CS3D5A – Assignment 4 - Dijkstra's Algorithm

Anton Gerdelan gerdela@scss.tcd.ie and Peter Lavin peter.lavin@scss.tcd.ie

Due midnight **13 December**.
Weight: 1/4 of CA grade. 10% of module.

1 Overview

The goal of this assignment is to write a program that represents the graph, shown in Figure 1, as a data structure, and write C or C++ implementations of **Depth-First Search** and **Dijkstra's Algorithm**, which you will then call to traverse the graph.

Exact input and output requirements are given. Comment out any test code that produces other output before submission.

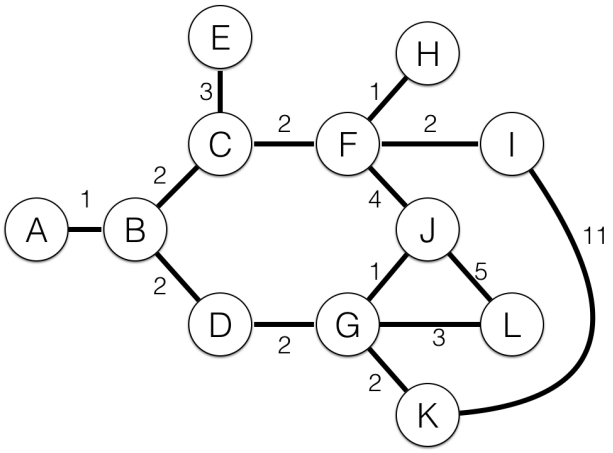
Traversal sequences should be correct with respect to your system for ordering the edges - therefore the correct sequence for your program might not exactly match the example output.

Do not use any pre-made graph data structures or algorithms from libraries e.g. from Boost.

2 Deliverables

- Any *C* or *C++* source code files used {.c, .cpp, .h, .hpp} with your name and date in comments at the top of every file.
- Do not include project settings files or compiled files.
- Work is individual. Do not copy-paste code from others or the web.
- No written report for this assignment.

Figure 1: Our non-directed, weighted graph with data shown as characters.



3 Task - Graph Representation

Store the graph in memory using any structure of your choice. You must correctly represent **all** the vertices and **all** the edges connecting them. Sets of edges and vertices, or adjacency lists using either matrices, linked lists, or arrays are all good choices.

4 Task - Depth-First Search

Write a Depth-First Search function with a declaration similar to:

```
bool dfs(Graph* graph, char start_at, char find_value);
```

The function should return `true` if the character given is found in the graph, and `false` if it is not.
Call this function two times, with inputs:

- From 'A' to 'X' (which does not exist in the graph)
- From 'K' to 'B'

The function should print to the terminal the data value of each node that it investigates during the search, in the following output form:

```
DFS from 'A': A B C D E F H I K G D J L
```

If the value is found the function should print `target found - HALT` and stop.

5 Task - Dijkstra's Algorithm

Write a function to compute Dijkstra's search algorithm - the basic version discussed in the lecture - with a declaration similar to:

```
bool dijkstra(Graph* graph, char start_at, char find_value);
```

The function should return `true` if the character given is found in the graph, and `false` if it is not.
Call the function two times, with inputs:

- From 'A' to 'X' (which does not exist in the graph). This should tour the graph - computing and printing the shortest path to every node from A.
- From 'A' looking for a path to 'J'.

Output should be printed in the form:

```
Dijkstra path from 'A' length 0
B length 1, parent A
C length 3, parent B
... (and so on)
```

Where each node that is marked **permanent** is printed with its cost and its parent vertex (the node before it in the path). Thus, we can work backwards to find the shortest path from A to any vertex in the graph. If the sought vertex is found and made permanent - print `target found - HALT` and stop.

6 Grading Scheme

- Your program must compile.
- Your program has a sound model for storing all of a graph's contents. **(2 Marks)**
- Your program produces correct output for Depth-First Search. **(2 Marks)**
- Your program produces correct output for Dijkstra's Algorithm. **(3 Marks)**
- Your program is robust to bugs and infinite loops, even if different start and end points are used in traversals. **(2 Marks)**
- Your program has appropriate comments that clearly explain your intentions. Your solution is easy to follow. It has consistent indentation and variable names, the number of functions and files is not excessive, and the solutions are not overly-complex. **(1 Marks)**

7 Suggestions

- Don't forget to write your name and the date into the top of every source file.
- Turn on all compiler warnings and re-write your code to silence them all. It's a good idea to try compiling with different compilers or analysis tools to catch additional bugs. Both Clang and GCC should be available in the Linux labs and have the same commands. NB: if you are using a Mac, typing `gcc` actually just runs Clang.
- Use `assert()` to validate your assumptions or function inputs.
- Try different search start and end points to make sure that your implementations are robust - don't stop when they seem to work - try to break them!
- You will have to think about program structure in this assignment. There is a risk of over-complicating your program, which means it will take you a very long time to write. If your solution is longer than 300 lines of code, think about compressing the code - are some loops or functions unnecessary or could be combined? Do you have too much structure? Have you made it easy for yourself to debug?
- If you are not comfortable with pointers, you might decide to re-write the function arguments, and make use of global variables for your graph structure instead. That is absolutely fine for this assignment.
- When your solution seems to work - double check your assumptions against the theory. It's easy to miss finer points of the algorithms.
- By all means ask for help with programming problems, tricky tasks, or clarifications to these instructions. You can paste confusing or broken snippets of your code into the discussion boards to ask for advice. Don't wait until the last few days, so that we have time to help you.