Queues and Stacks

2501ICT/7421ICTNathan

René Hexel

School of Information and Communication Technology
Griffith University

Semester 1, 2012

Outline

Lists Review

- Lists decouple logical structure from memory layout
  - $\rightarrow$ important if contiguous memory is at premium
- O(1) insertion
  - $\rightarrow$ even if resizing is necessary!
- Accessing a random element takes O(n)
  - $\rightarrow$ "intrinsic O(n) overhead"
- Singly Linked Lists
- Doubly Linked Lists
- Circular Linked Lists

Structured Collections

Stacks and Queues

Stacks and Queues

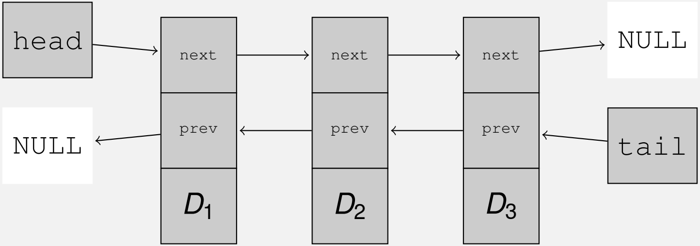
- Stacks and Queues are structured linear collections
- Can be implemented in different ways, e.g.
  - Arrays
  - Linked Lists
- Are often used in operating systems and run time environments
  - function/method calling Stacks in almost every programming language
  - process priority Queues in almost every operating system

Queues

- FIFO collections
  - first element enqueued is first to be dequeued
  - insertions occur at one end, the *rear*
  - removals occur at the other end, the *front*
- Priority Queue Extensions
  - reordering according to priority
  - few priorities: multiple Queues
  - many priorities: insert according to priority

Queues using Lists

- Use the head and tail Pointers



- Enqueue: Insert from the tail (rear)
- Dequeue: Remove from the head (front)

Priority Queues using Multiple Lists

- One list per priority
- Enqueue on the corresponding list
- Dequeue starting at the highest priority list
  - $\rightarrow$ go to next lower priority if empty, etc.
- Does not require search operation to enqueue
- Requires two pointers (head/tail) per priority:
  - faster `enqueue`, but slower `dequeue` operation
  - $\Rightarrow$ useful if number of different priorities is small

Priority Queues using Priority Ordering

- Only one list
  - enqueue according to priority
  - higher priority entries "overtake" lower ones
  - dequeue always from the head
- Requires linear search to enqueue: O(n)
  - dequeue doesn't require search: O(1)
  - ⇒ useful if number of different priorities is large

Queue Applications

- First-Come-First Serve Algorithms
  - → OS process scheduling
  - → event handling
  - → modelling and simulation of Real-World Processes
    - e.g. checkout queue in a supermarket

Stacks

- LIFO collection
  - the last element pushed onto the stack is the first to be retrieved
  - both push (insertion) and pop (deletion) operations occur at the top of the stack
- Implementation
  - arrays
  - singly linked lists

Array Implementation

- Index: `i = 0;`
- Push: `stack[i++] = element;`
- Pop: `element = stack[--i];`
- Peek: `element = stack[i-1];`
  - peek is a “sneak preview,” without changing i

→ Watch Preconditions!

Stack Example: Parsing Matching Parentheses

Example (Matching Parentheses)

```
expression = { Letter } | "(" expression ")"
```

"either a letter or an expression in brackets"

- Legal examples
  - `a ab (a) (ab) [(a)] [a(b)cd]`
- Illegal examples
  - `a) )( a( [[()] [a)a]`

Matching () Algorithm

- For each character `c` in the String
- If `c` is (
  - push `c` onto the stack
- Else if `c` is)
  - if the stack is empty: return false
  - else pop the top element and check that it is an opening bracket
- Once the string is exhausted, return true if the stack is empty