Walter Villa
May 5, 2021
DSA - Final Report

GOATS - the not-so goat-ed Algorithm

Motivation:

For my final project in Data Structures and Algorithms, I wanted to take the chance to explore an area that was personal and interesting to me: photography. As an engineer that has a sweet spot for content creation and design, I wanted to see if I was able to create an algorithm that would match a specific user with a specific spot within my home city: New York! This inspiration for this specific algorithm came from a program that I started my last year of high school called "GOATS", which stands for "Go Out And Take Shots". GOATS was supposed to build a community of aspiring high school models and photographers that joined every Friday afternoon after a long, stressful week at school to take photos and create amazing art. I have had incredible experiences in the program; from meeting with Alexandria Daddadrio, to traveling to unknown spots in the city.

I wanted to contribute to this program as much as I could, and seeing that I am inclined towards becoming a software engineer, I wanted to see if I could make a matching algorithm that matched a student (user) to a specific spot somewhere in the city through certain features that the student desired (parameters to the algorithm). I set my eyes on creating this algorithm that could look through information on spots to be able to locate the best one for the user. I initially wanted to denote the "best spot" being the spot that had an "end weight" equal to the total number of features that the user had requested because that would mean that that specific spot met all of the user's requirements and is the best suitable for the student. For complexity purposes, I wanted to use a hashmap of locations in certain boroughs (i.e Queens, Manhattan, Brooklyn, Bronx, and Staten Island [debatable if it's a part of NYC lol]) in order to call that specific borough to have access to locations that were also dictionaries.

I was able to complete the matching in a couple of days, and it seemed to work well, but I spoke with a member of the teaching team who told me that what I had originally thought of being an algorithm was perhaps more of a query than an algorithm. With the advice that they gave me, I set out to pose and solve a different type of question while still being able to use my 'matching' algorithm. In this new algorithm, I am trying to optimize a potential "photo-run" (a collection of photography spots) by trying to find the path with the most locations whilst traveling the shortest distance. This was a different type of question than what I had going into the project, but it offered a significant challenge for me in terms of finding the shortest path while looking for the max number of locations that one could take.

Design:

For this new problem that I wanted to create an algorithm for, I needed to switch my sources for research and information. Because I realized that I was trying to maximize the number of things I can have in a specific path while optimizing and minimizing for the distance traveled, I tried to revisit the Knapsack problem. While I believed this would be the answer to a majority of my algorithm, I was mistaken. I was calculating the distances to every single location that made it into the "matched - valid Spots" list, so I found it to have a really hard time to mimic the Knapsack problem in this case. I had a notion that I may want to use dynamic programming to store distances that I already visited, but I realized that it was more of a pre-computation step on my end. Moreover, I looked into path finding algorithms

because I was trying to find the most optimal path, but I did not plan on using a graph or nodes in my project. I looked over the notes on the class website that allowed me to find ways in which to store values within the locations I was trying to calculate the most optimal path in; an adjacency matrix was one of these bits that I found while looking at the class notes.

My background research on websites like Geeks for Geeks and Cracking the Coding Interview, as well as the consultation done to the teaching team and class notes led me to implement a brute-force algorithm from scratch to actually understand what I am doing and to have something working by the end of this project. One of my design choices was to be able to create and use different helper functions in order to compute the final answer, and I was able to learn how to optimize certain functions in order for them to be callable again during other instances in the code. I was very pleased to find out that I did not need to scratch everything that I had done in the initial matching algorithm, and I worked on top of this work.

I found the readings and speaking with the teaching team incredibly helpful when initially figuring out how to model my problem. I was positive that I didn't want to use graphs and nodes, and I was able to sketch out a couple of steps in which I was to go about the problem. Having a solid understanding of the approach that I wanted to take to make sure that I accounted for double counting, missed locations, and even invalid "next locations" (i.e not returning to the same spot that it just came back from because it was the next shortest distance) Overall, the experience and exposure to different methods presented in class helped with structuring my final implementation.

Analysis:

For my final result of my algorithm, I wanted the program to be able to return a list of the locations that it can go with the constraint that it would take the shortest path while still being under the bike's range. My algorithm keeps track of the current shortest path, and while it also prints out the locations, I figured it would be easy to check the result through unit tests.

I can tell that my algorithm does work as intended. Checking against it with my own adjacency matrix, I realize that it does correctly return the shortest path that maximizes the number of locations that a user can visit. My algorithm's time complexity is O(n^2), where n is the number of items in my 'nodeList' variable, which include the user location plus the other matched locations that the user had. My space complexity is O(n^2) because we have to precompute the distances from each location to the other. This takes up n^2 space.

Conclusions:

After pivoting from an initial matching algorithm, I am proud of the work that I have done with my final DSA project. While my implementation is a brute-force approach in finding the shortest path without graphs, I know that I may be able to potentially use some dynamic programming and memoization to be able to not have to calculate certain distances over and over again. I believe further looking into resembling this as a graph problem may also allow for other potential methods in solving for the shortest current path. Ultimately, while this algorithm may not be time or space efficient, I do believe that it may serve well for my "kids" (baby GOAT members). I may want to expand and perhaps not find the shortest path next time but perhaps the **safest** path for students, and researching metrics and data for this may require more development and time.