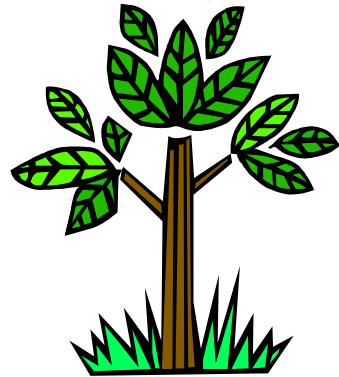


Lecture P10: Trees



Overview

Culmination of the programming portion of this class.

- Solve a database search problem.

Tree data structure.

- Useful.
- Versatile.
- Naturally recursive.

Searching a Database

Database entries.

- Names and social security numbers.

Desired operations.

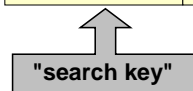
- Insert student.
- Delete student.
- Search for name given ID number.

Goal.

- All operations fast, even for huge databases.

Data structure that supports these operations is called a **SYMBOL TABLE**.

SS #	Last
192042006	Arac
201211991	Baron
177999898	Bergbreiter
232871212	Buchen
122993434	Durrett
162882273	Gratzer



Other Symbol Table Applications

Other applications.

- Online phone book looks up names and telephone numbers.
- Spell checker looks up words in dictionary.
- **Internet domain server looks up IP addresses.**
- Compiler looks up variable names to find type and memory address.

Web Site	IP Address
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55
www.amazon.com	208.216.181.15
www.simpsons.com	209.052.165.60

Representing the Database Entries

Define `Item.h` file to encapsulate generic database entry.

- Insert and search code should work for any item type.
 - ideally `Item` would be an ADT
- Key is field in search.

```

ITEM.h
typedef int Key;
typedef struct {
    Key ID;
    char name[30];
} Item;

Item NULLitem = {-1, ""};

int eq(Key k1, Key k2);
int less(Key k1, Key k2);
Key key(Item item);
void show(Item item);
    
```

```

item.c
#include "ITEM.h"

int eq(Key k1, Key k2) {
    return k1 == k2;
}

int less(Key k1, Key k2) {
    return k1 < k2;
}

Key key(Item item) {
    return item.ID;
}

void show(Item item) {
    printf("%d %s\n", item.ID,
           item.name);
}
    
```

5

Symbol Table ADT

Define `ST.h` file to specify database operations.

- Make it a true symbol table ADT.

```

ST.h (Sedgewick 12.1)
Item STsearch(Key k); // search for Key in database
void STinsert(Item item); // insert new Item into database
void STshow(void); // print all Items in database
int STcount(void); // number items in database
void STdelete(Item item); // delete Item from database
    
```

6

Unsorted Array Representation of Database

Maintain array of Items.

- Use SEQUENTIAL SEARCH to find database Item.

```

STunsortedarray.c
#define MAXSIZE 10000
static Item st[MAXSIZE];
static int size = 0;

Item STinsert(Item item) {
    st[size] = item;
    size++;
}

Item STsearch(Key k) {
    int i;
    for (i = 0; i < size; i++)
        if (eq(k, key(st[i])))
            return st[i];
    return NULLitem;
}
    
```

elements →

← Array of database Items.

← Key k found.

← Key k not found.

7

Unsorted Array Representation of Database

Maintain array of Items.

- Use SEQUENTIAL SEARCH to find database Item.

Advantage.



Key drawback.



Extra problem.



8

Sorted Array Representation of Database

Maintain array of Items.

- Store in sorted order (by Key).
- Use BINARY SEARCH to find database Item.



Array of
database Items.

Key k not found.

Key k found.

Divide-and-
conquer.

```

STsortedarray.c (Sedgewick 12.6)
#define MAXSIZE 10000
static Item st[MAXSIZE];
static int size = 0;

Item search(int l, int r, Key k) {
    int m = (l + r) / 2;
    if (l > r)
        return NULLitem;
    else if eq(k, key(st[m]))
        return st[m];
    else if less(k, key(st[m]))
        return search(l, m-1, k);
    else
        return search(m+1, r, k);
}
    
```

9

Sorted Array Representation of Database

Maintain array of Items.

- Store in sorted order (by Key).
- Use BINARY SEARCH to find database Item.

"Wrapper" for
search function.

```

STsortedarray.c (Sedgewick 12.6)
Item STsearch(Key k) {
    return search(0, size-1, k);
}
    
```

10

Cost of Binary Search

How many "comparisons" to find a name in database of size N?

- Divide list in half each time.
5000 ⇒ 2500 ⇒ 1250 ⇒ 625 ⇒ 312 ⇒ 156 ⇒ 78 ⇒ 39 ⇒ 18 ⇒ 9 ⇒ 4 ⇒ 2 ⇒ 1
- $\lceil \log_2(N+1) \rceil$ = number of digits in binary representation of N.
- $5000_{10} = 1001110001000_2$

The log functions grows very slowly.

- \log_2 (thousand) ≈ 10
- \log_2 (million) ≈ 20
- \log_2 (billion) ≈ 30

$$2^x = N$$

$$x = \log_2 N$$

Without binary search (or if unsorted): may examine all N items.

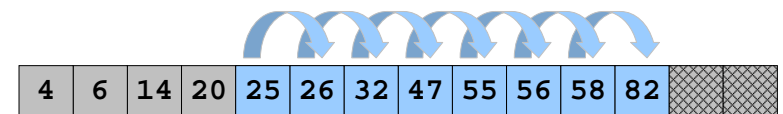
- N vs. $\log_2 N$ savings is staggering for large files.
- Milliseconds vs. hours (or more!).

11

Insert Using Sorted Array Representation

Key Problem: insertion is slow.

- Want to keep entries in sorted order.
- Have to move larger keys over one position to right.



Demo: inserting 25 into a sorted array.

12

Sorted Array Representation of Database

Maintain array of Items.

- Store in sorted order (by *key*).
- Use BINARY SEARCH to find database *Item*.

Advantage.



Key drawback.



Extra problem.

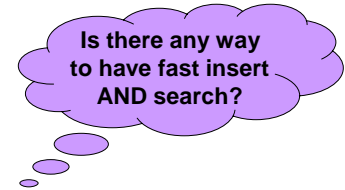


13

Summary

Database entries.

- Names and social security numbers.



Desired operations.

- Insert, delete, search.

Goal.

- Make all of these operations FAST even for huge databases.

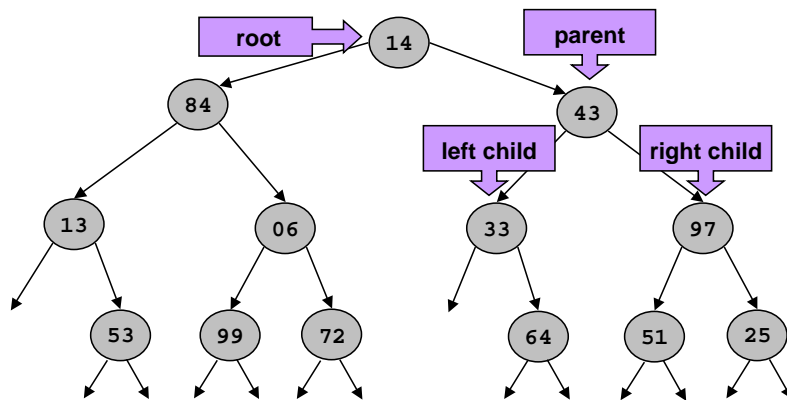
	asymptotic time			computer time		
	search	insert	delete	search	insert	delete
sorted array	log N	N	N	instant	2 hour	2 hour
unsorted array	N	1	1 *	2 hour	instant	instant
goal	log N	log N	log N	instant	instant	instant

* assumes we know location of node to be deleted

14

Binary Tree

Yes. Use TWO links per node!

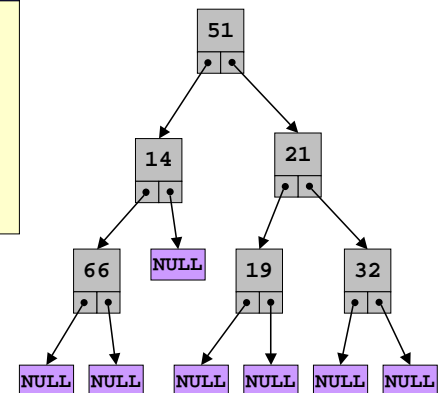


15

Binary Tree in C

```

STbst.c
typedef struct STnode* link;
struct STnode {
    Item item;
    link left;
    link right;
};
static link root;
    
```



Represent with TWO links per node.

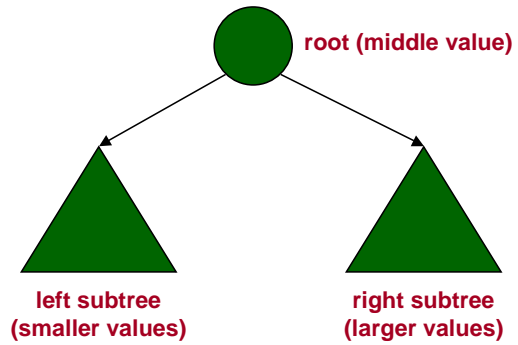
- Left arrow for first link.
- Right arrow for second link.

16

Binary Search Tree

Binary tree in "sorted" order.

- Maintain ordering property for tree and ALL sub-trees.

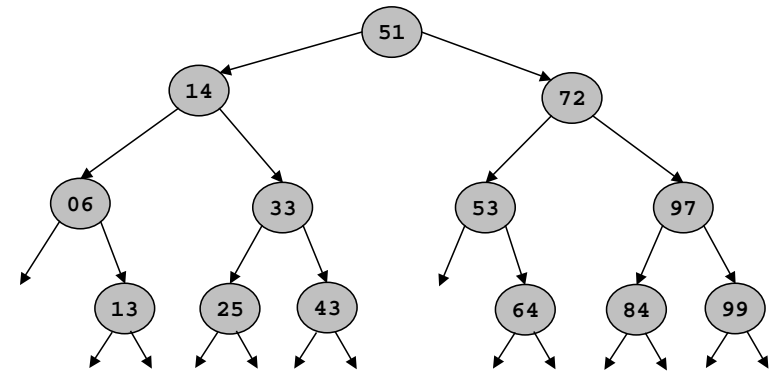


17

Binary Search Tree

Binary tree in "sorted" order.

- Maintain ordering property for tree and ALL sub-trees.

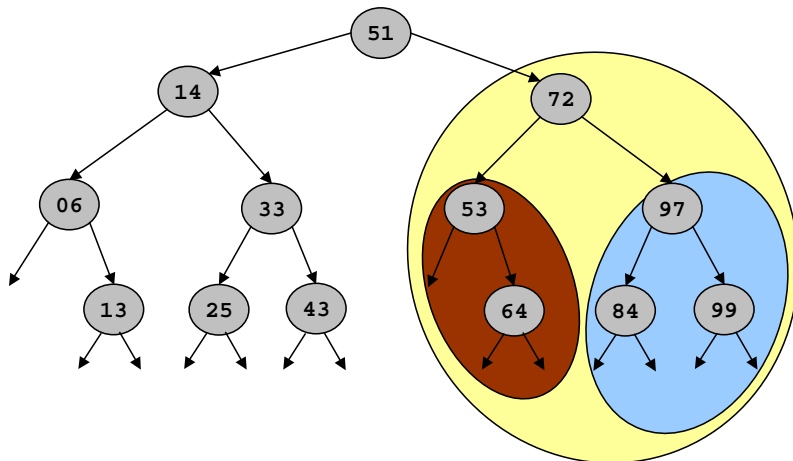


18

Binary Search Tree

Binary tree in "sorted" order.

- Maintain ordering property for tree and ALL sub-trees.

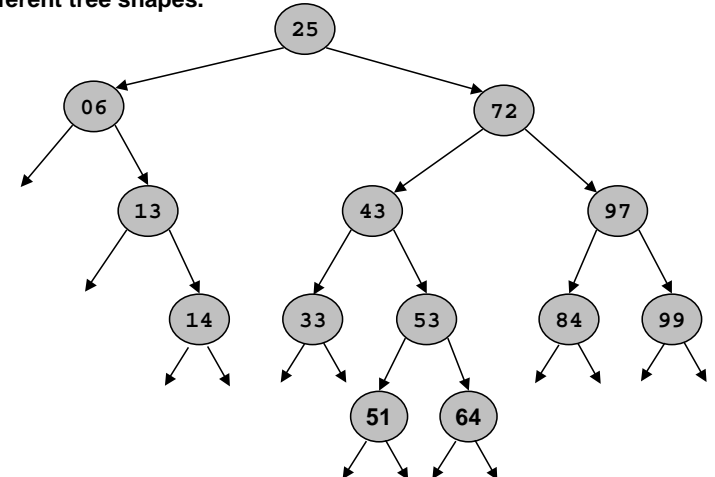


20

Binary Search Tree

Binary tree in "sorted" order.

- Many BSTs correspond to same input data.
- Have different tree shapes.



21

Search in Binary Search Tree

Search for key k in binary search tree.



- Analogous to binary search in sorted array.

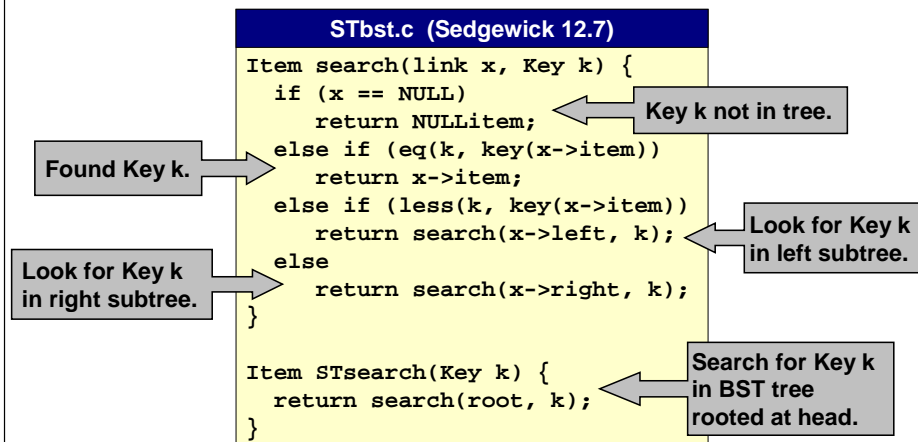
Search algorithm:

- Start at root node.
- If key of current node is k , return node.
- Go LEFT if current node has Key $< k$.
- Go RIGHT if current node has Key $> k$.

22

Search in BST's

Search for Key k .



23

Cost of BST Search

Cost of BST search depends on tree shape.

- Proportional to length of path from root to Key.
- If "balanced"
 - $\log_2 N$ keys are compared
 - proportional to binary search cost
- But if "unbalanced"
 - takes N comparisons for degenerate tree shapes
 - can be as slow as sequential search

Tree shape depends on insertion method.

24

Insert Using BST's

How to insert new database Item.



- Search for key of database Item.
- Search ends at NULL pointer.
- New Item "belongs" here.
- Allocate memory for new Item, and link it to tree.

25

Insert Using BST's

STbst.c (Sedgewick 12.7)

```
link insert(link x, Item item) {
    if (x == NULL)
        return NEWnode(item, NULL, NULL);

    else if (less(key(item), key(x->item))
        x->left = insert(x->left, item);
    else
        x->right = insert(x->right, item);
    return x;
}

void STinsert(Item item) {
    head = insert(root, item);
}
```

Insert new node here.

Divide-and-conquer.

Wrapper function.

26

Insert Using BST's

STbst.c (Sedgewick 12.7)

```
link NEWnode(Item item, link left, link right) {
    link x = malloc(sizeof *x);
    if(x == NULL) {
        printf("Error allocating memory.\n");
        exit(EXIT_FAILURE);
    }
    x->item = item;
    x->left = left;
    x->right = right;
    return x;
}
```

Allocate memory and initialize.

27

Insertion Cost in BST

Cost of BST insertion depends on tree shape.

- Cost is proportional to length of path from root to node.

Tree shape depends on order keys are inserted.

- Insert in "random" order.
 - leads to "well-balanced" tree
 - average length of path from root to node is $1.44 \log_2 N$
- Insert in sorted or reverse-sorted order.
 - degenerates into linked list
 - takes $N - 1$ comparisons

With cleverness can ensure tree is always (sufficiently) balanced.

- See red-black trees in COS 226.

28

Summary

Database entries.

- Names and social security numbers.

Desired operations.

- Insert, delete, search.

Goal.

- Make all of these operations FAST even for huge databases.

	asymptotic time			computer time		
	search	insert	delete	search	insert	delete
sorted array	log N	N	N	instant	2 hour	2 hour
unsorted array	N	1	1 *	2 hour	instant	instant
BST	log N	log N	log N	instant	instant	instant

29

Question

Current code searches for a name given an ID number.

What if we want to search for an ID number given a name?

ITEM.h

```
typedef char Key[30];
typedef struct {
    int ID;
    Key name;
} Item;

Item NULLitem = {-1, ""};

int eq(Key, Key);
int less(Key, Key);
Key key(Item);
```

item.c

```
#include <string.h>
int eq(Key k1, Key k2) {
    return strcmp(k1, k2) == 0;
}

int less(Key k1, Key k2) {
    return strcmp(k1, k2) < 0;
}

Key key(Item item) {
    return item.name;
}
```

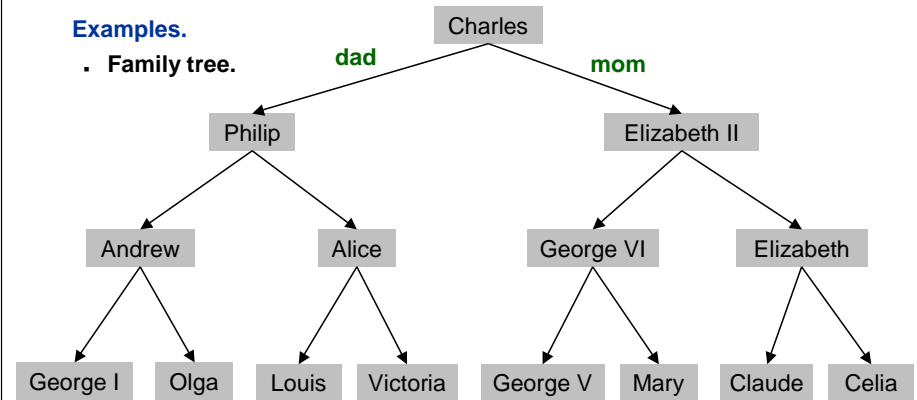
31

Other Types of Trees

Trees.

Examples.

- Family tree.



32

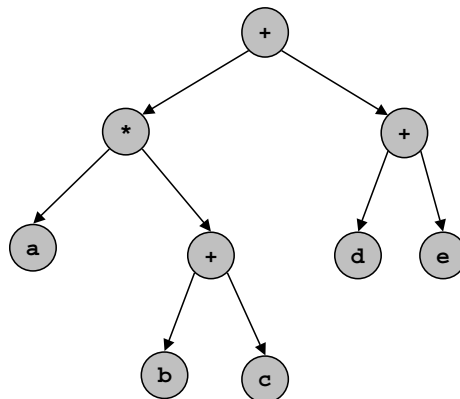
Other Types of Trees

Trees.

- Order of children may not be important.

Examples.

- Family tree.
- Parse tree.
 $(a * (b + c)) + (d + e)$



33

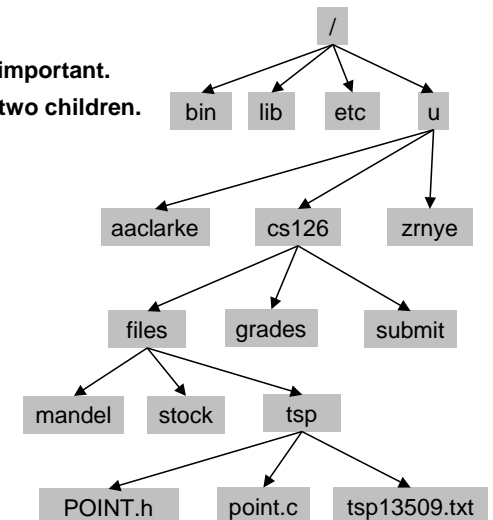
Other Types of Trees

Trees.

- Order of children may not be important.
- Nodes need not have exactly two children.

Examples.

- Family tree.
- Parse tree.
- Unix file hierarchy.



34

Summary

How to insert and search a database using:

- Unsorted array.
- Sorted array.
- Binary search tree.

Performance characteristics using different data structures.

Binary tree is fundamental data structure in computer science.

35

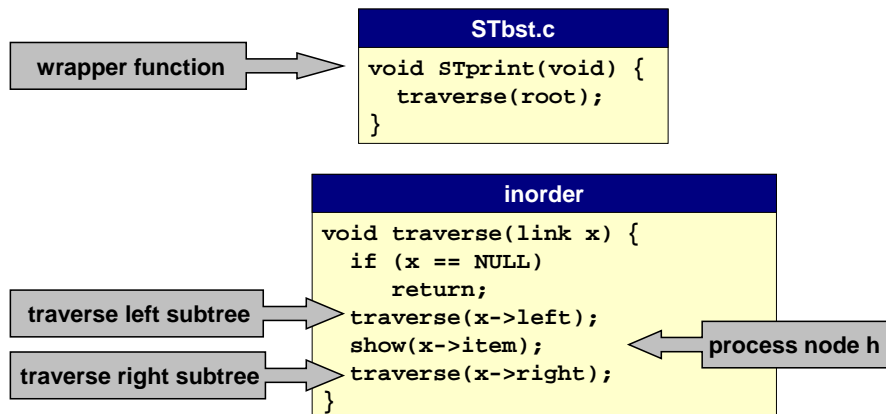
Lecture P10: Extra Notes



Traversing Binary Trees

Goal: visit (process) each node in tree in some order.

- "Tree traversal."



37

Traversing Binary Trees

Goal: visit (process) each node in tree in some order.

- "Tree traversal."
- Goal realized no matter what order nodes are visited.
 - **inorder**: visit between recursive calls

```
inorder
void traverse(link x) {
    if (x == NULL)
        return;
    traverse(x->left);
    show(x->item);
    traverse(x->right);
}
```

38

Traversing Binary Trees

Goal: visit (process) each node in tree in some order.

- "Tree traversal."
- Goal realized no matter what order nodes are visited.
 - inorder: visit between recursive calls
 - preorder: visit before recursive calls

```
preorder
void traverse(link x) {
    if (x == NULL)
        return;
    show(x->item);
    traverse(x->left);
    traverse(x->right);
}
```

39

Traversing Binary Trees

Goal: visit (process) each node in tree in some order.

- "Tree traversal."
- Goal realized no matter what order nodes are visited.
 - inorder: visit between recursive calls
 - preorder: visit before recursive calls
 - postorder: visit after recursive calls

```
postorder
void traverse(link x) {
    if (x == NULL)
        return;
    traverse(x->left);
    traverse(x->right);
    show(x->item);
}
```

40

Traversing Binary Trees

Goal: visit (process) each node in tree in some order.

- "Tree traversal."
- Goal realized no matter what order nodes are visited.
 - inorder: visit between recursive calls
 - preorder: visit before recursive calls
 - postorder: visit after recursive calls



41

Preorder Traversal With Explicit Stack

Visit the top node on the stack.

- Push its children onto stack.



Push right node
before left, so that left
node is visited first.

preorder traversal with stack

```
void traverse(link x) {
    STACKpush(x);
    while (!STACKempty()) {
        x = STACKpop();
        show(x->item);
        if (x->right != NULL)
            STACKpush(x->right);
        if (x->left != NULL)
            STACKpush(x->left);
    }
}
```

42

Level Traversal With Queue

Q. What happens if we replace stack with QUEUE?

- Level order traversal.
- Visit nodes in order from distance to root.



level traversal with queue

```
void traverse(link x) {
    QUEUEput(x);
    while (!QUEUEisempty()) {
        x = QUEUEget();
        show(x->item);
        if (x->left != NULL)
            QUEUEput(x->left);
        if(x->right != NULL)
            QUEUEput(x->right);
    }
}
```