CSE 332: Data Structures

Spring 2016
Richard Anderson
Lecture 1

CSE 332 Team

- Instructors: Richard Anderson
  - anderson at cs
- TAs: Hunter Zahn, Andrew Li
  - hzahn93 at cs
  - lia4 at cs

Today's Outline

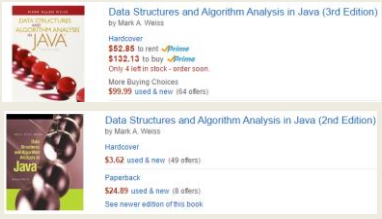
- Introductions
- Administrative Info
- What is this course about?
- Review: queues and stacks

Course Information

http://www.cs.washington.edu/332

Weiss, *Data Structures & Algorithm Analysis in Java*, 3rd Edition, 2012.

(or buy 2nd edition—1/3 price on Amazon!)



Communication

Staff
- cse332-staff@cs.washington.edu
- (or our individual addresses)

Announcements
- cse332a_sp16@u
- (you are automatically subscribed @u)

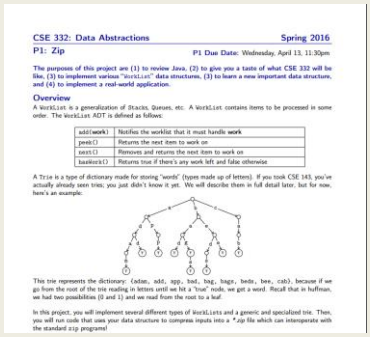
Written homeworks

Written homeworks (8 total)
- Assigned weekly
- Due at the **start of class** on due date
- No late homeworks accepted

Projects

- Programming projects (3 total, some with phases)
  - In Java
  - Eclipse encouraged
  - Turned in electronically
  - Work on individually
  - Start work early
    - You have two to three weeks on the projects
    - They are going to be very hard to get done in two to three days
  - Issue to watch out for: Java generics

Project 1 out today



CSE 332: Data Abstractions — Spring 2016

P1: Zip — P1 Due Date: Wednesday, April 13, 11:30pm

The purposes of this project are (1) to review Java, (2) to give you a taste of what CSE 332 will be like, (3) to implement various "WorkList" data structures, (3) to learn a new important data structure, and (4) to implement a real-world application.

Overview

A WorkList is a generalization of Stacks, Queues, etc. A WorkList contains items to be processed in some order. The WorkList ADT is defined as follows:

add(work)	Notifies the worklist that it must handle work
peek()	Returns the next item to work on
next()	Removes and returns the next item to work on
hasWork()	Returns true if there's any work left and false otherwise

A Trie is a type of dictionary made for storing "words" (types made up of letters). If you took CSE 143, you've actually already seen tries; you just didn't know it yet. We will describe them in full detail later, but for now, here's an example:

This trie represents the dictionary: {ado, add, app, bad, bag, bags, beds, bee, cab}. because if we go from the root of the trie reading in letters until we hit a "true" node, we get a word. Recall that in huffman, we had two possibilities (0 and 1) and we read from the root to a leaf

In this project, you will implement several different types of WorkLists and a generic and specialized trie. Then, you will run code that uses your data structure to compress inputs into a *.zip file which can interoperate with the standard zip programs!

Overall grading

Grading

20% - Written Homework Assignments

30% - Programming Assignments

20 % - Midterm Exam (Apr 29)

30% - Final Exam (June 6, 2:30-4:20 pm)

Collaboration

- HWs and Projects must be done solo
  - But you can discuss problems with others as long as you follow the Gilligan's island rule

Section

Meet on Thursdays

What happens there?

- Answer questions about current homework
- Previous homeworks returned and discussed
- Discuss the project (getting started, getting through it, answering questions)
- Finer points of Java, eclipse, etc.
- Reinforce lecture material

Homework for Today!!

Reading in Weiss

Chapter 1 – (Review) Mathematics and Java

Chapter 2 – (Next lecture) Algorithm Analysis

Chapter 3 – (Project #1) Lists, Stacks, & Queues

Today's Outline

- Introductions
- Administrative Info
- What is this course about?
- Review: Queues and stacks

13

Common tasks

- Many possible solutions
  - Choice of algorithm, data structures matters
  - What properties do we want?

14

Why should we care?

- Computers are getting faster
  - No need to optimize
- Libraries: experts have done it for you

15

Program Abstraction

Problem defn:

Algorithm:

Implementation:

16

Data Abstraction

Abstract Data Type (**ADT**):

Data Structure:

Implementation:

17

Terminology

- Abstract Data Type (ADT)
  - Mathematical description of an object with set of operations on the object. Useful building block.
- Algorithm
  - A high level, language-independent, description of a step-by-step process.
- Data structure
  - A specific organization of the data to accompany algorithms for an abstract data type.
- Implementation of data structure
  - A specific implementation in a specific language.

18

A starting problem: Prefix Sum

- Input: Array arr of size n
- Methods:
  - arr.sum(i) – find the sum of arr[0]…arr[i]
  - arr.update(i, value) – update arr[i] to value

19

Solutions

- Naïve
  - arr.sum(i): Loop through and add values
  - arr.update(i, value): arr[i] = value;
- Prefix array
  - Compute pre[i] = arr[0] + . . . + arr[i] for all i
  - arr.sum(i): return pre[i]
  - arr.update(i, value): recompute prefix array

20

Examples

- Naïve:

1	3	-4	6	3	5	-4	9

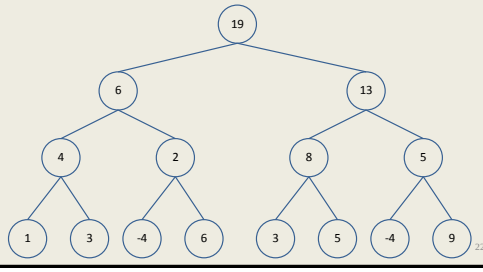
- Prefix Array:

1	3	-4	6	3	5	-4	9
1	4	0	6	9	14	10	19

21

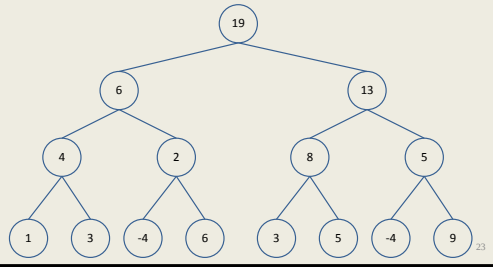
Better solution: Tree of partial sums

1	3	-4	6	3	5	-4	9



22

Sum and Update in O(log n) time



23

Today's Outline

- Introductions
- Administrative Info
- What is this course about?
- Review: queues and stacks

24

First Example: Queue ADT

- FIFO: First In First Out
- Queue operations
  - create
  - destroy
  - enqueue
  - dequeue
  - is_empty

25

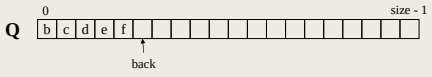
Queues in practice

- Print jobs
- File serving
- Phone calls and operators

(Later, we will consider "priority queues.")

26

Array Queue Data Structure



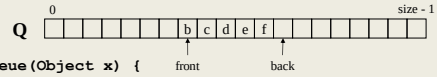

```
enqueue(Object x) {
   Q[back] = x
   back = (back + 1)
}

dequeue() {
   x = Q[0]
   shiftLeftOne()
   Back = (back – 1)
   return x
}
```

What's missing in these functions?

How to find K-th element in the queue?

27

Circular Array Queue Data Structure




```
enqueue(Object x) {
   assert(!is_full())
   Q[back] = x
   back = (back + 1)
}

dequeue() {
   assert(!is_empty())
   x = Q[front]
   front = (front + 1)
   return x
}
```

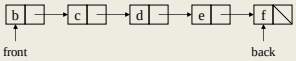
How test for empty/full list?

How to find K-th element in the queue?

What to do when full?

28

Linked List Queue Data Structure




```
void enqueue(Object x) {
  if (is_empty())
      front = back = new Node(x)
  else {
      back->next = new Node(x)
      back = back->next
  }
}
bool is_empty() {
  return front == null
}
```

```
Object dequeue() {
   assert(!is_empty())
   return_data = front->data
   temp = front
   front = front->next
   delete temp
   return return_data
}
```

29

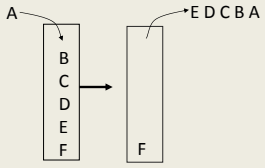
Circular Array vs. Linked List

- Advantages of circular array?
- Advantages of linked list?

30

Second Example: Stack ADT

- LIFO: Last In First Out
- Stack operations
  - create
  - destroy
  - push
  - pop
  - top
  - is_empty



31

Stacks in Practice

- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating postfix or "reverse Polish" notation

32

Assigned readings

Reading in Weiss

Chapter 1 – (Review) Mathematics and Java

Chapter 2 – (Next lecture) Algorithm Analysis

Chapter 3 – (Project #1) Lists, Stacks, & Queues

33