Advanced AI Assignment 1

This folder contains the problem statement and the basic code for the assignment. You are allowed to to attempt this assignment either in python, or in C/C++. You are provided with minimum boilerplate code to begin with, including code to read observations from a file. You need to fill up the missing code such that all conditions mentioned in the question are met.

Problem statement:

A detective is investigating a series of burglaries that have occurred in a wealthy neighbourhood over the past month. The burglaries have been happening at different part of the city and the detective believes that he has found the suspect, but does not have enough evidence to make the arrest yet, so he starts gathering all information about the burglar to notice any pattern in the suspect's actions. The detective believes that at any given time the suspect can be in the following states

- **Planning** : The suspect is planning where to steal from
- **Scouting** : The suspect is trying to scout the location planned beforehand
- **Burglary** : The suspect is performing the burglary
- **Migrating** : The suspect is moving elsewhere
- **Misc** : The suspect is not doing anything related to burglary

The detective has gathered all observations from various camera footage, witness testimony, investigating the places visited by the suspect, etc. His observations are composed of a tuple of Daytime and Observed action. The Daytime can be either of **Day, Evening, Night** and The Observed action can be any of:

- **Roaming** : Suspect is seen roaming the streets
- **Eating**: Suspect is observed to be eating outside in a restaurant or an eatery.
- **Home**: Suspect is observed to be at home
- **Untracked**: Suspect's whereabouts are unknown

Currently the suspect is on the run, with their whereabouts unknown. The detective believes that once found, this suspect would be likely doing the same activities in some other part of the city. While the search is ongoing, the detective decides to use the concept of Hidden Markov Model in order to better understand the profile of the suspect. His first course of action would be to make a simple implementation of a HMM defined upon the given states and observations.

The following assumptions are made for the suspect:

- The suspect always plans for a few days, then scouts for a few days. Thereafter, the suspect breaks into a house, and then immediately migrates the next day. This patterns is to be expected, with a few miscellaneous activity done in before the burglary.
- The suspect remains untracked most of the time while burgling, or migrating.
- While planning, the suspect spends most of their time in the house.
- The suspect can scout at any time of the day roaming in the streets, and mostly prefers scouting during the night.
- The suspect prefers to have food at eateries in the evening. It is believed that there is no significance of this action with respect to the burglary.
- On arriving at a new area after migration, the suspect begins with planning and other miscellaneous activities in the new place all over again.

These assumptions should hold after obtaining the trained model from the implemented program. Now, there are 3 additional use cases where the detective wants to use this model:

Part I

Complete the code to read a given database of observations on the suspect in order to prepare a Hidden Markov Model that represents the behaviour of the suspect.

The detective has spent a lot of time gathering evidences regarding the burglaries. With the collected data, the detective wishes to learn an HMM model that represents the behaviour of the suspect. The code to read the database is provided, so you only need to work on the algorithm to get the HMM model from the observed data. The observation is given as a list of sequence, where each sequence represents what the suspect did during their entire stay, up until moving to a different location.

The function that needs to be implemented is `LearnModel()` in the file `hmm.cpp` or `hmm.py`.

Part II

Complete the code such that given a HMM model and a sequence of observation, the likelihood that the observation has been generated by the given model is evaluated.

Now that the HMM model has been learned to mimic the behaviour of the suspect, the detective now notices that his co-workers are also working in burglary cases in other areas of the city. The detectives has a hunch that this suspect is also the cause of these other thefts. So he asks his co-workers for the observations that they noted in their investigations.

Now, with the prepared HMM model that represents his suspect, and a sequence of observations that were noted by other detectives in some other burglary cases, the detective now wants to see the likelihood of his current suspect's involvement in the other cases.

The function that needs to be implemented is `Liklihood()` in the file `hmm.cpp` or `hmm.py`.

Part III

Complete the code such that given a HMM model and a sequence of observation, the most likely sequence of states within the HMM is evaluated.

Now, given a list of observations of the suspect, the detective wishes to obtain the most likely sequence of the hidden states the suspect went through to produce that sequence of observations. This will come in handy when the suspect is found, allowing the detective to know the current state of suspect, and allowing him to catch and arrest him in the act.

The function that needs to be implemented is `GetHiddenStates()` in the file `hmm.cpp` or `hmm.py`.

Part IV

Create a new HMM model on some additional custom hidden states, and report the performance compared with the previous model.

Come up with some additional hidden states to the already defined enum `SuspectState`, in a newly defined enum, say `UpdatedSuspectState`. Next, make some updates to the basic assumptions as provided above such that all newly added hidden states are accounted for. Next, create a class for a new HMM model (say `Updated_HMM`) that works on all the newly defined

hidden states and on the same `Observation` structure as before. Use the boilerplate code to read the database, and implement functions to Learn the new `Updated_HMM` instance (Similar to **Part I**), check likelihood of a given observation sequence (**Part II**), and getting the most likely hidden state sequence for a given list of observations (Similar to **Part III**).

Now with two instances of the different HMM models (one of `HMM` and one of `Updated_HMM`), compare the results of the functions defined above on the same sequence of observation, and report your observations in the PDF report. You are required to write the code used to showcase the difference in behaviour of the two models in the file `new_hmm.cpp` or `new_hmm.py`.

General Guidelines for the assignment

- You can attempt the assignment in either C/C++ or python.
- The sections **Part I, Part II, Part III** are to be done in the file `hmm.cpp` or `hmm.py`. The section **Part IV** is to be done in the file `new_hmm.cpp` or `new_hmm.py`
- Additionally, prepare a PDF report for your code, explaining the problem solution and the logic within your code for the first three parts. For Part IV, you should state your updated assumptions, and give a brief overview of comparison between the old and the new HMM models.
- This database to read the observation is provided to you. The function to read and parse this file is also included in the boilerplate code. You only need to implement the functions related to the HMM.
- Complete the given source code files, and upload the two implemented files along with the report. (Do not include any additional folders or compress in a the zip file)
    - For students implementing in C/C++, upload the following on google classroom

      ```
      ├── hmm.cpp
      ├── new_hmm.cpp
      └── Report.pdf
      ```

    - For students implementing in python, upload the following on google classroom

      ```
      ├── hmm.py
      ├── new_hmm.py
      └── Report.pdf
      ```

- For evaluation of your assignment, your code will be tested with additional hidden test statements. They will be of the same format as the test cases provided to you. So ensure that the provided test cases are passing in your machine.
- The tests require reading the file `database.txt`. So ensure that this file is present in the same folder in your current working directory.
- Do not edit `tests.cpp` or `tests.py`. Do not add any other file than the ones already present in the boilerplate code.
- Needless to say, you should also not edit the defined name of HMM class, enums, or the given function names. Doing so will ensure that your code will fail the hidden test cases once you submit your code.

- The hidden tests will only use valid types, so there is no need for error handling within the code. There is also no need for any performance optimisation. Your code is expected to return the correct answer for queries on the functions that you need to complete, and you will be graded accordingly.
- For students implementing in C/C++, compile and run the file `tests.cpp` with `g++`. Run the executable file with the `dataset.txt` in the same folder. Similarly, students implementing in python should run the file `tests.py` while keeping the `database.txt` in the same folder.
- It is expected from all students to maintain their integrity and do the assignment on their own. Any cases of cheating/copying will be awarded 0 marks.
