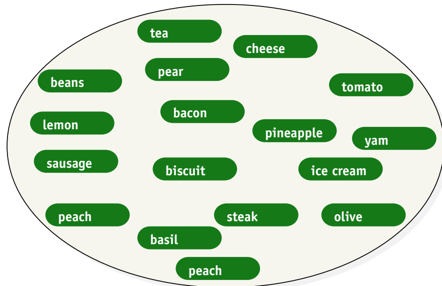
Trees and Graphs

February 1–2, 2012
CS 60: Principles of Computer Science

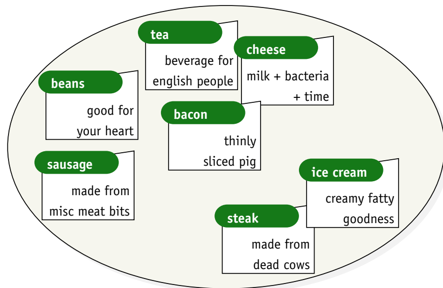
Assignment 2 due Monday: Higher-Order Functions, Trees, Graphs, and Java(!)

Two Abstract Interfaces

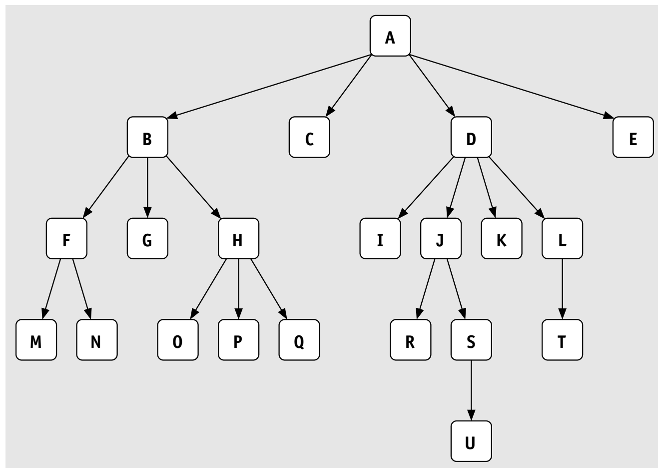
Set: An unordered collection



Map: Associates "keys" with "values"

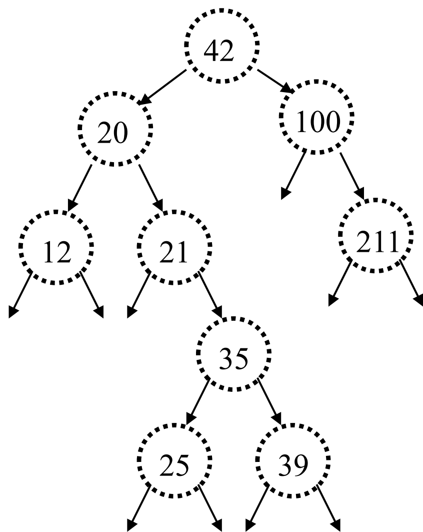


Trees in General



Nomenclature: node, root, leaves, child, parent, anscestor, descendant, ...
Applications: sets/maps, files and folders, graphics, ...

A Specific Kind of Tree: Binary Search Trees



Identifying Features:

- ✓ Every node has two (possibly empty) subtrees
- ✓ Each node has a "key"
- ✓ The root key is always greater than all nodes in a left subtree
- ✓ The root key is always less than all nodes in a right subtree

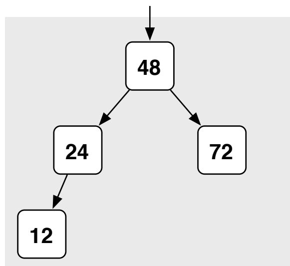
But Racket only has lists...?

Representing Trees as Lists

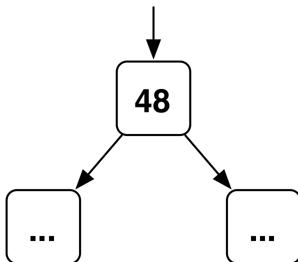
- Empty tree:
  the empty list `'()`

- Nonempty tree:
  a three-element list `(ROOT-KEY LEFT-TREE RIGHT-TREE)`

What Racket list represents this tree?



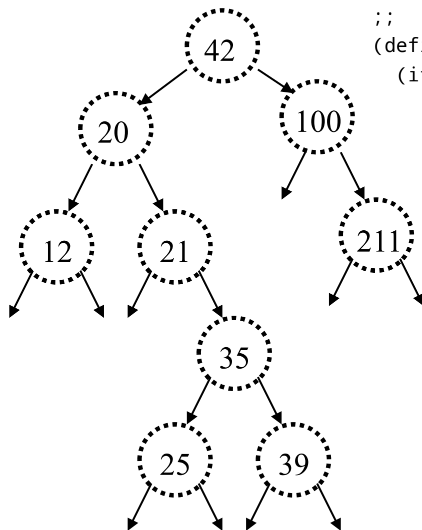
Searching in BSTs



We have this tree. Where should I look for these values (and why?)

- ✓ 42
- ✓ 60
- ✓ 48

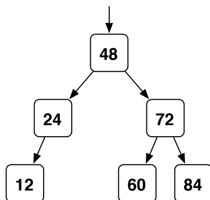
Searching in BSTs




```
;; (find item BST) ===>
;;    #t if item in BST,
;;    #f otherwise
(define (find item BST)
  (if (null? BST)

      (let* ( [ root (first BST) ]
              [ LEFT (second BST) ]
              [ RIGHT (third BST) ])
            (
```

Inserting a new value in a BST

Consider the tree



Let **LEFT** be the left subtree of the root

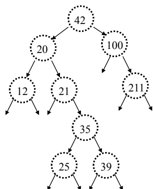
1. If we take **just** **LEFT** and insert **36**, what slightly-larger tree should we expect to produce? Draw it!
2. If we take ***the entire tree*** and insert **36**, what slightly-larger tree should we expect to produce? Draw it!
3. How are these two trees related?

NAME:

"QUIZ" TWO FUNCTIONS FOR BSTS

```
;; Given a BST, count the number of nodes
(define (size BST)
```

```
;; Given a BST and a new item *not in BST*,
;;   create a new BST containing everything
;;   in the given BST plus the new item.
(define (insert item BST)
  (if (null? BST)
      (                                  )
      (let* ( [ root (first BST) ]
              [ LEFT (second BST) ]
              [ RIGHT (third BST) ] )
            (
```



Trust in recursion...

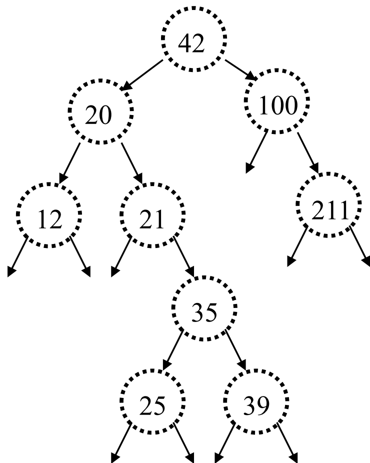
size

```
(define (size BST)
   (if (null? BST)
       0
       (let* ( [ LEFT  (second BST) ]
               [ RIGHT (third BST)  ] )
             (+ 1 (size LEFT) (size RIGHT)))))
```

insert

```
(define (insert item BST)
   (if (null? BST)
       (list item '() '())
       (let* ( [ root  (first BST)  ]
               [ LEFT  (second BST) ]
               [ RIGHT (third BST)  ] )
             (if (< item root)
                 (list root (insert item LEFT) RIGHT)
                 (list root LEFT (insert item RIGHT))))))
```

DELETING!



To delete an item N (producing a smaller tree)

- ✓ If the tree is empty?
- ✓ If the root is less than N?
- ✓ If the root is bigger than N?
- ✓ If the root is N?
  - If $N = 25$?
  - If $N = 21$?
  - If $N = 42$?

It was the best of cases, it was the worst of cases...

Suppose our tree has n nodes.

In the *best case*,

$O(\log n)$ { `find`, `insert`, `delete` }

In the *worst case*,

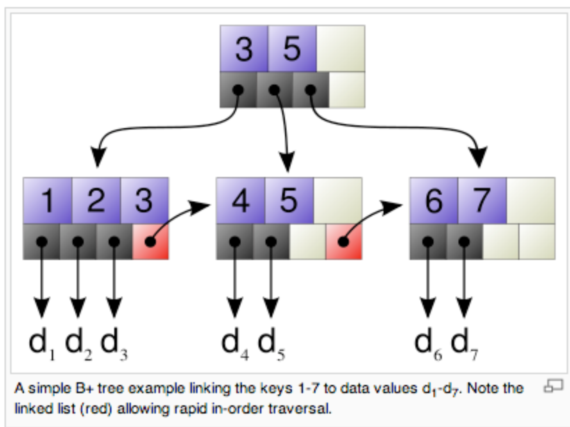
$O(n)$ { `find`, `insert`, `delete` }

Trees can be fast!

But what if the data conspires against you (*e.g.*, a sorted list)?

- ✓ Self-balancing binary trees (*covered in CS 70*)
- ✓ E.g., `http://www.cs.hmc.edu/~stone/Applets/avltree.html`

B-Trees (actually B+ Trees)



A simple B+ tree example linking the keys 1-7 to data values d_1-d_7. Note the linked list (red) allowing rapid in-order traversal.

wikipedia, B+ trees

The ReiserFS filesystem (for Unix and Linux), XFS filesystem (for IRIX and Linux), JFS2 filesystem (for AIX, OS/2 and Linux), and NTFS all use this type of tree for block indexing. Relational databases such as PostgreSQL and MySQL also often use this type of tree for table indices.

File systems and databases tend to use trees with higher branching factors. (Why?)

The Mac's File System

CHAPTER 12

The HFS Plus File System

The HFS Plus file system (or simply HFS+) is the preferred and default volume format on Mac OS X. The term *HFS* stands for *Hierarchical File System*, which replaced the flat Macintosh File System (MFS) used in early Macintosh operating systems. HFS remained the primary volume format for Macintosh systems before Mac OS 8.1, which was the first Apple operating system to support HFS+. Also called the *Mac OS X Extended* volume format, HFS+ is architecturally similar to HFS but provides several important benefits over the latter.[1] Moreover, HFS+ itself has evolved greatly since its inception—not so much in fundamental architecture but in its implementation. In this chapter, we will discuss features and implementation details of HFS+ in Mac OS X.

1. Two of the major limitations in HFS were that it was largely single threaded and that it supported only 16-bit allocation blocks.

Looking Back

Apple filed a patent for the Macintosh Hierarchical File System (U.S. Patent Number 4,945,475) in late 1989. The patent was granted in mid-1990. The original HFS was implemented using two B-Tree data structures: the Catalog B-Tree and the Extents B-Tree. As we will see in this chapter, HFS+ uses both of these B-Trees. Lisa OS—the operating system for Apple's Lisa computer (1983)—used a hierarchical file system before the Macintosh. Indeed, the HFS volume format benefited from work done on the Lisa's file system.

As we noted in Chapter 11, one hallmark of HFS was that it lent support to the graphical user interface by providing a separate data stream in a file—the *resource fork*—for storing application icons, resources, and other auxiliary data independently of the file's "main" data.

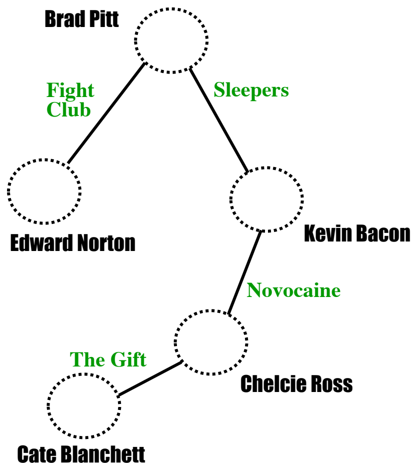
Noteworthy features of HFS+ include the following:

- Support for files up to 2^{63} bytes in size
- Unicode-based file/directory name encoding, with support for names containing up to 255 16-bit Unicode characters[2]
- A B+ Tree (the *Catalog B-Tree*) for storing the file system's hierarchical structure, allowing tree-based indexing
- Extent-based allocation of storage space using 32-bit allocation block numbers, with delayed allocation of physical blocks
- A B+ Tree (the *Extents Overflow B-Tree*) for recording files' "overflow" extents (the ninth and subsequent—for files with more than eight extents)
- Multiple byte-streams (or forks) per file, with two predefined forks and an arbitrary number of other, named forks that are stored in a separate B-Tree (see next item).
- A B+ Tree (the *Attributes B-Tree*) for storing arbitrary metadata[3] per file, thus providing native support for extended file system attributes (the names of which are Unicode strings up to 128 16-bit Unicode characters in length)
- Metadata journaling through the kernel's VFS-level journaling mechanism
- Multiple mechanisms to allow one file system object to refer to another: aliases, hard links, and symbolic links

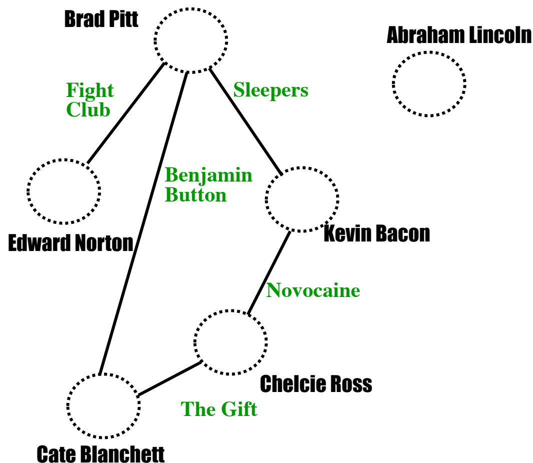
2. HFS+ stores Unicode characters in canonical, fully decomposed form.

3. The size of the data associated with a single extended attribute is limited to slightly less than 4KB in Mac OS X 10.4.

Another Tree?

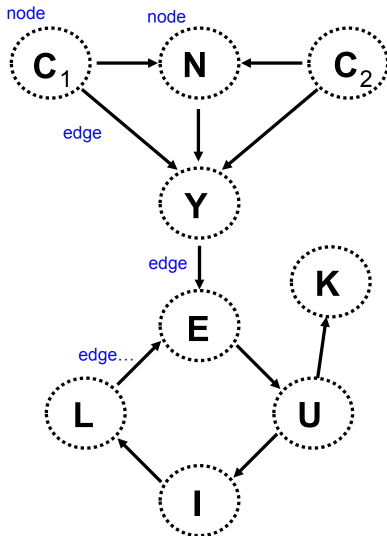


Undirected Graphs



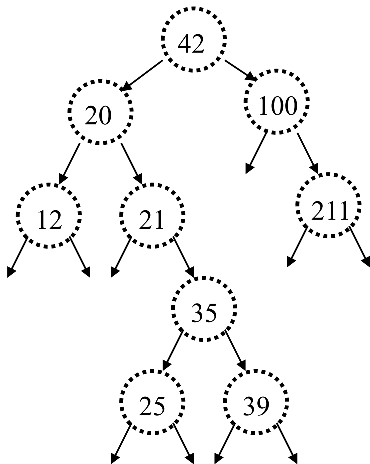
The "Oracle of Bacon" at **oracleofbacon.org/**

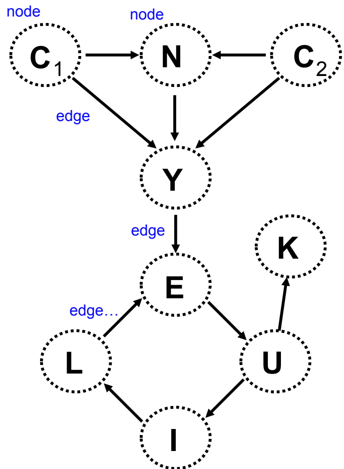
Directed Graphs



Trees vs. Directed Graphs

How are they different?



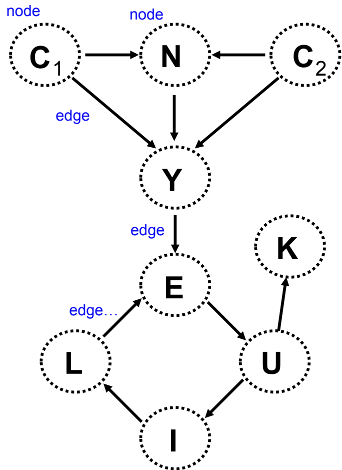


Graphical Programming?

Lists are our only building block in Racket.

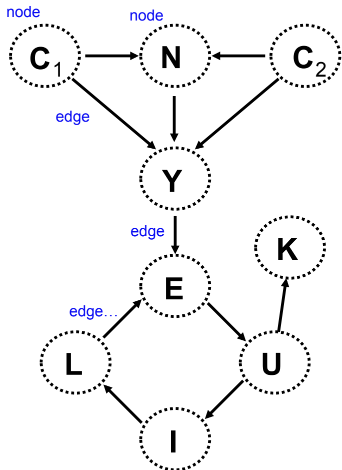
- ✓ BSTs: `'(42 (2 () ()) (50 (45 () ()) ()))`
- ✓ Graphs: ???

Implementation Strategy 1: Adjacency Lists



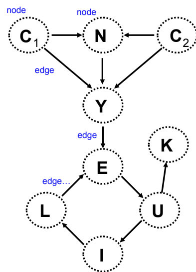

```
(define G '( [ c1 (n y) ]
             [ n  (y)   ]
             [ c2 (n y) ]
             [ k  ()    ]
             [ y  (e)   ]
             [ e  (u)   ]
             [ l  (e)   ]
             [ u  (i k) ]
             [ i  (l)   ]
           )
)
```

IMPLEMENTATION STRATEGY 2: EDGE LISTS




```
(define G '( (c1 n) (c1 y)
             (n  y) (c2 n)
             (c2 y) (y  e)
             (e  u) (u  i)
             (i  l) (l  e)
             (u  k)
           )
)
```

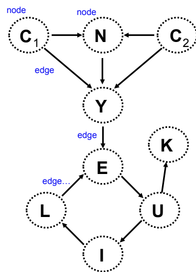
Find all the nodes in an edge list G




```
'( (c1 n) (c1 y) (n  y) (c2 n)
   (c2 y) (y  e) (e  u) (u  i)
   (i  l) (l  e) (u  k)       ))
```

```
;; Hint: there's a remove-duplicates function on lists.

(define (nodes G)

   (remove-duplicates (flatten G))
```

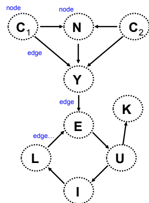
FIND THE CHIDREN OF A NODE n IN EDGE LIST G




```
'( (c1 n) (c1 y) (n  y) (c2 n)
   (c2 y) (y  e) (e  u) (u  i)
   (i  l) (l  e) (u  k)        ))
```

```
(define (kids n G)
   (map second
        (filter (lambda (e) (equal? (first e) n)) G)))
```

How about the parents?




```
'( (c1 n) (c1 y) (n  y) (c2 n)
   (c2 y) (y  e) (e  u) (u  i)
   (i  l) (l  e) (u  k)        ))
```

```
(define (parents n G)
    (kids n (map reverse G))
```

"Use it or lose it" — A general problem-solving strategy

1. Single "it" out
2. Recursively solve the problem **without** "it"
3. Recursively solve the problem **with** "it"
4. Combine (as appropriate)

All Sublists

```
;; (sublists '(a b)) ===> '( () (a) (b) (a b) )
;; (sublists '(a b c )) ===> '( () (a) (b) (a b)
                                (a c) (b c) (a b c))
;;    output possibly in a different order

(define (sublists L)
   (if (null? L)
       ;; all sublists of the empty list L
       '( () )

       ;; all sublists of a non-empty list L
       (let* ( [it     (first L)]
               ;; sublists without (first L)
               [loseit (sublists (rest L)) ]
               ;; sublists including (first L)
               [useit  (map (lambda (s) (cons (first L) s))
                            loseit) ] )
             (append loseit useit)) ))
```

Homework Problem: make-change

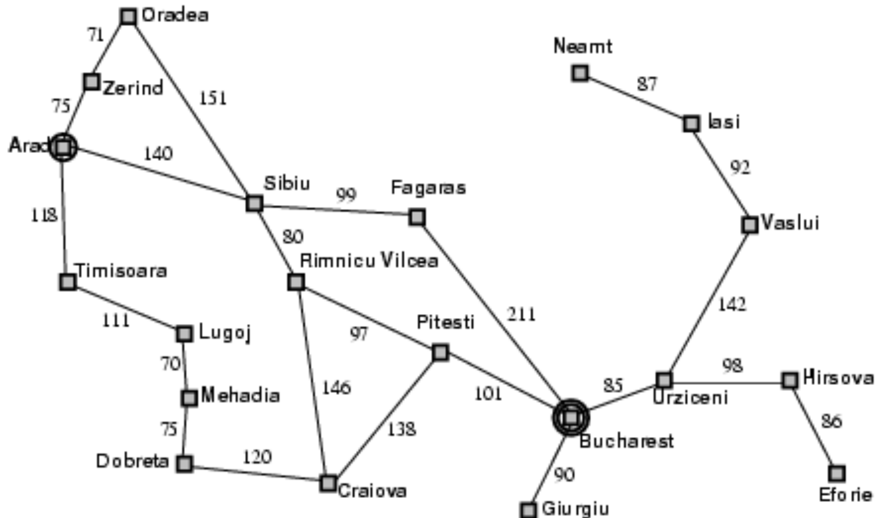
```
(make-change  42  '(2 5 10 25 50))  ;; ===> #t

(make-change  42  '(3 5 10 25 50))  ;; ===> #f
```

Reachability

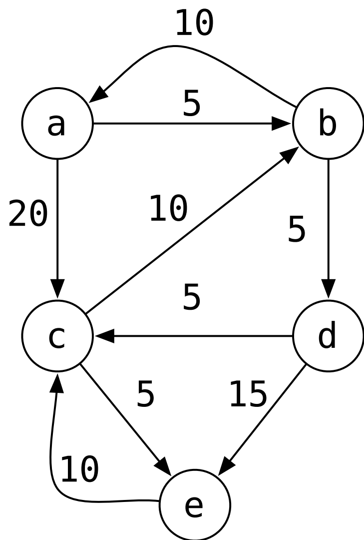
```
;; Is there a path from a to b in G?
(define (reach? a b G)
 (cond
  ((equal? a b) #t)
  ((null? G)    #f)
  ( else (let* ( [ EDGE   (first G)        ] ;; "it"
                 [ R      (rest G)         ]
                 [ loseit (reach? a b R)   ]
                 [ c      (first EDGE)     ]
                 [ d      (second EDGE)    ]
                 [ useit  (and (reach? a c R)
                               (reach? d b R)) ] )
           (or useit loseit) ))))
```

Homework: Weighted Graphs and Shortest Paths

Find the shortest path from Bucharest to Arad.



Representing Weighted Graphs




```
'( [a b 5]  [b a 10]
   [a c 20] [c b 10]
   [b d 5]  [d c 5]
   [c e 5]  [d e 15]
   [e c 10] )
```