Project 1 Report - CPSC 335 - Algorithm Implementations

0. Team Members

Joshua Lopez , Sepehr Nourbakhsh , Lynae Mercado

1. Introduction

This project implements two algorithms designed to solve distinct computational problems related to seating arrangements and fuel optimization in a circular road trip. Both algorithms were developed using Python and focus on efficient solutions using data structures and greedy techniques.

2. Algorithm 1: Connecting Pairs of Persons

2.1 Problem Overview

The goal of this algorithm is to rearrange individuals seated in a row so that couples (defined as consecutive pairs of numbers) are sitting side by side with a minimum number of swaps. This problem can be framed as finding the minimum number of swaps needed to group the couples together.

2.2 Approach

We used a **Union-Find** (or Disjoint Set Union, DSU) to efficiently track the seating pairs and their respective groups. The Union-Find helps in grouping the couples based on their IDs, ensuring that unnecessary swaps are avoided.

2.3 Algorithm Design

- Each individual is treated as a node, and couples are connected via the union operation.
- The algorithm checks the seating arrangement and identifies where partners are seated.
- Using the Union-Find structure, the algorithm tracks mismatched couples and computes the minimum number of swaps necessary.
- The problem is solved by ensuring each couple is connected and sitting adjacent to one another.

2.4 Complexity Analysis

- **Time Complexity**: The algorithm runs in **O(n)** time, where `n` is the number of individuals. This is due to the efficient operations provided by the Union-Find data structure.
- **Space Complexity**: The space complexity is **O(n)** for storing the Union-Find structure.

2.5 Example Input and Output

- **Input**: `[0, 2, 1, 3]` (row of people IDs)
- **Output**: `1` (minimum number of swaps)

3. Algorithm 2: Greedy Approach to the Hamiltonian Problem

3.1 Problem Overview

This algorithm solves the problem of determining the best city to start a circular road trip, where each city has a gas station, and the goal is to complete the entire loop without running out of fuel. The challenge is to identify the "preferred starting city" that ensures the car can travel through all cities and return to the starting point with non-negative fuel.

3.2 Approach

We implemented a **greedy strategy** to solve this problem. The algorithm iterates over the cities, calculating whether there is enough fuel to continue to the next city. If a deficit is encountered, the starting point is shifted to the next city.

3.3 Algorithm Design

- The algorithm calculates the net fuel balance at each city (gas provided minus distance to the next city).
- A greedy strategy is used to identify the point where a fuel deficit occurs, shifting the starting point to the next city.
- Once all cities have been considered, the algorithm checks if the total fuel is non-negative, ensuring a valid starting city.

3.4 Complexity Analysis

- **Time Complexity**: The algorithm runs in **O(n)**, where `n` is the number of cities. This is due to the single-pass traversal over the cities.
- **Space Complexity**: The space complexity is **O(1)** as no additional data structures are used apart from storing intermediate results.

3.5 Example Input and Output

- **Input**:
  - `city_distances = [5, 25, 15, 10, 15]`
  - `fuel = [1, 2, 1, 0, 3]`
  - `mpg = 10`
- **Output**: `4` (index of the preferred starting city)

4. Implementation

4.1 Algorithm 1: Connecting Pairs of Persons

The algorithm is implemented in `algorithm1.py` using Python 3.x. It takes an input array representing the seating arrangement and outputs the minimum number of swaps required to seat all couples together.

4.2 Algorithm 2: Greedy Hamiltonian Problem

The algorithm is implemented in `algorithm2.py`. It takes three inputs: distances between cities, fuel available at each city, and miles per gallon (MPG). The output is the index of the preferred starting city.

5. Conclusion

This project demonstrates the implementation of two distinct algorithms that solve real-world optimization problems. Both algorithms are designed for optimal performance and handle their respective problem constraints effectively.

Project 1 Report - CPSC 335 - Algorithm Implementations

File Descriptions:

- **a1.py:** Pseudocode of Algorithm 1.
- **a2.py:** Pseudocode of Algorithm 2.
- **algorithm1.py**: Python implementation of the "Connecting Pairs of Persons" algorithm.
- **algorithm2.py**: Python implementation of the "Greedy Hamiltonian Problem" algorithm.
- **README.txt**: Instructions for running the project.
- **Project_Report.pdf**: This document, containing detailed analysis of the algorithms.