CS 261 Final Exam

CS 261 – Fall 2023 – 10 points
Due on Canvas by 11:59 pm on Wednesday, 12/13/2023

This final exam is a take-home final exam. You may work on it on your own time and submit it via Canvas by the due date specified above. This is an "open-book" exam: you are free to use any resources that are helpful to you, including course lecture notes, internet articles, etc. If you use information from a 3rd-party source (e.g. from a Wikipedia article), make sure to cite your source (e.g. "According to Wikipedia's article on queues…"), and, if the source is an internet article, include a link.

Choose a data structure to implement a real-world feature

For the final exam, choose ***ONE*** of the software features described below. For your chosen feature, pick the data structure from the list below that you think would be best suited to implement that feature, and describe why that data structure is the best choice to implement your chosen software feature.

Here is the list of data structures you can choose from to implement your software feature. Note that this list ***DOES NOT*** include the dynamic array or the linked list, so you may not choose either of those data structures as the basis for your chosen software feature's implementation. (Some of the data structures listed here can be built *on top of* a dynamic array or linked list. That's OK.)

- Stack
- Queue
- Deque
- Binary search tree
- AVL tree
- Priority queue
- Binary heap (minimizing or maximizing)
- Map/hash table

Assignment requirements

After you've chosen one of the software features described below and selected the best data structure to implement it, write a short document (1-2 pages at most) that satisfies the following requirements:

- Your document should outline all the requirements of your chosen software feature that are relevant to choosing a data structure to implement that feature. For example, does

your chosen software feature have specific memory or runtime requirements? Does the software feature require data to be ordered a certain way? Be as detailed as possible here, and include every requirement of your chosen feature you think might be relevant.

- Your document should name the data structure from the list of choices above that you think is best suited to implement your chosen software feature, and it should describe *how* that data structure can be used to implement that feature. If a special configuration of the data structure is needed to implement your chosen software feature (e.g. multiple instances of that data structure are needed to implement the software feature), make sure to describe that special configuration and to explain why it's necessary.
- Your document should explain how your data structure satisfies all of the requirements you listed above. If there are any trade-offs involved here (e.g. your chosen data structure is still the best choice even though it doesn't satisfy a particular requirement as well as a different data structure might), make sure to describe those trade-offs, too. Again, be as detailed as possible here.
- **Your document should be concise!** Imagine you're writing this document for your busy engineering manager in your first software development job. You want to convince your manager what the right data structure is without making them spend too much time reading your document. Assume your manager knows what feature you're working on and that they know what all the possible data structure candidates are and how each data structure works. Focus on telling your manager ***what specific characteristics*** of your chosen data structure make it the right one to satisfy the requirements of your chosen feature and, if needed, how the data structure needs to be configured to implement the feature.

Software features to choose from

Choose ***ONE*** of the following software features, and write about the data structure you think is best suited to implement your chosen feature, as described above.

Undo/redo in a 3D world building application

You are working on a team creating a 3D world building application that will be used by game developers to build worlds for their games. The application allows the user to execute various actions to build a world piece by piece, e.g. create a new shape/object in the world, drag a shape/object into place, resize a selected shape/object, change the color/texture of a selected shape/object, etc. Within the application code, every time the user takes one of these actions, an instance of a structure called `struct action` is generated to describe all of the relevant information about the action.

You have been assigned to work on an undo/redo feature within this application. This feature will allow the user to undo actions they've already taken and to redo actions they've undone, just like an undo/redo feature in a code editor or word processor. In particular, when the user performs an "undo", your feature will undo the most recent action taken by the user. The user

can continue to "undo" as many times as they want, and each subsequent "undo" will undo the next most recent action taken by the user. Similarly, when a user performs a "redo", it will re-execute the most recently undone action. A user may also "redo" as many times as they want, and each subsequent "redo" will re-execute the most recently undone action.

Your first task in designing this feature is to choose a data structure to use in its implementation. Each time the user takes an action within the application, a reference to the corresponding instance of the `struct action` structure will be passed to your undo/redo feature, and your implementation must somehow store and organize these `struct action` instances so that when the user executes an "undo" or a "redo", the corresponding `struct action` can be quickly and easily accessed and used to undo or redo the action it represents. Choose a data structure to use for storing and organizing these `struct action` instances to implement the undo/redo feature.

Load balancer for a web service

You are working on a team developing a large-scale web service. At a high level, this service accepts requests from machines on the internet, processes those requests, and sends responses back to the machines that initiated the requests. Because your team expects your web service to receive a high volume of requests, you plan to have several physical web servers powering your service, each of which will run the service and process requests.

You have been assigned to work on a load balancer for this web service. The load balancer's job will be to serve as the "gateway" from the outside world to your web service, distributing traffic to the individual physical web servers powering the service. Specifically, the load balancer will be the part of your web service that accepts new requests from the internet. It will store these requests, and when one of the physical web servers powering your web service is ready to process a new request, your load balancer will send the next waiting request to that server for processing. Importantly, the load balancer must ensure that requests are sent to servers in the same order they are received. This is the only criterion used to order requests.

Your first task in designing the load balancer is to choose a data structure to use in its implementation. Specifically, each time your load balancer receives a new request, it generates an instance of a structure called `struct request` to represent details about that request. Your load balancer must then store and organize these `struct request` instances so that when one of your service's physical web servers is ready for a new request, the load balancer can quickly and easily access the `struct request` instance representing the next request waiting to be processed so it can send the details of that request to the waiting server. Choose a data structure to use for storing and organizing these `struct request` instances within your load balancer.

Collection of data records in a database service

You are working on a team developing a new database service. This service will store data records, and clients will be able to issue commands to your database service to generate and store new data records, to search for existing data records, and to modify or delete existing records. Each data record will be represented by two pieces of information:

- An identifier represented as a 128-bit integer value. Each identifier will be automatically generated by the database service and will uniquely identify the data record it is associated with.
- An instance of a structure called `struct record`. This structure will be capable of representing the specific data associated with each data record.

Within your database, data records will be divided into "collections", which are simply sets of related data records. You have been assigned to work on designing and implementing a representation for a "collection" of data records within your database service, and your first task in doing so is to choose a data structure to use to represent a collection.

In particular, your implementation of a collection will receive the unique identifier and `struct record` instance representing each new data record created within the collection, and it must store and organize these values so they can be quickly and easily accessed later. Importantly, the unique identifier will be the only value by which a data record can be accessed within a collection (i.e. searching for, modifying, and deleting data records will always require the unique identifier associated with the record to be found, modified, or deleted). Each collection must be capable of storing millions or even billions of data records while still allowing records to be created, retrieved, modified, and deleted very quickly. Choose a data structure to use to store and organize the unique identifiers and `struct record` instances within a collection.

Submission

To submit your final exam, type up your analysis in a separate document, save that document as a PDF, and submit the PDF on Canvas under the "Final Exam" assignment by the due date specified above.

Grading rubric

Here is the rubric that will be used to grade your submission:

- 5 points – assessment of the requirements of the chosen software feature
    - 5 points – clearly describes all of the relevant requirements of the chosen feature
    - 3 points – misses important information and/or identifies only some of the relevant requirements of the chosen feature but misses others
    - 1 point – incoherently, inadequately, or inaccurately describes the requirements of the chosen feature or only identifies inconsequential requirements
    - 0 points – does not identify any meaningful requirements of the feature

- 5 points – description of and justification for chosen data structure
    - 5 points – describes a data structure that could be used to implement the chosen software feature, and correctly and completely describes how that data structure can be used to implement the feature while satisfying all of its relevant requirements
    - 3 points – partially describes how the chosen data structure can be used to implement the chosen software feature while satisfying its relevant requirements, but either misses relevant points or makes erroneous claims
    - 1 point – incoherently, inadequately, or inaccurately describes how the chosen data structure satisfies the relevant requirements of the chosen feature
    - 0 points – does not meaningfully describe how a particular data structure satisfies the relevant requirements of the chosen feature

Plagiarism review

Note that your final exam submission will be reviewed for plagiarism, so make sure you do your own work and write your analysis in your own words. Any student found to commit plagiarism on the final exam will be penalized as described in the course syllabus.