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_2.tar.gz --one-top-level=lab4_2
cd /mnt/c/Users/user/Downloads/lab4_2 && 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/
|   └── dagGraph.json
├── obj/
|   ├── Color.o
|   ├── Graph.o
|   └── main.o
├── outputs/
|   ├── dots/
|   |   └── dagGraph.dot
|   |   └── dagGraphCyclic.dot
|   └── img/
|       └── dagGraph.png
|       └── dagGraphCyclic.png
├── src/
|   ├── Color.cpp
|   ├── Graph.cpp
|   └── main.cpp
├── instructions.md
```

```
└── Makefile
```

Implementation Details

1. DAG Operations

1.1 Checking if Graph is DAG (Directed Acyclic Graph)

- **Purpose**: Determine if the graph is a DAG (has no cycles)
- **Method**: `bool isDAG()`
- **Helper Method**: `bool isDAGUtil(int v, std::vector<bool> &visited, std::vector<bool> &recStack)`
- **Return**: `true` if graph is a DAG, `false` if it contains cycles
- **Example**:

```
Input: Graph from dagGraph.json
Output: true (graph has no cycles)
```

1.2 Topological Sort

- **Purpose**: Produce a linear ordering of vertices such that for every directed edge u->v, vertex u comes before v in the ordering
- **Method**: `bool topologicalSort()`
- **Helper Method**: `bool topologicalSortUtil(int v, std::vector<bool> &visited, std::vector<int> &result, std::vector<bool> &recStack)`
- **Parameters**:
  - `visited`: Keep track of visited vertices
  - `result`: Store the topological sort result
  - `recStack`: Track recursion stack for cycle detection
- **Return**: `false` if graph has cycles, `true` if successful sort
- **Example**:

```
Input: Graph from dagGraph.json
Output: 3 2 1 0
// Meaning: This is a valid topological ordering where:
// - vertex 3 must be processed before 2
// - vertex 2 must be processed before 1 and 0
// - vertex 1 must be processed before 0
```

Implementation Notes:

1. **isDAG & isDAGUtil**:
   - Uses DFS with a recursion stack to detect cycles
   - A graph with a cycle cannot be a DAG
   - Time Complexity: O(V + E)
   - Space Complexity: O(V)

2. **topologicalSort & topologicalSortUtil**:
    - Only works on DAGs
    - Uses modified DFS to produce ordering
    - Prints vertices in topologically sorted order
    - Time Complexity: O(V + E)
    - Space Complexity: O(V)

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!