

22AIE112 DATA STRUCTURES & ALGORITHMS -1

(L-T-P-C:2-1-3-4)

UNIT-1-PART-1

Introduction to Data Structures

April 16, 2023

Dr.Remya S
Assistant Professor
Department of Computer Science



AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

School of Computing

Amritapuri Campus, Clappana P.O., Kollam - 690525, Kerala,
India. Ph: +91 (476) 280 2100 Email: csoffice@am.amrita.edu
www.amrita.edu/school/computing/amritapuri

Syllabus

Unit 1

Data Structure Hierarchy – primitive and non-primitive, Array data structure, properties and functions, single and multi-dimensional arrays, simple problems, Basics of Algorithm Analysis, big-O notation, notion of time and space complexity, dynamic arrays

Agenda

- ① **What** is a Data structure?
- ② **Why** it is important in programming languages?
- ③ Different **categories** of Data structure

What is a Data structure ?

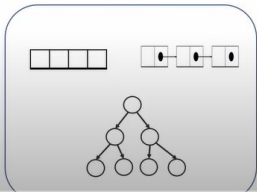
- A data structure is a way of **storing** data in a computer so that it can be used efficiently and it will allow the most efficient algorithm to be used.
- A data structure should be seen as a **logical concept** that must address two fundamental concerns:
 - ① **How the data will be stored ?**
 - ② **What operations** will be performed on it ?

Why Datastructures are important?

- Data structure and algorithms are two of the most important aspects of computer science.
- Data structures allow us to **organize and store data** in different ways, based on your needs.
- while algorithms allow us to process that data in a meaningful way.
- Learning data structure and algorithms will help you become a better programmer.



+ Home building instructions =



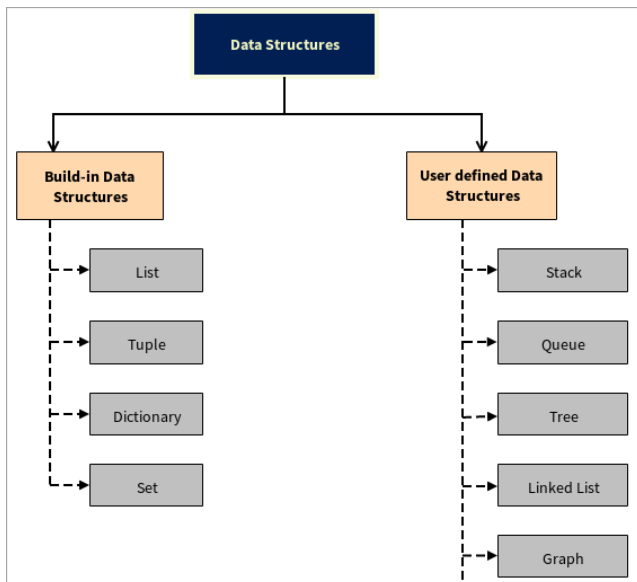
+ Code instructions =



Use right datastructure for a problem



Classification of Datastructures

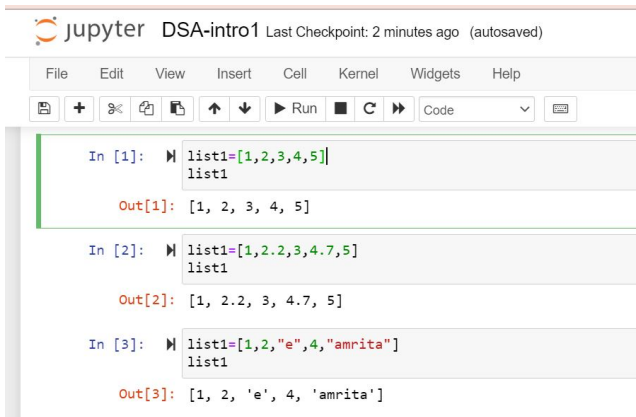


List

- [1, 2, 3, 4, 5]
- Mutable- dynamic-can change the order of items in a list or reassign them
- Linear Data structure-ordered elements
- Mixed type elements
- Variable length
- Zero-based indexing-can access list elements using indexing

0:	10
1:	2.3
2:	50
3:	44.66
4:	5.66
5:	40
6:	555

List



The image shows a Jupyter Notebook window titled "DSA-intro1" with a status bar indicating "Last Checkpoint: 2 minutes ago (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook contains three code cells, each with input and output.

```
In [1]: list1=[1,2,3,4,5]
list1

Out[1]: [1, 2, 3, 4, 5]
```

```
In [2]: list1=[1,2.2,3,4.7,5]
list1

Out[2]: [1, 2.2, 3, 4.7, 5]
```

```
In [3]: list1=[1,2,"e",4,"amrita"]
list1

Out[3]: [1, 2, 'e', 4, 'amrita']
```

```
In [3]: list1=[1,2,"e",4,"amrita"]  
list1
```

```
Out[3]: [1, 2, 'e', 4, 'amrita']
```

```
In [5]: list1[0]
```

```
Out[5]: 1
```

```
In [6]: list1[3]
```

```
Out[6]: 4
```

```
In [7]: list1[4]
```

```
Out[7]: 'amrita'
```

```
In [8]: 1 list1[-1]
```

```
Out[8]: 'amrita'
```

```
In [9]: list1[-2]
```

```
Out[9]: 4
```

```
>>> list1 = [1,2,3,4]  
>>> list1  
[1, 2, 3, 4]  
>>> list1.append(5)  
>>> list1  
[1, 2, 3, 4, 5]
```

```
>>> list1 = [[1,2],1,2,3]  
>>> list1  
[[1, 2], 1, 2, 3]  
>>> |
```

Tuples

- (1, 2, 3, 4, 5)
- Immutable
- Stores the data of different types
- can access list elements using indexing
- zero indexing
- negative indexing possible
- tuples can be nested
- Tuples are faster than lists
- tuples are defined as objects with the data type 'tuple'

```
>>> tuple1 = (1,2,3,4,6)
>>> type(tuple1)
<class 'tuple'>
>>> tuple2 = 1,2,3,4,5
>>> tuple2
(1, 2, 3, 4, 5)
>>>
```

```
>>> tuple1 = tuple()
>>> tuple1
()
>>> tuple2 = (2)
>>> tuple2
2
>>> type(tuple2)
<class 'int'>
>>> tuple3 = (2,)
>>> tuple3
(2,)
>>> type(tuple3)
<class 'tuple'>
```

```
>>> tuple1 = (1,2,3,4)
>>> tuple1.append(5)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    tuple1.append(5)
AttributeError: 'tuple' object has no attribute 'append'
```

Dictionary

- Dictionary is defined using key value pairs, separated by commas, enclosed within curly braces
- key and values are separated by colons

Dictionary_name = { key1:value1 , key2:value2... }

- Keys -Unique, immutable
- values-mutable or immutable

```
>>> d = {}
>>> d
{}
>>> d = {"amul": "amul@xyx.xom", "jia": "jia@xyz.xom"}
>>> d
{'amul': 'amul@xyx.xom', 'jia': 'jia@xyz.xom'}
>>> d = dict()
>>> d
{}
>>>
```

```
>>> dictionary = {"j":1,"k":2,"j":3}
>>> dictionary
{'j': 3, 'k': 2}
>>> d = {[1,2]: "a"}
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    d = {[1,2]: "a"}
TypeError: unhashable type: 'list'
>>>
```

```
>>> d = {"1":1,"2":2}
>>> d
{'1': 1, '2': 2}
>>> d1 = {"2":2,"1":1}
>>> d1
{'2': 2, '1': 1}
>>> d==d1
True
>>> d["1"]
1
>>>
```

```
>>> d = {}
>>> d["1"] = 1
>>> d
{'1': 1}
>>> d = {"1":{1:2}}
>>> d
{'1': {1: 2}}
```

Set

- Collection of unique elements- no items can be repeated
- sets cannot be nested


```

>>> s = set()
>>> s
set()
>>> s = set("hello")
>>> s
{'h', 'e', 'o', 'l'}
>>> s = set([1,2,3,4])
>>> s
{1, 2, 3, 4}

```

```

>>> s2 = {1,2,3,4}
>>> s2
{1, 2, 3, 4}
>>> type(s2)
<class 'set'>
>>> s = {1,2,3,1,2,3}
>>> s
{1, 2, 3}

```

```

>>> s = {1,2,3,4}
>>> s1 = {2,3,1,4}
>>> s==s1
True
>>> s.add(3)
>>> s
{1, 2, 3, 4}
>>> s.add(34)
>>> s
{1, 2, 3, 4, 34}

```

```

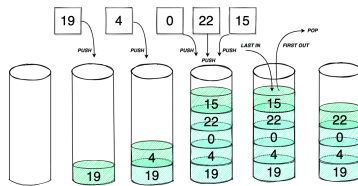
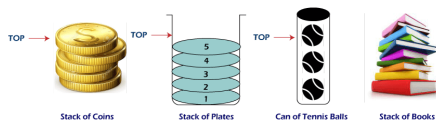
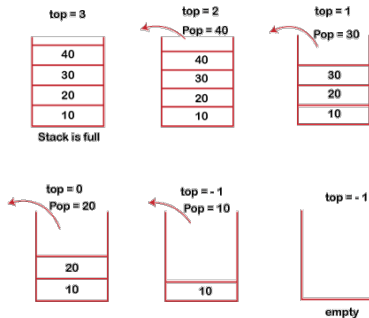
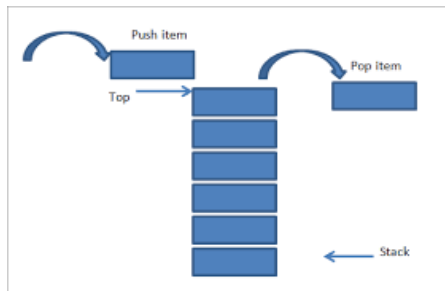
>>> s = {[1,2,3],3}
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    s = {[1,2,3],3}
TypeError: unhashable type: 'list'
>>> s = {{1,2},3,4}
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    s = {{1,2},3,4}

```

```
>>> s = {"hhf", 1, 1.3}
>>> s
{1, 'hhf', 1.3}
>>> 1 in s
True
>>> for i in s:
        print(i)

1
hhf
1.3
>>>
```

Stack



Last In First Out(LIFO)

OR

First In Last Out(FILO)

Stack

- Stack is an ordered list of the same type of elements.
- It is a linear list where all insertions and deletions are permitted only at one end of the list.
- Stack is a LIFO (Last In First Out) structure.
- In a stack, when an element is added, it goes to the top of the stack.

Queue

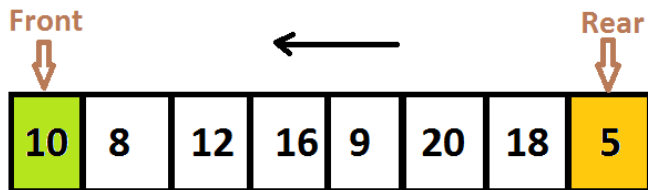


Queue

- Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other end called as FRONT.
- Front points to the beginning of the queue and Rear points to the end of the queue.
- Queue follows the FIFO (First - In - First Out) structure.
- According to its FIFO structure, element inserted first will also be removed first.
- In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue), because queue is open at both its ends.
- The enqueue() and dequeue() are two important functions used in a queue.

Queue

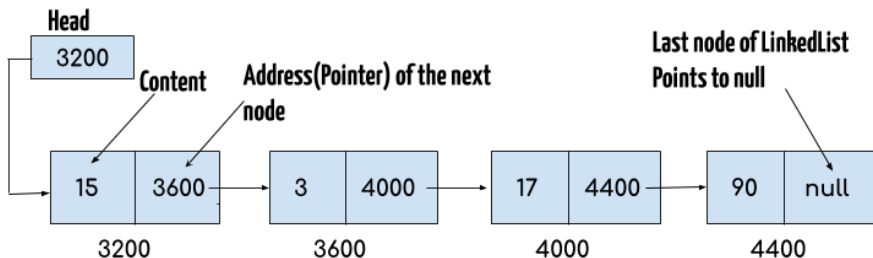
Queue Data Structure (First In First Out)

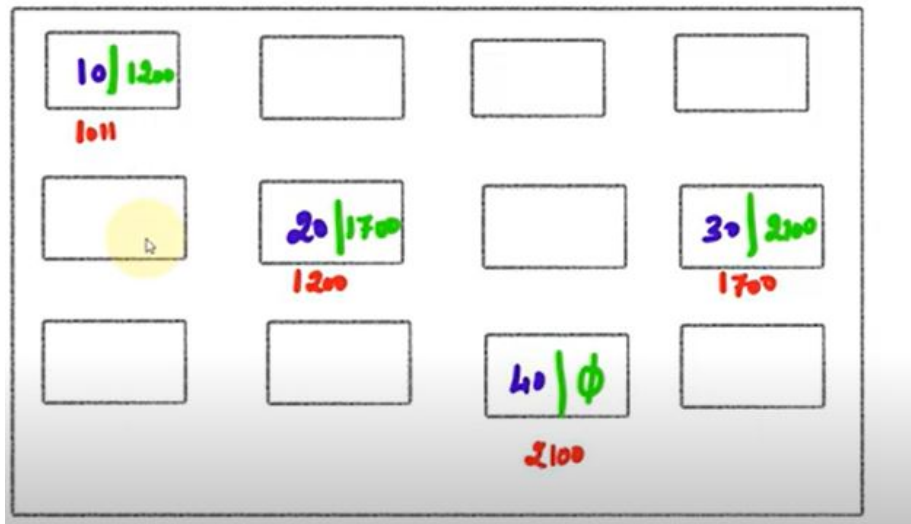


```
Enqueue(10)
Enqueue(8)
Enqueue(12)
Enqueue(16)
Enqueue(9)
Enqueue(20)
Enqueue(18)
Dequeue() -->10
Dequeue() -->8
Dequeue() -->12
Dequeue() -->16
Dequeue() -->9
Dequeue() -->20
Dequeue() -->18
```


Linked List

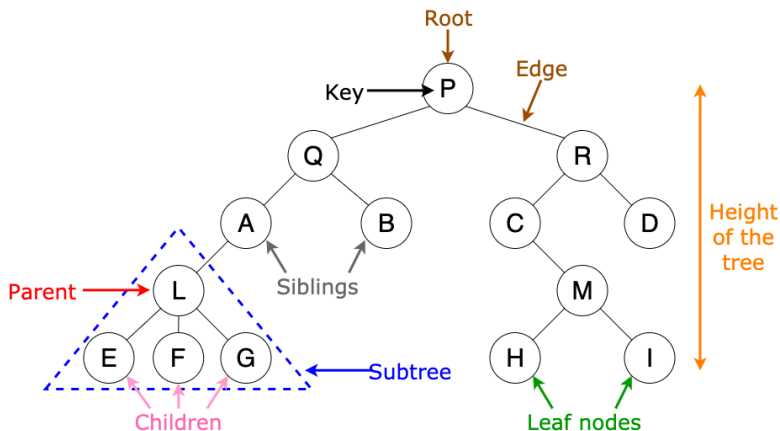
- A linked list is a linear data structure in which the elements are not stored at contiguous memory locations.
- The elements in a linked list are linked using pointers as shown in the figure





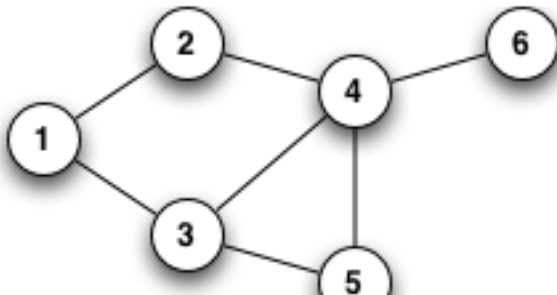
Tree

- Non linear data structure
- Represents relationship between nodes
- Nodes are connected by edges

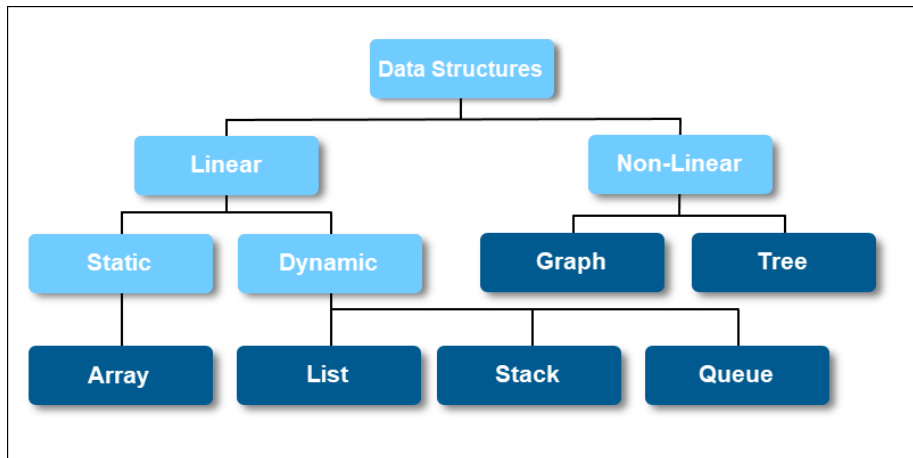


Graph

- A Graph is a non-linear data structure consisting of vertices and edges.
- The vertices are sometimes referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph
- Tree is a special type of graph(Every trees are graphs), but not all graphs are tree
- Multiple paths between nodes are possible in graphs



Classification of Data structures



Linear vs Non linear data structure

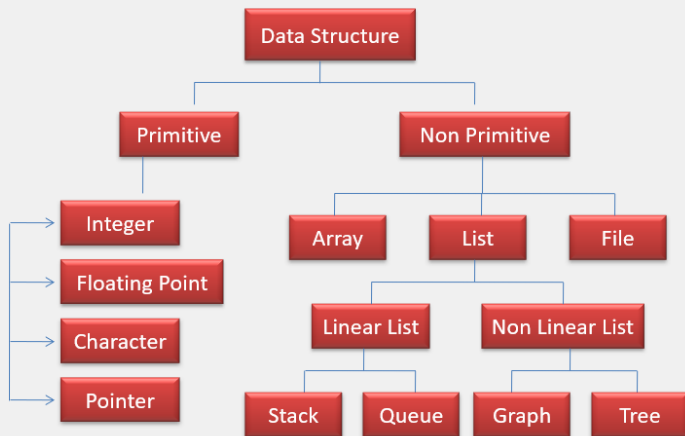
① Linear Data Structure

- Linear data structures can be constructed as a continuous arrangement of data elements in the memory.
- It can be constructed by using array data type.
- In the linear Data Structures the relationship of adjacency is maintained between the data elements

② Non-linear structure

- Non-linear data be constructed as a collection of randomly distributed set of data item joined together by using a special pointer (tag).
- In non-linear Data structure the relationship of adjacency is not maintained between the data items.

Classification of Data structures



Primitive vs Non primitive data structure

- Primitive data structure is a fundamental type of data structure that stores the data of only one type
- Non-primitive data structure is a type of data structure which is a user-defined that stores the data of different types in a single entity.