CSE211 DATA STRUCTURES

LAB 4 FALL 2024

GRAPH OPERATIONS

Prerequisites

Open the terminal and execute the following commands after downloading the `tarball` file:

```
cd /mnt/c/Users/user/Downloads && tar -xvf lab4_1.tar.gz --one-top-level=lab4_1
cd /mnt/c/Users/user/Downloads/lab4_1 && make all
code .
```

Introduction

In this lab, you will implement advanced operations on a Graph data structure using C++. The Graph is implemented as a template class that can store elements of any type T. Your task is to implement the following challenging operations:

1. `processParallel`: Schedule tasks across multiple processors
2. `processOrders`: Process restaurant orders across cooking stations

Project Structure

```
.
├── bin/
│   └── graph
├── deps/
│   └── nlohmann/
│       └── json.hpp
├── include/
│   ├── Color.hpp
│   └── Graph.hpp
├── inputs/
│   ├── bellmanFordGraph.json
│   ├── bellmanFordGraphCycle.json
│   └── dijkstraGraph.json
├── obj/
│   ├── Color.o
│   ├── Graph.o
│   └── main.o
├── outputs/
│   ├── dots/
│   │   ├── bellmanFordGraph.dot
│   │   ├── bellmanFordGraphCycle.dot
│   │   └── dijkstraGraph.dot
│   └── img/
│       ├── bellmanFordGraph.png
│       ├── bellmanFordGraphCycle.png
│       └── dijkstraGraph.png
├── src/
```

```
|   ├── Color.cpp
|   ├── Graph.cpp
|   └── main.cpp
├── instructions.md
└── Makefile
```

Implementation Details

1. Dijkstra's Algorithm

- **Purpose**: Find shortest paths in graphs with non-negative weights
- **Method**: `bool dijkstra(int src, std::vector<T> &distances)`
- **Parameters**:
  - `src`: Source vertex
  - `distances`: Vector to store shortest path distances
- **Return**: `true` if successful
- **Example**:

```
Input: Graph from dijkstraGraph.json, source = 0
Output distances: [0, 2, 5, 6]
// Meaning:
// Vertex 0 -> 0: distance 0
// Vertex 0 -> 1: distance 2
// Vertex 0 -> 2: distance 5
// Vertex 0 -> 3: distance 6
```

2. Bellman-Ford Algorithm

- **Purpose**: Find shortest paths and detect negative cycles
- **Method**: `bool bellmanFord(int src, std::vector<T> &distances)`
- **Parameters**:
  - `src`: Source vertex
  - `distances`: Vector to store shortest path distances
- **Return**: `false` if negative cycle exists, `true` otherwise
- **Example**:

```
// Normal graph
Input: Graph from bellmanFordGraph.json, source = 0
Output distances: [0, 2, 4, 5]
// Graph with negative cycle
Input: Graph from bellmanFordGraphCycle.json, source = 0
Output: false (indicates negative cycle detected)
```

Testing

1. Build and run:

```
make deps # Download dependencies (first time only)
make clean # Clean previous builds
make all # Compile all files
make run # Execute the program
```

2. Visualization (requires Graphviz):

- DOT files are generated in `outputs/dots/`
- PNG visualizations in `outputs/img/`
- For installation of Graphviz, refer to the Graphviz Installation Guide

Restrictions

❌ Do not modify:

- Graph.hpp interface
- main.cpp test cases
- Project structure
- Build system

❌ Do not use:

- External libraries (except nlohmann/json)
- Global variables
- Additional data structures (except where specified)

Academic Integrity

- Individual work only
- No code sharing
- No plagiarism
- Violations result in zero grade

Submission

1. Test thoroughly
2. Clean build files: `make clean`
3. Submit only the `Graph.cpp` file

Good luck with your implementation!