CSE 373: Trees

Chapter 4

Summary of Last Week

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 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
- Procedure call chains
Tree Terminology

root:
leaf:
child:
parent:
sibling:
grandparent:
grandchild:
ancestor:
descendant:

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
typedef struct _TreeNode {
    NodeType data;
    struct _TreeNode *child[MAX_DEGREE];
} TreeNode;
```
General Tree Implementation

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

```c
typedef struct _TreeNode {
    NodeType data;
    struct _TreeNode *firstchild;
    struct _TreeNode *sibling;
} TreeNode;
```

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
typedef struct _TreeNode {
    NodeType data;
    struct _TreeNode *parent;
    struct _TreeNode *firstchild;
    struct _TreeNode *sibling;
} TreeNode;
```
Tree Operations

- Not a well-defined ADT...
- Possible Operations
  - Tree operations:
    ```c
    TreeNode *Root(Tree);
    TreeNode *Find(Tree, NodeType);
    ```
  - Node operations:
    ```c
    void AddChild(TreeNode *, NodeType);
    int NumChildren(TreeNode *);
    TreeNode *GetKthChild(TreeNode *);
    void DeleteKthChild(TreeNode *, int);
    ```
  - Also traversal operations...

Well-defined Traversals

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

**post-order:**
1) process children
2) process node
Traversing 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 applications”
- **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

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

Decision Trees:

"if I move"

here... there...

"then s/he might move"

here... there...

"then s/he might move"

here... there...

"then s/he might move"

here... there...