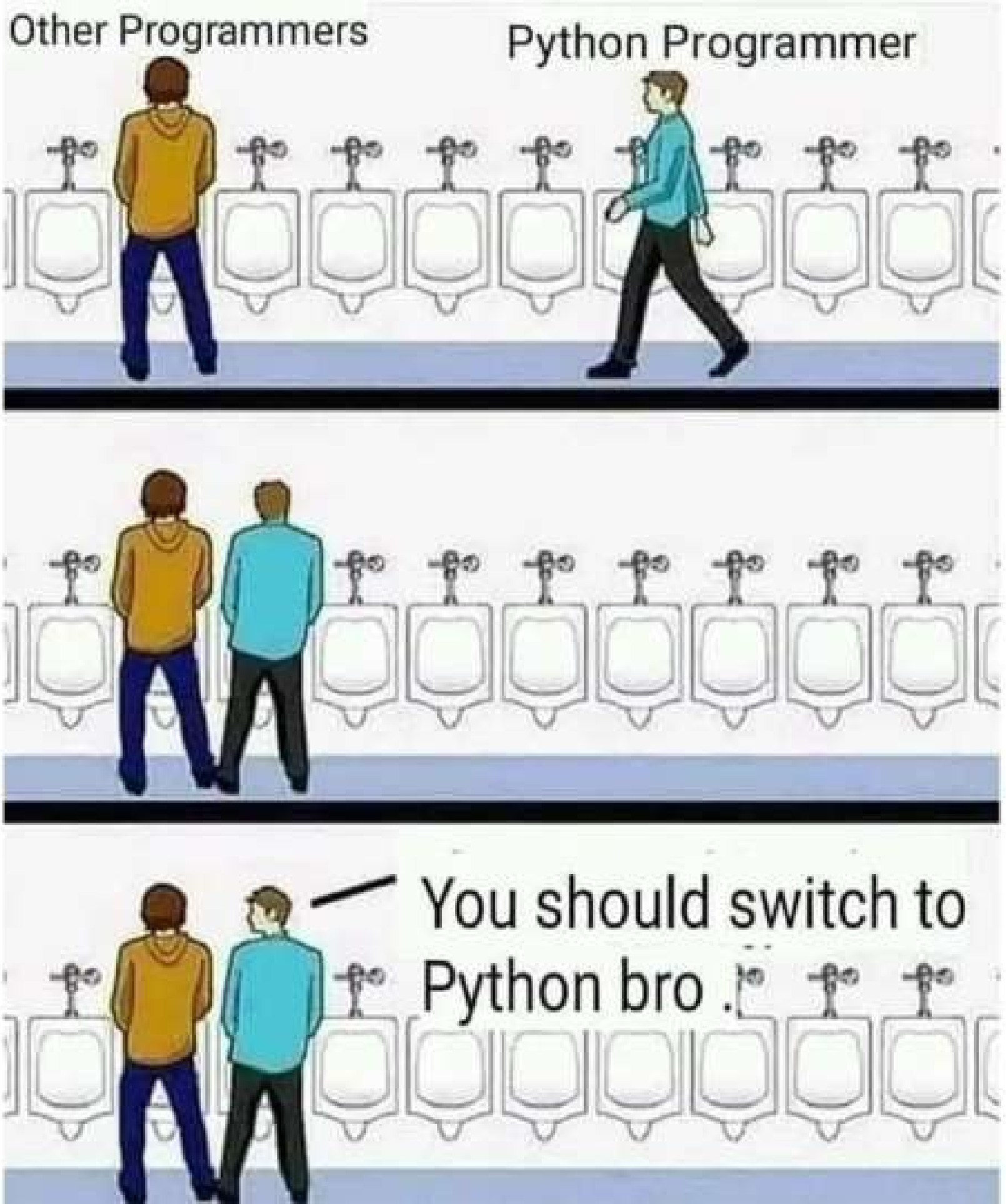

Other Programmers
Python Programmer
You should switch to Python bro .

DATA STRUCTURE Q/A

Instagram - @codeatul

Q What are data structure ?

⇨ Data Structure are the methods and techniques used to maintain data in an Organized fashion. This is primarily done to ensure that data can be manipulate and accessed in an efficient manner.

Q What is difference between Data Structure and File Structure ?

⇨

Data Structure	File Structure
• Data stored on both (Disk and Ram.	• Data Stored only on Disk.
• Costomized storage Policies	• Standard file Storage Polices
• High compatibility with external apps	• Low compatibility with external apps.

Q What is Linked List ?

⇨ Linked List is a data structure consist of individual entities called nodes. These nodes have capability of to connect other nodes. and create chain in the process. These continuous chain structure forms a linked list as the name suggest.

Q What are the types of searching used in Data Structure?

way that it has the ability to split the data unit into chunks and then perform search operation.

Q How are individual elements accessed in an array?

⇨ Each of the values in an array is given an index position from 0 to n-1, where "n" is the number of elements in the array. Individual element can be accessed by using the index element or operations. Multidimensional arrays have more than one dimension to work with.

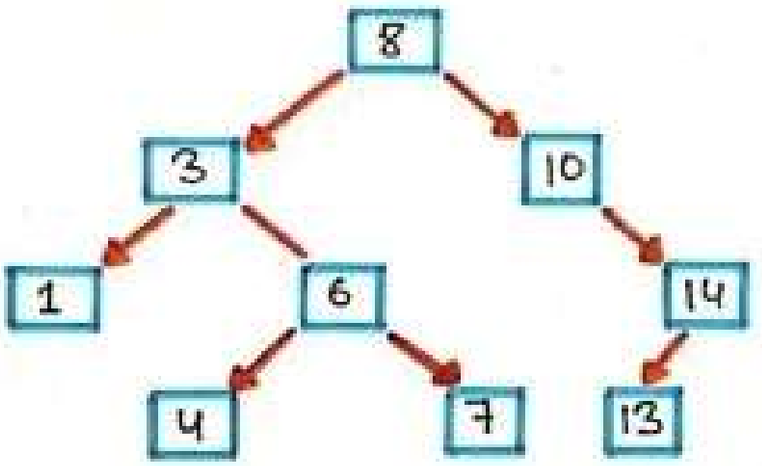
Q How does binary search work?

⇨ Binary search is used when there is primarily a creation of efficiency. It involves working on already ordered data, which is stored either in ascending order or descending order. To begin with the middle element of the array is find out and search begin from there. The array is searched in two parts based on a search value being higher or lower than middle element. It is the key to know the order of the arrangement to help search the value accordingly.

Q What is Queue in Data Structure?

⇨ A Queue is widely used data structure that is used to denote the ordered access and manipulation of an element. The operation of this data structure is exactly same as literal queue in the real world. Elements are added one after the other and are

A Binary search tree is a data structure that stores data in every efficient manner. It consist of two primary nodes from root node. The main thing here is that the values of the nodes in the left subtree are less in number than the values of the root node and the values of the nodes on the right of the root node are correspondingly higher than the root. Also individually both of these left and right subtree are their own binary seayrh trees at all points of time.

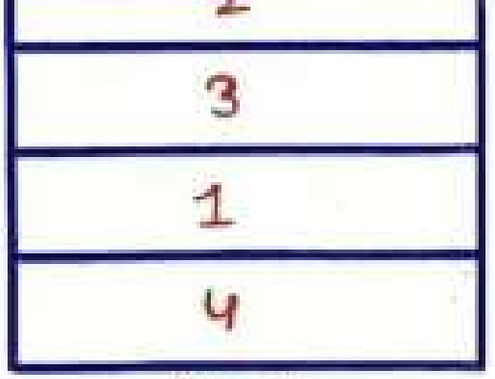


Q What is the meaning of FIFO?

FIFO is also known as First In First Out is a way of representing a data operation on factors such as how data is accessed and in what order here the data is first put into the list will be the first entity to exit from the ordered data structure.

Q What is the difference between void and Null?

Void :- Void is data type identifier in data structure.
Null :- Null is considered to be value with no physical



Stack

Q What is the working of LIFO?

⇨ LIFO stands for the Last In First Out access order. It is directly corresponding to how the data can be worked on and modified. The data entity is stored or pushed in last is the first one to be worked on at any point in time. If there is requirement to access the very first element stored then first you have to retrive all of the data that came after that element.

Q What are Multidimentional array?

⇨ Multidimentional arrays are the arrays that with more than one level or dimension for every point of storage. This is primarily used in cases where data cannot be represented or storage using only one dimension.

Q Are Linked list Linear or non linear data Structure?

⇨ Linked list are considered to be the best words here (Both). Based on usage, If it is a storage policy then it is considered as non-linear. whereas If

⇨ Dynamic memory management is a technique in which storage units are allocated based on requirements continuously. Using dynamic memory allocation, individual data structures can be either stored separately or combined to form entities called composites. These composites can be worked on when required.

Q What are Push and PoP operations in Data Structure?

⇨ Both Push and PoP operations denote how data can be stored and used when required in a stack. The **Push** operations denotes that users are adding data into structure, and the **PoP** operation denotes data being pulled or removed from the structure. Usually, the top most element is considered when performing Push and PoP operations.

Q How variable is stored in memory when using data structure?

⇨ A variable is stored based on amount of memory that is needed. First the required quantity of memory is assigned and later it is stored based on the data structure being used. Using concept such as dynamic allocation ensures high efficiency and that the storage unit can be supplied based on requirements in real time.

Q What is Merge sort?

⇨ Merge sort is method of sorting which is based on the devide and conquer technique. Here the data entities adjecent to each other are first merged and sorted in every ittration to create sorted lists. These smaller sorted lists are combined at the end to form the completely sorted list.

List

Intro :

- List is a sequential collection of objects.
- Elements are positioned using zero-based index.
- Elements can be inserted or removed or retrieved from any arbitrary position using an integer index.

Popular Implementations :

- ArrayList, Vector And LinkedList

Internal Structure :

- **ArrayList :** Internally uses re-sizable array which grows or shrinks as we add or delete elements.
- **Vector :** Same as ArrayList but it is synchronized.
- **LinkedList :** Elements are stored as Nodes where each node consists of three parts - Reference To Previous Element, Value Of The Element and Reference To Next Element.

Null Elements :

- **ArrayList :** Yes
- **Vector :** Yes
- **LinkedList :** Yes

Duplicate Elements :

- **ArrayList :** Yes
- **Vector :** Yes
- **LinkedList :** Yes

Order Of Elements :

- **ArrayList :** Insertion Order
- **Vector :** Insertion Order
- **LinkedList :** Insertion Order

Synchronization :

- **ArrayList :** Not synchronized
- **Vector :** Synchronized
- **LinkedList :** Not synchronized

Performance :

- **ArrayList :** Insertion -> O(1) (If insertion causes restructuring of internal array, it will be O(n)), Removal -> O(1) (If removal causes restructuring of internal array, it will be O(n)), Retrieval -> O(1)
- **Vector :** Similar to ArrayList but little slower because of synchronization.
- **LinkedList :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(n)

When to use?

- **ArrayList :** Use it when more search operations are needed then insertion and removal.
- **Vector :** Use it when you need synchronized list.
- **LinkedList :** Use it when insertion and removal are needed.

Queue

Intro :

- Queue is a data structure where elements are added from one end called tail of the queue and elements are removed from another end called head of the queue.
- Queue is typically FIFO (First-In-First-Out) type of data structure.

Popular Implementations :

- PriorityQueue, ArrayDeque and LinkedList (Implements List also)

Internal Structure :

- **PriorityQueue :** It internally uses re-sizable array to store the elements and a Comparator to place the elements in some specific order.
- **ArrayDeque :** It internally uses re-sizable array to store the elements.

Null Elements :

- **PriorityQueue :** Not allowed
- **ArrayDeque :** Not allowed

Duplicate Elements :

- **PriorityQueue :** Yes
- **ArrayDeque :** Yes

Order Of Elements :

- **PriorityQueue :** Elements are placed according to supplied Comparator or in natural order if no Comparator is supplied.
- **ArrayDeque :** Supports both LIFO and FIFO

Synchronization :

- **PriorityQueue :** Not synchronized
- **ArrayDeque :** Not synchronized

Performance :

- **PriorityQueue :** Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(1)
- **ArrayDeque :** Insertion -> O(1) , Removal -> O(n), Retrieval -> O(1)

When to use?

- **PriorityQueue :** Use it when you want a queue of elements placed in some specific order.
- **ArrayDeque :** You can use it as a queue OR as a stack.

Set

Intro :

- Set is a linear collection of objects with no duplicates.
- Set interface does not have its own methods. All its methods are inherited from Collection interface. It just applies restriction on methods so that duplicate elements are always avoided.

Popular Implementations :

- HashSet, LinkedHashSet and TreeSet

Internal Structure :

- **HashSet :** Internally uses HashMap to store the elements.
- **LinkedHashSet :** Internally uses LinkedHashMap to store the elements.
- **TreeSet :** Internally uses TreeMap to store the elements.

Null Elements :

- **HashSet :** Maximum one null element
- **LinkedHashSet :** Maximum one null element.
- **TreeSet :** Doesn't allow even a single null element

Duplicate Elements :

- **HashSet :** Not allowed
- **LinkedHashSet :** Not allowed
- **TreeSet :** Not allowed

Order Of Elements :

- **HashSet :** No order
- **LinkedHashSet :** Insertion order
- **TreeSet :** Elements are placed according to supplied Comparator or in natural order if no Comparator is supplied.

Synchronization :

- **HashSet :** Not synchronized
- **LinkedHashSet :** Not synchronized
- **TreeSet :** Not synchronized

Performance :

- **HashSet :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **LinkedHashSet :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **TreeSet :** Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(log(n))

When to use?

- **HashSet :** Use it when you want only unique elements without any order.
- **LinkedHashSet :** Use it when you want only unique elements in insertion order.
- **TreeSet :** Use it when you want

Map

Intro :

- Map stores the data in the form of key-value pairs where each key is associated with a value.
- Map interface is part of Java collection framework but it doesn't inherit Collection interface.

Popular Implementations :

- HashMap, LinkedHashMap And TreeMap

Internal Structure :

- **HashMap :** It internally uses an array of buckets where each bucket internally uses linked list to hold the elements.
- **LinkedHashMap :** Same as HashMap but it additionally uses a doubly linked list to maintain insertion order of elements.
- **TreeMap :** It internally uses Red-Black tree.

Null Elements :

- HashMap : Only one null key and can have multiple null values.
- LinkedHashMap : Only one null key and can have multiple null values.
- TreeMap : Doesn't allow even a single null key but can have multiple null values.

Duplicate Elements :

- **HashMap :** Doesn't allow duplicate keys but can have duplicate values.
- **LinkedHashMap :** Doesn't allow duplicate keys but can have duplicate values.
- **TreeMap :** Doesn't allow duplicate keys but can have duplicate values.

Order Of Elements :

- **HashMap :** No Order
- **LinkedHashMap :** Insertion Order
- **TreeMap :** Elements are placed according to supplied Comparator or in natural order of keys if no Comparator is supplied.

Synchronization :

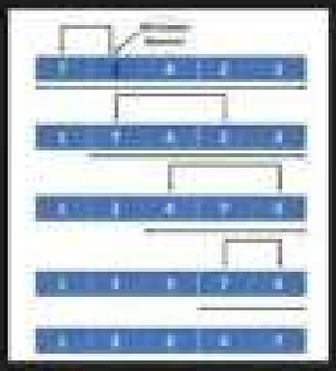
- **HashMap :** Not synchronized
- **LinkedHashMap :** Not Synchronized
- **TreeMap :** Not Synchronized

Performance :

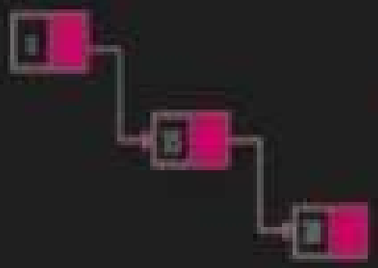
- **HashMap :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **LinkedHashMap :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **TreeMap :** Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(log(n))

When to use?

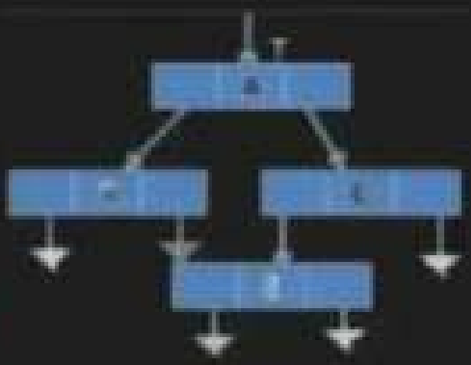
DATA STRUCTURES
DIAGRAMS



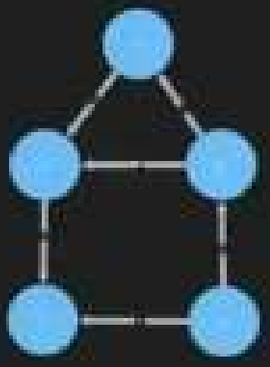
SORTING



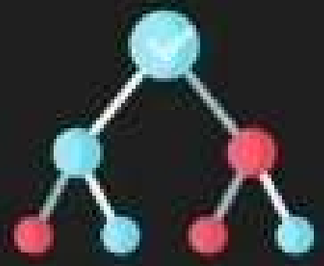
LINK LIST



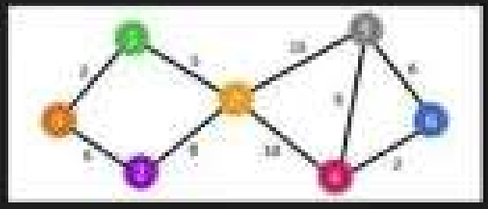
LIST



SPANNING TREE

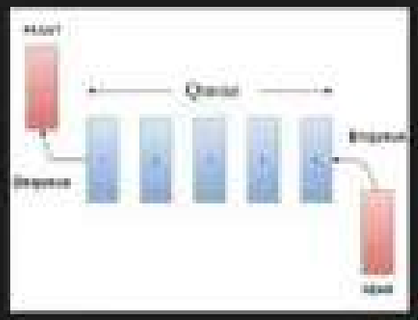


TREE

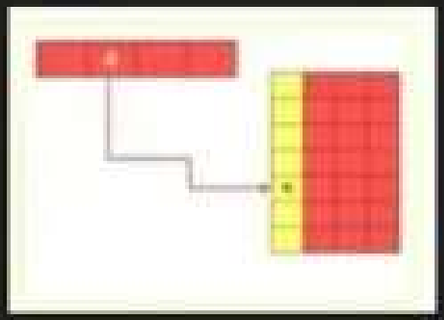


GRAPH

STACK



QUEUE



HASHING