Lecture 1: Intro, Stacks & Queues

CSE 332: Data Structures & Parallelism

Winston Jodjana

Summer 2023

Data Structures?

Clever ways to organize information in order to enable *efficient* computation over that information

Trade-Offs

- A data structure strives to provide many useful, efficient operations
- But trade-offs!
  - Time vs. Space
  - One operation more efficient if another less efficient
  - Generality vs. Simplicity vs. Performance
- That is why there are many data structures

Terminologies

- Abstract Data Type (ADT)
  - Mathematical description of a "thing" with set of operations on that "thing"
- Data Structures
  - A specific organization of data and family of algorithms for implementing an ADT
- Implementation of a data structure
  - The actual code implementation in a specific language
- Algorithm
  - A high level, language-independent description of a step-by-step process

Terminology Example: Stack

- The ***Stack*** ADT supports operations:
  - **push**: adds an item
  - **pop**: raises an error if isEmpty, else returns *most-recently pushed item* not yet returned by a pop
  - **isEmpty**: initially true, later true if there have been same number of pops as pushes
  - etc.
- A Stack data structure could use a linked-list or an array or something else, and associated algorithms for the operations
- One implementation is in the library **java.util.Stack**

Why useful

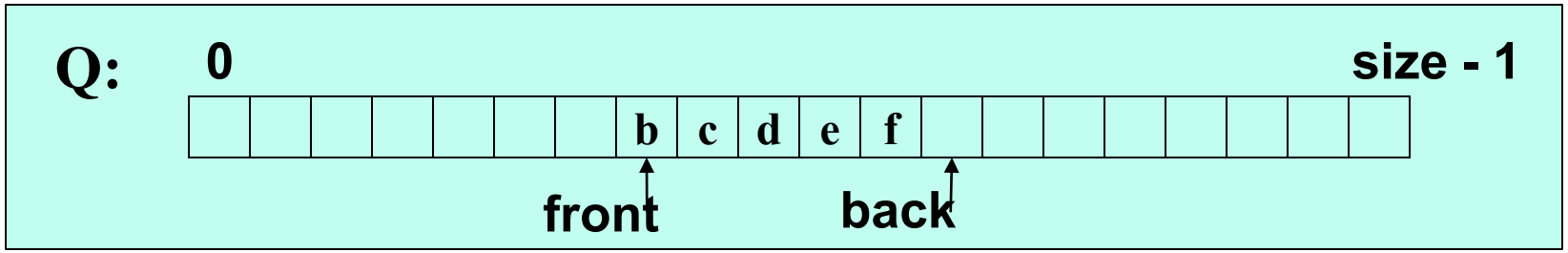
The ***Stack*** ADT is a useful abstraction because:

- It arises all the time in programming (see Weiss for more)
  - Recursive function calls
  - Balancing symbols (parentheses)
  - Evaluating postfix notation: 3 4 + 5 *
  - Clever: Infix ((3+4) * 5) to postfix conversion (see Weiss)
- We can code up a reusable library
- We can communicate in high-level terms
  - "Use a stack and push numbers, popping for operators…"
  - Rather than, "create a linked list and add a node when…"

Terminology Example: Queue

- The ***Queue*** ADT supports operations:
  - **enqueue**: adds an item at the end
  - **dequeue**: raises an error if isEmpty, else returns item at the start
  - **isEmpty**: initially true, later true if there have been same number of enqueue as dequeues
  - etc.
- A Queue data structure could use a linked-list or an array or something else, and associated algorithms for the operations
- One implementation is in the library **java.util.Queue**

Circular Array Queue Data Structure

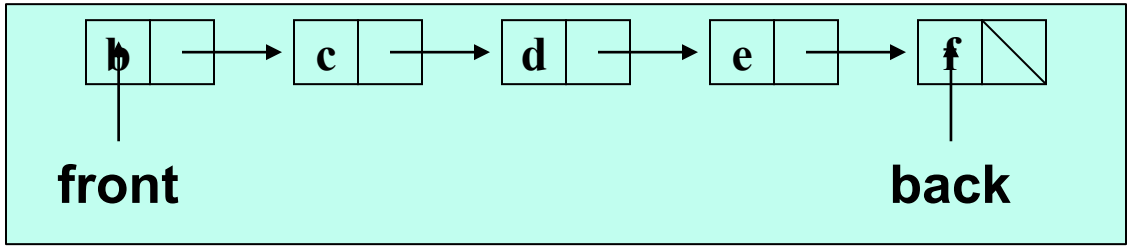



```
// Basic idea only!
enqueue(x) {
  Q[back] = x;
  back = (back + 1) % size
}
```

```
// Basic idea only!
dequeue() {
  x = Q[front];
  front = (front + 1) % size;
  return x;
}
```

- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- What if ***array*** is full?
- How to *test* for empty?
- What is the *complexity* of the operations?

Linked List Queue Data Structure




```
// Basic idea only!
enqueue(x) {
  back.next = new Node(x);
  back = back.next;
}
```

```
// Basic idea only!
dequeue() {
  x = front.item;
  front = front.next;
  return x;
}
```

- What if ***queue*** is empty?
  - Enqueue?
  - Dequeue?
- Can ***list*** be full?
- How to *test* for empty?
- What is the *complexity* of the operations?