Data Structures in Java

Session 4

Instructor: Bert Huang

http://www1.cs.columbia.edu/~bert/courses/3134

Announcements

- Homework 1 is due on 9/22 by class time; that is in a little less than 5 days

Review

- Algorithm Analysis
- Big-Oh notation
- Maximum Subsequence example

Today's Plan

- Extra Big-Oh analysis example
- List Abstract Data Type
- Array Lists
- Linked Lists

- An algorithm takes 1 ms for size 8. How long will it take for input size 16 if the running time is

linear $\Theta(N)$

  - T(N) = c N
  - 1 ms = c 8
    - c = 1/8
  - ? ms = c 16
  - T(16) = 2 ms

- An algorithm takes 1 ms for size 8. How long will it take for input size 16 if the running time is **quadratic** $\Theta(N^2)$

  - T(N) = c N^2
  - 1 ms = c 8^2 = 64 c
    - c = 1/64
  - ? ms = c 16^2 = 256 c
  - T(16) = 4 ms

- An algorithm takes 1 ms for size 8. How long will it take for input size 16 if the running time is

  logarithmic $\Theta(\log N)$

  - T(N) = c log N
  - 1 ms = c log 8 = 3
    - c = 1/3
  - ? ms = c log 16 = 4 c
  - T(16) = 4/3 ms

Code Analysis

- ```
  sum = 0;
  for (i=0; i<n; i++)
      for (j=0; j<i*i; i++)
          for (k=0; k<j; k++)
              sum++;
  ```

$$\sum_{i=0}^{N}\sum_{j=0}^{i^2}\sum_{k=0}^{j}1$$

$$\sum_{i=0}^{N}\sum_{j=0}^{i^2}j$$

$$\sum_{i=0}^{N}\frac{i^2(i^2+1)}{2}$$

$$\frac{1}{2}\sum_{i=0}^{N}i^4+i^2$$

Getting a little too complicated...
  ```

Code Analysis

Be really pessimistic :(

In the worst case, we overestimate, but our big-Oh bound isn't wrong

- ```
  sum = 0;
  for (i=0; i<n; i++)
      for (j=0; j<i*i; i++)
          for (k=0; k<j; k++)
              sum++;
  ```

n of these

n*n of these

n*n of these

$$O(NN^2N^2) = O(NNNNN) = O(N^5)$$
  ```

Abstract Data Types

- Defined by:
  - What information it stores
  - How the information is organized
  - How the information can be accessed
- Doesn’t specify **implementation**

Vs. Implementation

- What information it stores
  - What classes/types of variables
- How the information is organized
  - How it is stored in memory
- How the information can be accessed
  - What methods (and algorithms)

Abstract Data Type: Lists

- An ordered series of objects
- Each object has a previous and next
  - Except **first** has no prev., **last** has no next
- We can insert an object (at location *k*)
- We can remove an object (at location *k*)
- We can read an object from (location *k*)

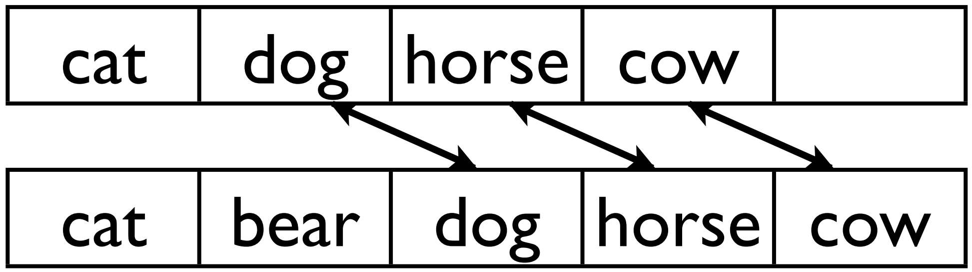
Applications for Lists

- To Do: insert tasks, remove when done
- Word Processor:
  - typing text inserts to list,
  - deleting text removes (using a simple array will leave gaps)
- Shopping: insert needed items, remove when bought, order by priority

List Methods

- Insert object (at index)
- Delete by index
- Get by index

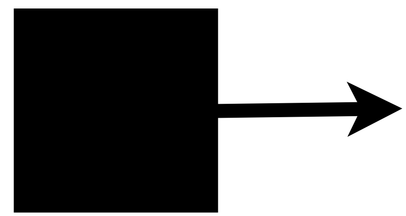
Array Implementation of Lists

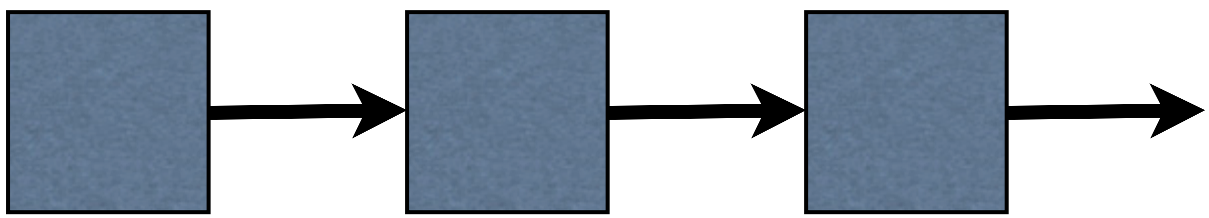


- Insert - need to shift higher-indexed elements →
- Delete - need to shift higher-indexed elements ←
- Get - easy
- How to insert more than array size?
  - Create new, larger array. Copy to new array.

Linked List Implementation

- Store elements in objects
- Each object has a reference to its next object
- Insert - rearrange references

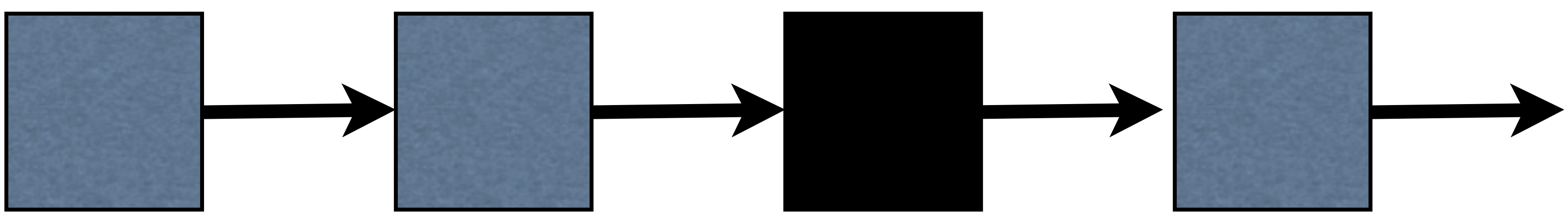




- But we need to find the previous element

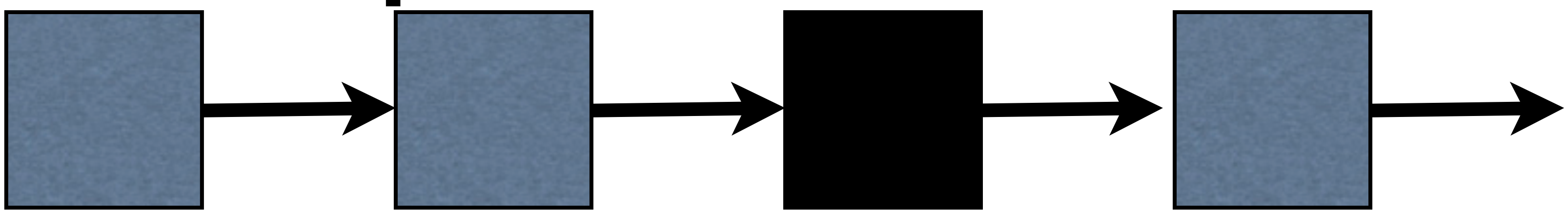
Linked List Implementation

- Store elements in objects
- Each object has a reference to its next object
- Insert - rearrange references



- But we need to find the previous element

Linked List Implementation



- Finding an element in a linked list is slower
- If we keep a **head** reference, finding the last element takes N steps
- If we keep a head and a **tail** reference*, finding the middle element takes N/2 steps
- Be careful iterating; navigate the list smartly

Linked Lists vs. Array Lists

- Linked Lists
- No additional penalty on size
- Insert/remove O(1)*
- get kth costs O(N)*
- Need some extra memory for links

- Array Lists
- Need to estimate size/grow array
- Insert/remove O(N)*
- get kth costs O(1)
- Arrays are compact in memory

Lists in Java

- **Collection** Interface extends **Iterable**
- A Collection stores a group of objects
- We can add and remove from a Collection
- **Iterator** objects let us iterate over objects in a Collection (also enhanced **for** loop)
- Built in **LinkedList** and **ArrayList** implementations of Collection

Reading

- Weiss Ch. 3
- Homework due next class!