ITP 20002 Discrete Mathematics, Fall 2020

Homework 5. Implementing Graph Algorithms

Shin Hong

1. Overview

In this homework, you are asked to implement two algorithms, the Dijkstra's algorithm (Section 3.1) for finding shortest paths between two vertices and the Fleury's algorithm (Section 3.2) for finding a Euler path that work with the given C library for undirected multi-graphs (Section 2).

You need to construct `dijkstra.c` and `euler.c` for implementing the two algorithms, respectively. These must find correct answers, and properly use the given data structure (`graph.c` and `graph.h`) that represent undirected pseudographs (i.e., multigraphs with loops) with positive integer weights.

Submission deadline is **11:59 PM, 24 Dec (Thur)**. All related code is found at the `hw5` branch of the course Github repository:

`https://github.com/hongshin/DiscreteMath/tree/hw5`

2. Graph Library

A C library for representing an undirected, weighted multi-graph is given for this homework. You can find its data structure and interfaces `graph.h` and their implementations at `graph.c`. This library supports the following interface functions:

- `graph_create (int n_vertices)`: constructs and returns an object that represents graph with `n_vertices` number of vertices. Each vertex of this graph is referred as a non-negative integer identifier between 0 and `n_vertices - 1`. Initially, this graph does not have any edge.
- `graph_free (graph_t * g)`: reclaims all memories allocated for representing a graph object `g`.
- `graph_read_stdin ()`: constructs and returns a graph object as given through the standard input. An input specifying a graph must have first two numbers representing the number of vertices and the number of undirected edges. After that, the definitions of the edges follow. Each edge is represented as three numbers where the first and the second represent the vertex identifiers and the third represents a weight given to the edge. You can find input data examples at `data/graph1.txt` and `data/graph2.txt`.
- `graph_add_edge (graph_t * g, int init, int term, int w)`: adds a new edge between two verteces `init` and `term` with a weight `w` to graph `g`.
- `graph_remove_edge (graph_t * g, int init, int term, int w)`: removes an edge of a weight `w` between vertices `init` and `term` if exists in graph `g`.
- `graph_print_stdout (graph_t * g)`: displays graph `g` via the standard output. Each printed line shows the two vertices connected with one or multiple edges, and the weights given to these edges.

Note that you are given a complete implementation of this library, so you do not need implement these functions. In addition, you must not change these function definitions.

The following is the structure of the `graph_t` type whose object represents an weighted, undirected multigraphs with loop:

```
typedef struct {
            int n_vertices ;
            int ** m ;
            int *** w ;
        }
        graph_t ;
```

`n_vertices` represents the number of vertices. The vertices in this graph are named as 0 to `n_vertices - 1`. `m[v1][v2]` represents the number of edges (i.e., multiplicity) between two vertices `v1` and `v2` for $0 \leq$ `v1` $<$ `n_vertices` and $0 \leq$ `v2` $<$ `n_vertices`. Since it's a undirected graph, `m[v1][v2]` must be the same as `m[v2][v1]`. For there are multiple edges betwen `v1` and `v2`, the weights of these edges are stored in `w[v1][v2][0]` to `w[v1][v2][m[v1][v2]-1]`. Note that `w[v1][v2]` is null if `m[v1][v2]` is zero.

In this homework, you must understand this library and use it when you implement the two graph algorithms. You must not have an independent data structure for representing graphs in your program.

3. Graph Algorithm

3.1. Dijkstra's algorithm for finding shortest paths

The Dijkstra's algorithm for a shortest path problem can find the length of a shortest path connecting two given vertices in a graph with positive weights. This algorithm determines a shortest path to each vertex in the order of their closeness with the starting vertex. The intuition behind this algorithm is that the shortest path of the i-th closest vertex can contain only the first to (i-1)-th closest vertices. You can find the detail of this algorithm in Section 10.6.2. of our textbook and many other resources. You need to implement this algorithm to work correctly for a multigraph that may have loops.

3.2. Fleury's algorithm for finding a Euler path

The Fleury's algorithm finds a Euler path (or Euler circuit) which covers all edges exactly once if such path exists in the given graph. The detail of this algorithm is explained in Section 10.5.2 in our textbook. This algorithm first determines whether the given graph has a Euler path or not by counting the degree of each vertex. Also, if the graph is determined to have a Euler path but not Euler circuit, the algorithm must find the starting and the ending vertices. If a Euler circuit or path exists, the algorithm works as follows:

Step 1. Starting from a vertex, construct a path by selecting an arbitrary sequence of consecutive edges until it is impossible to extend the path. Then, remove these edges from the graph.

Step 2. If edges remain in the graph, start Step 1 at one of vertex in the path to obtain another path. Merge two paths into one path. Repeat this until no edge remains.

For finding a Euler path, you do no need to consider weights given to the edges of the graph.

4. Your Task

You are asked to implement the algorithms at `dijkstra.c` and `euler.c` in the given baseline code. You can find that these source code files already have the stub for loading an input data as a `graph_t` object. Each of your algorithm must properly access this graph object to find the expected answer, respectively. Note that your program must not have another redundant data structure that represents the same target graph. Note also that you are not allowed to use other libraries for constructing these algorithms.

In doing this homework, you can reference materials that explain the Dijkstra's algorithm and the Fleury's algorithm other than our textbook. If so, you must clearly mention the reference at the presentation.

4.1. Implementing Dijkstra's algorithm

As shown in the source code `dijkstra.c`, this program is designed to receive the target graph from the standard input via `graph_read_stdin()`. In addition, this program receives the identifiers of the starting and ending vertices as command-line arguments.

This program must print out the length of the shortest paths connecting two vertices given as command-line arguments. The program must not print anything if there is no path connecting the two vertices. It is not mandatory to print out the path of the shortest length.

4.2. Implementing Fleury's algorithm

The source code `euler.c` shows that this program receives the target graph from the standard input via `graph_read_stdin()`. For given graph, this program must print out a sequence of vertex identifiers which represent a Euler path (or Euler circuit) that covers all edges exactly once. It is okay that your program finds an arbitrary Euler path even if there exist multiple ones.

4.3. Presentation video

As a report of your result, you must record a video where you present the two algorithm implementations (source code files) and demo of your programs.

Your video must be 4 minutes to 6 minutes long, and one video must cover two algorithms at the same time. In your video, you first explain how you implement the algorithms by presenting the source code. In addition to that, you need to show the execution results of your programs to demonstrate its correctness.

5. Submission

You must submit **(1) the source code files** and **(2) the presentation video** via Hisnet. Since a video file would be too large to be contained in a Hisnet submission, it is recommended to upload the video to YouTube or Google Drive, and then specify the link to it at the `README` file in your source code submission. The presentation video can be placed at YouTube, Google Drive, or other streaming service. The submission deadline is **11:59 PM, Dec 24 (Thur)**. This is strict and no late submission will be accepted.