CompSci 100
Prog Design and Analysis II

Sept 23, 2010
Prof. Rodger
Announcements

• Markov II and extra, Apt-two due today
• Apt-three due Sept 30

• Recitation this week is related to APT-three
  – Bring APT-three handout to class (descriptions of APTs)

• Exam 1 is Oct 7
From data to information to knowledge

• Data that’s organized can be processed
  – Is this a requirement?
  – What does “organized” mean?

• Purpose of map in Markov assignment?
  – Properties of keys?
  – Comparable v. Hashable

• TreeSet v. HashSet
  – Speed v. order
  – Memory considerations

• *Use versus build*
Plan

- Understand linked lists from the bottom up
  - Not as clients of java.util.LinkedList
  - Using linked lists to implement different structures
  - Using linked lists to leverage algorithmic improvements

- Self-referential structures and recursion
  - Why recursion works well with linked-structures

- Setting up the DNA-linked-list assignment
Foundations for Hash- and Tree-Set

• Typically *linked lists* used to implement hash tables
  – List of frames for film: clip and insert without shifting
  – Nodes that link to each other, not contiguous in memory
  – Self-referential, indirect references, *confusing*?

• Why use linked lists?
  – Insert and remove without shifting, add element in constant time, e.g., $O(1)$ add to back
    • Contrast to ArrayList which can double in size
  – Master pointers and indirection
  – Leads to trees and graphs: structure data into information
Linked lists as recombinant DNA

- Splice three GTGATAATTC strands into DNA
  - Use strings: length of result is $N + 3 \times 10$
  - Generalize to $N + B \times S$ (# breaks x size-of-splice)

- We can use linked lists instead
  - Use same GTGATAATTC if strands are immutable
  - Generalize to $N + S + B$, is this an improvement?
Getting in front

• Suppose we want to add a new element
  – At the back of a string or an ArrayList or a …
  – At the front of a string or an ArrayList or a …
  – Is there a difference? Why? What's complexity?

• Suppose this is an important problem: we want to grow at the front (and perhaps at the back)
  – Think editing film clips and film splicing
  – Think DNA and gene splicing

• Self-referential data structures to the rescue
  – References, reference problems, recursion, binky
Goldilocks and the Hashtable

• A hashtable is a collection of *buckets*
  – Find the right bucket and search it
  – Bucket organization?
    • Array, linked list, search tree
Structuring Data: The inside story

• How does a hashtable work? (see SimpleHash.java)
  – What happens with `put(key, value)` in a HashMap?
  – What happens with `getValue(key)`?
  – What happens with `remove(key)`?

```java
ArrayList<LinkedList<Combo>> myTable;
public void put(String key, int value) {
    int bucketIndex = getHash(key);
    ArrayList<Combo> list = myTable.get(bucketIndex);
    if (list == null) {
        list = new ArrayList<Combo>();
        myTable.set(bucketIndex, list);
    }
    list.add(new Combo(key, value));
    mySize++;
}
```

CompSci 100 Fall 2010
How do we compare times? Methods?

**Dual 2Ghz Power PC**
- *King James Bible*: 823K words
- Time to arraylist hash: 5.524
- Time to default hash: 6.137
- Time to link hash: 4.933
- Arraylist hash size = 34027
- Default hash size = 34027
- Link hash size = 34027

**Linux 2.4 Ghz, Core Duo,**
- *Wordlist*: 354K words
- Time to arraylist hash: 1.728
- Time to default hash: 1.416
- Time to link hash: 1.281
- Arraylist hash size = 354983
- Default hash size = 354983
- Link hash size = 354983

**Linux 2.4 Ghz, Core Duo,**
- *King James Bible*: 823K words
- Time to arraylist hash: 1.497
- Time to default hash: 1.128
- Time to link hash: 1.03
- Arraylist hash size = 34027
- Default hash size = 34027
- Link hash size = 34027

**OS X Laptop 2.4 Ghz, Core Duo,**
- *King James Bible*: 823K words
- Time to arraylist hash: 1.894
- Time to default hash: 1.315
- Time to link hash: 1.335
- Arraylist hash size = 34027
- Default hash size = 34027
- Link hash size = 34027
What’s the Difference Here?

• How does find-a-track work? Fast forward?
Contrast LinkedList and ArrayList

- **ISimpleList, SimpleLinkedList, SimpleArrayList**
  - Meant to illustrate concepts, not industrial-strength
  - Very similar to industrial-strength, however

- **ArrayList --- why is access O(1) or constant time?**
  - Storage in memory is contiguous, all elements same size
  - Where is the 1\(^{st}\) element? 40\(^{th}\)? 360\(^{th}\)?
  - Doesn’t matter what’s in the ArrayList, everything is a pointer or a reference (what about null?)

![Diagram of an ArrayList with arrows pointing to elements and a box representing storage in memory.](image-url)
Linked lists, CDT and ADT

- As an ADT
  - A list is empty, or contains an element and a list
    - ( ) or (x, (y, ( )) )

- As a picture

- CDT (concrete data type) pojo: plain old Java object

```java
public class Node{
    String value;
    Node next;
}

Node p = new Node();
p.value = "hello";
p.next = null;
```
What about LinkedList?

- Why is access of $N^{th}$ element linear time?
  - Keep pointer to last, does that help?
- Why is adding to front constant-time $O(1)$?
ArrayLists and linked lists as ADTs

• As an ADT (abstract data type) ArrayLists support
  – Constant-time or $O(1)$ access to the $k$-th element
  – Amortized linear or $O(n)$ storage/time with add
• Total storage used in $n$-element vector is approx. $2n$, spread over all accesses/additions (why?)
  – Adding a new value in the middle of an ArrayList is expensive, linear or $O(n)$ because shifting required
• Linked lists as ADT
  – Constant-time or $O(1)$ insertion/deletion anywhere, but...
  – Linear or $O(n)$ time to find where, sequential search
• Good for sparse structures: when data are scarce, allocate exactly as many list elements as needed, no wasted space/copying (e.g., what happens when vector grows?)
Linked list applications

- Remove element from middle of a collection, maintain order, no shifting. Add an element in the middle, no shifting
  - What’s the problem with a vector (array)?
  - Emacs visits many files, internally keeps a linked-list of buffers
  - Naively keep characters in a linked list, but in practice too much storage, need more esoteric data structures

- What’s \((3x^5 + 2x^3 + x + 5) + (2x^4 + 5x^3 + x^2 + 4x)\) ?
  - As a vector \((3, 0, 2, 0, 1, 5)\) and \((0, 2, 5, 1, 4, 0)\)
  - As a list \(((3,5), (2,3), (1,1), (5,0))\) and \_________\?
  - Most polynomial operations sequentially visit terms, don’t need random access, do need “splicing”

- What about \((3x^{100} + 5)\)?
Linked list applications continued

• If programming in C, there are no “growable-arrays”, so typically linked lists used when # elements in a collection varies, isn’t known, can’t be fixed at compile time
  – Could grow array, potentially expensive/wasteful especially if # elements is small.
  – Also need # elements in array, requires extra parameter
  – With linked list, one pointer accesses all elements

• Simulation/modeling of DNA gene-splicing
  – Given list of millions of CGTA... for DNA strand, find locations where new DNA/gene can be spliced in
    • Remove target sequence, insert new sequence
Building linked lists

• Add words to the front of a list (draw a picture)
  – Create new node with next pointing to list, reset start of list

```java
public class Node {
    String value;
    Node next;
    Node(String s, Node link) {
        value = s;
        next = link;
    }
}
// ... declarations here
Node list = null;
while (scanner.hasNext()) {
    list = new Node(scanner.next(), list);
}

• What about adding to the end of the list?
Dissection of add-to-front

- List initially empty
- First node has first word

Each new word causes new node to be created
  - New node added to front

Rhs of operator = completely evaluated before assignment

```
list = new Node(word, list);
Node(String s, Node link)
{ info = s; next = link; }
```
Standard list processing (iterative)

• Visit all nodes once, e.g., count them or *process* them

```java
public int size(Node list) {
    int count = 0;
    while (list != null) {
        count++;
        list = list.next;
    }
    return count;
}
```

• What changes if we generalize meaning of *process*?
  – Print nodes?
  – Append “s” to all strings in list?
Nancy Leveson: Software Safety

Founded the field

- Mathematical and engineering aspects
  - Air traffic control
  - Microsoft word

"C++ is not state-of-the-art, it's only state-of-the-practice, which in recent years has been going backwards"

- Software and steam engines once extremely dangerous?
  - [http://sunnyday.mit.edu/steam.pdf](http://sunnyday.mit.edu/steam.pdf)

- THERAC 25: Radiation machine killed many people
Building linked lists continued

• What about adding a node to the end of the list?
  – Can we search and find the end?
  – If we do this every time, what’s complexity of building an N-node list? Why?

• Alternatively, keep pointers to first and last nodes
  – If we add node to end, which pointer changes?
  – What about initially empty list: values of pointers?
    • Will lead to consideration of header node to avoid special cases in writing code

• What about keeping list in order, adding nodes by splicing into list? Issues in writing code? When do we stop searching?
Standard list processing (recursive)

• Visit all nodes once, e.g., count them
  
  ```java
  public int recsize(Node list) {
    if (list == null) return 0;
    return 1 + recsize(list.next);
  }
  ```

• Base case is almost always empty list: null pointer
  – Must return correct value, perform correct action
  – Recursive calls use this value/state to anchor recursion
  – Sometimes one node list also used, two “base” cases

• Recursive calls make progress towards base case
  – Almost always using `list.next` as argument
Recursion with pictures

- Counting recursively

```java
int recsize(Node list){
    if (list == null)
        return 0;
    return 1 + recsize(list.next);
}
```

```java
System.out.println(recsize(ptr));
```
Linked list Practice

• What is a list? Empty or not: mirrored in code

```java
public Node copy(Node list) {
    if (null == list) return null;
    Node first = new Node(list.next,null);
    first.next = copy(list.next);
    return first;
}
```

• How can we replace last three lines with one?
  – return new Node(list.next,copy(list.next));
  – When constructing a list, make sure to assign to .next field!

• What about iterative version? Issues? Advantages?
Changing a linked list recursively

• Pass list to method, return altered list, assign to list
  – Idiom for changing value parameters

```java
list = change(list, "apple");
public Node change(Node list, String key) {
    if (list != null) {
        list.next = change(list.next, key);
        if (list.info.equals(key)) return list.next;
        else return list;
    }
    return null;
}
```

• What does this code do? How can we reason about it?
  – Empty list, one-node list, two-node list, \( n \)-node list
  – Similar to proof by induction