CSC 220 Data Structures

Stacks, Queues, and Deques

Parkland College Fall 2016

20160921
Stacks, Queues, and Deques

• Defined by their interface (API) more than their underlying implementations

• Will base on Python lists today
  • primitive arrays in C
  • std::vector in C++
  • ArrayList in Java
Stacks

• Stack of dishes, PEZ dispensers, undo lists, browser history, etc.

• LIFO/FILO
  • last in, first out/first in, last out

• API:
  • `push()` – put item on stack
  • `pop()` – remove item from stack

• `array_stack.py`
  • adapter design pattern
Stack complexities

• (chart in web notes/textbook)
• Consequence of underlying implementation
• push/pop could trigger resize, so amortized
• alternative: implement as set size and live with consequences
  • e.g. function stack and stack smashing/uncontrolled recursion
Stack applications

• Common helper data structures
• **Call stack**
• open/close matching
  • parentheses
  • html/xml tags
  • embedded comments
• HP calculators, reverse Polish notation
• Lisp, scheme, Polish notation
Queues

- Theater/cafeteria/amusement park lines
- Print queues
- First in, first out
  - FIFO
- API
  - `enqueue()` – enter
  - `dequeue()` – leave
- `array_queue.py`
  - circular buffer instead of adaptor pattern
Queue complexities

• (chart in web notes/textbook)
• enqueue/dequeue could trigger resize, so amortized
Queue applications

- event driven schedulers
- GUI responses
- Printer queues
- Will expand with “priorities” later in semester
  - Priorities: way to cut in line
Deques, “decks”

- Can enter/leave either end
  - Restaurant lines, can decide wait is too long
- Same complexities as queue
- `collections.deque`
  - uses list-like API
Other Implementations

• Recall amortization only if time available for spreading cost, otherwise, we do pay for the cost!
• May need to guarantee complexities
• Can do this with next topic, Linked Lists.