ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 458E
FUNCTIONAL PROGRAMMING

HOMEWORK NO : 3

PREPARED BY:

150220096 : CEYDA AKIN

SPRING 2025

Contents

1 INTRODUCTION

This assignment involves the development of a Haskell-based system to compute the shortest path between two entities within a directed semantic graph constructed from Freebase data. The graph is derived from structured relationships between machine-readable entities (MIDs), and the system must accurately determine the minimal connection between any two specified nodes using Breadth-First Search (BFS). The final output includes both the shortest path and its human-readable equivalent.

2 PROBLEM DEFINITION

The core objective is to design a Haskell program that finds the shortest semantic path between two Freebase entities, represented by Machine IDs (MIDs), in a large directed graph. The system is expected to:

- Parse and construct a directed graph from Freebase relationship data.
- Map MIDs to their corresponding human-readable names using a reference table.
- Accept a source and a target MID as input and compute the shortest path via BFS.
- Output both the numeric path length and the sequence of node names, or report an error if no path exists.

3 IMPLEMENTATION MOTIVATION

This task offers an opportunity to apply functional programming paradigms to real-world data problems. Haskell, with its strong emphasis on purity, immutability, and static typing, provides a robust framework for developing a concise and reliable solution. Working with large datasets and graph structures in a purely functional language reinforces practical problem-solving skills and deepens understanding of advanced language features such as lazy evaluation, higher-order functions, and algebraic data types.

4 IMPLEMENTATION EXPLANATION

4.1 Data Preparation

The dataset consists of two files:

- `mid2name.tsv`: Maps MIDs to human-readable entity names. Only the first occurrence of each MID is retained to ensure consistency.

- `freebase.tsv`: Contains semantic relationships in triplet format. Source and destination MIDs are extracted to construct a directed graph.

4.2 Graph Representation

A graph is represented as a `Map MID [MID]`, facilitating fast access and efficient traversal. Each node maps to a list of directly connected neighbors.

4.3 Shortest Path Search

Breadth-First Search is employed due to its suitability for unweighted graphs. It ensures the shortest path (in terms of number of edges) is found. The search continues level by level, recording visited nodes and maintaining paths. If no path is found, a custom `NoPathException` is thrown.

4.4 Functional Concepts Applied

- Use of immutable structures and recursion.
- Type-safe exception handling via custom data types.
- Safe and lazy I/O using `withFile` and UTF-8 encoded reading.
- Clear separation between pure and impure functions.

5 RESULTS

Trial 1: Valid Path

- **Source:** /m/04jpl (London, england)
- **Target:** /m/0d0vqn (Swedish climate)
- **Distance:** 2

Trial 2: Short Path

- **Source:** /m/0kfv9 (The Sopranos)
- **Target:** /m/01l1sq (Steven Van Zandt)
- **Distance:** 1

Trial 3: No Connection

- **Source:** /m/075pwf (Decanoic acid)
- **Target:** /m/08ns5s (Long snapper)
- **Output:** Exception thrown – NoPathException

Output Screenshot

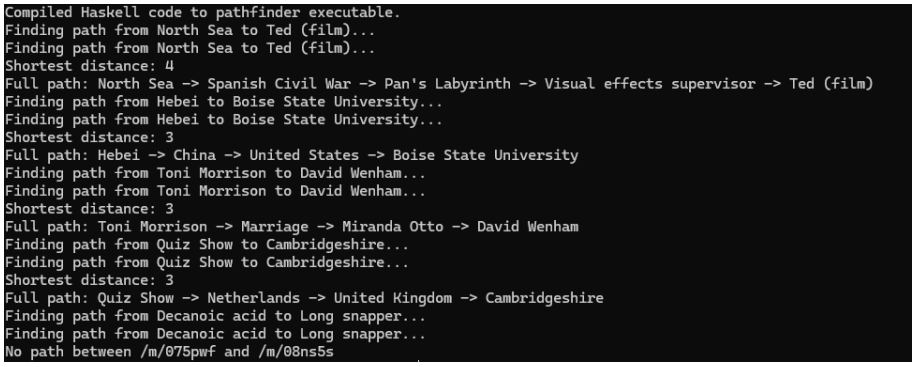

Figure 1: Sample terminal output for valid path

6 DISCUSSION

The assignment showcases the practical use of functional programming techniques in handling large-scale semantic datasets. Particular attention was required to maintain consistency in name resolution between different programming environments. This was resolved by ensuring that only the first appearance of each MID was preserved during parsing.

Directed edge construction proved essential for correctly modeling the Freebase graph. BFS was selected for its optimality and completeness. The lazy evaluation strategy of Haskell allowed for efficient memory usage during processing.

Custom exception handling further improved the usability and reliability of the program. By explicitly detecting unreachable nodes, the system provides clear and user-friendly error feedback.

7 OVERALL CONCLUSION

This project provided valuable experience in:

- Modeling semantic relationships through graph theory.
- Leveraging Haskell's functional features for safe and reliable data processing.
- Implementing classical graph algorithms within a pure functional context.
- Debugging and aligning program behavior across multiple language environments.

The resulting application is robust, efficient, and consistent in its outputs, successfully achieving all the goals defined in the assignment specification. The project exemplifies the strengths of functional programming in managing structured data and performing graph traversal at scale.

REFERENCES

[1] Istanbul Technical University Department of Computer Engineering. Blg 458e – functional programming homework 3 booklet. *ITU Course Materials*, 2025. Prepared by Dr. Yusuf H. Şahin.