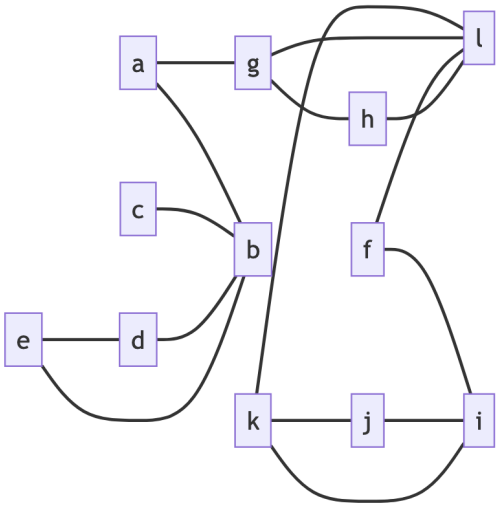
CSCI 200
due 04/05/2024

Name:

Problem Set 6

Problem 1 (10 points, 5 points for each part)

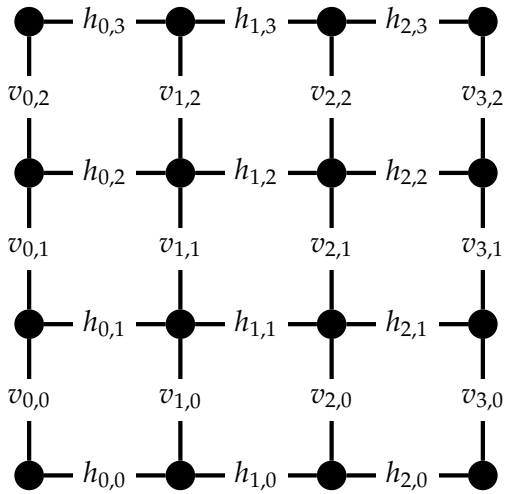
Starting at vertex a, perform (a) breadth-first and (b) depth-first search to create a spanning tree of the following graph. Use alphabetical order to determine the order in which to visit neighboring vertices. Please show your work and **be sure to list the order in which vertices are traversed**. If you would like to draw the tree, see how the `mermaid` commands are used below to draw the original graph.



Problem 2 (10 points)

Let G be the graph shown below with the horizontal edges labelled $h_{i,j}$ and vertical edges labelled $v_{j,i}$ for $0 \leq i \leq 2$ and $0 \leq j \leq 3$. The weights on the horizontal edges are $w(h_{i,j}) = 4i + j$ whereas those on the vertical edges are $w(v_{j,i}) = 100 + 4j + i$. Construct the minimum spanning tree for using the algorithm we used in Lecture 6W (Prim's algorithm). Instead of starting at $h_{0,0}$ (which is the edge with minimum weight), start with the edge $h_{0,2}$.

In your answer, please **list the sequence in which the edges are added to the MST** (using the original $h_{i,j}$ and $v_{j,i}$ labels). You do not need to include your drawing of the MST, but (if you want) you can take a picture of your work and add it to your repository (see the "Add file" button above).



Problem 3 (10 points)

During a team workout of **nine** athletes, the trainer must create a schedule for each exercise. There are **eight** exercises, each of which can be performed by **two or three** athletes at any given time. The assigment of athletes to each of the eight exercises is given below:

- E_1: Nadia, Eduardo, Charlie
- E_2: Nadia, Diane, Eve
- E_3: Eduardo, Kento
- E_4: Greta, Diane, Hamza
- E_5: Greta, Jiao, Eve
- E_6: Jiao, Kento
- E_7: Jiao, Diane
- E_8: Eduardo, Kento, Eve

Clearly, no athlete can be doing different exercises at the same time. We need to help the trainer by determining the minimum number of time slots required such that all athletes complete all their assigned exercises.

(a) Recast this problem as a question about coloring the vertices of a particular graph. Draw the graph and explain what the vertices, edges, and colors represent.

(b) Show a coloring of this graph using the fewest possible colors. What athlete-excercise schedule does this imply?

To include the graph in your submission, please either (1) take a picture and upload to your repository (using the "Add file" button above), naming your picture `problem3.png` (or .jpg) or (2) draw the graph using `mermaid` (see Problem 1 for an example, as well as the Mermaid documentation).

When uploading a picture, please either use a `.png` or `.jpg` format - do not upload `.HEIC` files because they cannot be viewed in the GitHub interface.

Problem 4 (10 points)

Consider the relation $R = \{(x, y) : x - y \text{ is an integer}\}$ defined on the set of real numbers.

(a) Show that R is an equivalence relation.

(b) What is the equivalence class that contains 2 for this equivalence relation? Express your result as a set.

(c) What is the equivalence class that contains $\frac{1}{2}$ for this equivalence relation? Express your result as a set.

CSCI 200
due 04/05/2024

Problem Set 6

Name:

Problem 5 (5 points)

The goal of this problem is to become familiar with running a `Python` script, specifically in a GitHub codespace.

`Python` code is *interpreted* by a `Python` interpreter, which is a separate program that you will call to run your code (similar to how you would use `java` to run your compiled `Java` program). Scripts are run from top to bottom, unlike having a `public static void main` entry point (if you have not taken CS 201, don't worry about this). To run your script (suppose it's saved to a file called `myscript.py`), you would type (in a Terminal):

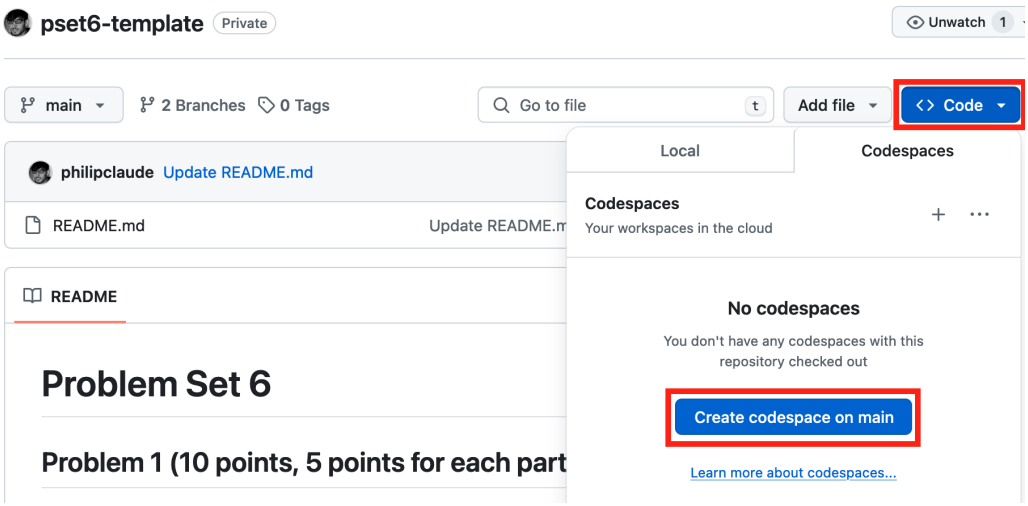
```
$ python myscript.py
```

Note: You wouldn't actually type the dollar sign (`$`) - it's only there because that's what you'll see if you type this in a Terminal.

Instead of running the script *locally* (on your computer), we'll run this script using something called a GitHub codespace (in the cloud!). A codespace is a virtual machine (in the cloud), which you interact with from your web browser. This codespace will have Visual Studio (VS) Code (a fancy programming editor) built directly into it, so you will be able to write code, run your code (within a Terminal) while also having some extensions that make it convenient to save changes to your repository.

Here's what to do:

(1) At the top-right of your repository, click on the **Code** button. Then click on **Create codespace on main**.



(2) A new tab will open in your browser with your newly created codespace. It will have a random name like "silver orbit" or "bookish space barnacle" (see the bottom-left of the editor). You should also see something like the following in your **Terminal**:

```
@philipclaude -> /workspaces/pset6-template (main) $
```

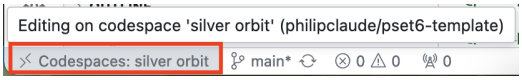
Now type `python problem5.py`:

```
@philipclaude -> /workspaces/pset6-template (main) $ python problem5.py
```

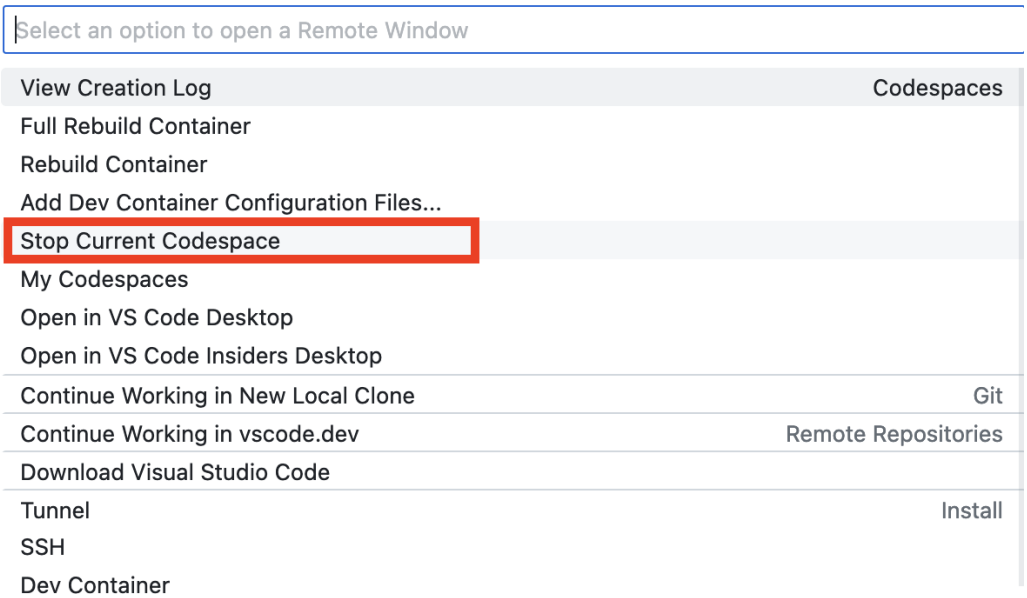
Then press the **Enter** key, and you should see some stuff printed in the terminal.

(3) Copy the output from Step (2) (anything in between the triple-backticks) and paste it into your `README.md` file for Problem 5 (the `mermaid` block is already set up for you). When pasting the `mermaid` block into your `README.md` file, be sure to do this back in your original repository (not in the `README.md` in your codespace). Otherwise the changes won't be synced with your original problem set repository (unless you follow the steps described in Problem 6).

(4) That's it! If you want to practice programming, you'll use this codespace for Problem 6. Otherwise stop your codespace by clicking on your codespace name at the bottom-left of the editor:



Then select **Stop Current codespace**:



Note: Stopping your codespace is a good idea so we don't waste computing resources. You can also delete your codespace by navigating to: https://github.com/codespaces. I think it will be auto-deleted after a certain amount of time anyway.

Have a look at `problem5.py`, which is running depth-first-search (`dfs`) on a graph similar to the one we used in Lecture 6W (note that the edge `('g', 'h')` is missing from the initial graph). Feel free to modify the input graph if you want to check your answer to Problem 1.

CSCI 200
due 04/05/2024

Problem Set 6

Name:

Problem 6 (optional! but can be used to check your answers)

The main goal of this (optional) problem is to practice programming with graphs (in `Python`), as well as pushing changes to GitHub. Please complete Problem 5 before attempting this problem, then open up your codespace again.

In your repository, you should see the `middgraph.py` file, which exposes a light interface for representing graphs. We'll call this interface the `middgraph` "module." Modules are useful for keeping functions, and class definitions in a separate file. In `Python` they can be *imported* with the `import` keyword (usually at the top of your *script*). The `middgraph` module mainly has a class definition for a `Graph`. The constructor (the `__init__` function) expects two sets: vertices and edges.

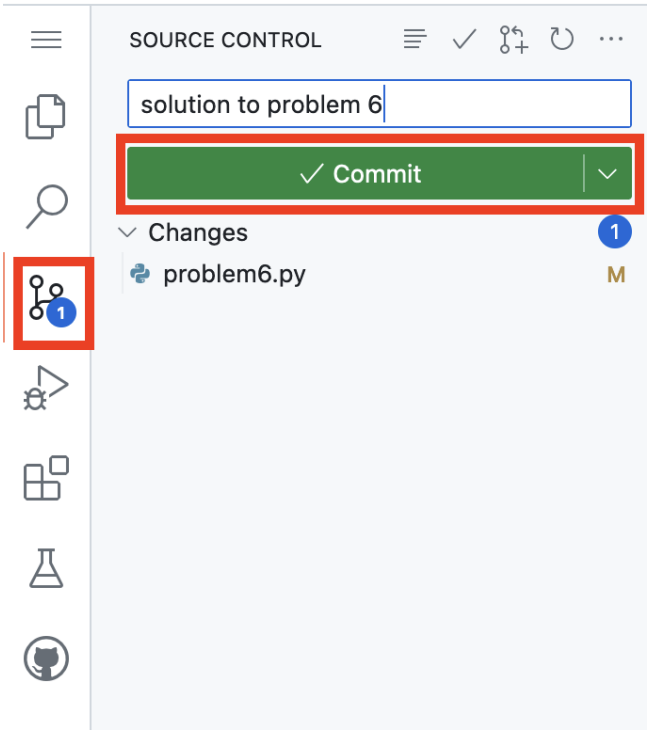
To get started, you might want to see how the `dfs` function is written using `middgraph` in the `problem5.py` script. Then, implement Prim's algorithm for finding the minimum spanning tree (MST) of a graph. The file `problem6.py` has been set up with the graph from Lecture 6F and the `prims_mst` function has been defined but not implemented. Note that the graph from the in-class example is defined and the `sort_edges_by_weight` function is called. Please complete the `prims_mst` function, using the pseudocode from class (and the notes) to guide your implementation.

Once you've implemented this, it will be saved in your codespace, but it won't be saved to your repository. This is where we will use `git` to push the changes to your repository:

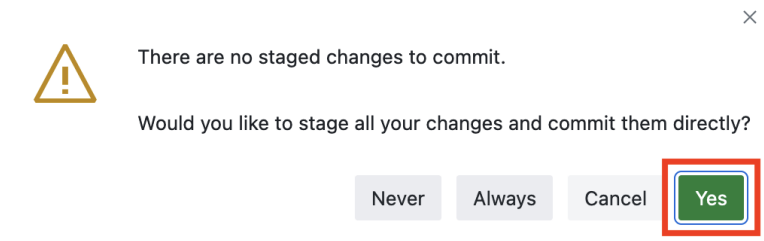
(1) Click on the Source Control icon 

.

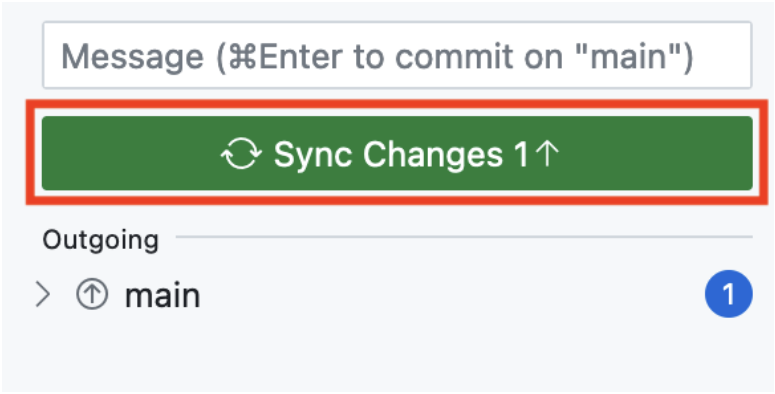
(2) All the files you have changed in your codespace should be listed here. Type in a message in the text box (e.g. `solution to problem 6`) and then click **Commit**:



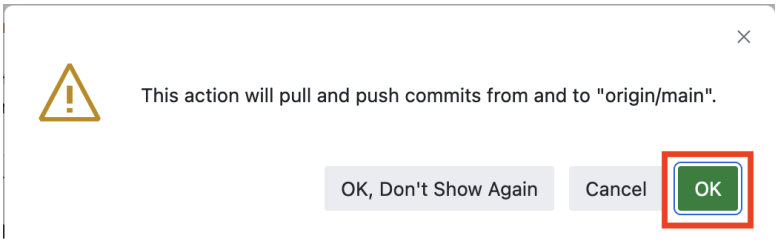
(3) You will probably be asked about staging all your changes, click **Yes**:



(4) Now you're ready to *push* your changes back to your problem set repository. Click on the **Sync Changes** button:



(5) You'll probably be asked about pulling/pushing your commits - click **OK**:



(6) Go back to your original problem set repository on GitHub and check that your changes are there.

There is nothing to write in the problem set `README` for Problem 6, but make sure that your repository includes your changes to `problem6.py`.

Extra problem: try to implement BFS too! Then you can use the `dfs` and your `bfs` function to check your answer to Problem 1. You may also want to try implementing the graph coloring algorithm we saw in Lecture 6F.