mp_stories

Sneaky Stories

Extra credit: Apr 22, 23:59 PM
Due: Apr 29, 23:59 PM

Doxygen

Partner MP mp_stories is a **partner MP**!

- The creative story (submitted with the last MP and will be used in this MP) must be completed by yourself and must be unique (and different from your partner's work).
- Everything else in this MP can be completed with a partner!

You should denote who you work with in the PARTNERS.txt file in this MP. If you worked alone, include only your NetID in PARTNERS.txt.

Goals and Overview

In this MP you will:

- Merge your creative story into the course-wide CYOA graph
- Implement an Adjacency List with correct running times
- Find interesting sub-structures in the course-wide CYOA story using the shortest path algorithm

Checking Out the Code

From your CS 225 git directory, run the following on EWS:

```
git fetch release
git merge release/mp_stories -m "Merging initial mp_stories files"
```

If you're on your own machine, you may need to run:

```
git fetch release
git merge --allow-unrelated-histories release/mp_stories -m "Merging initial mp_stories files"
```

Part 1: Merge Your Story to the Course-wide Graph

For Part 1, we will combine our graphs using **forked repo** (and **git pull requests**), a common way to merge changes in code when working as part of a large team.

Part 1.1: Add your files

Part 1.1a: Fork `_graph` repo outside of your CS 225 course repo by clicking the `Fork` button at the top right corner.

- If you are in a folder with directories with mp1, mp2, etc, you are still inside of your CS 225 repo. Move up another directory.
- Once you are outside of your CS 225 course repo directory, run:

`git clone https://github-dev.cs.illinois.edu/<NETID>/_graph.git mp_stories-graph`

Part 1.1b: Move into the `mp_stories-graph` repo.

- Run `cd mp_stories-graph`

Part 1.1c: Copy your entire `story_data` directory from your `mp_mazes` folder (eg: the directory of all the .md files that contain your story) into the `mp_stories-graph` folder that is the course-wide graph. Your `mp_stories-graph` repo will now contain the `story_data` directory.

Part 1.1d: Validate your graph (again) by running `story_validator.py`

- On EWS, from the base of the `mp_stories-graph` directory:

```
module load python3/3.7.0
python3 story_validator.py story_data
```

- On your own machine with Python installed: `python3 story_validator.py story_data`

More information can be found here.

Part 1.1e: Commit to your fork:

- Inside of the `mp_stories-graph` directory, run the following:
- Add all your files by running: `git add story_data`
- Commit your files by running: `git commit -m "<NETID> story"`
- Push your files by running: `git push origin master`

Part 1.2: Pair Up and Combine

Pair up with your friend, lab partner, or other CS 225 student to combine your forks into one larger graph. The two of you should work together to combine the graphs. Specifically, one of you will create a "pull request" on GitHub, and the other will review the request, and approve the request in order to merge the two branches. This is a common practice in project collaboration to make sure that every change is being reviewed before being merged into the main code base.

Part 1.2a: Make sure your story is pushed to your fork by checking `https://github-dev.cs.illinois.edu/<NETID>/_graph`

Part 1.2b: Make a pull request to the fork you want to merge into:

- Click the "Pull requests" tab
- Click "New pull request" next to the branch button
- In the "base" fork selection, select the branch you fork to merge into
- Make sure that the "compare" branch is your branch

Part 1.2c: Review and accept the pull request, thus merging the two forks (by the other person):

- The other person will see the pull request and any difference it will make
- After reviewing the changes and making sure there's no conflicts, click the arrow next to "Merge pull request", and select "Rebase and merge". Then press "Rebase and merge" to merge your changes
    - While not strictly necessary, this will greatly lower the chances of having conflicts automatically resolved in a way which breaks the story by reporting more conflicts and failing more often. If you are having trouble merging the PR by rebasing and know what you are doing, feel free to merge the PR in a different way.

Part 1.2d: Modify and/or add new files to ensure that the two graphs are connected. Remember that there must be exactly one starting vertex (no incoming edges). This means you must at least modify or add one node, though integrating your stories in an interesting way is ideal!

Part 1.2e: Validate your graph again by running `story_validator.py`

Part 1.3: Merge with other forks or let others merge with you!

Let others merge with you

Suppose you already have your graph with at least 2 people's work. You can make a pull request to the `cs225-sp19/_graph/master` to make your graph pending to be combined with the course-wide graph!

- Make sure your graph is valid by running the validator
- Click "New pull request" from your branch, select `cs225-sp19/_graph/master` as the base (see Part 1.2a&b)
- Create the pull request to `cs225-sp19/_graph/master`

This way, your fork will be a pending branch to be forked into `cs225-sp19/_graph/master`(the final course-wide graph). Others can also make pull requests to your branch! If you accept their pull request (make sure that the changes don't break anything), your stories will become one larger story! You can start by inviting your friends to merge into your branch!

As the owner of the branch, this will require a bit of extra work since you have to review and accept other people's pull requests in order to make your story larger. You are also responsible for maintaining your branch and combine others' stories with the current one (see Part 1.2d).

Merge with other forks' pull requests

Alternatively, you can merge into other people's stories by making pull requests to other people's forks!

- Navigate to the top-right corner of your fork of `_graph` and click the number count next to the `Fork` button.
- Check for open pull requests to `cs225-sp19/_graph/master` and find a fork that you like
- Make a pull request from your fork, select as the base

This way, if your pull request is accepted by the owner of the other fork, your story will be integrated into their story, and it will be combined with the course graph later when their pull request to `cs225-sp19/_graph/master` gets accepted. However, you don't have control over whether they accept your pull request. Again, try starting with your friends!

Extra Credits

You'll earn **+7 extra credit** for having your story in the largest pull request by Monday, Apr. 22nd at 11:59pm! Each smaller pull request will earn progressively less extra credit and CS 225 course staff will merge all remaining large-ish pull requests after Wednesday to generate the full, course-wide dataset.

Part 2: Adjacency List

You have been provided with an incomplete `Graph` implementation for an adjacency list implementation. You must complete all functions in `Graph.hpp` without editing `Graph.h`. This means you **must** make use of the `std::unsorted_map` that maintains the vertices and `std::list` that maintains the edges.

Open the doxygen for class `Graph`.

> ℹ In this MP, we will not be testing for memory leaks. However, we do require that your code is free of any memory **errors**.

You should see the following output in `valgrind -v`.

```
==7555== For counts of detected and suppressed errors, rerun with: -v
==7555== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Runtime Required

In this MP, tests are designed to ensure you have an implementation that runs in the running time described in lecture of the adjacency list, with the **exception** of vertex/edge removal. You will find several test cases require a specific running time to be met.

Most importantly, ensure your `Graph::incidentEdges` and `Graph::isAdjacent` functions run in times described in lecture.

Testing Your Story With Your MP

The `main.cpp` file is used to compile the `./mp_stories` that loads your interactive story from your `story_data` directory. Copy your `story_data` directory from the last MP and replace the `story_data` folder in this MP. A complete implementation of Part 2 will allow you to verify the format of your story from your story_data folder.

Tip on std::reference_wrapper

`std::reference_wrapper<class T>` is a useful utility that lets `std::container` based structures like `std::list` to be able to store references to data instead of needing to internally copy data into the container data structure. To access the reference from the `std::reference_wrapper<class T>`, please use the `std::reference_wrapper<class T>` member function `T& get() const noexcept` to retrieve the wrapped reference.

Test Suite

Additional unit tests similar to how we will grade your MP are provided for you in `tests`.

```
make test
./test
```

Part 3: Creative Sub-Structures

The next thing to do in mp_stories is to create a shortest path algorithm! You must find and return the shortest path between a given `start` vertex and `end` vertex.

There are different implementations for the shortest path algorithm. Compare them and use the one that you see fit the most!

Complete `Graph<K,V>::shortestPath(...)` within `Graph2.hpp`. This function will allow you to find your shortest path in the course-wide graph!

Grading Information

The following files are used to grade mp_stories so far:

- `Graph.hpp`, for Part 2
- `Graph2.hpp`, for Part 3

All other files will not be used for grading.

No Memory Leak Grading! You do not need to worry about leaking memory in this MP – we will not test (and you will have) unfree'd memory when you program exists.