Summary:
Given two projects, what are the behaviors for merging? Can they merge, and if so, is it a FFM (Fast-forward Merge) and 3WM (3-Way Merge)?

Branching Clarification:

- After reading G1 and speaking to Wes, we are under the impression that branching is not required;
- There are only two things that can be merged at any given time: A local project and a remote project. These are two projects, possible on separate machines, that share a project id.
    - Projects share a project id if one was created by cloning another;

Notation:
MRCA, RH, LH are defined in "Terms for Merging"
*_predicates are defined in "Predicates for Merging"
(MRCA, RH] = the set of commits between MRCA and RH. Not including MRCA, including RH.

Design Decision:

- Merging will overwrite future commits if LH is not at leaf
    - This is nice because it is the same behavior as moving HEAD back then making another commit.

Merge: (remote_project, HEAD_REMOTE) --into--> (local_project, HEAD_LOCAL)

- Fail
    a. Occurs when
        - Project ids are not identical
        - There are conflicts: both projects changed the same part of the same file
    b. Report Error to User
- FFM
    a. Occurs when:
        - FFM predicate returns true

b. Copy commits in (MRCA, RH] remote project
c. Change outgoing pointer *of* MRCA (in local) *to* first copied commit
d. Change parent pointer *of* first copied commit *to* MRCA (in local)
e. Update internal structures to reflect the change
    - HEAD Pointer
    - Potential meatadata/data structures if there is any

- 3WM
    a. Occurs when:
        - 3WM_predicate returns True
    b. Copy commits in (MRCA, RH] in remote project
        - This amount to copying these directories into .dvcs
    c. Pointer Manipulation
        - In MRCA
            - Add a child pointer → first copied commit
        - In first copied commit in (MRCA, RH]
            - Clear parent pointers
            - Add parent pointer → MRCA
        - In last copied commit in (MRCA, RH] (== RH)
            - Clear child pointers
            - Add child pointer → merge commit
        - In Merge commit
            - Add parent pointer → LH
            - Add parent pointer → RH
        - Move HEAD to Merge commit

findMRCA:= finds the most recent common ancestor

- Takes two projects and returns their mrca

MergeResolveDiff: commit->commit->commit

- Given the 3WM_predicate returns True, how should the merge commit be formed?
    - Find MRCA

- Perform diffs of **commit 1** with **MRCA** and **commit 2** with **MRCA**
- Walkthrough the change blocks identified in the two diffs, check if there are conflicts. React accordingly:
    - Conflict situations:
        - Can only happen when both commits are not empty
            - print out the locations of conflicts, and indicate the users to resolve manually
            - but do we create a commit here?
    - No conflict situations:
        - Both empty
            - either create an empty commit or do nothing
        - One commit is empty, while the other is not
            - create a new commit which contains content of the non-empty commit
        - Two commits both are not empty
            - create a new commit by combining the two commits

Reference:
https://www.atlassian.com/git/tutorials/using-branches/git-merge
https://stackoverflow.com/questions/14961255/how-does-git-merge-work-in-details