CSE 373: Trees

Chapter 4

Summary of Chapter 3

Lists, Stacks, and Queues...
- are composed of elements in a sequential order
  - Lists – arbitrary order
  - Stacks – LIFO
  - Queues – FIFO
- implementations are usually array- or link-based
- operations add, remove, find, iterate over elements
- usually, searching for a specific element is $O(n)$
  - counterexample?
Trees

Trees allow the expression of non-sequential relationships

Real-life Instances of Trees

- Family trees
- Organization Charts
- Classification trees
  - what kind of flower is this?
  - what’s wrong with my car?
- File directory structure
  - folders, subfolders in Windows
  - directories, subdirectories in UNIX
- Non-recursive procedure call chains
Tree Terminology

- **root:**
- **leaf:**
- **child:**
- **parent:**
- **sibling:**
- **grandparent:**
- **grandchild:**
- **ancestor:**
- **descendent:**

More Tree Terminology

- **path:**
- **depth:**
- **height:**
- **degree:**
Implementation of Trees

- Trees can’t be implemented with lists (easily)
- Why not?

Naive Tree Implementation

If we can bound the degree of a tree’s nodes, it has a simple implementation:

```c
struct TreeNode {
    Object data;
    TreeNode *child[MAX_DEGREE];
};
```
**General Tree Implementation**

Since a tree can have any number of children…
- Parent links to first child
- Siblings link to one another

```c
struct TreeNode {
    Object data;
    TreeNode *firstchild;
    TreeNode *sibling;
};
```

**Design Decision: Parent Pointer**

- For most operations, only pointers to children are needed
- Some implementations may also store a pointer to a node’s parent:

```c
struct TreeNode {
    Object data;
    TreeNode *parent;
    TreeNode *firstchild;
    TreeNode *sibling;
};
```
Tree Operations

- Like List, not a well-defined ADT...
- Possible Operations
  - Tree operations:
    TreeNode *root();
    TreeNode *find(Object);
  - Node operations:
    void addChild(Object);
    int numChildren();
    TreeNode *getKthChild(int);
    void deleteKthChild(int);
  - Also traversal operations...

Well-defined Traversals

**pre-order:**
1) process node
2) process children

**post-order:**
1) process children
2) process node
Traversal Applications

- Print Directory Listing
  - www/
    - index.html
    - handouts/assignments/
    - ch1.ps
    - hw1.html
    - hw2.html

- Print Disk Usage

Tree Applications

- Storing data for the “real life instances of trees”
- **CAD/drawing:** Storing hierarchies of objects
  (a wheel is made of a tire and spokes; a car is made...)
- **graphics:** Storing a scene’s geometry/structure
- **languages:** Storing a class hierarchy (*e.g.*, C++)
Application: Storing Expressions

\[ a[i] = (b + c[i,j,k]) \times \text{pow}(3,n); \]

```
array_ref
  
  *
    
    function_call
      
      pow
        3
        n
```

Decision Trees:

```
"if I move"

  here...

  "then s/he might move"

    here...

    there...

    "then s/he might move"

      here...

      there...

      "then s/he might move"
```

UW, Autumn 1999  CSE 373 – Data Structures and Algorithms  Brad Chamberlain