CS 225: Exam 4

Review Session

Content

Very similar to last time!

- MCQ
- FRQ
- Coding Tips
- Questions

MCQ

Heap Structure

What characteristic of Heaps allow them to be stored efficiently in an array?

- (a) None of the other choices is a sufficient explanation.
- (b) Heaps are complete trees.
- (c) Heaps are perfect trees.
- (d) Heaps contain comparable keys.
- (e) Heaps are binary trees.

MCQ

Graphs

For a simple connected graph G with 51 vertices, which of the following options cannot be the number of edges of G?

- (a) All of the options are possible
- (b) 69
- (c) 48
- (d) 1328
- (e) 510
- (f) At least two of the options are impossible

FRQ

You are working on a small embedded system and have only a fixed amount of memory to work with. You need build the system that buffers request for information in case more come in than can be handled in a given time. The requests need to be handled in order that they are received. If more requests are received than you have space for you can ignore them.

You want to optimize for your speed of adding new requests as well as the speed of removal of the oldest request.

Explain your answer in the form of English sentences. As a guideline for your answer, be sure to:

1. Explicitly state the data structure you would implement to solve this problem.
2. Explicitly define variables for data size in the context of Big O. (For example, to say something is O(N), you must explicitly define N.)
3. Describe the interface (ADT) of the data structure as well as how you would use this interface to construct your data structure and solve the problem. **No implementation details about the functions are needed, just how you would use them.**
4. For each function described in (3), explicitly state the Big O of each function.
5. Justify why your solution is the optimal solution by highlighting what is good about your choice.

FRQ - Explicitly state the data structure needed

Key information:

- Fixed amount of memory
- Handle requests in order
- Do not need to accept requests if full
- Optimize adding and removing (oldest request)

Based on this information, which data structure is suitable?

FRQ - Explicitly state the data structure needed

Key information:

- Fixed amount of memory
- Handle requests in order
- Do not need to accept requests if full
- Optimize adding and removing (oldest request)

Based on this information, which data structure is suitable?

A queue, specifically based on the last point

FRQ - Explicitly state BigO variables

What should we measure in terms of?

Is there a specific upper or lower bound to this variable?

FRQ - Explicitly state BigO variables

What should we measure in terms of?

Is there a specific upper or lower bound to this variable?

Our BigO variable is n, which is the number of requests we receive (note that *n* is at most *m*, the maximum number of requests our fixed memory allows for)

FRQ - 1) Describe ADT and 2) How you would use it

We would use two functions in our queue ADT

- Enqueue, which adds an item to the back of the queue as soon as it is processed
- Dequeue, which removes an item from the top of the queue

FRQ - 1) Describe ADT and 2) How you would use it

We would use Enqueue() and Dequeue() as follows:

- Enqueue - as each request comes in, we will add it to the back of the queue *if space permits*
- Dequeue - when we need to handle a request, we will remove it from the front of the queue

Make sure to use this section to connect the data structure to the real life task at hand – explain why this structure is helpful in the context

FRQ - Describe the BigO of the ADT functions

Enqueue?

Dequeue?

FRQ - Justify why your choice is correct

Since all of our ADT functions run in O(1) time, we cannot improve upon our runtime. By using a fixed size queue, we make sure that we never add more requests than we have memory for. Since it is fixed size, we never have to resize the queue either.

Make sure to emphasize the runtime/space complexity (depending on which is being asked), and why that one cannot be improved upon. Also, if there are additional aspects of the question, like the fixed memory case here, make sure to explain how you handle that, and why it is the best way to handle it

Coding

General Tips:

- Spin up your coding workspace while reading the question
- Skim through the header file
- Be careful with the function declaration (use spiral rule)

  e.g int const * p vs int * const p

- #include <bits/stdc++.h>
  - Includes all stl headers
- Run local tests/memcheck in main.cpp
  - You can use print statements, valgrind, and gdb just like you can on an MP or Lab

Coding

Exam 4 Tips:

- The exam question will be related to heaps
- Watch out for these key concepts when working with heaps
  - Is it a min or max heap?
  - Is it 0 or 1 indexed?

Coding

To refresh your memory on heaps, you can reference the following:

- Lectures: 6/27, 6/30
- Assignments: Lab_heaps
- POTDs: #34, 35
- Other: Heaps notes in “Resources” tab