Week III Study Guide:

**QUEUES AND DEQUES:**

A queue is a simple data structure, yet it is distinctly more complex than the predecessor, the stack.

It can be easily visualized as the familiar line, such as one at the grocery store, and a movie theater.

Whereas a stack is FILO, or (LIFO as you prefer) a queue is FIFO. There is an alternative queue called a priority which arranges the order of things on the basis of a priority value.

There is also a variation of queue called the DEQUE (pronounced DECK), which stands for *double-ended queue.* In a Deque, data can be inserted from the front or the back.

These are both valuable for computer applications and computer science, as a queue can be used to set up task and job arrangements, such as a printer queue and a deque can handle other tasks that may require data to be pulled from both ends of the data structure.

Most of the commands relating to queues are as follows:

addBack(element) – *push() / addTop() / etc.*

front() – *peek() / top()*

removefront() – *pop()*

isEmpty() – *isEmpty()*

Notice the fundamental similarities to a stack. Many of the commands are similar and have similar functions. I have italicized the equivalent above.

A deque adds the following controls:

addFront(element)

back()

removeBack()

As far as we have learned so far, there are no equivalents. Deques are used as specialized queues and queues can be considered specialized deques.

As with a stack, the FIFO structure is the defining characteristic of a queue.

In a deque, the defining characteristic is that things can be removed either from the front or back directly.

**Applications of Queues**

Remember the following applications for queues:

\* Searching a maze

\* Printer Queues

\* Depth First Search – which moves through the depths of a structure before examining the alternatives. Consider this versus a Breadth-First Search, which looks at the breadth of a structure before going deeper. …this is important for searches and sorts!

C/P’d from chapter:

**How to search a maze using breadth-first search:**

Keep a list of squares you have visited, initially empty.

Keep a queue of squares you have yet to visit.

Put the starting square in this queue

While the queue is not empty: Remove an element from the queue If it is the finish, then we are done

Otherwise if you have already visited this square, ignore it

Otherwise, mark the square on your list of visited positions, and add all the neighbors of this queue to your stack.

If you eventually reach an empty queue and have not found the start, there is no solution

Linked List Queues:

Whereas a stack only needs a link to maintain one end of the chain of values, both insertions and removals occur on the side. In a queue, the insertions and removals happen on opposing sides. Thus, it is critical to build and maintain links from both the front and back of the collection. You thus will have to build a first link and last link.

Enter the *sentinel* – a special link that does not contain a value. This will be used to mark either the beginning or end of the chain. When a sentinel is in the front of a linked list queue, it is often called the *list header*.

Sentinels are useful for handling special cases and ensures there is never an empty link that might cause a segmentation fault or other issue with the program. Afterwards, a new value is inserted and the links are updated to represent the new value.

Values are removed from the front, like the stack, but because of the sentinel, these elements are right after the sentinel.

**A Linked List Deque**

This version of a linked list queue uses double links. They have double sided links and allow us to remove things from the back of the linked list.

In a doubly linked list, each link maintains *two­* pointers. Now things are getting tricky!

The other link, the prev link, points backwards, and the next link points forwards. Anticipating future applications, we now also keep a count of the number of elements in the list.

**Dynamic Array Queues and Deques:**

And just like our old friend, the dynamic array, we now can have dynamic array deques. Implementing the array as a stack will not work for queues or deques, so we have to create a new way of doing this.

Imagine if you will, trying to implement the old loop processes for the dynamic array stack. It is unfortunately much to slow O(n) for our purposes. Any insertion or removal operation will be much too slow and frustrating, so we have to find a new method of approach. NOTE: ADDING IS EXPENSIVE, putting stuff to the back isn’t necessarily so.

The first issue we encounter is trying to apply our old models of thinking to dynamic array deques/queues. We have to forget the array collection starts at 0 rule. This is where we place our sentinel. Also, there will be two integer array fields (as before), except this time it will be the size and starting location instead of size and capacity.

Our starting location will be where we begin inserting data. Of course, given that this is a queue or deque, we have to obey their rules. When we add a value or bit of information, we increase the size and place the element at the end or before the starting location (depends if we are adding front or back).

Adding the value to the front is simple, decrement starting point by one, add element.

Removing elements will undo these operations and therefore, are performed like so. If you reach capacity, capacity doubles, and values will be copied in to the new array.

This will require a circular buffer – which circles around the queue or deque and begins again at its respective side.

**SELF-STUDY GUIDE:**

1. What are the defining characteristics of the queue ADT?

First In, First Out ordering.

1. What do the letters in FIFO represent? How does this describe the queue?

First In, First Out, it explains the queue like a line-up.

1. What does the term deque stand for?

Double Ended Queue.

1. How is the deque ADT different from the queue abstraction?

The Deque allows you to add things to the back as well as remove them from the back. In addition, you can peek at the back.

1. What will happen if an attempt is made to remove a value from an empty queue?

If you have the IsEmpty() set up, it will respond that the queue is empty and doesn’t exist. Otherwise you could potentially hit a segmentation fault or other program termination.

1. What does it mean to perform a depth-first search? What data structure is used in performing a depth-first search?

A depth first search means the queue is looking at the ‘depth’ of a structure before looking for alternatives. It banks on the chance of finding the desires element in the depth of the queue.

1. How is a breadth-first search different from a depth-first search? What data structure is used in performing a breadth-first search?

A breadth-first search is far more thorough than the depth-first search. A breadth-first search will look for as many alternatives as possible before moving onto the next step and explores those at the same time. It also takes far more time than a depth-first search. They are used in queues.

1. What is a sentinel in a linked list?

It is a placeholder node, essentially, which does not contain a value. The sentinel is used to mark either the beginning or end of a chain of links. Thus, it prevents a list from being empty, even if it is logically empty.

1. What does it mean to say that a list is singly-linked?

When a list is singly linked, it goes through data linearly, and it is impossible to back up. We used sentinels to help with this issue, but you can’t do anything to look at the preceeding point.

1. How is a doubly-linked list different from a singly-linked list? What new ability does the doubly-linked feature allow?

A doubly linked list, each node/link maintains two pointers. You have a forward and back link, usually called prev and next. This allows you to traverse the list far more easily. It allows you to insert in the middle of a list as well.

1. Why is it difficult to implement a deque using the same dynamic array implementation that was used in the dynamic array stack?

Using the dynamic array implemenetation of the arraystack is a slow and ponderous exercise. You also need to use an o(n) loop to insert anything into the queue/deque. Adding things also takes a considerable amount of time. Thus we use a sentinel/placeholder point in the array, and move it through the list as a start point.