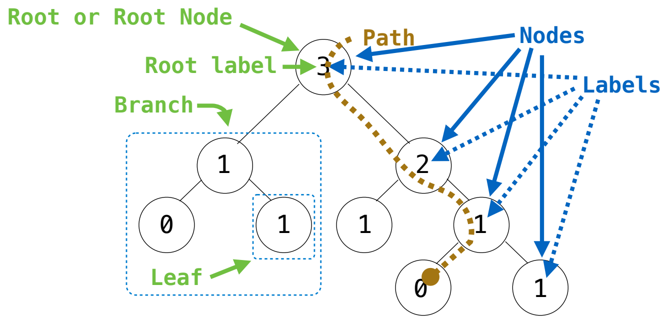
Data C88C Final Exam Study Guide



Recursive description:
- A **tree** has a root **label** and a list of **branches**
- Each branch is a **tree**
- A tree with zero branches is called a **leaf**

Relative description:
- Each location is a **node**
- Each **node** has a **label**
- One node can be the **parent/child** of another

```
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches
```

Built-in isinstance function: returns True if branch has a class that *is* **or** *inherits from* Tree

```
def leaves(tree):
   "The leaf values in a tree."
   if tree.is_leaf():
        return [tree.label]
    else:
        return sum([leaves(b) for b in tree.branches], [])
```

```
def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_Tree(n-2)
        right = fib_Tree(n-1)
        fib_n = left.label+right.label
        return Tree(fib_n,[left, right])
```

Exponential growth. E.g., recursive fib $O(b^n)$
Incrementing *n* multiplies *time* by a constant

Quadratic growth. E.g., overlap $O(n^2)$
Incrementing *n* increases *time* by *n* times a constant

Linear growth. E.g., slow exp $O(n)$
Incrementing *n* increases *time* by a constant

Logarithmic growth. E.g., exp_fast $O(\log n)$
Doubling *n* only increments *time* by a constant

Constant growth. Increasing *n* doesn't affect time $O(1)$

A **column** has a name and a type

A **table** has columns and rows

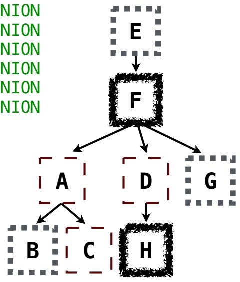
Latitude	Longitude	Name
38	122	Berkeley
42	71	Cambridge
45	93	Minneapolis

A **row** has a value for each column

```
SELECT [expression] AS [name], [expression] AS [name], ... ;

SELECT [columns] FROM [table] WHERE [condition] ORDER BY [order];
```

```
CREATE TABLE parents AS
  SELECT "daisy" AS parent, "hank" AS child UNION
  SELECT "ace"            , "bella"         UNION
  SELECT "ace"            , "charlie"       UNION
  SELECT "finn"           , "ace"           UNION
  SELECT "finn"           , "dixie"         UNION
  SELECT "finn"           , "ginger"        UNION
  SELECT "ellie"          , "finn";
```



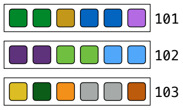
```
CREATE TABLE dogs AS
  SELECT "ace" AS name, "long" AS fur UNION
  SELECT "bella"      , "short"       UNION
  SELECT "charlie"    , "long"        UNION
  SELECT "daisy"      , "long"        UNION
  SELECT "ellie"      , "short"       UNION
  SELECT "finn"       , "curly"       UNION
  SELECT "ginger"     , "short"       UNION
  SELECT "hank"       , "curly";
```

```
SELECT a.child AS first, b.child AS second
  FROM parents AS a, parents AS b
  WHERE a.parent = b.parent AND a.child < b.child;
```

First	Second
bella	charlie
ace	dixie
ace	ginger
daisy	ginger

```
CREATE TABLE lift AS
  SELECT 101 AS chair, 2 AS single, 2 AS pair UNION
  SELECT 102         , 0          , 3         UNION
  SELECT 103         , 4          , 1;

SELECT chair, single + 2 * pair AS total FROM lift;
```



String values can be combined to form longer strings

```
sqlite> SELECT "hello," || " world";
hello, world
```

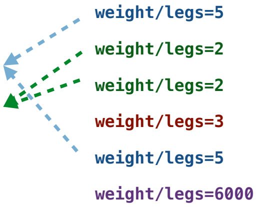
Basic string manipulation is built into SQL

```
sqlite> CREATE TABLE phrase AS SELECT "hello, world" AS s;
sqlite> SELECT substr(s, 4, 2) || substr(s, instr(s, " ")+1, 1)
        FROM phrase;
low
```

The number of groups is the number of unique values of an expression
A having clause filters the set of groups that are aggregated

```
SELECT weight/legs, count(*) FROM animals
                             GROUP BY weight/legs
                             HAVING COUNT(*)>1;
```

weight/ legs	count(*)
5	2
2	2



kind	legs	weight
dog	4	20
cat	4	10
ferret	4	10
parrot	2	6
penguin	2	10
t-rex	2	12000

An aggregate function in the [columns] clause computes a value from a group of rows:

- MAX([expression]) evaluates to the largest value of [expression] for any row in a group
- COUNT(*) evaluates to the number of rows in a group
- MIN, SUM, & AVG are also aggregate functions similar to MAX

With no GROUP BY clause, aggregation is performed over all rows:

```
select max(legs) from animals;
```

max(legs)
4