Sorting, Stacks, Queues

Bryce Boe

2013/11/06

CS24, Fall 2013

Outline

- Finish Lab 5 Part 2 Review
- Lab 6 Review / Sorting / C++ Templates
- Stacks and Queues

$O(n^2)$ Sorting Algorithms

- Bubble sort
  - Bubble the largest element to the end in each pass
- Insertion sort
  - Insert the next element into the sorted portion of the list
- Selection sort
  - Find the smallest (or largest) item and put it in its proper location

Templates

- Before templates:
  - Need a version of the data structure for each type it contains OR store (void*) pointers in the structure
- With templates:
  - Declare functions / classes that can operate on arbitrary types

STACKS

Stack

- A First-in Last-out data structure (FILO)
  - Operates like a stack of papers
- Operations
  - void push(T item)
    - Add an item to the stack
  - T pop()
    - Remove and return the most recently added item from the stack

Linked-List Implementation

- push(item)
  - Use insert_at(0, item) for a O(1)
- pop(item)
  - Use remove_at(0) for a O(1)

Array-based implementation

- push(item)
  - Use insert_at(-1, item) for an O(1) insertion
  - O(n) when the array must expand
- pop()
  - Use remove_at(-1) for an O(1) removal

QUEUES

Queues

- What is a queue?
  - A data structure that allows access to items in a first in, first out manor (FIFO)
- What are its operations?
  - **enqueue** (add to the queue)
  - **dequeue** (remove the *oldest* item in the queue)
- What are some example queues?
  - Waiting in line, task scheduling, data buffering

Linked List Implementation

- Stacks add and remove from the same side, thus for queues it makes sense to add and remove from opposite sides
- BUT, adding and removing from the end of the list is O(n)

Make the linked list smarter

- Add a *tail* pointer
  - Gives us immediate access to the end of the list
  - Can we improve these functions' efficiency?
    - insert_at(-1, item)? YES
    - remove_at(-1)? NO

Linked-List Implementation

- enqueue(item)
  - Use insert_at(-1, item) for a O(1)
    - Assumes we have a working tail pointer in the list
- dequeue(item)
  - Use remove_at(0) for a O(1)

Array-based implementation

- To implement an unbounded queue on top of the array-based implementation of a list requires treating the array as circular
- Rather than using 0 as a base index, the queue needs to keep track of which index should be the base, and use modular arithmetic to wrap around
- When the array needs to grow, the values must be manually “reset” such that the base index is at the zero position

Array-based implementation

- enqueue(item)
  - Use insert_at((BASE + size) % allocated, item) for an O(1) operation
- dequeue(item)
  - Use remove_at(BASE) for an O(1) operation and make sure to increment the BASE

Problems we can now solve

- Write a program to determine if a given text is a palindrome:
  - racecar, stats
  - **p**oor**d**an**i**s**i**n**ad**roop

- Take a few minutes to solve it with your neighbor

Palindrome Solution

```
bool is_palindrome(char *word) {
      Queue queue;
      Stack stack;
      int index = 0;
      //iterate through the word adding to the queue
      while(word[index] != ‘\0’) {
            stack.push(word[index]);
            queue.enqueue(word[index++]);
      }
      while(!queue.is_empty())
            if (stack.pop() != queue.dequeue()
                  return false;
      return true;
}
```