Analysis of Algorithms
Lab 8

Andrés Calderón, Ph.D.

April 2, 2025

1 Introduction

In this lab, we delve into some of the most fundamental and widely used algorithms in graph theory, with a special focus on shortest paths and flow networks. These concepts form the backbone of numerous real-world applications, from GPS navigation and logistics to network routing and resource allocation.

We begin by exploring Dijkstra's Algorithm, a classic method for finding the shortest path between nodes in a graph with non-negative weights. This hands-on section includes a video walkthrough to help reinforce your understanding. Then, we turn to key theoretical foundations through selected readings from the CLRS textbook, followed by exercises that deepen your insight into all-pairs shortest paths and network flow transformations.

Finally, we tackle a practical scenario involving multiple trucking companies sharing a road system, where the goal is to determine whether each company can reach a common destination using edge-disjoint paths. This challenge will test your ability to design efficient graph algorithms that solve real logistical problems.

Through this lab, you'll strengthen your algorithmic thinking and gain experience applying advanced graph concepts to both theoretical and practical problems. Let's get started!

2 How Dijkstra's Algorithm Works

In this first section, we will watch a video from the Spanning Tree YouTube Channel that provides a clear explanation of the well-known Dijkstra Algorithm for finding the shortest path between two nodes in a graph. The video is in English, but remember that you can enable the auto-generated subtitles and translate them into Spanish. You can access the video here. There is no particular hand-in for this section—just take notes and make sure you understand how the algorithm works. Enjoy!

3 Reading

In the next section of this lab, we will focus on reading a couple of sections from the CLRS book (4th edition). Specifically, you are asked to read Sections 23.1 Shortest paths and matrix multiplication and 24.1 Flow networks. You may extract these sections from the book and use any translation service to convert the text into Spanish if that makes you feel more comfortable. After your reading, you will answer the following exercises:

Exercises

1. Run SLOW-APSP on the weighted, directed graph from Figure 23.2 (in the book), showing the matrices that result from each iteration of the loop. Then, do the same for FASTER-APSP.

2. Show that splitting an edge in a flow network yields an equivalent network. More formally, suppose the flow network G contains the edge (u, v), and define a new flow network G' by creating a new vertex x and replacing (u, v) with the new edges (u, x) and (x, v), where $c(u, x) = c(x, v) = c(u, v)$. Show that a maximum flow in G' has the same value as a maximum flow in G.

4 Disjoint Roads

A number k of trucking companies, $\langle c_1, c_2, \cdots, c_k \rangle$, want to use a common road system, which is modeled as a directed graph, to deliver goods from source locations to a **common target location**. Each trucking company c_i has its own source location, modeled as a vertex s_i in the graph, and the common target location is another vertex t. (All these $k + 1$ vertices are distinct.)

The trucking companies want to share the road system for delivering their goods, but they want to avoid getting in each other's way while driving. Thus, they want to find k edge-disjoint paths in the graph, each connecting a source s_i to the target t. We assume that there is no problem if trucks from different companies pass through a common vertex.

Design an algorithm that the companies can use to determine whether such k paths exist, and if not, return *"impossible"*.

5 What Do We Expect?

You will compile all your answers into a well-structured report and submit it before the lab on **April 16, 2025**. Please submit your report in **PDF format** and email it to me with the subject line formatted as `[ADA] Lab 8`, and include your names in the body of the email.

Happy Hacking 😎!