Graph Traversal with a Web API

Erik Saule

Goal: Implement a breadth-first search (BFS) traversal algorithm in C++ that interacts with a web-based graph server. The program should take a starting node and a traversal depth as input, query the server for neighboring nodes, and return all nodes reachable within the given depth.

1 Programming

For reference, here is an overview of breadth-first search (BFS) from Geeks for Geeks:

Breadth-First Search (BFS) - GeeksforGeeks

Here's a step-by-step explanation of how BFS should work:

- Start from the given source node.
- Use a queue to explore nodes level by level.
- Keep track of visited nodes to avoid cycles.
- Expand nodes up to the given traversal depth.
- Return all nodes visited within the depth limit.

Dataset Information. The graph used in this assignment is built from the Bridges Actor/Movie Dataset. This dataset contains actors and the movies they have acted in, with edges representing the relationship between them. You can learn more about this dataset at:

`https://bridgesuncc.github.io/tutorials/Data_WikiDataActor.html`

TODO. Implement a BFS traversal in C++ that interacts with a graph server.

- The program should take the starting node and traversal depth as command-line arguments. (Starting node will be a string representing actor/movie name, e.g., "Tom Hanks," and depth will be an integer value.)
- It should make API calls to fetch neighbors dynamically.
- The program should parse JSON responses from the API.
- It should return all nodes within the given depth.

Suggestions.

- First, figure how to use `curl` to make web requests.
- Then, parse JSON responses using a C++ library like `rapidjson`.
- Write a function that, for a particular node of the graph (identified by a string), will make the curl query and return a set of neighbors.
- Write BFS using that function as the neighbor discovery.
- Finally, add time measurement.

2 Interacting with the Web API

The graph server has been set up and is accessible at:

`http://hollywood-graph-crawler.bridgesuncc.org/neighbors/`

The API provides a single endpoint:

- **GET /neighbors/{node}**: Returns a JSON response containing all immediate neighbors of the given node.

Example API Call:

```
curl -s http://[ENDPOINT]/neighbors/Tom_Hanks
```

Example Response:

```
{
    "neighbors": ["Forrest_Gump", "Saving_Private_Ryan", "Cast_Away"],
    "node": "Tom_Hanks"
}
```

3 Using RapidJSON and libcurl

3.1 Including and Using libcurl

libcurl is a library for making HTTP requests in C++. To install it:

```
# On Ubuntu/Debian:
sudo apt install libcurl4-openssl-dev

# On macOS:
brew install curl
```

To use it in your C++ code, include the header:

```
#include <curl/curl.h>
```

For detailed instructions on how to use libcurl, refer to the libcurl tutorial (read up to ”When it does not work” section):
`https://curl.se/libcurl/c/libcurl-tutorial.html`
If you have any issues compiling your code, try linking libcurl explicitly using:

```
g++ your_program.cpp -o your_program -lcurl
```

3.2 Including and Using RapidJSON

RapidJSON is a fast JSON parser for C++. To install it using GitHub:

```
git clone https://github.com/Tencent/rapidjson.git
```

Once cloned, install it as follows:

For Ubuntu/Debian:

```
cd rapidjson
mkdir build && cd build
cmake ..
sudo make install
```

For macOS:

```
cd rapidjson
mkdir build && cd build
cmake ..
sudo make install
```

If this does not work, you can directly include the RapidJSON headers by specifying the path in your compilation command. Assuming the cloned folder is in your home directory:

```
g++ your_program.cpp -o your_program -I ~/rapidjson/include
```

For detailed instructions on how to use RapidJSON, refer to the RapidJSON tutorial:
`https://rapidjson.org/md_doc_tutorial.html`

4 Benchmarking and Testing

Your program should handle graphs of various sizes efficiently. Test your implementation by running it with different nodes and depths.

TODO. Evaluate the performance of your BFS implementation.

- Run the program with different starting nodes and traversal depths.
- Test for "Tom Hanks"/2 on Centaurus where node name = "Tom Hanks" and distance to crawl = 2.
- Measure execution time for different depths and graph sizes.

5 Submission

TODO. Submit an archive containing:

- Your C++ source code.
- A `Makefile` for compiling the code.
- A `README` explaining how to run the program.
- Example output files.