).

6 Library testing

It is possible to test graph-library and Dijkstra's algorithm, through the developed main code (graph-test). In order to that, it creates a test-graph structure to try out and test graph-library. The first stage in graph creation, consists in arches and nodes allocation inside heap (names and costs dynamically taken from two vectors, containing streets and crosses info to make the example of a road network). During second stage, arches and nodes (streets and crosses) are connected together forming a graph-structure [Chapter 4]. Then, once the structure has been created, testing software will ask the user to choose which testing option to adopt. There are two different testing options:

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

After having the testing-mode choice carried out, the first gnuplot script [Chapter 5] will graphically display the allocated structure, to give the user a brief overview of the created network. By closing this screen, the user can proceed in selected testing mode. Consequently, Dijkstra's algorithm [Chapter 3] will be applied to detect all the min-cost paths, then reconstructing the specifically requested one. Here the second gnuplot script [Chapter 5] will be called aiming to display the allocated graph (gray coloured), highlighting the elaborated shortest path between source and destination nodes (specified during tesing option choice). After graphical representation is closed, the whole structure and all the dynamic memory allocated inside heap, will be cleared immediately before closing test software.

7 Conclusions

To sum up, in this particular test, Dijkstra's algorithm has been implemented to find shortest path between two crosses inside a road network, but the **library can be utilized in very different applications** thanks to the importance of the algorithm itself. That's because it still remains one of the best and most diffused algorithms in all its variants, to find shortest paths from a single-source node inside a graph. To conclude with a citation, "The Dijkstra's algorithm has its own shortcomings when seeking an optimal path between two points, but it has irreplaceable advantages." *(by DongKai Fan, Ping Shi)* [1]

8 Images

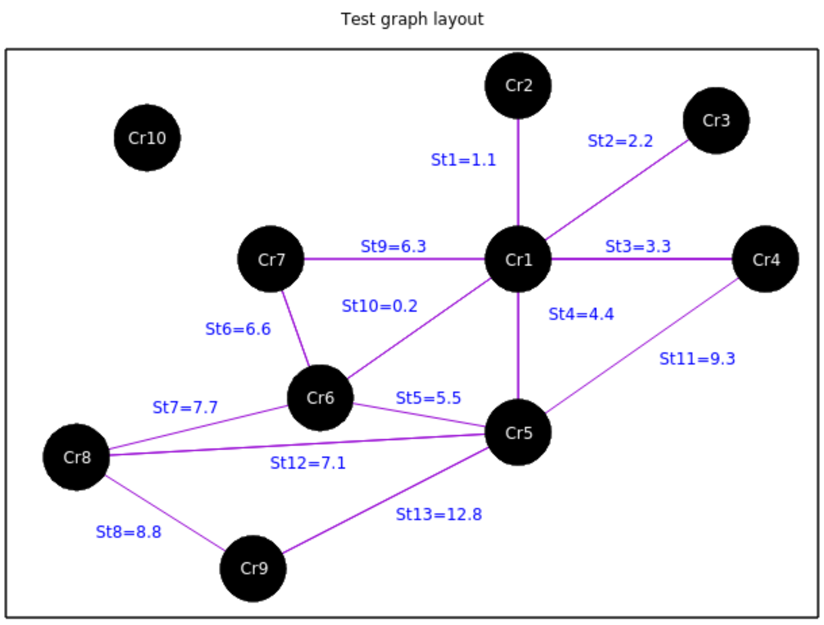

Figure 2: *Graph-structure (exported from gnuplot)*

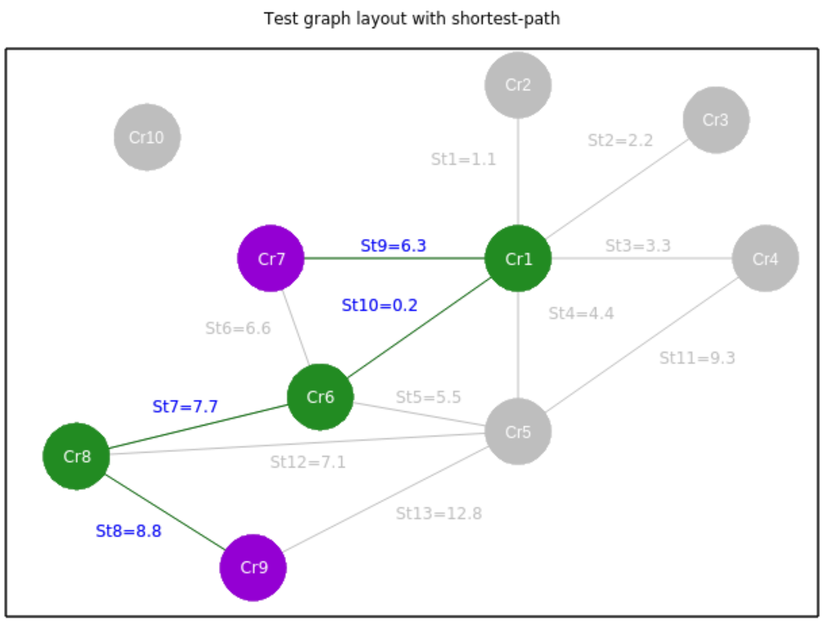

Figure 3: *Graph-structure with shortest path between 'Cross9' and 'Cross7' (exported from gnuplot)*

References

[1] DongKai Fan and Ping Shi. Improvement of dijkstra's algorithm and its application in route planning. In *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery*, volume 4, pages 1901–1904, 2010.