CS124-and-a-bit, Session 1: Graphs, DFS, SCCs, BFS, SSSP

CS124 - Spring 2023

Andrew Holmes - 8 February 2023 - version 0.1

Contents

1 Introduction

Welcome to CS124-and-a-bit! This is a new initiative for this year that I am running alongside 124.5. Here's some information about what it is (or perhaps better, what I hope it becomes) and why I'm trying to introduce it.

1.1 What is CS124-and-a-bit

CS124-and-a-bit is going to be a **series** of **optional sections** focused on **implementing** some of the key algorithms, data structures and ideas from the class. Some key points:

- These sessions are planned to be **heavily problem focused**. I don't intend on lecturing extensively or just showing you coded implementations. I want it to be essentially a colloborative coding session working in small groups, and I will circle and help out.
- I tried to write problems to reinforce concepts from the class, rather than generally extending too far beyond the class material. If you want to review or understand the algorithms and data structures better, actually implementing and modifying them certainly seems like a good way of understanding them better.
- The problems will hopefully also be useful for you in the future for interview preparation or general experience etc. However, this is **not a pure Leetcode session**. I may steal a few questions from other websites to save myself some time writing questions and solutions, but the aim is to reinforce class material primarily.
- I'm currently aiming for sessions to last **60-90mins**, within which hopefully you can tackle a few problems. I will write my own solutions to as many problems as I have time to, and these will be shared afterwards.
- Currently I plan on running sessions on:
  1. Session 1: Graphs, DFS, SCCs, BFS, SSSP
  2. Session 2: Divide Combine, Dynamic Programming

1.2 Why CS124-and-a-bit

- CS124 is very theory focused. This is aimed to appeal to those of you who want to try to implement some concepts from class - whether to review understand the content, for interview preparation, bragging rights or to procrastinate the work you actually need to do.
- We know that the course moves fast, and fully understanding all the material as we go can be a challenge. This hopefully will act as a review session to grapple with topics you have seen but may not have fully processed yet.
- **CS124.5** has been traditionally aimed at extension topics **beyond** the syllabus of CS124. While we want to continue offering these opportunities, we know that CS124 is a lot of work without additional material. Only a small subset of students generally attend 124.5, so this is aimed at being an extension opportunity that is more appealing/open to the wider CS124 community.

1.3 Please remember

- This is a new initiative. It won't be perfect first time around, but I'd love your feedback to see whether this is a worthwhile initiative to continue in the future.
- There's **only one of me** working on this. It's extra time that I'm putting in to try to create something helpful. Please don't harass me if I make a mistake or typo etc, or you just don't think it was what you were looking for. :)

2 Graph problems

We've talked in class about different representations of graphs. These questions will be about working with graphs and their representations.

2.1 Switching graph representations

Problem 1: Switching graph representations

We generally assume we are given an adjacency list in this class. Sometimes it may be more helpful to have an adjacency matrix, or sometimes we might be provided an adjacency matrix when we desire an adjacency list. Let's implement code to switch between graph representations:

a) Implement **list_to_matrix** to convert an adjacency list to an adjacency matrix.

b) Implement **matrix_to_list** to convert an adjacency matrix to an adjacency list.

3 DFS problems

3.1 Finding back edges

Problem 2: Finding back edges

We claimed in class that we could use DFS to find cycles by finding back edges in a graph. Let's actually try to implement this:

a) Implement **find_back_edges** to find back edges in a graph. You may assume edges is an adjacency list storing all the edges in the graph. *You may add* vertices *as an input to your function if you would like.*

b) Add another test to check what back edges your code finds on a different graph, or from a different source in the graph.

c) What is the runtime of your implementation?

3.2 Number of islands

Problem 3: Number of islands

Let's practice applying DFS to a problem where we might have to model the graph ourselves.
For this problem you are given an $m \times n$ grid, where each position on the grid is either a 0 or a 1. 0 corresponds to water, 1 corresponds to land. The challenge is to count the **number of islands** present in the grid. An island is formed by pieces of land that lie adjacent to each other (horizontally or vertically, **not diagonally**). For example:

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

This grid has two islands. The bottom left piece of land is its own island, and there is a central island formed by the other 1's.

a) Implement **count_islands**. You may assume that the grid provided is rectangular (each row is of the same length).

b) What is the runtime of your algorithm?

c) What is the space complexity of your algorithm? Can you do better?

Problem 4: Lowest Common Ancestor of a Binary Tree

Given a binary tree and two values that are in the tree, find the lowest common ancestor node of the two values.
FINISH THIS

4 Strongly Connected Components

4.1 Kosaraju's algorithm

Problem 5: Kosaraju's SCC algorithm

We have seen in class an algorithm to find SCCs (called Kosaraju's algorithm). Here we will implement it!

a) Implement **find_SCCs** using Kosaraju's algorithm to find SCCs in a graph.

b) What is the runtime of your algorithm?

C) What is the space complexity of your algorithm?