DATA STRUCTURES

15

Introduction

Prof. Khalil Al-Shqeerat

CS Dept. – Qassim University

INTRODUCTION

- What is data?
  - Quantities, characters or symbols on which operations are performed by a computer
    - Stored on external memory
    - May be and transmitted in the form of electrical signals.
  - Example: C = A + B (A & B is data)
- When data becomes information?
  - If data is arranged in a systematic way then it gets a structure and become meaningful.
  - MEANINGFUL OR PROCESSED DATA IS CALLED **INFORMATION**.

DATA

- Values or set of values.
- A data item refers to a single unit of values.
- Data items that are divided into sub items are group items; those that are not are called elementary items.
  - For example, an employee's name may be divided into three sub items – first name, middle name and last name
- An entity is something that has certain attributes or properties which may be assigned values.
  - The values may be either numeric or non-numeric.

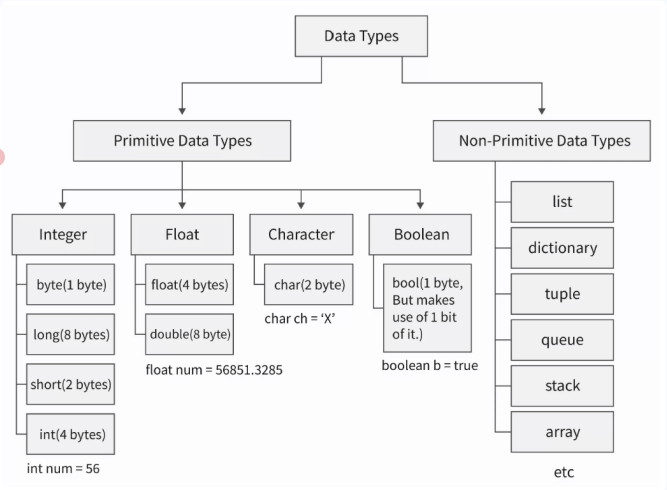
WHAT IS A DATA STRUCTURE?

- A data structure is the systematic way to organize data so that it can be used efficiently.
- It is a group of data elements stored together under one name
  - Example: arrays
    - Instead of creating multiple variables of same type.
    - Storing strings is equivalent of storing a sequence of characters.
- It is also referred as "Abstract Data Type (ADT)"
  - ADT is a type (or class) for objects whose behavior is defined by a set of values and a set of operations
- The challenge is to select the most appropriate data structure for the problem

CLASSIFICATION OF DATA STRUCTURES

- The data structure can be classified into two categories
  - primitive data structure
    - Storing the value of only one data type
  - Non-primitive data structure.
    - Storing the value of more than one data type

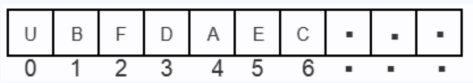
* غير تقليدية



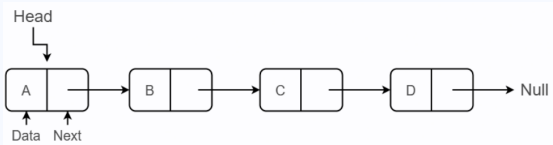
PHYSICAL DATA STRUCTURES

* نوع اخر فيزكل

- Actual implementations of data structures in memory.
  - Array, Linked list



Array



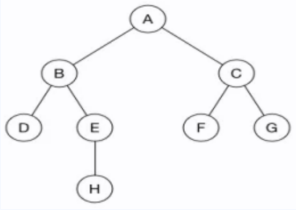
* تكون منتشرة Linked list

LOGICAL DATA STRUCTURES

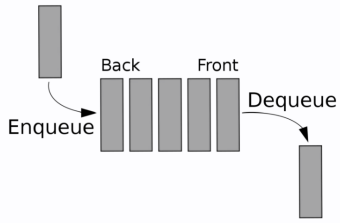
* مبنيه على اللي فوق

- Logical implementation of data over physical memory.
  - Stack, Queue, Tree

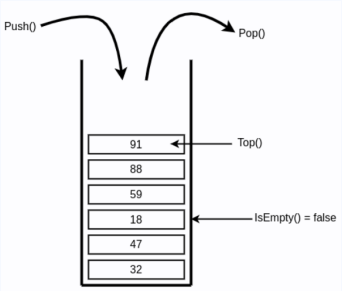
* هو هنيه اشكال حقيقه بالذاكرة



Tree



Queue



Stack

STATIC VS. DYNAMIC DATA STRUCTURES

ثابت

- Static data structure
  - Collection of data in memory which have a fixed size
  - It can store a limited amount of elements or data in memory.
  - Array is an example of static data structure.

ثابت

مثال: array

- Dynamic Data Structure
  - Collection of data in memory which do not have a fixed size
  - Its size can be modified during the operations performed on it
  - It can store a variable amount of elements or data in memory.
  - Linked List is an example of dynamic data structure.

* عكس اغير الحجم وهو يشتغل

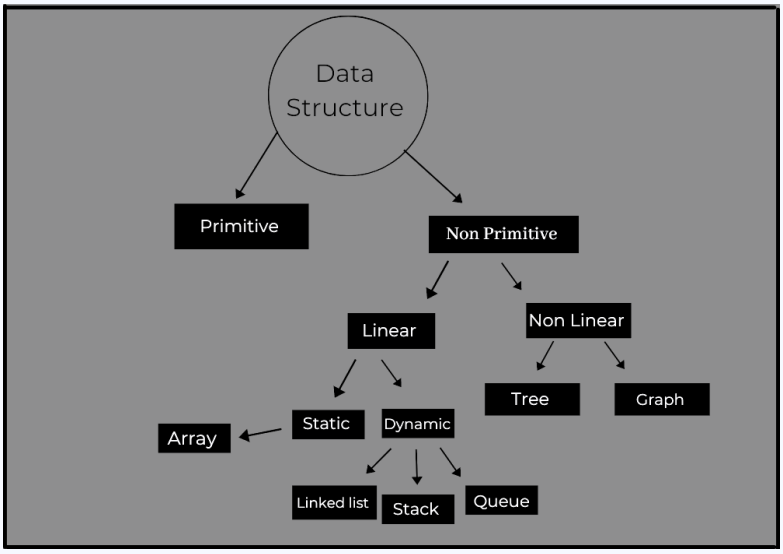
LINEAR DATA STRUCTURES

- Data elements are arranged sequentially or linearly
- Every element is attached to its previous and next adjacent
- Single level is involved.
  - It can traverse all the elements in single run only.
- Easy to implement because computer memory is arranged in a linear way.
- Array, Stack, Queue, Linked list are examples of Linear data structures.

NON-LINEAR DATA STRUCTURES

- Data elements are not arranged sequentially or linearly.
- Hierarchical multiple levels are involved.
  - It can't traverse all the elements in single run only.
- Its implementation is complex.
- It utilizes computer memory efficiently in comparison to a linear data structure.
- Its examples are trees and graphs

SUMMARY



DATA STRUCTURE OPERATIONS

- Different types of operations can be performed for the manipulation of data in every data structure.
  - Traversal: accessing each record/node exactly once so that certain items in the record may be processed. اعر على كل العناصر
  - Search: Finding the location of the desired node with a given key value, or to satisfy one or more conditions. ابحث
  - Insertion: Adding a new node/record to the structure. اضيف
  - Deletion: Removing a node/record from the structure.
  - Sort: arranging all data items in a particular order. ترتيب
  - Merge: combining the data items of two different sorted list into a single sorted list. دمج

* كيف اختار DS كويس

HOW TO CHOOSE A DATA STRUCTURE?

- The choice of a appropriate data structure depends on the following factors:
  - Based on the operations that will be performed
    - A linked list used to perform insertions and deletions.
    - An array is used to perform indexing. تخزن وتوصل لها بسهوله
  - Based on the time complexity of the operations ← الكفاءة تاخذ كم وقت
    - For example, to perform searches frequently, a binary search tree is used.
  - Based on the space complexity of the operations كم المكان اللي بياخذ
    - For example, to store a lot of data, you should use an array.
  - based on the amount of memory used ← مساحه ذاكرة كم تاخذ
    - For example, to store a lot of data in memory, you should use a linked list
      ما تحجز مساحه كبيرة بالذاكرة

DATA STRUCTURES

25

Linear Array

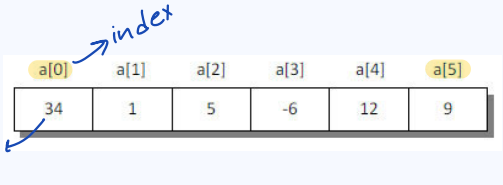
Prof. Khalil Al-Shqeerat

CS Dept. – Qassim University



REVIEW!

- An **Array** is a finite collection of similar elements stored in adjacent memory locations.
  - It contains ***n*** number of elements
    - It is referenced using an index that varies from 0 to n-1
    - An array ***arr[n]*** contains ***n*** elements
      - where **0** is the lower bound, and **n - 1** is the upper bound.
    - Example:



كل العناصر متصلة

OPERATIONS ON THE ARRAY

- There are several operations that can be performed on an array.

Operation	Description
Traversal	Processing each element in the array
Search	Finding the location of an element with a given value
Insertion	Adding a new element to an array
Deletion	Removing an element from an array
Sorting	Organizing the array elements in some order
Merging	Combining two arrays into a single array
Reversing	Reversing the elements of an array

تمرير كل عنصر لتوجيهه شيء

اضيف عنصر

اعيد ترتيب

ادمج عناصر في arr

اعكس الترتيب

ROW AND COLUMN ARRANGEMENT

كلام نظري *

- Rows and columns of a matrix are only a matter of imagination.
  - When a matrix gets stored in memory all its elements are stored linearly since computer's memory can only be viewed as consecutive units of memory locations.
- Two possible arrangements of elements in memory-Row Major Arrangement and Column Major Arrangement.
- Since the array elements are stored in adjacent memory locations, we can access any element of the array once we know the **base address** (starting address) of the array and **number of rows and columns** present in the array.

ROW MAJOR ARRANGEMENT - FORMULA

- Row Major Arrangement Formula is used to calculate the memory address of a given array index location.
  - The Location of element A[i, j] can be obtained by evaluating the expression:
    - LOC (A [i, j]) = base_address + W [N * i + j] ذي قاعدة تخليني اعرف مكان اي عنصر *
      - base_address = address of the first element in the array.
      - W = Word size, means a number of bytes occupied by each element of an Array. int = 4byte *
      - M = Number of rows in the array. X
      - N = Number of columns in the array.

COLUMN MAJOR ARRANGEMENT - FORMULA

- Column Major Arrangement Formula is used to calculate the memory address of a given array index location.
  - The Location of element A[i, j] can be obtained by evaluating the expression:
    - LOC (A [i, j]) = base_address + W [M * j + i]
    - **Example:** to calculate the address of element **121**
      - LOC (a[3, 4]) = 502 + 4 *[3*(3) + 1]
        = 502 + 4 * [9 + 1]
        = 502 + 4 * 10
        = 502 + 40
        = 542

COMMON MATRIX OPERATIONS

- Common matrix operations are addition, multiplication and transposition.

DATA STRUCTURES

27

Linked List

Prof. Khalil Al-Shqeerat

CS Dept. – Qassim University

ARRAY LIMITATIONS

مشاكل الاري

- Arrays have a fixed dimension.
  - Once the size of an array is decided it cannot be increased or decreased during execution.
- Insertion of a new element in an array is tedious because during insertion each element after the specified position has to be shifted one position to the right.

اضافه

- Deletion of an existing element in an array is inefficient because during deletion each element after the specified position has to be shifted one position to the left.

SOLUTION!

- Linked list overcomes all these disadvantages.
- A linked list can grow and shrink in size during its lifetime.
- No maximum size of a linked list.
- Unlike arrays, while inserting or deleting elements in a linked list shifting of existing elements is not required.

LINKED LIST DEFINITION

- Unlike array, linked list elements are not constrained to be stored in adjacent locations.
  - The order of the elements is maintained by explicit links between them.
- **Linked list** is a collection of elements called nodes, each of which stores two items of information—an element of the list and a link.
- The link part is a pointer to the next node.
  - It is shown using an arrow.
  - The NULL (N) in the last node indicates that it is the last node in the list.

OPERATIONS ON LINKED LIST

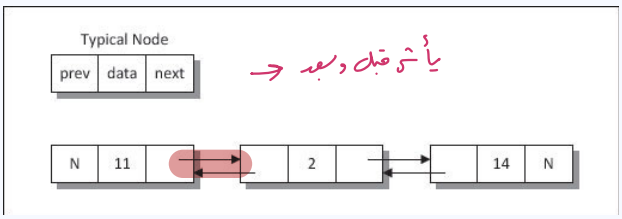
- Several operations can be performed on linked lists:
  1. building a linked list by adding new node;
     - At the beginning,
     - At the end or
     - In the middle of the linked list),
  2. Deleting a node.
  3. Display contents of all nodes.

RECURSIVE OPERATIONS ON LINKED LIST

- A function is called ‘recursive’ if a statement within the body of a function calls the same function.
- Some of the operations that are carried out on linked lists can be easily implemented using recursion.
- Recursive functions in the following code;
  - to count nodes in a linked list,
  - comparing two linked lists, and
  - cloning a linked list

DOUBLY LINKED LISTS

- So far. each node in the linked list provides information about where is the next node in the list.
  - It has no knowledge about where the previous node lies in memory.
  - To reach any node in the list we have to traverse the list right from the first node.
- To avoid this we store in each node not only the address of next node but also the address of the previous node in the linked list.



DATA STRUCTURES

13

Stacks

Prof. Khalil Al-Shqeerat

CS Dept. – Qassim University

STACK DEFINITION

- Stack is a data structure in which addition of new element or deletion of an existing element always takes place at the top of stack.
- When an item is added to a stack, the operation is called **push**, and when an item is removed from the stack the operation is called **pop**.
- Stack is also called last-in-first-out (LIFO) list.
- A stack data structure can be maintained as an array or as a linked list.

حجم ثابت

عيه ي اعدل حجمه

STACK AS A LINKED LIST

- Limitation of an array
  - its size cannot be increased or decreased once it is declared
  - Reserving either too much memory or too less memory for an array and in turn for a stack.
- This problem can be overcome if we implement a stack using a linked list.
- The pointer to the beginning of the list serves the purpose of the top of the stack

DATA STRUCTURES

22

QUEUE

Prof. Khalil Al-Shqeerat

CS Dept. – Qassim University

QUEUE DEFINITION

- Queue is a linear data structure that permits insertion of new element at one end and deletion of an element at the other end.
- The end at which the deletion of an element takes place is called **front**, and the end at which insertion of a new element takes place is called **rear**.
- The first element that gets added into the queue is the first one to get removed from the list.
  - Queue is also referred to as **first-in-first- out (FIFO)** list

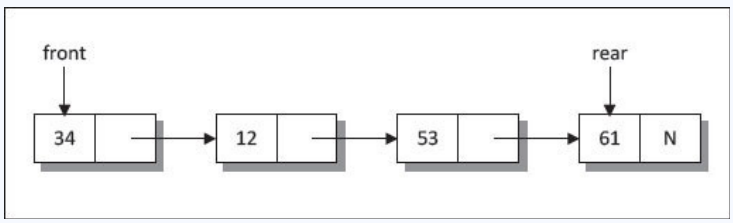
QUEUE AS AN ARRAY

- An array can store a fixed number of elements.
- Queue keeps on changing as element is removed from the front end or a new element is added at the rear end.
- Declaring an array with a maximum size would solve this problem.
- The maximum size should be large enough for a queue to expand or shrink.

* احط حجم كبير احتياط

QUEUE AS A LINKED LIST

- Linked lists do not have any restrictions on the number of elements it can hold.
- Space for the elements in a linked list is allocated dynamically.
  - It can grow as long as there is enough memory available for dynamic allocation.

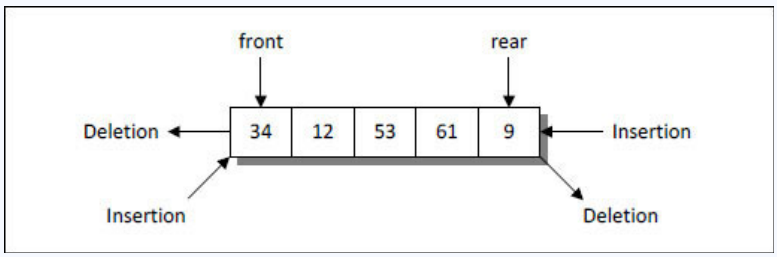


CIRCULAR QUEUE

- The queue using an array suffers from one limitation.
  - The queue is reported as full (since rear has reached the end of the array), even though in actuality there might be empty slots at the beginning of the queue
- To overcome this limitation the queue can be implemented as a circular queue.
  - When elements are added to the queue and the end of the array is reached, the next element is stored in the first slot of the array.
- In a circular queue, the queue would be reported as full only when all the slots in the array stand occupied.

DEQUE

- The word deque is a short form of double-ended queue
  - Items can be added or deleted at either the front or rear
  - No changes can be made elsewhere in the list.
- It is a generalization of both a stack and a queue.



DEQUE – CONT.

- Two variations of a deque
  - Input-restricted deque
    - It restricts the insertion of elements at one end only, but the deletion of elements can be done at both the ends of a queue
  - Output-restricted deque.
    - It restricts the deletion of elements at one end only, and allows insertion to be done at both the ends of a deque.

PRIORITY QUEUE

- A priority queue is a collection of elements where the elements are stored according to their priority levels.
  - The order in which the elements should get added or removed is decided by the priority of the element.
- Rules are applied to maintain a priority queue.

(a) The element with a higher priority is processed before any element of lower priority.

(b) If there are elements with the same priority, then the element added first in the queue would get processed

DATA STRUCTURES

46 TREES

Prof. Khalil Al-Shqeerat

CS Dept. – Qassim University

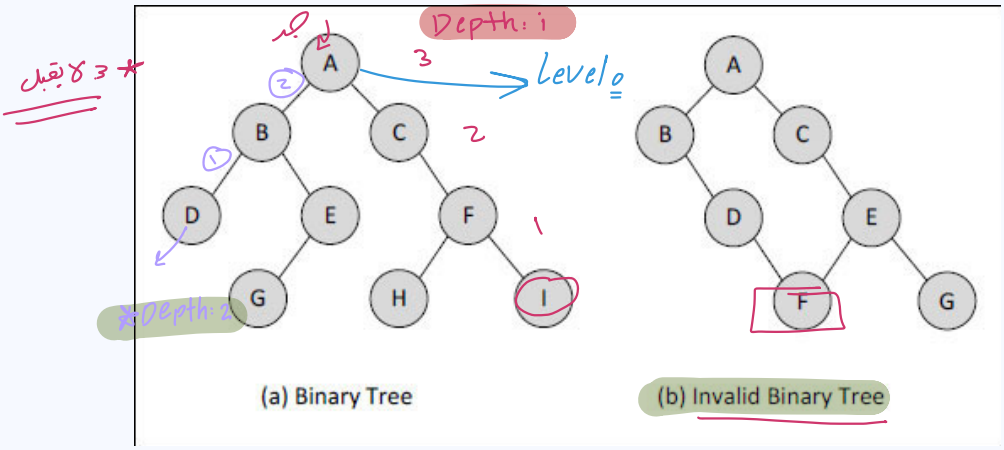
INTRODUCTION

- A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search.
- It is a collection of nodes that are connected by **links** and has a hierarchical relationship between the nodes.
- The topmost node of the tree is called the **root**, and the nodes below it are called the **child** nodes.
- Each node can have multiple child nodes, and these child nodes can also have their own child nodes, forming a **recursive structure**.

BINARY TREES

- A tree that has at the most two pointers to the other tree nodes is called **Binary**.
- A **binary tree** is a finite set of elements that is either empty or is partitioned into three disjoint sub-sets.
  - The first sub-set contains a single element called the **root** of the tree.
  - The other two sub-sets are themselves binary trees, called the **left** and **right** sub-trees of the original tree.
    - A left or right sub-tree can be empty.
- Each element of a binary tree is called a **node** of the tree.

BINARY TREE

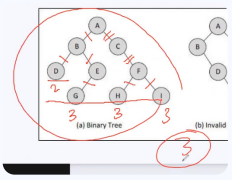


SOME TERMINOLOGY

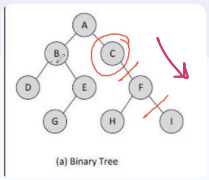
- **Parent**: If A is the root of a binary tree and B is the root of its left or right sub-tree then, A is parent of B and B is left or right child of A
- An **ancestor** of a node is any other node on the path from the node to the root.
  - An ancestor node of a node is either the parent of the node or the parent of some ancestor of the node.
- A **descendant** of a node is either a child of the node or a child of some descendant of the node (also called **successors**).

TERMINOLOGY – CONT.

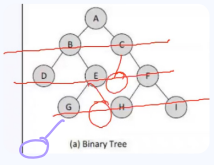
- **Climbing**, **Descending**: The root of the tree is at the top and the leaves at the bottom.
  - **Climbing** the tree when going from the leaves to the root.
  - **Descending** the tree when going from the root to the leaves.
- **Degree** of a node is the number of nodes connected to this node.
- **Depth** of a node is the maximum number of links from root to that node.
  - The depth of a binary tree is the maximum level of any leaf in the tree.
    - The length of the longest path from the root to any leaf.



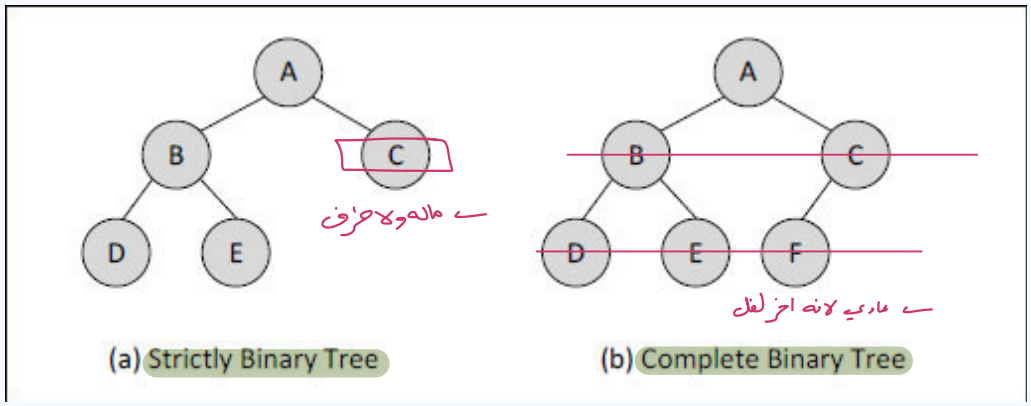
TERMINOLOGY – CONT.



- **Height** of a node is the maximum number of links from that node to leaf node.
  - Height of a binary tree is height of its root node.
- **Strictly binary tree**: If every non-leaf node in a binary tree has nonempty left and right sub-trees.
- **Complete binary tree**: the maximum number of possible nodes at all levels except the last level, and all the nodes of the last level appear as far left as possible.

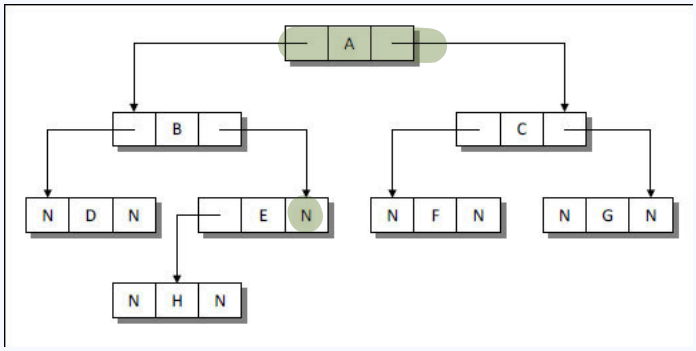


STRICTLY AND COMPLETE BINARY TREE



LINKED REPRESENTATION OF BINARY TREES

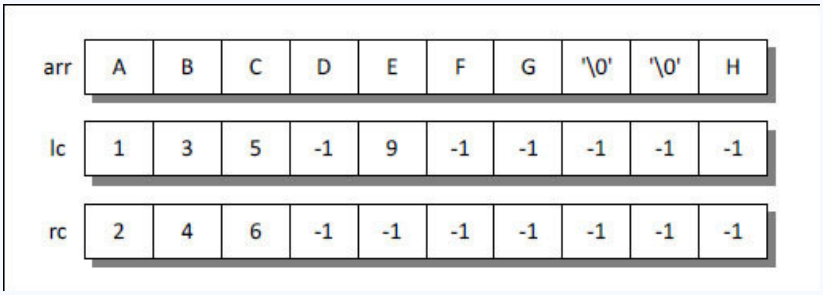
- Each node contains addresses of its left child and right child.
  - If a child is absent, the link contains a NULL value.



★ اضافی بلاری

ARRAY REPRESENTATION OF BINARY TREES

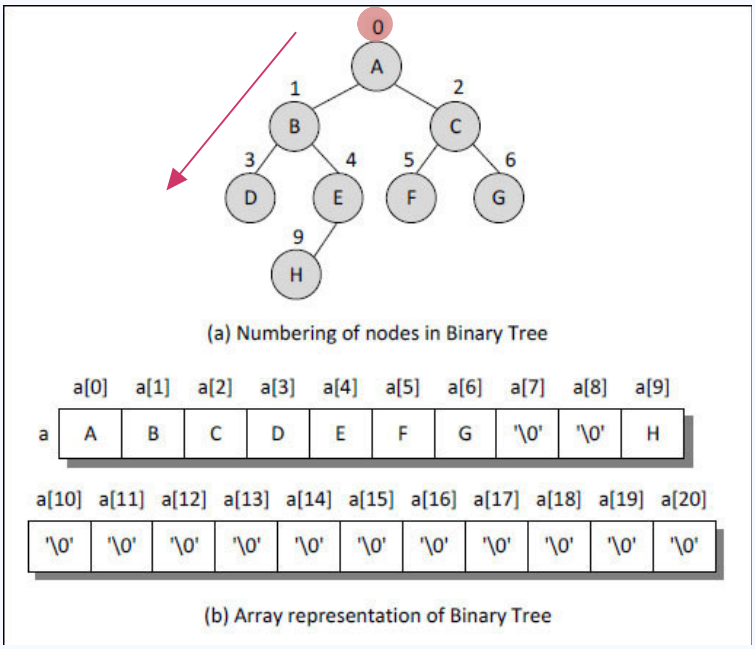
- When a binary tree is represented by arrays three separate arrays are required.
  - One array stores the data fields of the trees.
  - The other two arrays represent the left child and right child of the nodes.



arr	A	B	C	D	E	F	G	'\0'	'\0'	H
lc	1	3	5	-1	9	-1	-1	-1	-1	-1
rc	2	4	6	-1	-1	-1	-1	-1	-1	-1

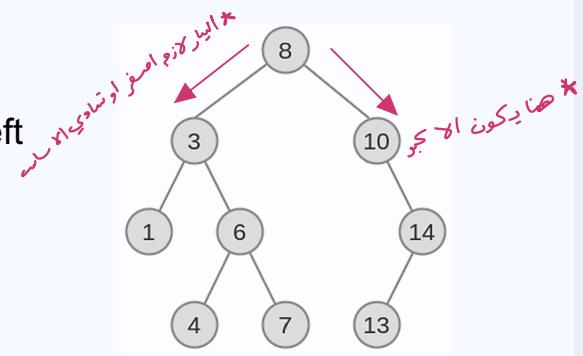
CONT.

- A binary tree can also be represented using one single array.
- Nodes are numbered from left to right level by level from top to bottom.



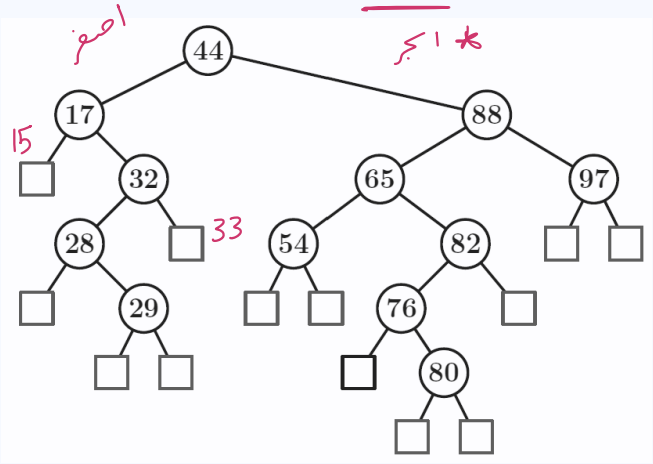
BINARY SEARCH TREES

- Binary search tree (BST) is a variant of binary tree in which the nodes are **arranged** in a particular manner.
  - The value of left node must be smaller than or equal to the parent node
  - The value of right node must be greater than or equal to the parent node.
- This rule is applied recursively to the left and right subtrees of the root.



EXERCISE..

- Complete the following tree to satisfy BST data structure.

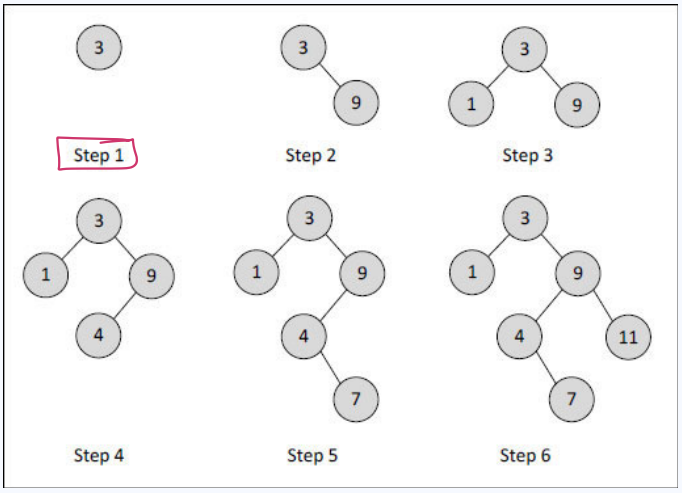


OPERATIONS ON A BINARY SEARCH TREE

- There are many operations that can be performed on binary search trees (such as Insertion, Traversal, Searching and Deletion).

- **Insertion of a Node**
  - While inserting a node in a BST the value being inserted is compared with the root node.
    - A left sub-tree is taken if the value is smaller than the root node.
    - A right sub-tree is taken If the value is greater than or equal to the node.
  - this operation is repeated at each level till a node is found whose left or right sub-tree is empty.
  - Finally, the new node is appropriately made the left or right child.

EXAMPLE



OPERATIONS ON A BINARY SEARCH TREE – CONT.

Traversal of a BST

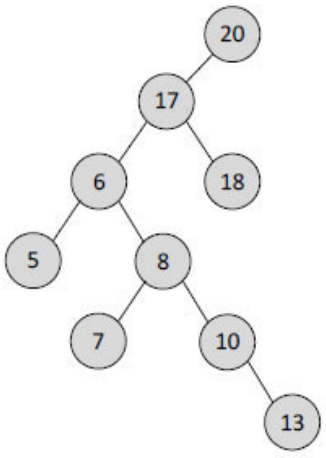
- The traversal of a BST is to visit each node in the tree exactly once.
- There are three popular methods of BST traversal: inorder traversal, preorder traversal and postorder traversal.
- Traversing a BST involves visiting the root node and traversing its left and right sub-trees.
- To traverse a non-empty BST in preorder:
  (1) Visit the root
  (2) Traverse the left sub-tree in preorder
  (3) Traverse the right sub-tree in preorder

TRAVERSAL OF A BST – CONT.

- To traverse a non-empty BST in inorder (or symmetric order):
  (1) Traverse the left sub-tree in inorder
  (2) Visit the root
  (3) Traverse the right sub-tree in inorder
- To traverse a non-empty BST in postorder:
  (1) Traverse the left sub-tree in postorder
  (2) Traverse the right sub-tree in postorder
  (3) Visit the root

* ترتيب قراءة العناصر

TRAVERSAL OF A BST – EXAMPLE



* طرق قراءة

Inorder : 5, 6, 7, 8, 10, 13, 17, 18, 20

Prorder : 20, 17, 6, 5, 8, 7, 10, 13, 18

Postorder : 5, 7, 13, 10, 8, 6, 18, 17, 20 → L R Root

OPERATIONS ON A BINARY SEARCH TREE – CONT.

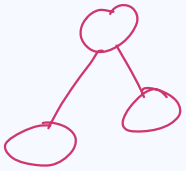
Searching of a Node

- To search any node in a binary tree, initially the value is compared with the root node.
  - If they match then the search is successful.
  - If the value is greater than the root node then searching process proceeds in the right sub-tree of the root node,
  - Otherwise, it proceeds in the left sub-tree of the root node.
- BST search operation is very efficient because while searching an element we do not need to traverse the entire tree.

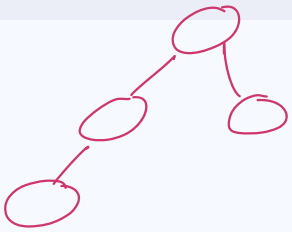
OPERATIONS ON A BINARY SEARCH TREE – CONT.

Deletion of a Node

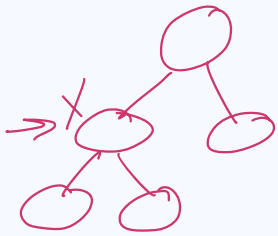
- Four cases are considered when deleting a node from a BST:
- Case 1: Node to be deleted is absent.
  - If the node is not found when traversing the BST, we only need to display the message that the node does not exist.
- Case 2: Node to be deleted has no children (leaf)
  - Simply delete the node from the tree
  - The left link or the right link of the parent of this node should be set to NULL.



DELETION OF A NODE – CONT.



- Case 3: Node to be deleted has one child
  - Replace that node with its child node
  - The pointer of the parent of the node to be deleted should points to the child of the node being deleted.
  - Remove the child node from its original position



- Case 4: Node to be deleted has two children
  - Get the inorder successor of that node.
  - Replace the node with the inorder successor.
  - Remove the inorder successor from its original position.

AVL TREES

An AVL (Adelson-Velskii and Landis) tree is a binary search tree with a balance condition.

- It controls the height of the binary search tree
- The balance condition ensures that the depth of the tree is O(logN).

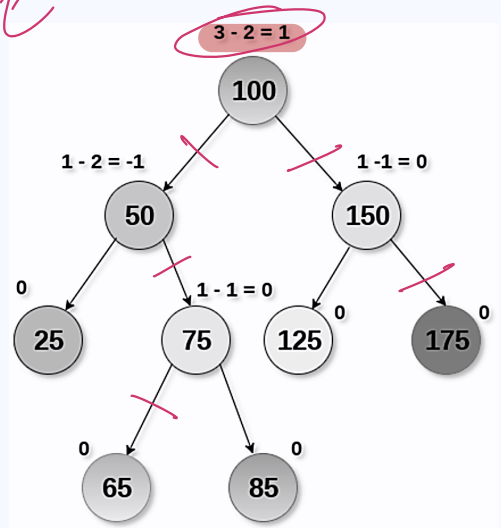
The difference between the heights of left and right subtrees, (**Balance Factor**), must be 0 or at most one (1, -1).

BALANCE FACTOR = HEIGHT(LEFT SUBTREE) – HEIGHT(RIGHT SUBTREE)

- Once the difference exceeds one, the tree automatically executes the balancing algorithm until the difference becomes one again.

BALANCE FACTOR

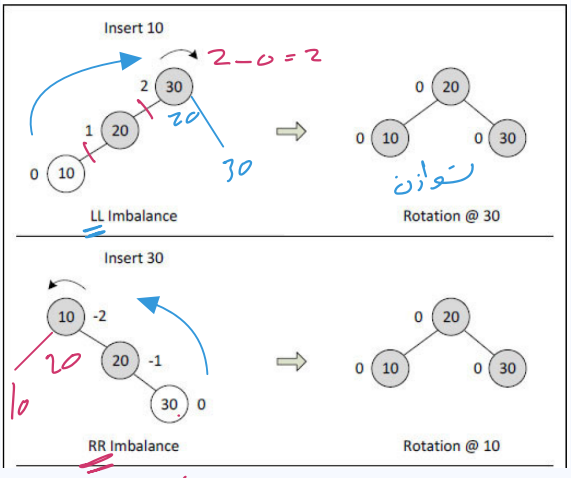
- If balance factor of any node is 1:
  - The left sub-tree is one level higher than the right sub-tree.
- If balance factor of any node is 0:
  - The left sub-tree and right sub-tree contain equal height.
- If balance factor of any node is -1:
  - The left sub-tree is one level lower than the right sub-tree.

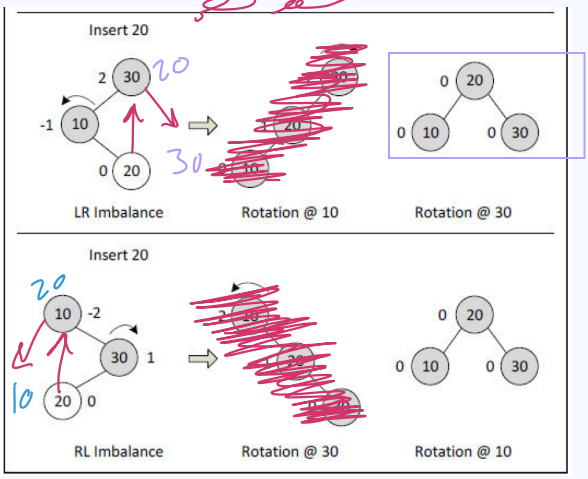


BALANCE ROTATIONS

- If balance factor of any node is other than these three values then the tree is not balanced.
- To re-balance and make it an AVL tree the nodes need to be properly adjusted.
- One of the 4 types of rotations – Left rotation, Right rotation, Left Right rotation, and Right Left rotation.
  - First two involve a 1 step process (single rotation), whereas the next two involve a 2 step process (double rotations).
- If 2 nodes are unbalanced then we need to rotate about the first **ancestor** that caused imbalance.

AVL ROTATIONS



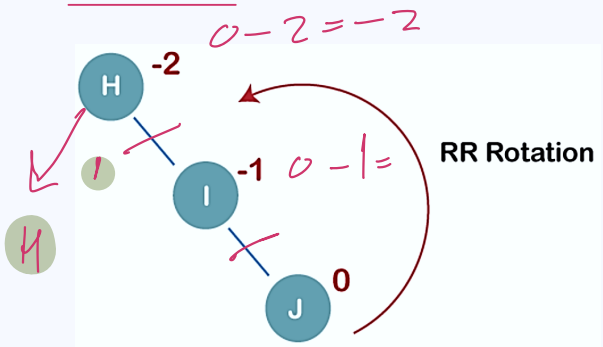


EXAMPLE

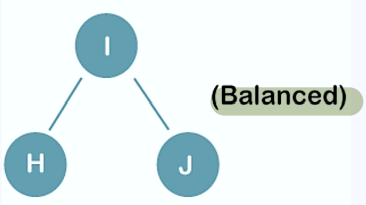
Construct an AVL tree having the following elements:

H, I, J, B, A, E, C, F, D, G, K, L

1. Insert H, I, J

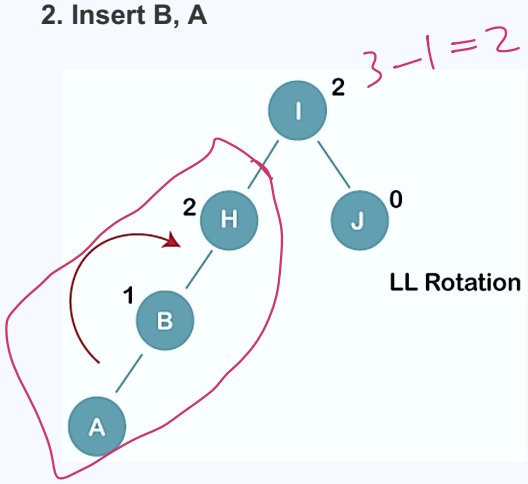


The resultant balance tree is:

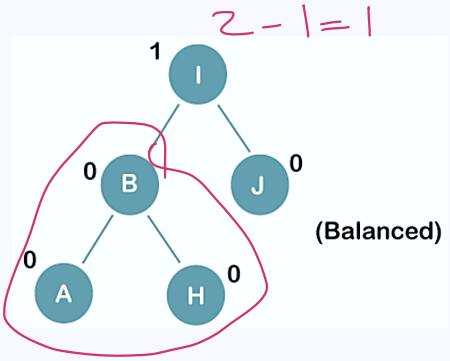


EXAMPLE – CONT.

2. Insert B, A

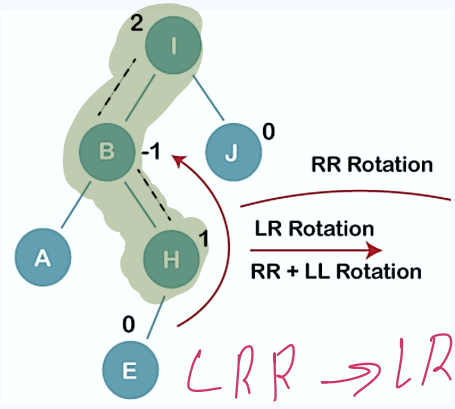


The resultant balance tree is:

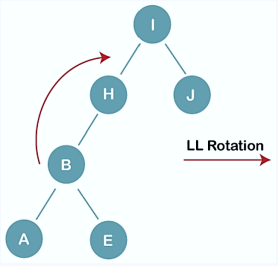


EXAMPLE – CONT.

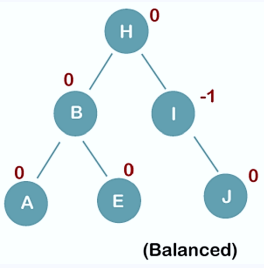
3. Insert E



The resultant tree after RR rotation is:

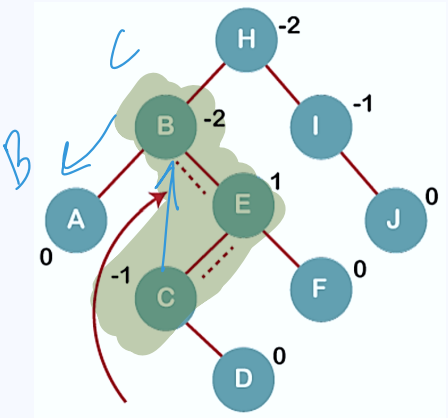


The resultant balanced tree after LL rotation is:

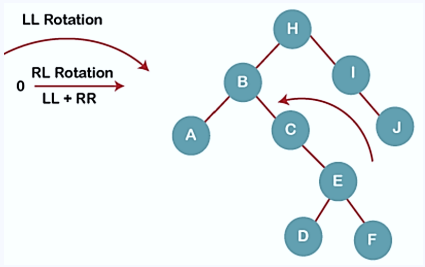


EXAMPLE – CONT.

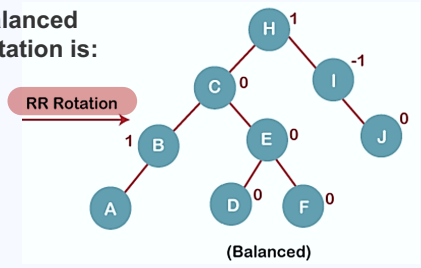
4. Insert C, F, D



The resultant tree after LL rotation is:

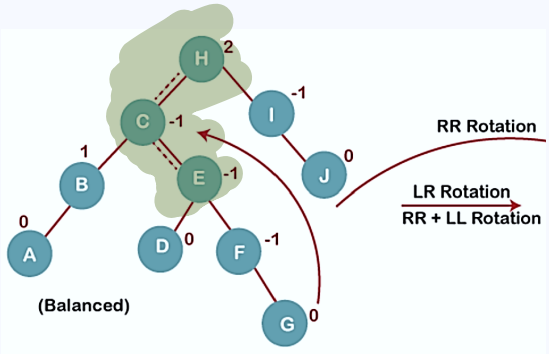


The resultant balanced tree after RR rotation is:

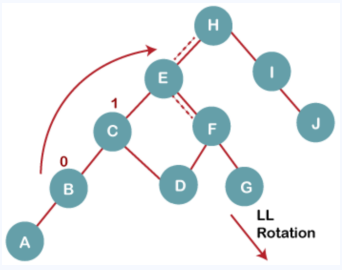


EXAMPLE – CONT.

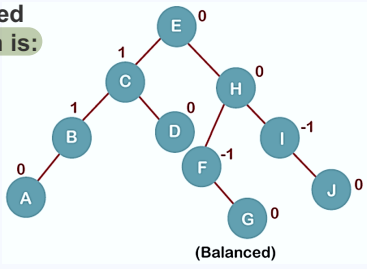
5. Insert G



The resultant tree after RR rotation is:

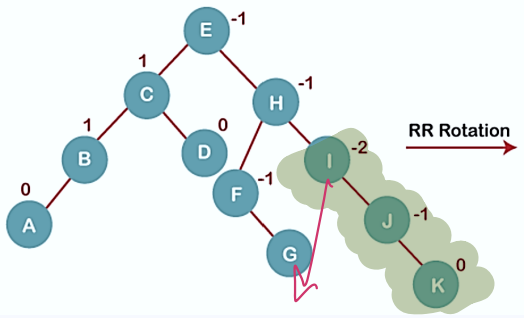


The resultant balanced tree after LL rotation is:

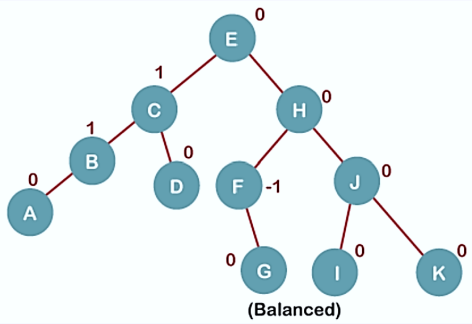


EXAMPLE – CONT.

6. Insert K

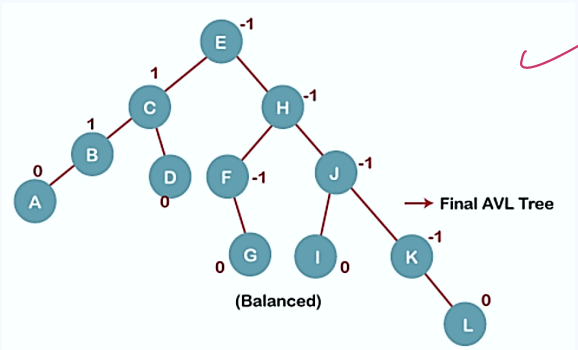


The resultant balanced tree after RR rotation is:



EXAMPLE – CONT.

7. Insert L



EXERCISE

Construct an AVL tree having the following elements:

15, 20, 24, 10, 13, 7, 30, 36, 25