CS180 - Trees

Announcements

- HW3 is due in 1 week
- No class Monday
Last time: Sorting - (Ch. 10)

- **Insertion sort**: - Simple & easy to code
  - Useful for smaller lists
  - $\mathcal{O}(n^2)$
  - $\mathcal{O}(n + k)$

- **Merge Sort**: - Fastest worst case: $\mathcal{O}(n \log n)$
  - Difficult to run in-place

- **Quick sort**: - Worst case $\mathcal{O}(n^2)$
  - But $\mathcal{O}(n \log n)$ in practice

- **Bubble sort**: - Slow - $\mathcal{O}(n^2)$
  - Again, OK for small problems (but less useful than insertion sort)
Bucket Sort $O(n+N)$

Suppose we have $n$ numbers all between $[0, N-1]$.

Turn things around: use $0...N-1$ as keys.

Put element with key $i$ in spot $A[i]$

Radix Sort

Suppose we have n ordered pairs all numbers 'b/t 0 ≤ N - 1

(1,1), (1,3), (1,5)

1 2 3 4 5 ... 11

(1,1) (1,3) (1,5) (2,5) (3,5) (5,1)
\((1,1) \ (3,1) \ (1,3) \ (1,5) \ (2,5) \ (3,5) \ (5,1)\)

Bucket sort again: (on 1st coord)

\[
\begin{align*}
\frac{1}{(1,1)} & \quad \frac{2}{(2,5)} & \quad \frac{3}{(3,1)} & \quad \frac{4}{(3,5)} & \quad \frac{5}{(5,1)}
\end{align*}
\]
Ch 6 - Trees

All data structures so far have expressed linear orderings:

\[ \text{lists, vectors, stacks, queues} \]

Some structures require more complex relations.
Examples:

- File systems:
  
  - WWW
  - CS140
  - CS14S
  - CS150
  - CS180
  
  - Fall 08
  - Spring 09
  - Fall 09
  
  - CS14S
  - CS190
  
  - Programs
  - Labs
Ex:

Family tree (Patrachice)

Abraham

Isaac

Jacob  Esau

Ishmael
Definitions

A tree is a set of nodes storing elements in a parent-child relationship.

- It has a special node, \( \text{r} \), called the root.
- Each node (except \( \text{r} \)) has a unique parent.
More dfs

- child
- siblings - share a common parent
- leaves - have no children
- internal nodes - have at least 1 child
- rooted subtree
- ancestor - parent of a parent
- descendant - children of child
Tree Data Structure

What sort of data might a tree class need?

Tree class will need a root.

```
Node* _root;
int _size;
```

**Node struct:**

```
    Node* _parent;
    Child pointers < 2 pointers
    Object _data;
    int _aux;
    _left + _right
    depth or height
```
Iterators in Trees

up

down left

down right

private data:

Node* location;

BinaryTree* mytree;
Code for trees

We'll come back to this after fall break.

Our first implementation will be of a special kind of tree, since we can avoid pointers in some cases.
Binary Trees

Here, every node has 0 or 2 children.
Array Based implementation:

Root is #6

For any node v with number n, left child gets number 2*n and right child gets 2*n+1
Each array will have size
and max capacity.
You have to double array if a
new level is added to free.
Depth of a tree

**depth**: defined recursively:
- root has depth: 0
- every other node:
  \[ \text{depth}(v) = \text{depth}(\text{parent}(v)) + 1 \]

(Easy to give recursive algorithm.)

\[ O(\text{depth of tree}) \]
Height of a tree

Height of a leaf = 0

Height(v) = max(Height of children) + 1

leads to recursive algo.

How long?

$O(\text{size of subtree rooted at } v)$