Matrix and Tree using List

Matrix is any doubly subscripted array of elements arranged in rows and columns.

$$\begin{array}{cc} & \text{columns} \\ \text{rows} & \begin{bmatrix} a & b \\ c & d \end{bmatrix} \end{array}$$

$$A = \begin{bmatrix} a_{11}, \dots, a_{1n} \\ a_{21}, \dots, a_{2n} \\ \dots\dots\dots\dots \\ a_{m1}, \dots, a_{mn} \end{bmatrix} = \{A_{ij}\}$$

m rows
n columns
m X n matrix

Row Vector: [1 x n] matrix ➔

$$A = \begin{bmatrix} a_1 a_2, \dots, a_n \end{bmatrix} = \{a_i\}$$

Column vector: [m x 1] matrix ➔

$$A = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_m \end{bmatrix} = \{a_i\}$$

Basic Matrix Operations

- Addition, Subtraction, Multiplication: creating new matrices (or functions)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

Just add elements

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a-e & b-f \\ c-g & d-h \end{bmatrix}$$

Just subtract elements

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

Multiply each row by each column

Matrix Addition and Subtraction

Addition

- Commutative: **A**+**B**=**B**+**A**
- Associative: (**A**+**B**)+**C**=**A**+(**B**+**C**)

$$\mathbf{A}+\mathbf{B}=\begin{bmatrix}2 & 4\\ 2 & 5\end{bmatrix}+\begin{bmatrix}1 & 0\\ 3 & 1\end{bmatrix}=\begin{bmatrix}2+1 & 4+0\\ 2+3 & 5+1\end{bmatrix}=\begin{bmatrix}3 & 4\\ 5 & 6\end{bmatrix}$$

Subtraction

- By adding a negative matrix

$$\mathbf{A}-\mathbf{B}=\begin{bmatrix}2 & 4\\ 5 & 3\end{bmatrix}-\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}=\begin{bmatrix}2 & 4\\ 5 & 3\end{bmatrix}+\begin{bmatrix}-1 & -2\\ -3 & -4\end{bmatrix}=\begin{bmatrix}1 & 2\\ 2 & -1\end{bmatrix}$$

Matrix Muplication
A x B = C

[2 x 2] [2 x 3]

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}$$

$$C = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} & a_{21}b_{13} + a_{22}b_{23} \end{bmatrix}$$

[2 x 3]

Square Matrix:Same number of rows and columns

$$B = \begin{bmatrix} 5 & 4 & 7 \\ 3 & 6 & 1 \\ 2 & 1 & 3 \end{bmatrix}$$

Identity Matrix

Square matrix with ones on the diagonal and zeros elsewhere.

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Identity matrix

Worked example $\boldsymbol{A I_3 = A}$ for a 3x3 matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1+0+0 & 0+2+0 & 0+0+3 \\ 4+0+0 & 0+5+0 & 0+0+6 \\ 7+0+0 & 0+8+0 & 0+0+9 \end{bmatrix}$$

- In Matlab: **eye(r, c)** produces an r x c identity matrix

Two Dimensional Arrays

- Some data can be organized efficiently in a **table** (also called a **matrix** or **2-dimensional array**)
- Each cell is denoted with two subscripts, a row and column indicator

B[2][3] = 50

B	0	1	2	3	4
0	3	18	43	49	65
1	14	30	32	53	75
2	9	28	38	50	73
3	10	24	37	58	62
4	7	19	40	46	66

2D Lists in Python

```
data = [ [1, 2, 3, 4],
         [5, 6, 7, 8],
         [9, 10, 11, 12]
       ]
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
>>> data[0]
[1, 2, 3, 4]
>>> data[1][2]
7
>>> data[2][5]   index error
```

2D List Example in Python

- Find the sum of all elements in a 2D array

```
def sum_matrix(table):
     sum = 0
     for row in range(0,len(table)):
         for col in range(0,len(table[row])):
             sum = sum + table[row][col]
    return sum
```

number of rows in the table

number of columns in the given row of the table

In a rectangular matrix, this number will be fixed so we could use a fixed number for row such as len(table[0])

Tracing the Nested Loop

```
def sum_matrix(table):
    sum = 0
    for row in range(0,len(table)):
        for col in range(0,len(table[row])):
            sum = sum + table[row][col]
    return sum
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
len(table) = 3
len(table[row])= 4 for every row
```

row	col	sum
0	0	1
0	1	3
0	2	6
0	3	10
1	0	15
1	1	21
1	2	28
1	3	36
2	0	45
2	1	55
2	2	66
2	3	78

2D Array Creation using List [1/2]

Static Allocation

```
# create a 2d list with fixed values (static allocation)
a = [ [ 2, 3, 4 ] , [ 5, 6, 7 ] ]
print(a)
```

Dynamic Allocation (1)

```
# Create a variable-sized 2d list
rows = 3
cols = 2

a=[]
for row in range(rows): a += [[0]*cols]

print("This IS ok.  At first:")
print("   a =", a)

a[0][0] = 42
print("And now see what happens after a[0][0]=42")
print("   a =", a)
```

2D Array Creation using List [2/2]

Dynamic Allocation (2)

```
rows = 3
cols = 2

a = [ ([0] * cols) for row in range(rows) ]

print("This IS ok.  At first:")
print("   a =", a)

a[0][0] = 42
print("And now see what happens after a[0][0]=42")
print("   a =", a)
```

Dynamic Allocation (3)

```
def make2dList(rows, cols):
    a=[]
    for row in range(rows): a += [[0]*cols]
    return a

rows = 3
cols = 2

a = make2dList(rows, cols)

print("This IS ok.  At first:")
print("   a =", a)

a[0][0] = 42
print("And now see what happens after a[0][0]=42")
print("   a =", a)
```

Manipulating 2D-Array made by List [1/3]

Getting 2d List Dimensions

```
# Create an "arbitrary" 2d List
a = [ [ 2, 3, 5] , [ 1, 4, 7 ] ]
print("a = ", a)

# Now find its dimensions
rows = len(a)
cols = len(a[0])
print("rows =", rows)
print("cols =", cols)
```

Nested Looping over 2d Lists

```
# Create an "arbitrary" 2d List
a = [ [ 2, 3, 5] , [ 1, 4, 7 ] ]
print("Before: a =", a)

# Now find its dimensions
rows = len(a)
cols = len(a[0])

# And now loop over every element
# Here, we'll add one to each element,
# just to make a change we can easily see
for row in range(rows):
    for col in range(cols):
        # This code will be run rows*cols times, once for each
        # element in the 2d list
        a[row][col] += 1

# Finally, print the results
print("After:  a =", a)
```

Manipulating 2D-Array made by List [2/3]

Printing over 2d Lists

```
# Helper function for print2dList.
# This finds the maximum length of the string
# representation of any item in the 2d list
def maxItemLength(a):
    maxLen = 0
    rows = len(a)
    cols = len(a[0])
    for row in range(rows):
        for col in range(cols):
            maxLen = max(maxLen, len(str(a[row][col])))
    return maxLen

# Because Python prints 2d lists on one row,
# we might want to write our own function
# that prints 2d lists a bit nicer.
def print2dList(a):
    if (a == []):
        # So we don't crash accessing a[0]
        print([])
        return
    rows = len(a)
    cols = len(a[0])
    fieldWidth = maxItemLength(a)
    print("[ ", end="")
    for row in range(rows):
        if (row > 0): print("\n  ", end="")
        print("[ ", end="")
        for col in range(cols):
            if (col > 0): print(", ", end="")
            # The next 2 lines print a[row][col] with the given fieldWidth
            formatSpec = "%" + str(fieldWidth) + "s"
            print(formatSpec % str(a[row][col]), end="")
        print(" ]", end="")
    print("]")

# Let's give the new function a try!
a = [ [ 1, 2, 3 ] , [ 4, 5, 67 ] ]
print2dList(a)
```

Manipulating 2D-Array made by List [3/3]

Accessing a whole row

```
# alias (not a copy!); cheap (no new list created)
a = [ [ 1, 2, 3 ] , [ 4, 5, 6 ] ]
row = 1
rowList = a[row]
print(rowList)
```

Accessing a whole column

```
# copy (not an alias!); expensive (new list created)
a = [ [ 1, 2, 3 ] , [ 4, 5, 6 ] ]
col = 1
colList = [ ]
for i in range(len(a)):
    colList += [ a[i][col] ]
print(colList)
```

Manipulating 3D-Array made by List

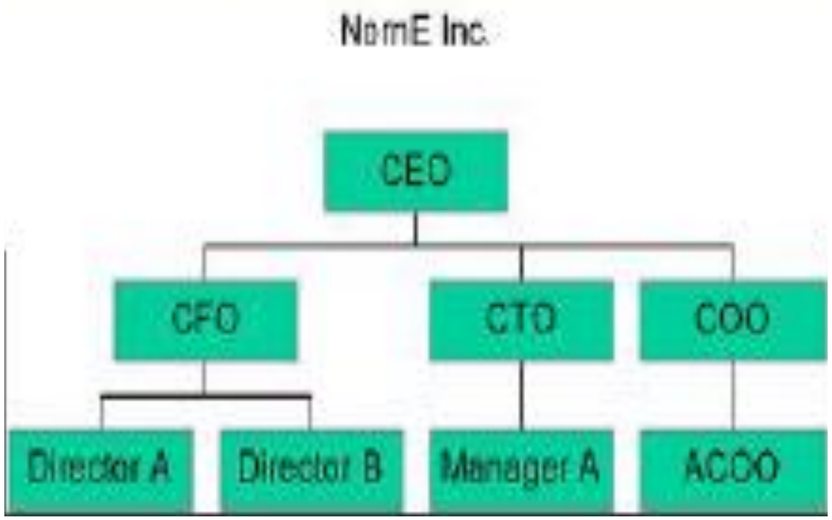
```
# 2d lists do not really exist in Python.
# They are just lists that happen to contain other lists as elements.
# And so this can be done for "3d lists", or even "4d" or higher-dimensional lists.
# And these can also be non-rectangular, of course!
```

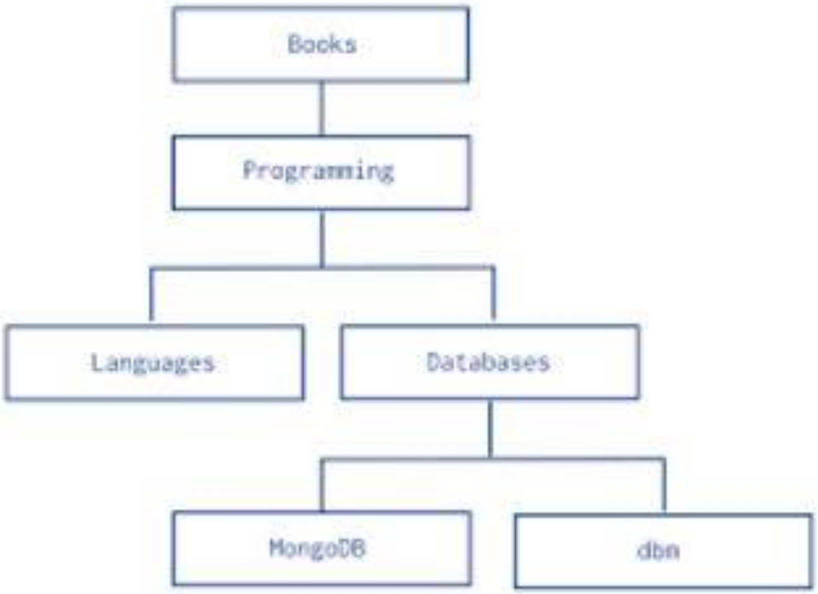
```
a = [ [ [ 1, 2 ],
        [ 3, 4 ] ],
      [ [ 5, 6, 7 ],
        [ 8, 9 ] ],
      [ [ 10 ] ] ]

for i in range(len(a)):
    for j in range(len(a[i])):
        for k in range(len(a[i][j])):
            print("a[%d][%d][%d] = %d" % (i, j, k, a[i][j][k]))
```

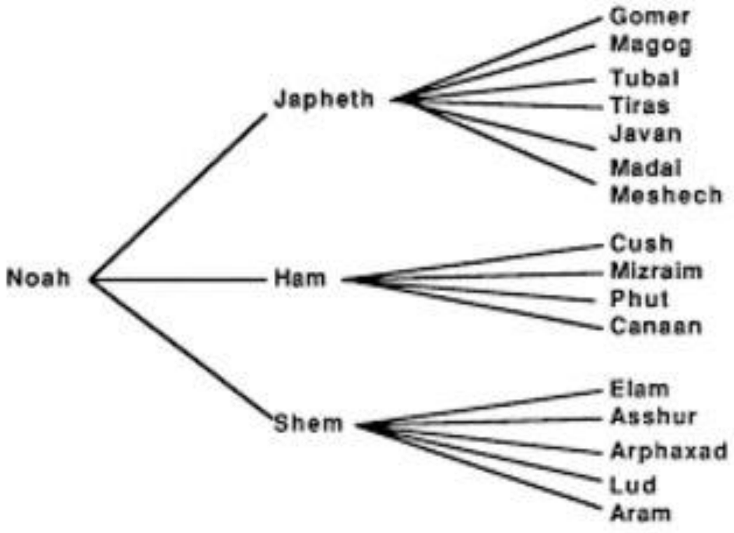
Better Ways for 2D Array, 3D Array,…..

- Array Module
- NumPy Module

Tree Structure using List



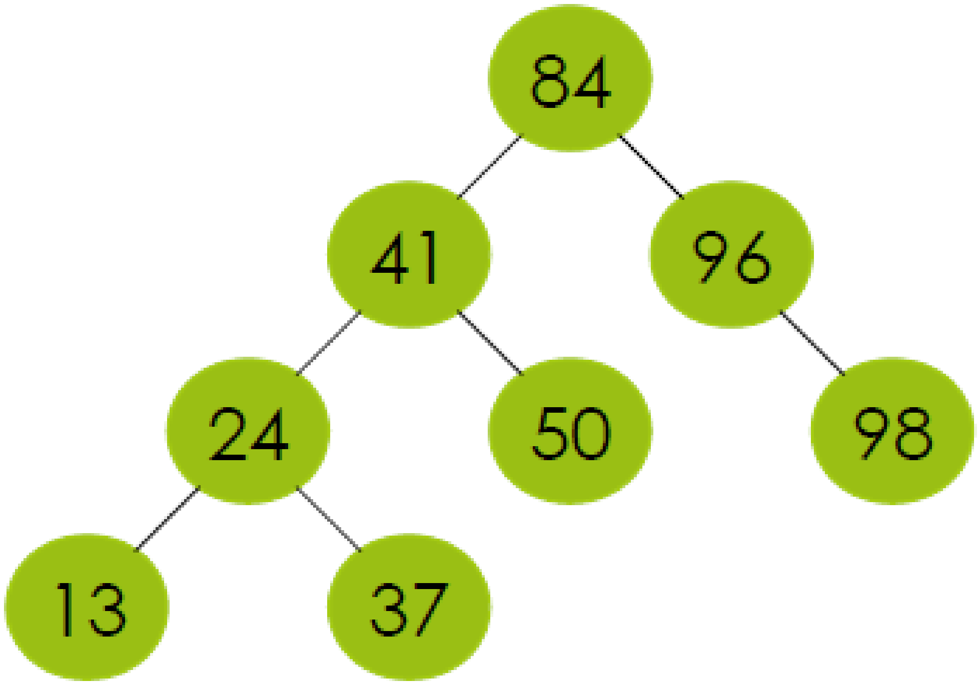




Trees

- A **tree** is a hierarchical data structure.
  - Every tree has a **node** called the **root**.
  - Each node can have 1 or more nodes as **children**.
  - A node that has no children is called a **leaf**.
- A common tree in computing is a **binary tree**.
  - A binary tree consists of nodes that have at most 2 children.
- Applications: data compression, file storage, game trees

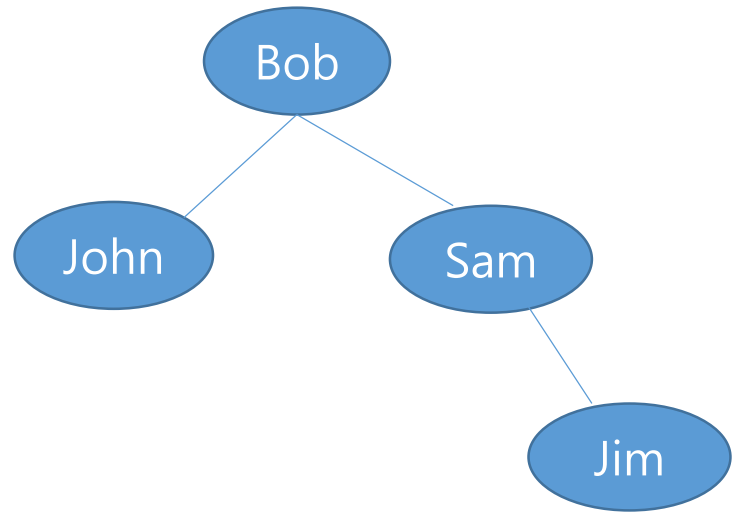
Binary Tree



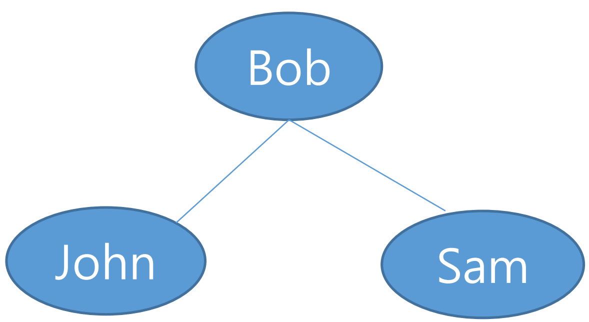
The root contains the data value 84.
There are 4 leaves in this binary tree: nodes containing 13, 37, 50, 98.
There are 3 internal nodes in this binary tree: nodes containing 41, 96, 24
This binary tree has height 3 – considering root is at level 0,
the maximum level among all nodes is 3

Trees and Their List Representations

[3 , [ ], [ ]]



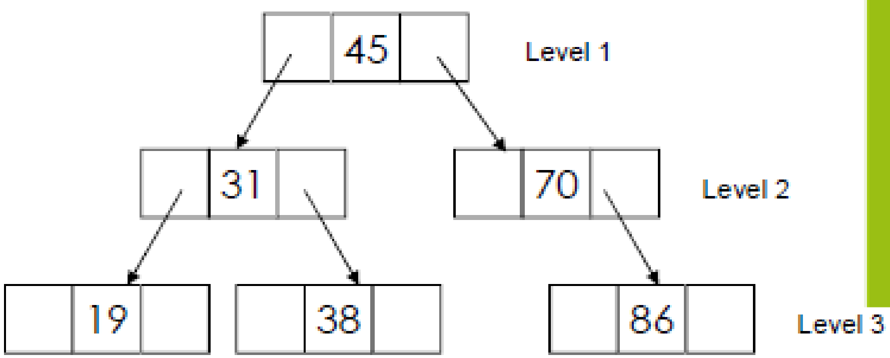
[Bob , [John, [ ]. [ ]],
[Sam, [ ], [Jim, [ ], [ ]]]

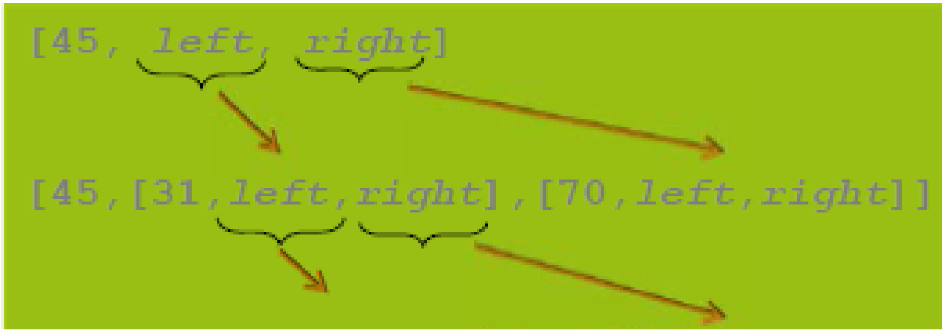


[Bob , [John, [ ]. [ ]], [Sam, [ ], [ ]]

Binary Trees: Implementation

- One common implementation of binary trees uses nodes like a linked list does.
  - Instead of having a “next” pointer, each node has a “left” pointer and a “right” pointer.





[45, [31, [19, [ ], [ ]], [38, [ ], [ ]], [70. [], [86, [ ], [ ]]]