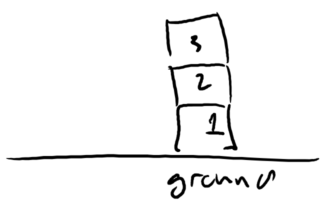
2019-09-26 Stacks and Queues

Thursday, September 26, 2019 8:59 AM

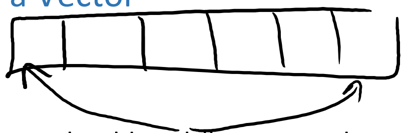
- Both stacks and queues define how we insert, access, and retrieve data
    - NOTE: neither stack nor queue specify how data should be contained (e.g. in Vector or LL)
- A stack has the following rules
    - The order in which things are inserted are the opposite of the order in which things are remove
    - In other words, what goes into the stack first must come out of the stack last (FILO)
    - In other words, what comes into the stack last must come out first (LIFO)
- A queue has the following rules
    - Items are removed from the queue in the order in which they are inserted
    - In other words, what goes in first, comes out first (FIFO)
    - In other words, what goes in last comes out last (LILO)
- The quintessential stack is a stack of boxes



- Stacks are pretty common in programming. In fact, every program has at least one stack - call stack
- Stacks are also used in recursion and searching (depth-first search)
- Stacks are also very handy implementing undo and redo
- Stacks have two operations:
    - Push() -> add item to "top" stack
    - Pop() -> remove item from "top" of stack
- Queues are also very common in real life
    - Lines of people
- Also popular in computing
    - Networking
    - Printing
- Queues are also used in searching (breadth-first search)
- Queue operations
    - Enqueue() / (C++) Push() -> add item to "end" of queue
    - Deqeue() / (C++) Pop() -> remove item from "front" of queue
- Bonus Data Structure: Deque (pronounced "deck")
    - Works like a deck of cards
        - Add to top of deque
        - Add to bottom of deque
        - Remove from top of deque
        - Remove from bottom of deque

Implementing A Stack

...using a Vector

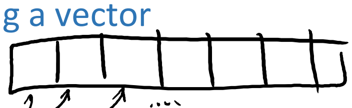
- 
- Where should push() put new elements?
    - Push to front is bad idea because it is O(N)
    - Push to end is good because O(1)
        - Pushing to end means that we must pop from end as well

...using a Linked List

- Can push to front or end
    - Push to front O(1)
    - Push to end O(1) assuming intelligent LL design
    - Regardless of choice O(1) on pop()

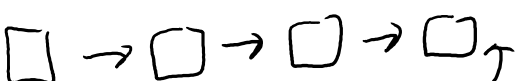
Implementing a Queue

...using a vector



- New items get added to end of vector O(1)
- Therefore, items must be dequeued from element 0
    - O(N)

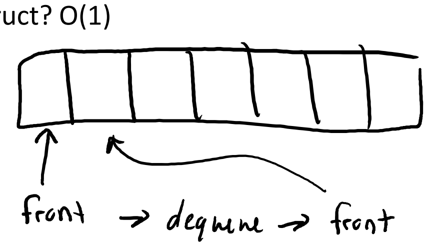
...using a linked list



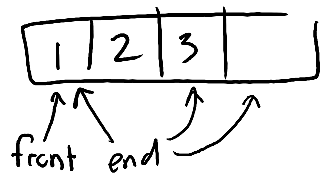
- New items are added to end of LL
    - O(1)
- Items are dequeued from the front
    - O(1)

Improve a vector-based queue's performance

- Observation: performance is slow because we must always dequeue from element 0 (physical constraint)
- Why must the front of the queue be tied to a physical location?
    - Why can't we make the "front" of the queue a logical construct? O(1)

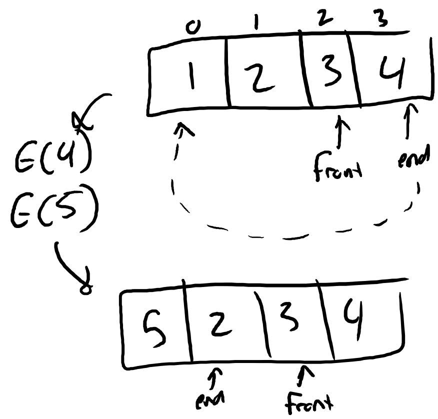


- Major downside: After a dequeue operation, the vector will contain empty wasted space.
    - E.g. if we do 1k enqueues and dequeues, there will be 1k empty boxes that can't ever be used again.
- Observation #2: make enqueue a logical operation
    - I.e. add to "logical" end of queue



Enqueue
data[end] = value;
end++;

- Operation: E(1), E(2), E(3), D(), D()



end = (end + 1) % size;

- Resize when end == front AND logical size == physical size
- This data structure has O(1) enqueue and O(1) dequeue
- This queue structure is called "circular queue"