STACKS & QUEUES

CS136: Data Structures & Advanced Programming

March 15, 2017

LAST CHANCE MIDTERM QUESTIONS

LINEAR STRUCTURES

- What if we want to impose an **ordering** to our lists?
- I.e., provide only one way to add and remove elements from list
  - No longer provide access to middle
- Order of removal depends on the order elements were added
  - LIFO: Last In First Out
  - FIFO: First In First Out

EXAMPLES

- FIFO
  - Line (queue) at grocery store
  - Line at dining hall (hopefully)
- LIFO
  - Stack of trays at dining hall
  - Stack of cups
  - Deck of cards

LINEAR INTERFACE

- We need another interface!
  - Should have fewer methods than List interface since we are limiting access…
- Methods:
  - addFront/Back(E value) - Add a value to the structure.
  - boolean empty() - Returns true iff the structure is empty.
  - E getFront/Back() - Preview the next object to be removed.
  - E removeFront/Back() – Remove the next value from the structure.
  - int size() - Returns the number of elements in the linear structure.

LINEAR STRUCTURES

- **No “random access” to list elements!**
  - This means no access to middle of list
- More restrictive than general List structures
  - More implementation freedom
  - More efficient for *some* uses
  - More choices to think about when building our programs

STACKS

- Applications:
  - TODO list, implementing recursion
- What methods do we need to define?
  - Stack interface methods
- New terms: push, pop, peek
  - **Push** = add to top (back) of stack
  - **Pop** = remove from top (back) of stack
  - **Peek** = look at top of stack (but do not remove)

STACK IMPLEMENTATIONS

- Fixed-length array
  - int top, Object data[ ]
  - Add/remove from index top

+ all operations are O(1)

– always wasted/run out of space

- Vector
  - Vector data
  - Add/remove from tail

+/– most ops are O(1) (push: O(n) worst case)

– potentially wasted space for capacity

- Linked List
  - SLL data
  - Add/remove from head

+ all operations are O(1)

– nodes guarantee high space overhead

EVALUATING ARITHMETIC EXPRESSIONS

- Computer processes use stacks to evaluate arithmetic expressions
- Example: x*y+z
  - First rewrite as xy*z+ (we'll look at this rewriting process on Friday)
  - Then:
    - push x
    - push y
    - mult (pop twice, multiply, push result)
    - push z
    - add (pop twice, add, push result)

QUEUES

- Applications:
  - Print jobs, GUI events, network messages
- Operations
  - Push back (“enqueue”)
  - Pop front (“dequeue”)
  - Size
  - Empty
- Many implementation choices…

QUEUE IMPLEMENTATIONS

- Fixed-length array
- “Circular buffer” fixed-length array
- Vector
- Circular buffer Vector
- List (with tail pointer)

DEQUE

- Applications:
  - Queue with regrets, work-stealing
- Push front
- Push back
- Pop front
- Pop back
- Size

SUMMARY

- *Limiting* a data structure to a specific usage pattern can paradoxically be powerful
  - Implementation freedom
  - Avoid usage bugs
- Stack = LIFO
- Queue = FIFO
- Good luck on the midterm tonight! Bronfman 7pm or 8:30pm