Binary Trees

Trees

- nodes=objects
  - data section
  - linkage info
- parent
- children
  - binary= max 2
  - left/right
- tree root
  - like listhead
SubTree

Fundamental property
- left subtree values <= value
- right subtree values >= node value

Binary Search Tree
Binary Search: look for value

- look for 'P'

Tree Balance

GOOD

BAD
Tree Operations: create

- similar to lists, but different linkage

```cpp
class treenode{
    int value;
    treenode* parent, lchild, rchild;
};
```

```cpp
treenode* root = new treenode;
root->parent=NULL;
root->lchild = root->rchild=NULL;
root->value = somevalue;
```

---

Tree Operations: insert

- create new node
- associate value
- insert the node into the tree based on value
  - fundamental property must be preserved
- insert value 7 in example
Tree Operations: delete

- delete node content: easy
- but linkage has to be handled
  - not so easy
- use successor() / predecessor()
  - to determine what nodes replace the deleted one
  - and perhaps continue replacements

Predecessor, Successor

- Predecessor(x) = highest value in the tree smaller or equal to x (but not the same node as x)
- Successor(x) = smallest value in the tree bigger or equal to x (but not the same node as x)
Min, Max

- Min = go deep on the left branch
- Max = go deep on the right branch

Searching the tree

- Binary search Trees very good for searching large amounts of data
- Search for value x
- start at root, repeat
  - compare $x$ with value
  - if found, return
  - if $x>$ value go on the right branch
  - if $x<$ value go on the left branch
Traversing: inorder

- recursion order: leftchild, node, rightchild
- void TraverseInorder (treenode* node) {
  • if (node==NULL) return;
  • TraverseInorder (node->lchild);
  • cout<<" "<<node->value;//process node
  • TraverseInorder (node->rchild);
  ...
}

Traversing: preorder

- recursion order: node, leftchild, rightchild
- same as DFS
- void TraversePreOrder (treenode* node) {
  • if (node==NULL) return;
  • cout<<" "<<node->value;//process node
  • TraversePreOrder (node->lchild);
  • TraversePreOrder (node->rchild);
  ...
}
Traversing : postorder

○ recursion: leftchild, rightchild, node

```cpp
void TraversePostorder (treenode* node) {
    if (node==NULL) {cout<< "NULL."; return;}
    cout<<"\ngoing left ..."); TraversePostorder (node->lchild);
    cout<<"\ngoing right ..."); TraversePostorder (node->rchild);
    cout" "<<"address="<<node" value="<<node->value;
}
```

Traversing Tree : BFS

○ "Breadth First Search"

○ nonrecursive : needs a queue

○ level by level in the tree (also called "waves"):    ○ first the root
    ○ then all root's children
    ○ then all the nodes 2-edges away from the root
    ○ all nodes 3-edges away from the root
    ○ etc.
Traversing Tree: DFS

- "Depth First Search"
- recursion order: node, leftchild, rightchild
- same as Pre-order