HEAP DATA STRUCTURE

Describe Python's built-in data structure?

Python has four basic inbuilt data structures namely Lists, Dictionary, Tuple and Set. These almost cover 80% of the our real world data structures. This article will cover the above mentioned topics.

Above mentioned topics are divided into four sections below.

Lists: Lists in Python are one of the most versatile collection object types available. The other two types are dictionaries and tuples, but they are really more like variations of lists.

Dictionary: In python, dictionary is similar to hash or maps in other languages. It consists of key value pairs. The value can be accessed by unique key in the dictionary.

Tuple : Python tuples work exactly like Python lists except they are immutable, i.e. they can't be
changed in place. They are normally written inside parentheses to distinguish them from lists (which use square brackets), but as you'll see, parentheses aren't always necessary. Since tuples are immutable, their length is fixed. To grow or shrink a tuple, a new tuple must be created.

Sets: Unordered collection of unique objects.

- Set operations such as union (|) , intersection(&), difference(-) can be applied on a set.
- Frozen sets are immutable i.e once created further data can't be added to them
- {} are used to represent a set.Objects placed inside these brackets would be treated as a set.
-

Describe the Python user data structure?

User-defined **data structures:** Data structures that aren't supported by python but can be programmed to reflect the same functionality using concepts supported by python are user-defined data structures. There are many data structure that can be implemented this way:

- Linked list: A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.

- Stack: A stack is a linear structure that allows data to be inserted and removed from the same end thus follows a last in first out(LIFO) system. Insertion and deletion is known as push() and pop() respectively.

- Queue: A queue is a linear structure that allows insertion of elements from one end and deletion from the other. Thus it follows, First In First Out(FIFO) methodology. The end which allows deletion is known as the front of the queue and the other end is known as the rear end of the queue.

- Tree: A tree is a non-linear but hierarchical data structure. The topmost element is known as the root of the tree since the tree is believed to start from the root. The

elements at the end of the tree are known as its leaves. Trees are appropriate for storing data that aren't linearly connected to each other but form a hierarchy.

- Graph: A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes.

- Hashmap: Hash maps are indexed data structures. A hash map makes use of a hash function to compute an index with a key into an array of buckets or slots. Its value is mapped to the bucket with the corresponding index. The key is unique and immutable. In Python, dictionaries are examples of hash maps.

Describe the stages involved in writing an algorithm?

Every algorithm needs a process in order to be created and utilized. Described below are the four stages of algorithm analysis and design:

Design

The first stage is to identify the problem and thoroughly understand it. This is where it's important you consult with everybody who has an interest in the problem. Speak with them and see how they see the problem and what they need out of the solution so their part of the project or program can progress.

After you obtain the input, break out the problem into stages and calculate what happens at each step so the next step can occur. All of this is elementary and you probably did this from the first computer science class you ever took, but the same basic rules apply.

This is also the point where you are going to flowchart and/or use pseudo code to work out the specific problems of solving the flow of operations within the code.

Analyze

Once you have the basic framework of the algorithm it's time to start analyzing how efficient the code is in solving the problem. Algorithm design is fluid and subject to individual plans. This is a step that some programmers like to attack after they have coded the algorithm and run it through the compiler. Others prefer to examine it prior to writing the code and analyze results based on their expectations from the design stage.

Either way, what you are doing is looking for the efficiency of the algorithm. Algorithms are measured in time and space for their efficiency. Look at the algorithm you're designing and see how it works with different size data structures and what kind of time it takes to work through those structures. The problem here is deciding when the algorithm has reached maximum efficiency for the project and produces acceptable results.

Implement

Writing and coding the algorithm is the next step in the process. If you are the one writing the algorithm, then you need to write it in the coding language you understand the best. In order for you to know how to write the algorithm efficiently you have to know exactly what each line of code is going to accomplish when the program is executed. Write the code to execute quickly but can also handle the input data that it will receive.

If you are part of a team then have the best programmer in your group write the initial code, notate it well so the lesser experienced programmers will understand what is happening as the application is executed.

Experiment

Once the algorithm is designed and coded go back and experiment with different variables in the algorithm. Try and enter data that will make it fail or try and re-write the code to work it out most efficiently. Experimentation in algorithmic design is really just another step of the analyzing of the algorithm. Keep attacking the efficiency aspect until it executes as much data as necessary in the smallest amount of time. When you find flaws in what you have written or ways to write the code better, then go back to the design step and redesign the algorithm.

The design and analysis of algorithms is a circular process. You may find yourself becoming involved in any one of the steps. An experiment on an existing algorithm might lead to a new design. Or a re-coding of an algorithm might lead to a more efficient execution. Wherever you find yourself, keep working towards the goal of efficiency of the algorithm.

Outline the components of a good algorithm

The key components / characteristics of a good Algorithm are :

1. Input - An algorithm should have 0 or more well defined inputs.
2. Output - An algorithm should have 1 or more well defined outputs, and should match the desired output.
3. Finiteness - Algorithms must terminate after a finite number of steps.
4. Definiteness - Each step of algorithm must be defined unambiguously.
5. Well-Ordered - The exact order of operations performed in an algorithm should be concretely defined.
6. Independent - An algorithm should have step-by-step directions, which should be independent of any programming code.

Describe the Tree traversal method?

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree −

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

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

Explain the difference between inorder and postorder tree traversal?

INORDER TRAVERSAL -

An inorder traversal is a traversal technique that follows the policy, i.e., Left -> Root -> Right.

Here, the left subtree of the root node is traversed first, then the root node, and then the right subtree of the root node is traversed.

POSTORDER TRAVERSAL -

A Postorder traversal is a traversal technique that follows the policy, i.e., Left -> Right -> Root.

Here, the left subtree of the root node is traversed first, then the right-side sub-tree, and then finally the node is traversed