CS 1501 Project 3

Released: Monday, October 14

Due: Tuesday, October 29, 11:59 PM

Goal

To explore an advanced application of priority queues in order to gain a deeper understanding of the data structure.

Background

You will be writing a basic application to help a user select an apartment to rent. You will write a menu-based user interface driver program (to be run in the terminal, no GUI), but most of the logic will be in implementing a priority queue-based data structure. You should write a PQ-based data structure that stores objects according to the relative priorities of *two* of their attributes, making it efficient to retrieve objects with the minimum or maximum value of *either* attribute (whether an attribute is a min attribute, such as price, or max attribute, such as size, is defined at initialization). Your data structure should further be indexable to allow for efficient updates of entered items. You will want users to be able to enter details about apartments that they are considering renting. The user should then be able to efficiently retrieve the apartment with the highest square footage or lowest rent. You should assume that the user is looking for apartments in multiple different cities. Thus, the retrievals should be possible on the set of all entered apartments or on the subset of all apartments within a specific city (e.g., “lowest rent in Pittsburgh”, “highest square footage in San Francisco”).

Specifications

1. First, you must create a class to store data about apartments. Specifically, this class must contain the following information:

    - A street address (e.g., 4200 Forbes Ave.)
    - An apartment number (e.g., 3601)
    - The city the apartment is in (e.g., Pittsburgh)
    - The apartment's ZIP code (e.g., 15213)

- The monthly cost to rent (in US dollars)
- The square footage of the apartment

2. You must write a terminal menu-based driver program (again, no GUI). Specifically, your driver must present the user with the following options:

    1. Add an apartment
        - This will prompt the user for each of the above-listed attributes of an apartment to keep track of.

    2. Update an apartment
        - This option will prompt the user for the street address, apartment number, and zip code of an apartment, and then ask the user if they would like to update the rent for the apartment.

    3. Remove a specific apartment from consideration
        - This option will prompt the user for the street address, apartment number, and zip code of an apartment to remove from the data structure (e.g., if it is no longer to rent)
        - Note that this mean you will need to support removal of apartments other than the minimum rent or maximum square footage.

    4. Retrieve the lowest rent apartment
    5. Retrieve the highest square footage apartment
    6. Retrieve the lowest rent apartment by city
        - This option will prompt the user to enter a city and then return the apartment with the lowest rent within that city.

    7. Retrieve the highest square footage apartment by city
        - This option will prompt the user to enter a city and then return the biggest apartment within that city.

    Note: Retrieval operations should *not* remove the apartment with minimum rent or maximum square footage from the data structure, just return information about that apartment. Apartments should *only* be removed via the “remove a specific apartment from consideration” menu option.

3. To aid in the testing of your application, you will find an example file with some test data stored in your folder (`apartments.txt`). Your progam should read in the contents of this file to initialize your data structure each time it is run. You can assume that this file will already exist, and you do not need to write an updated version of the data structure back to the file.

4. To ensure efficiency of operations, you must base your data structure around the use of heaps with indirection (making them indexable). Note that retrieval operations on either attribute (e.g., retrieve minimum rent, retrieve maximum square footage) **must have a $O(\log n)$ runtime** (both for all apartments and for those within a specific city). Updates and removals must also have a $O(\log n)$ runtime. Take care

in selecting your approach (especially the indirection data structure) to account for the types of keys you will need to store and the type and number operations that you will need to perform on them.

5. Because this project requires you to make a number of decisions about how to implement its requirements, write a documentation file explaining your implementation and **justifying your decisions**. Name this file `documentation.txt`. Document your approach carefully, to reduce the effort required to trace through your code during grading. State the runtime and memory requirements of your approach, and use this information as part of your explanation of why you chose your approach.

Submission Guidelines

- Upload your submission to the provided Box folder named `cs1501-p3-abc123`, where `abc123` is your Pitt username.
- **DO NOT** upload any IDE package files.
- You must name the primary driver for your program `AptTracker.java`, and it must be outside of any package.
- You must be able to compile your program by running `javac AptTracker.java`.
- You must be able to start your program by running `java AptTracker`
- You must document and justify your approach in `documentation.txt`.
- You must fill out `info_sheet.txt`.
- The project is due at 11:59 PM on Tuesday, October 29. Upload your progress to Box frequently, even far in advance of this deadline. **No late submissions will be accepted.** At the deadline, your Box folder will automatically be changed to read-only, and no more changes will be accepted. Whatever is present in your Box folder at that time will be considered your submission for this project—no other submissions will be considered.

Additional Notes and Hints

- You are free to use code provided by the book authors in implementing your solution. It is up to you to decide if it would be easier to modify the provided code to meet the requirements of this project or to start with a clean slate with entirely your own code.

- You may consider adding additional rows to `apartments.txt` to expand your testing. During grading, we will test by placing other data in `apartments.txt`, so you should be sure your code works for a wide variety of scenarios and corner-cases.

Grading Rubric

Feature	Points
Adding an apartment works properly	10
Updating an apartment works properly	10
Removing an apartment works properly	15
Retrieval for all apartments works properly	10
Retrieval for a given city works properly	15
Operations on either attribute are efficient due to heap-backed data structure	15
Validity of justifications	15
Menu-based driver program works properly and has appropriately labeled options	5
Info sheet/submission	5