Announcements

- HW9 out tonight, due 4/3
- Ants extra credit due 4/3
- See Piazza for submission instructions

Data Structure Applications

The data structures we cover in 61A are used everywhere in CS

More about data structures in 61B

Example: recursive lists (also called linked lists)
- Operating systems
- Interpreters and compilers
- Anything that uses a queue

The Scheme programming language, which we will learn soon, uses recursive lists as its primary data structure

Example: Environments

Trees with Internal Node Values

Trees can have values at internal nodes as well as their leaves.

```python
class Tree(object):
    def __init__(self, entry, left=None, right=None):
        self.entry = entry
        self.left = left
        self.right = right

def fib_tree(n):
    if n == 1:
        return Tree(0)
    if n == 2:
        return Tree(1)
    left = fib_tree(n - 2)
    right = fib_tree(n - 1)
    return Tree(left.entry + right.entry, left, right)
```

Implementing Sets

What we should be able to do with a set:
- Membership testing: Is a value an element of a set?
- Union: Return a set with all elements in set1 or set2
- Intersection: Return a set with any elements in set1 and set2
- Adjunction: Return a set with all elements in s and a value v

Union

Intersection

Adjunction
Sets as Unordered Sequences

**Proposal 1:** A set is represented by a recursive list that contains no duplicate items. This is how we implemented dictionaries.

```python
def empty(s):
    return s.is_Rlist.empty
def set_contains(s, v):
    if empty(s):
        return False
    elif s.first == v:
        return True
    return set_contains(s.rest, v)
```

**Set Intersection Using Ordered Sequences**

This algorithm assumes that elements are in order.

```python
def intersect_set2(set1, set2):
    if empty(set1) or empty(set2):
        return Rlist.empty
    e1, e2 = set1.first, set2.first
    if e1 == e2:
        rest = intersect_set2(set1.rest, set2.rest)
        return Rlist(e1, rest)
    elif e1 < e2:
        return intersect_set2(set1.rest, set2)
    elif e2 < e1:
        return intersect_set2(set1, set2.rest)
```

Sets as Ordered Sequences

**Proposal 2:** A set is represented by a recursive list with unique elements ordered from least to greatest.

```python
def set_contains2(s, v):
    if empty(s) or s.first > v:
        return False
    elif s.first == v:
        return True
    return set_contains2(s.rest, v)
```

Tree Sets

**Proposal 3:** A set is represented as a Tree. Each entry is:
- Larger than all entries in its left branch and
- Smaller than all entries in its right branch.

```python
def set_contains3(s, v):
    if s is None:
        return False
    elif s.entry == v:
        return True
    elif s.entry < v:
        return set_contains3(s.right, v)
    elif s.entry > v:
        return set_contains3(s.left, v)
```

Sets as Unordered Sequences

**Time order of growth**

- \(\Theta(n)\)
- \(\Theta(n^2)\)
- \(\Theta(n^2)\)
- \(\Theta(n^2)\)
- \(\Theta(n)\)
- \(\Theta(n)\)

**Set membership in Tree Sets**

Set membership tests traverse the tree:
- The element is either in the left or right sub-branch
- By focusing on one branch, we reduce the set by about half.

```python
def set_members3(s, v):
    if s is None:
        return False
    elif s.entry == v:
        return True
    elif s.entry < v:
        return set_members3(s.right, v)
    elif s.entry > v:
        return set_members3(s.left, v)
```

**Order of growth?**

- \(\Theta(n)\)
- \(\Theta(n)\)
Adjoining to a Tree Set

```plaintext
8
3 9
1 7 11
```

Sets as ordered sequences:
- Adjoining an element to a set
- Union of two sets

Sets as binary trees:
- Intersection of two sets
- Union of two sets

That's homework 9!

What Did I Leave Out?

Sets as ordered sequences:
- Adjoining an element to a set
- Union of two sets

Sets as binary trees:
- Intersection of two sets
- Union of two sets

That's homework 9!

Social Implications / Programming Practices

- Why things go wrong
- What can we do about this

Therac-25 Case Study

- Medical imaging device

![Therac-25 Case Study Diagram](image)

Figure 9 Typical Therac-25 Facility

Lesson to be learned

- Social responsibility in engineering
- First real incident of fatal software failure
- Bigger issue
  - No bad guys
  - Honestly believed there was nothing wrong

Therac-25 Case Study

- What happened?
- 6 serious injuries
- 4 deaths
- Otherwise effective – saved hundreds of lives

```
8
5
3 9
1 7 11
```

Right! Left! Right! Stop!
“Software Rot”

☐ Other engineering fields: clear sense of degradation and decay
☐ Can software become brittle or fractured?

A bigger picture

☐ All software is part of a bigger system
☐ Software degrades because:
  ☐ Other piece of software changes
  ☐ Hardware changes
  ☐ Environment changes

Ex: Compatibility Issues

A bigger issue

☐ The makers of the Therac did not fully understand the complexity of their software
☐ Complexity of constructs in other fields more apparent

A “simple” program

☐ This program can delete any file you can

Complexity in the Therac-25

☐ Abundant user interface issues
☐ Cursor position and field entry
☐ Default values
☐ Too many error messages
Too many error messages

![Image]

How can we solve these things?

- Know your user
- Fail-Soft (or Fail-Safe)
- Audit Trail
- Correctness from the start
- Redundancy

(More) Complexity in the Therac-25

- No atomic test-and-set
- No hardware interlocks

Correctness from the start

- Edsger Dijkstra: “On the Cruelty of Really Teaching Computing Sciences”
- CS students shouldn’t use computers
- Rigorously prove correctness of their programs
- Correctness proofs
- Compilation (pre-execution) analysis

Fail-Soft (or Fail-Safe)

```python
def mutable_rlist():
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'first':
            return first(contents)
        if message == 'rest':
            return rest(contents)
        if message == 'len':
            return len_rlist(contents)
        ...
        else:
            print('Unknown message')
        return dispatch
```
On debugging

- Black box debugging
- Glass box debugging
- Don't break what works
- Golden rule of debugging...

Golden rule of debugging

- “Debug by subtraction, not by addition”
  - Prof. Brian Harvey