


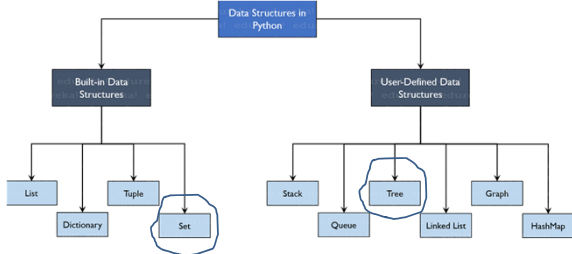
DASC / CSE 5300

Module II

Sharma Chakravarthy
 Information Technology Laboratory (IT Lab)
 Computer Science and Engineering Department
 The University of Texas at Arlington, Arlington, TX 76019
 Email: sharmac@cse.uta.edu
 URL: <http://itlab.uta.edu/sharma>

Python





```


graph TD
    A[Data Structures in Python] --> B[Built-in Data Structures]
    A --> C[User-Defined Data Structures]
    B --> D[List]
    B --> E[Dictionary]
    B --> F[Tuple]
    B --> G[Set]
    C --> H[Stack]
    C --> I[Queue]
    C --> J[Tree]
    C --> K[Linked List]
    C --> L[Graph]
    C --> M[HashMap]
    
```

4/13/2022

© your name


3

In Module II



- I will cover the following with examples so you can practice further on your own
 - Big-O complexity
 - Python data structures for analysis
 - Algorithms for data structures
 - Graphs
 - Dictionary
 - **Trees and Sets**
 - Stack, Queue

2



Tree Data Structure
(special case of a Graph)

Tree Data structure

- Tree is a non-linear data structure
- A tree is a graph with a root and nodes (a graph is not necessarily a tree)
- Root does **not have a parent**
- All other nodes (children) have a **unique** parent
- Number of children for a node determines the branching factor or fan-out of a tree (binary, n-ary)
- Nodes in a tree can be organized into levels
 - Root is at level (distance) 0
 - nodes with distance 1 from root are level 1
 - Nodes with distance 2 from root are in level 2 and so on
- Given the fan-out and # of total leaf nodes, the height (or max level) of the tree can be determined

5

Binary Tree Traversals

- Can be done in multiple ways
 - Pre-order
 - In-order
 - Post-order

pre, in, and Post always Refers to the root

PreOrder Traversal
 1. Visit root
 2. Visit Left
 3. Right

Node visit order
A B D E F G H J L M K N O

7

Tree examples

Binary Tree;
 complete binary tree
 Leaf nodes: 8; height 3
 Fan-out 2 (binary)
 $height = \log_{fan-out} \# \text{ total leaf nodes}$

For a binary tree
 $N_{max} = 2^0 + 2^1 + 2^2 + \dots + 2^h = 2^{h+1} - 1$
 Or $h = \log_2(N^{max} + 1) - 1$

4/13/2022 © Sharma Chakravarthy 6

Binary Tree Traversals

- Can be done in multiple ways A, B, D, E, F, C, G, H, J, L, M, K, O
- Pre-order
- In-order
- Post-order

InOrder Traversal
 1. Visit left
 2. Visit root
 3. Visit right

PreOrder Traversal
 1. Visit root
 2. Visit Left
 3. Right

DB F E A G C L J M H O K

8

Binary Tree Traversals

➤ Can be done in multiple ways A, B, D, E, F, C, G, H, J, L, M, K, O

- Pre-order
- In-order
- Post-order

PreOrder Traversal

1. Visit root
2. Visit Left
3. Right

PostOrder Traversal

1. Visit left
2. Visit right
3. Visit root

Node visit order
DFEBGLMJOKHCA

Pre: ABDEFCGHJLMKO
In: DBFEAGCLJMHOK
Post: DFEBGLMJOKHCA

Python Implementation

➤ Not supported natively in Python (in most PLS)
 ➤ But it is quite straightforward to implement it.

```

class Node(object):
def __init__(self, data):
    self.data = data
    self.children = []
def add_child(self, obj):
    self.children.append(obj)
n = Node(5)
p = Node(6)
q = Node(7)
n.add_child(p)
n.add_child(q)
print(n.children)
for c in n.children:
    print(c.data)
print(n.data)

```

6
7
5

Binary Search

- Items stored for lookup are usually not stored randomly in a (binary) tree
- They are stored in sorted order
 - Left tree has items < root value
 - Right tree has items > root value
- This reduces a $O(N)$ search to $\log_2 N$ search
- “Divide and conquer” algorithms use binary search
- Heap sort (or tournament sort) uses trees as heaps with specific properties
- Array representation can also be used for trees

Python Implementation

```

class Tree(object):
def __init__(self, name, left_subtree = None, right_subtree = None):
    self._name = name
    self._left_subtree = left_subtree
    self._right_subtree = right_subtree

def inorder(tree):
if tree is not None:
    inorder(tree._left_subtree)
    print tree._name
    inorder(tree._right_subtree)
a = Tree('a')
b = Tree('b')
c = Tree('c', a, b)
inorder(c)

```

HW: complete postOrder and preOrder traversals

a
c
b

Utility of Tree Traversals

- Can be used for searching for items in a tree in a systematic way
- You can also do depth first and breadth first (level order) traversals for searches
 - Used in many algorithms (remember shortest path)
- You can consider tree traversals as variants of depth first traversal
- Level order as a variant of breadth first traversal
- Monte Carlo search
 - Based on random sampling of the search space

13

Traversals on non-binary trees

- You can define the three traversals on a non-binary or n-ary tree
- Instead of left and right, you have to determine which subtrees are considered as left and which are right
- Beyond that, the algorithm is same
- Can implement as a recursive algorithm
- Or use a stack for backtracking

15

B and B+ trees

- Binary trees are not good for storing and searching **large number of objects**
- Binary trees are usually built **top down** and hence are susceptible to extremely unbalanced trees
- B and B+ trees were developed in the 1970's for avoiding the above
 - Built bottom up instead of top down
 - **Balanced, meaning the length of the path from root to any leaf node is the same**
 - Dynamically adjusts height (grows and shrinks)
 - Used in Database management systems (DBMSs)

14

Traversals on non-binary trees

- Trie or prefix tree is an application where n-ary trees are used
 - Words can be stored for easy lookup
 - Compact, as prefixes are stored **only ones**
 - Can be used for autocompletion
 - Look up $O(m)$ where m is the length of search string

16

The University of Texas ARLINGTON

Lab

Sets

The University of Texas ARLINGTON

Lab

Sets (2)

- Set builder notation or set comprehension (in Python)
 - Teens = {x | 20 < x < 12} “|” stands for “such that”
 - Evens = {x | n is even}
 - Primes = {p | p is prime} **is this set finite?**
- A set, by definition, does not have duplicates
- A set which allows duplicates is called a multi set or a bag
- You can have sets inside sets (element of a set can be a set) **only in a frozenset in Python**
- Ordinary Set elements have to be immutable in Python
- A set is **not** ordered (not a sequence)

19

The University of Texas ARLINGTON

Lab

Sets

- In mathematics, a set is a collection of elements
 - Elements need not be homogeneous (of the same type)
 - {"john", "coba 149", 55, 3.5} is a set
 - A set with no elements is an empty set
 - A set with a single element is a singleton set
 - A set can be finite or infinite (mathematically)
- Set theory has provided foundations for all branches of mathematics since 1950's
- Russel's paradox: set of all sets that **do not contain themselves cannot exist**
 - i.e., {x | x is a set and x <> x} cannot exist!
 - Remember set comprehension in Python?

18

The University of Texas ARLINGTON

Lab

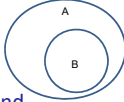
Sets (3)

- Sets form the basis for Relational algebra and SQL
- Sets also form the basis of the Relational Model (will study in Module III)
- Set operations
 - Membership (\in)
 - Subset and superset property (proper and improper subsets)
 - {1, 2} is a proper subset of {1, 2, 3} and
 - {1, 2, 3} is an improper subset of {1, 2, 3}
- Venn diagrams can be used to understand set operations

20

Sets (4)

- Sets form the basis for Relational algebra and SQL
- Sets also form the basis of the Relational Model (will study in Module III)
- Set operations
 - Membership
 - Subset and superset property (proper and improper subsets)
 - {1, 2} is a proper subset of {1, 2, 3} and
 - {1, 2, 3} is an improper subset of {1, 2, 3}
- Venn diagrams can be used to understand set operations



21

Sets Operations

- Difference ($A - B$)
 - $A - B \neq B - A$ (commutativity does not hold)
 - $A - (B - C) \neq (A - B) - C$
 - $A - A = \Phi$
 - $A - \Phi = A$
- Relative Complement
 - With respect to a finite set
 - $A \setminus B = A - B$
 - $A \setminus B \neq B \setminus A$ for $A \neq B$
 - $A \setminus A = \Phi$

23

Sets Operations



- Union ($A \cup B$)
 - $A \cup B = B \cup A$ (commutativity)
 - $A \cup (B \cup C) = (A \cup B) \cup C$ (Associativity)
 - $A \cup A = A$
 - $A \cup \Phi = A$
- Intersection ($A \cap B$)
 - $A \cap B = B \cap A$ (commutativity)
 - $A \cap (B \cap C) = (A \cap B) \cap C$ (Associativity)
 - $A \cap A = A$
 - $A \cap \Phi = \Phi$
- A subset B if and only if $A \cup B = B$

22

Sets Operations



- Cartesian product (x)
 - Cartesian product of two sets creates a new set by associating each element of first set with every element of the second set
 - $\{1, 2\} \times \{\text{red, white, green}\}$ is $\{(1, \text{red}), (1, \text{white}), (1, \text{green}), (2, \text{red}), (2, \text{white}), (2, \text{green})\}$
 - We will see the use of this in the join operation of SQL
- Cardinality
 - Of s set is the number of elements in it, denoted by $|s|$
 - $|\{a, b, c\}|$ is 3

24

 **Python Sets** 



- {} or set() function can be used to create sets
- To create an empty set use set(), not {} which creates an empty dictionary!
- X= set(<iter>) where <iter> is any iterable (e.g., list, tuple, string) that generates a list of objects to be included in the set.
 - set(["my", "name", "is", "sharma"]) creates the set {"my", "name", "is", "sharma"}
 - Set("abracadabra") creates the set {"a", "b", "r", "c", "d"}

25

 **Python Sets** 

- Modifying sets
 - A.update(B[, C ...]) or
 - A |= B [|C ...] # update with union
 - Similarly for intersection and other operators
 - .add(<elem>) # <elem> is a single immutable object
 - .remove(<elem>)
 - .pop() removes a random element
- Frozen sets
 - Python provides this as an immutable set which is otherwise a set
 - .frozenset(<iter>)
 - Useful when you need a set that is immutable
- **HW: Use set comprehension to create Cartesian product**

27

 **Python Sets** 

- Operations
 - | is union operation; also A.union(B)
 - & is intersection; also A.intersection(B)
 - - is difference; also A.difference(B)
 - A.isdisjoint(B) return True if A and B have no elements in common or $A \cap B = \Phi$
 - .issubset() improper subset
 - A < B return True if A is a proper subset of B
 - .issuperset()
 - A > B

26

Questions/comments




For more information visit:

<http://itlab.uta.edu>



Spring 2019 
 CSE 6331