DATA STRUCTURE ANSWERS

- By Dipesh Adelkar

1.1 Perform Binary Search In Array

- Binary search can be applied when the array is sorted
- Binary search works like dictionary
- To use binary search, we need to find the index of middle element
- Middle index = (ub + lb)/2

10	20	30	40	50
1	2	3	4	5

- Suppose we have this array (S) and we have to find 40
- lb = 1 , ub = 5
- middle index = (5 + 1)/2 = 3
- S[middle index] = 30 < 40
- So, we will set lb as 4 and ub as 5, Middle index = (5 + 4)/2 = 4
- S[middle index] = 40 = Desired element, therefore we will stop search

1. **Set start = lb, end = ub**
2. **Repeat steps 3 and 5 while start <= end**
3. **Set middle = (start + end)/2**
4. **If S[middle] = data then**
   Print “element found”
   Exit
5. **If S[middle] > data then set**
   Start = middle - 1
6. **Else**
   Start = middle + 1

   End loop
7. **Exit**

1.2 What is data structure?
explain following data types (linear/non-linear, static/dynamic, homo/non-homo)

- A data structure is a specialized format for organizing, processing, retrieving and storing data

- Elements in a linear structure forms a linear sequence
- Example of linear data structure : array, linked list, queue
- Elements in non linear data structure do not form a linear sequence
- E.g., tree and graph

- Homogeneous refers to elements of same data type e.g., array
- Non homogeneous refers to elements of different data type e.g., linked list
- Other data structures such as stack, queue and tree can be homogeneous or non-homogeneous depending on weather they are implemented using array or using linked list

- Static data structure refers to storing the data elements in a consecutive memory location
- e.g., array
- dynamic data structure refers to the data elements in non-consecutive memory location
- e.g., linked list

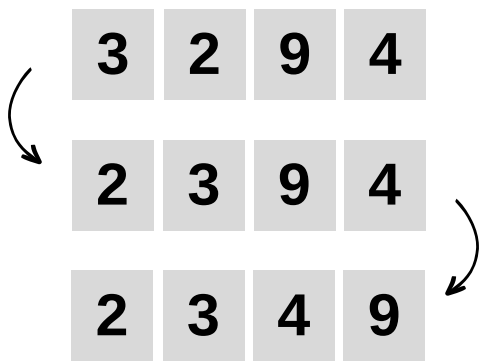
1.4 Explain Bubble Sorting of Array

- Sorting of array refers to arrange the elements of the array in some logical order
- this order can be in increasing or decreasing

S	10	20	30	40	50
	1	2	3	4	5

- Here S[1] < S[2] < S[3] < S[4] < S[5]

1. **Repeat For p = 1 to n - 1**
2. **For i = 1 to n - p**
3. **If S[i] > S[i + 1] Then**
   Exchange S[i] with S[i + 1]
   [End If]
   [End Loop]
   [End Loop]
4. **Exit**



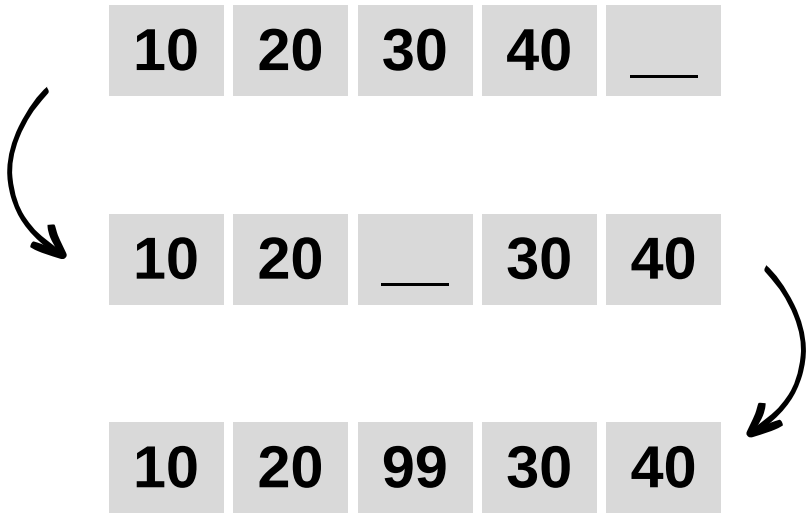
- Here a loop is executed for n - 1 times
- we have used a nested loop which is executed for certain steps
- later we have compared the two elements and exchanged if required

1.5 Perform Insertion In Array

- Insertion is to add new element in the array
- The new element can be added anywhere in the array
- Only possible if there is space for element in array
- Algorithm : inserting ‘New’ element at k where size of array ‘S’ is n

1. **while i = n to k**
   Set S[i+1] = S[i]
   Set i = i - 1
   [end loop]
2. **Set S[k] = New**
3. **Set n = n + 1**
4. **Exit**

- In this algorithm we are moving all the elements from k to n to one position at the right.
- When all the elements are moved one position to the right, we insert new element at kth position
- And then increase the size of array by 1
- Suppose we have to place 10 at the place of 40

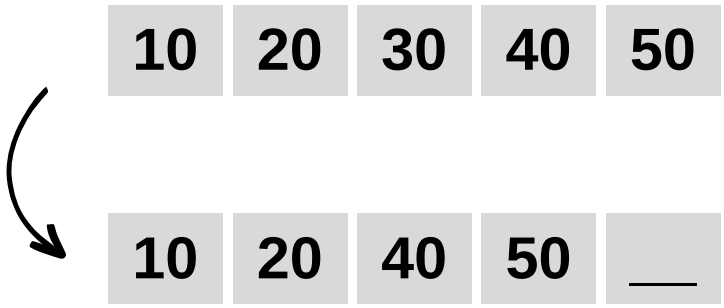


Perform Deletion In Array

- Deletion is to delete element in the array
- Algorithm : Deleting element at k where size of array ‘S’ is n

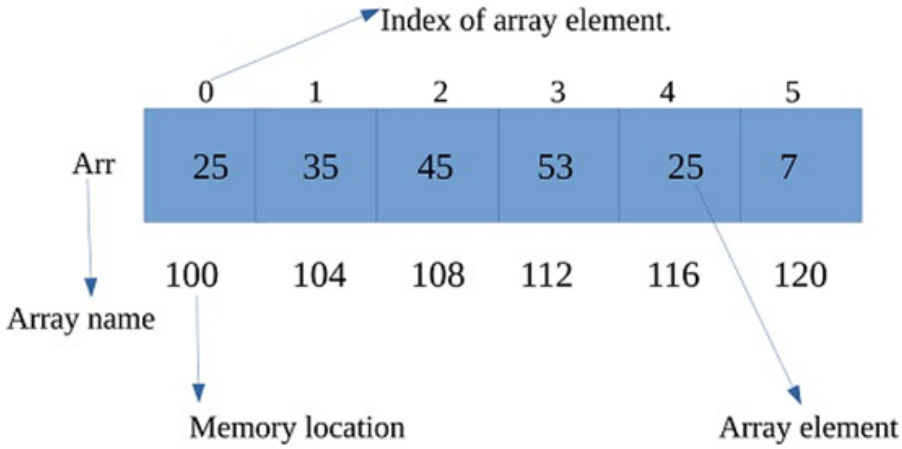
1. **While 2 and 3 for i = k to n**
   Set S[i] = S[i + 1]
   Set i = i + 1
   [end loop]
2. **Set n = n - 1**
3. **Exit**

- In this algorithm we are moving all the elements from k to n to one position at the left
- And then decrease the size of array by 1
- Suppose we have to remove 40



1.6 Explain Memory Representation of array

- The elements in the array are stored in consecutive memory location
- Let’s consider an array with Base(S)

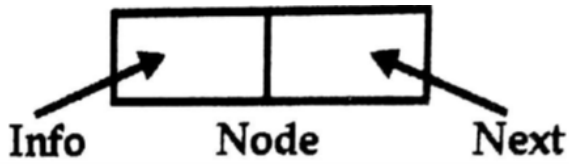


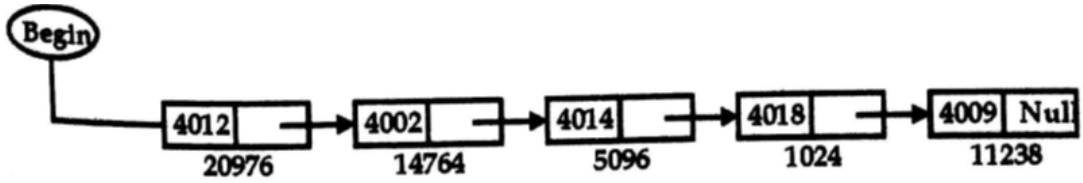
- Address of any other element of an array S can be calculated as
- Loc(Sk) = Base(S) + w(k – lb)
- Sk is Kth element of array and Loc(Sk) is location of Sk
- Base(S) is the base address of the array
- w is the size of data type of the array element
- (k - lb) is the distance of k from lower index(first index)

Example
Here Base(S) = 100 , w = 4 , lb = 0
Loc(Sk) = Base(S) + w(k – lb)
Loc(S3) = 100 + 4(3 – 0) ….finding memory location of elements at index 3
Loc(S3) = 100 + 4 x 3
Loc(S3) = 100 + 12
Loc(S3) = 112
Hence element at index 3 is stored at location 112 in memory

2.1 Explain structure of Linked List

- Linked list is the linear collection of elements at nonconsecutive memory location
- In linked list the element/data is stored in a node
- Node has two partitions one to store data and another one which stores address/location of the next node
- Last node of the linked list has Null value as the next which defines end of linked list





- The 1st node is held by special pointer known as begin/Start
- 1st node points to the address of 2nd node i.e., 14764
- 2nd node points to the address of 3rd node i.e., 5096
- This happens till the last node which points to Null value

2.2 Traversing of Single linked List and find largest element in the list

- It refers to visiting every element in linked list
- Traversing an linked list

1. **If Begin = Null Then**
   Print “Linked list is empty”
   Exit
   [End If]
2. **Set Pointer = Begin**
3. **Set max = 0**
4. **Repeat while Pointer != Null**
   if max < Pointer -> Info
   max = Pointer -> Info
   assign Pointer = Pointer -> Next
   [End Loop]
5. **Print : max**
6. **Exit**

- Here we check if the linked list is empty or not
- if the list is not empty we execute step 2 , Pointer is assigned to the first node which carries the address of the node
- Info part of the current element gets compared with the max value and if the max is less than info, info value is stored in max and then the pointer is assigned to the address of next node
- once pointer encounters Null value the loop ends

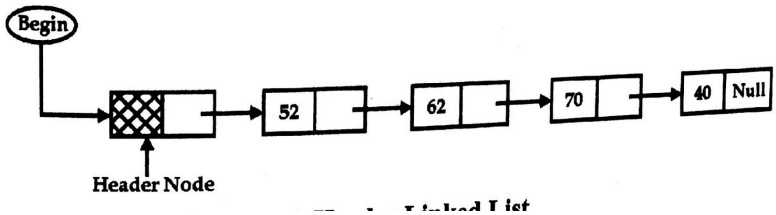
2.5 Algorithm to find the position of the given element of data in two way linked list by traversing it from end to beginning

1. If End = Null Then
   Print “Linked List is Empty”
   Exit
   [End If]
2. Set Pointer = End
3. Repeat while Pointer != Null
   If Pointer -> Info = Data
   Print “Data found at position”: Pointer
   Exit
   Set Pointer = Pointer -> Pre
   [End Loop]
2. Exit

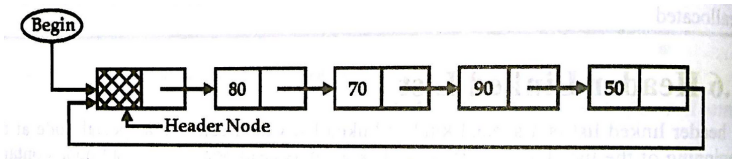
2.6 Short note on Header linked list

- Header Linked list is a special kind of linked list which contains a special node at the beginning of the lit
- This Special node is called as a head node
- This Head node contains some information about regarding the Linked List
- e.g. Number of nodes
- Header Linked List can be categorized into 4 parts
    - Grounded Header Linked List
    - Circular Header Linked List
    - Two way Header Linked List
    - Circular Two way Header Linked List

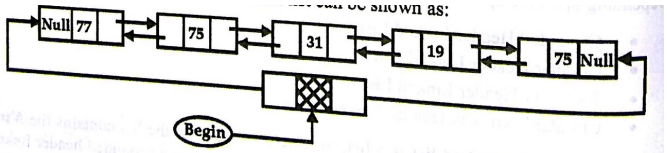
- Grounded Linked List - the last node contains Null in its Next Pointer



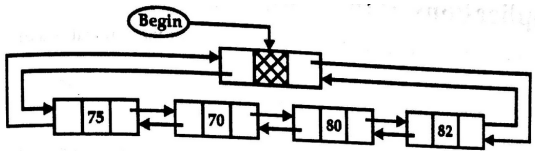
- Circular Linked List - The last node points back to the Header node



- Two Way Header Linked List - A Header node can also be Inserted in a Two way Linked List



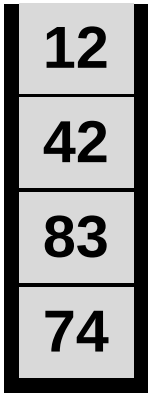
- Two Way Circular Header Linked List - The Last node of the Two way Linked list points back to the Header node



3.1 What is Stack ?algorithm to push in Linked List

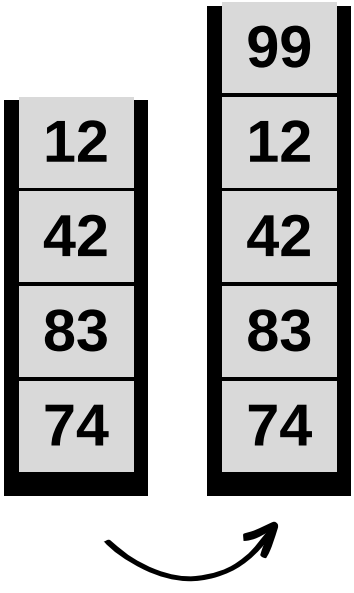
- Stack is also called as ‘LIFO’ (Last In First Out) It means that the last item added to the stack will be the first item to be removed from the stack
- In Stack Insertion and Deletion takes place from only one end
- In Stack insertion operation is called as “PUSH”
- and deletion operation is called as “POP”

- PUSH - Push operation refers to insertion of a new element into the stack
- we can perform PUSH operation only if the stack is not full i.e. the stack should have sufficient space for new element
- If the stack is already full and when we try to insert new element it is called as “Stack Overflow” condition



- POP - Pop operation refers to removal of an element from the top of the stack
- We can perform POP operation only if the stack is not empty
- We need to make sure that the Stack is not empty before applying the POP operation
- When we try to perform POP operation while the stack is empty it is called as “Stack Underflow” condition

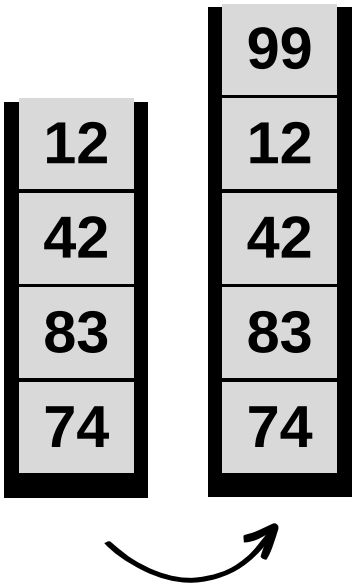
1. **If Free = Null then**
   print “Stack is already full”
   Exit
   [End if]
2. **Set New = Free and Free = Free -> Next**
3. **Set New -> Info = Data**
   and New -> Next = Top
4. **Set Top = New**
5. **Exit**



3.3 Algorithm for Push and Pop operation in Stack

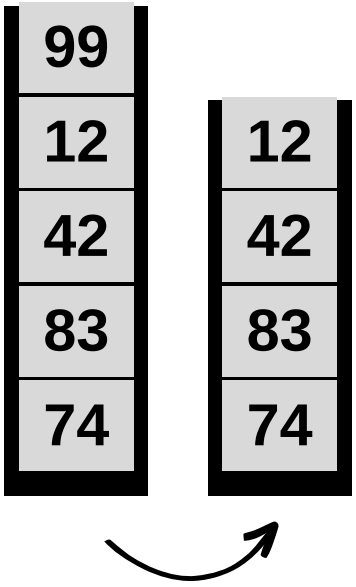
PUSH operation

1. If Top = Max then
   print “Stack is already full”
   Exit
   [End if]
2. Set Top = Top + 1
3. Set S[Top] = Data
4. Exit



POP operation

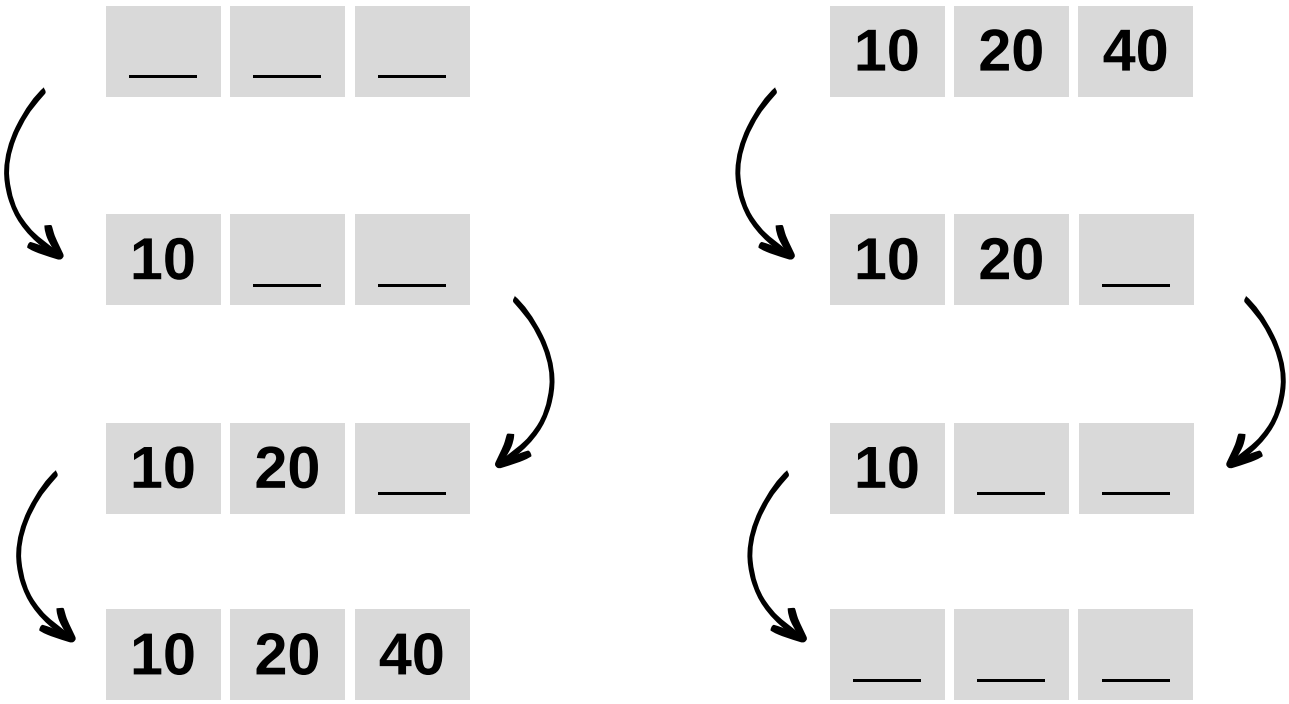
1. If Top = Null then
   print “Stack is Empty”
   Exit
   [End if]
2. Set Data = S[Top]
3. Set Top = Top - 1
4. Exit



3.4 How Insertion and Deletion take place in Queue With Example

- Queue is a Linear collection of elements in which insertion takes place at one end known as “rear” and deletion takes place at another end known as the “front” of the queue
- The elements of the queue are processed in the same order as they were added into the queue, that’s why queue is also known as FIFO (First in First out)
- Two types of operations are performed in a queue
    - Insertion (rear)
    - Deletion (front)

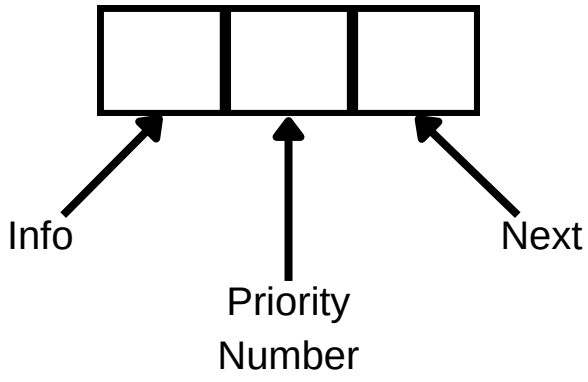
- If the queue is already full and when we try to insert new element in the queue this condition can be called as “Overflow” condition
- If the queue is empty and if we try to delete an element from the queue this can be called as “Underflow” condition



3.6 Explain Priority Queue using Linked List with Example

- Priority Queue is the kind of queue data structure in which insertion and deletion operation are performed according to some special rule rather than FIFO
- there are three ways to represent priority queue
    - Priority queue using linked list
    - priority queue using multiple queues
    - priority queue using heap structure

- Priority Queue using linked list :
    - in this priority queue the node of the linked list is divided into three parts
    - Info : this part contains the data element
    - Priority : this part hold the priority number of the element
    - Next : this part holds the address of the next node



4.1 Sort the Following using Selection Sort : 22,35,17,08,13,44,05,28

- Selection sorting algorithm is also called as in-place comparison sort
- in-place means that this algorithm does not take extra space for sorting the elements of the array
- in this algorithm the smallest element of the array is replace with the element at the first position
- Then the second smallest element is swapped with the element at 2nd position
- this process continues until the entire array is sorted

22	35	17	08	13	44	05	28

05	35	17	08	13	44	22	28

05	08	17	35	13	44	22	28

05	08	13	35	17	44	22	28

05	08	13	17	35	44	22	28

05	08	13	17	22	44	35	28

05	08	13	17	22	44	35	28

05	08	13	17	22	28	35	44

- 1st step : 5 is the smallest element so we will exchange its position with the element at position 1
- 2nd Step : 08 is the second smallest element so we will exchange its position with element at position 2
- We will continue this until the entire array is sorted

4.2 Algorithm of selection sort technique

- Selection sorting algorithm is also called as in-place comparison sort
- in-place means that this algorithm does not take extra space for sorting the elements of the array
- in this algorithm the smallest element of the array is replace with the element at the first position
- Then the second smallest element is swapped with the element at 2nd position
- this process continues until the entire array is sorted

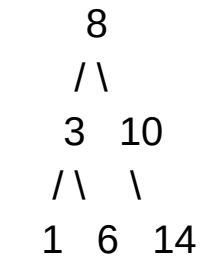
```
for i = 0 to i < n-1
    set min = 1
    for j = 1 to  j <n
        if ( a[j] < min)
            min = a[j];
        if ( min! = 1 )
            swap (a[i] , min)
Exit
```

- The code uses two nested loops to compare elements in an array 'a' and perform swapping operations.
- The outer loop iterates from i = 0 to i < n-1, where 'n' is the size of the array.
- Inside the outer loop, 'min' is set to 1
- The inner loop iterates from j = 1 to j < n.
- Inside the inner loop, it checks if the element at index 'j' is less than 'min'. If true, it sets min = a[j] and it swaps min and a[i]

4.5 Explain Binary search Tree with Example

- A binary search tree (BST) is a data structure used for organizing and storing a collection of elements, such as numbers, in a way that allows for efficient insertion, deletion, and searching of elements. The key characteristic of a binary search tree is that each node in the tree has at most two child nodes: a left child and a right child
- All nodes in its left subtree have values less than the node's value.
- All nodes in its right subtree have values greater than the node's value.
- This property ensures that elements are organized in a sorted manner within the tree, making it easy to search for specific values.

Example :



- Now, we have a binary search tree that satisfies the binary search tree property, making it easy to perform operations like searching for elements.
- For example, searching for the value 6 would involve comparing 6 to the root (8), then going to the left child (3), and finally to the right child (6), ultimately finding the desired value.
- This property allows for efficient searching and other operations like insertion and deletion.

5.1 What is Hashing? Explain Mid Square and Division remainder method with Example

- Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function
- Hashing is used to overcome collision of elements at same address

- Mid Square Method
    - in this method the key value is squared and the digits are taken from the middle of the squared value
    - This middle 3 digits values of the square of the key is the Relative address

Key value	Square of Key vale	Relative address
5010	25100100	100
5016	25160256	160
5301	28100601	100
5400	29160000	160

- Division Remainder Method
    - in this method the the addresses are n=mapped by dividing the key values by the number of available addresses and the remainder is taken as the relative address

Key value	Remainder after dividing the key by 997
1098	101
1120	123
1185	118
1230	233

5.2 Explain the Following collision resolution Techinique with Example (Linear Probing) : Hash table of size 10 Keys are 33,101,99,83,93

- Linear probing is a collision resolution technique in hash tables where, upon a collision, it searches for the next available slot by incrementing linearly until an empty slot is found.

$$Hash(K, p) = [\ H(K) + p\ ] \bmod m$$

- p is probe number which can be 0,1,2,..., m-1
- n is the size of the hash table
- Keys : 33,101,99,83,93

K = 33

Hash(K, p) = [H(K) + p] mod m

Hash(K, p) = [33 mod 10 + 0] mod 10

Hash(K, p) = 3 mod 10

Hash(K, p) = 3

K = 101

Hash(K, p) = [H(K) + p] mod m

Hash(K, p) = [101 mod 10 + 0] mod 10

Hash(K, p) = 1 mod 10

Hash(K, p) = 1

K = 99

Hash(K, p) = [H(K) + p] mod m

Hash(K, p) = [99 mod 10 + 0] mod 10

Hash(K, p) = 9 mod 10

Hash(K, p) = 9

0	___
1	101
2	___
3	33
4	___
5	___
6	___
7	___
8	___
9	99

K = 83

Hash(K, p) = [H(K) + p] mod m

Hash(K, p) = [83 mod 10 + 0] mod 10

Hash(K, p) = 3 mod 10

Hash(K, p) = 3

since 3 is occupied by 33 we will set p = 1

K = 83

Hash(K, p) = [H(K) + p] mod m

Hash(K, p) = [83 mod 10 + 1] mod 10

Hash(K, p) = 4 mod 10

Hash(K, p) = 4

K = 93

Hash(K, p) = [H(K) + p] mod m

Hash(K, p) = [93 mod 10 + 0] mod 10

Hash(K, p) = 3 mod 10

Hash(K, p) = 3

since 3 is occupied by 33 we will set p = 1

which will give 4 and 4 is also occupied so

set p = 2

K = 93

Hash(K, p) = [H(K) + p] mod m

Hash(K, p) = [93 mod 10 + 2] mod 10

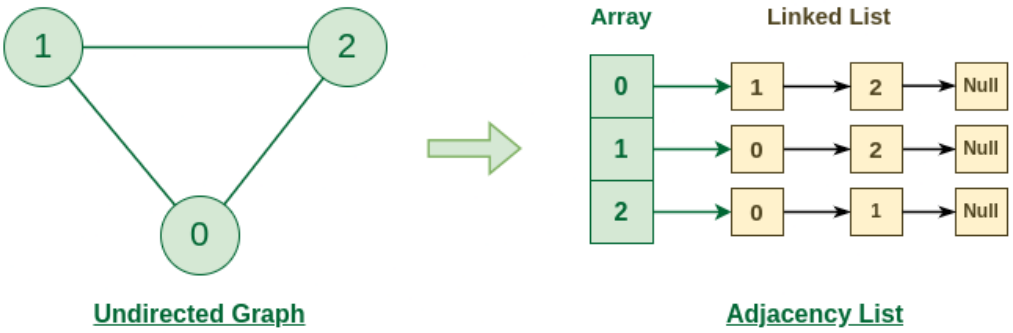
Hash(K, p) = 5 mod 10

Hash(K, p) = 5

0	___
1	101
2	___
3	33
4	83
5	93
6	___
7	___
8	___
9	99

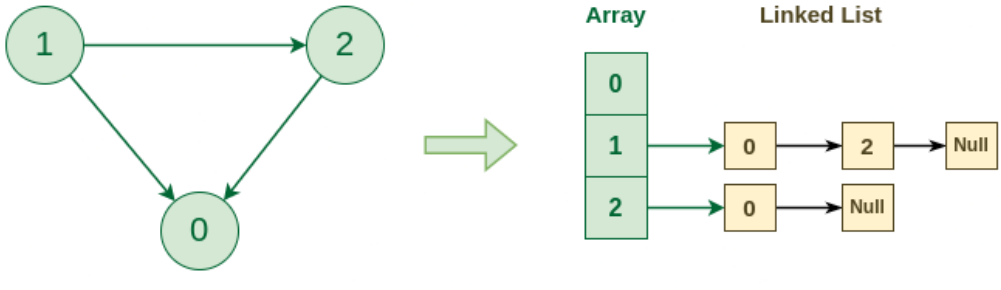
5.3 Explain Adjency List with Example (Linked list Representation)

The below undirected graph has 3 vertices. So, an array of list will be created of size 3, where each indices represent the vertices. Now, vertex 0 has two neighbours (i.e, 1 and 2). So, insert vertex 1 and 2 at indices 0 of array. Similarly, For vertex 1, it has two neighbour (i.e, 2 and 1) So, insert vertices 2 and 1 at indices 1 of array. Similarly, for vertex 2, insert its neighbours in array of list.



Undirected Graph Adjacency List

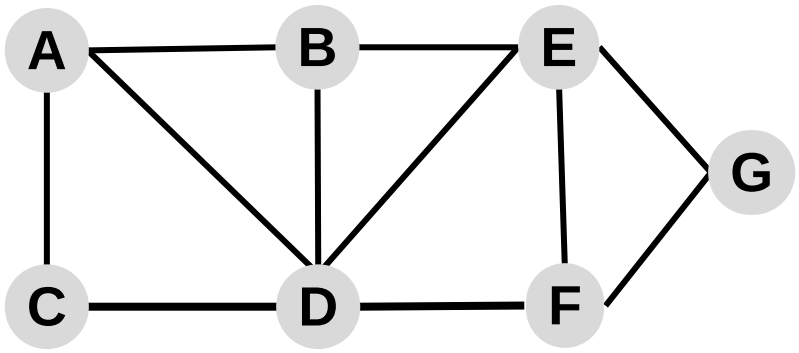
The below directed graph has 3 vertices. So, an array of list will be created of size 3, where each indices represent the vertices. Now, vertex 0 has no neighbours. For vertex 1, it has two neighbour (i.e, 0 and 2) So, insert vertices 0 and 2 at indices 1 of array. Similarly, for vertex 2, insert its neighbours in array of list.



Directed Graph Adjacency List

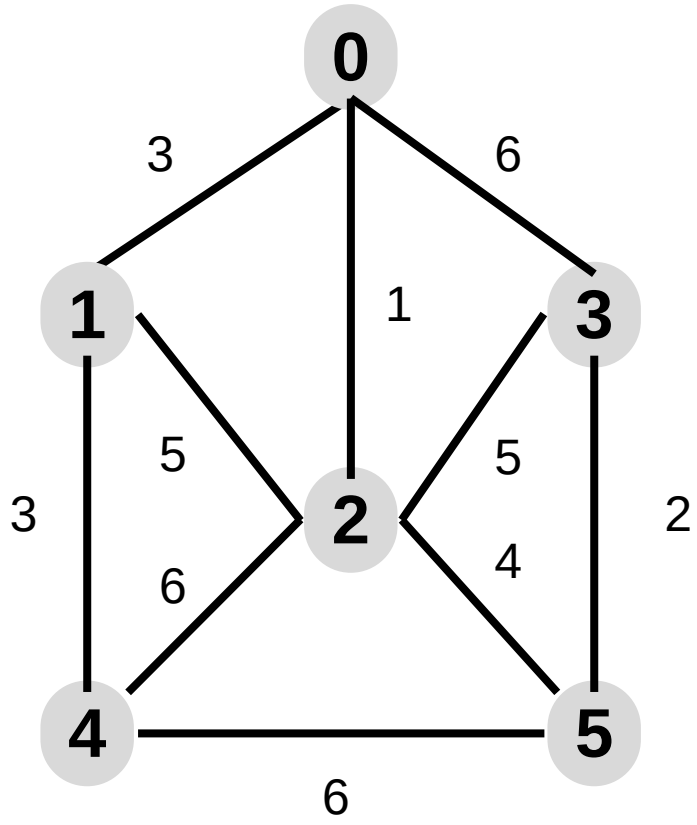
5.5 What is Adjency Matrix? Generate Adjency Matrix of the undirected Graph

- Undirected : Initially, the entire Matrix is initialized to 0. If there is an edge from source to destination, we insert 1 to both cases
- Directed : Initially, the entire Matrix is initialized to 0. If there is an edge from source to destination, we insert 1 for that particular destination

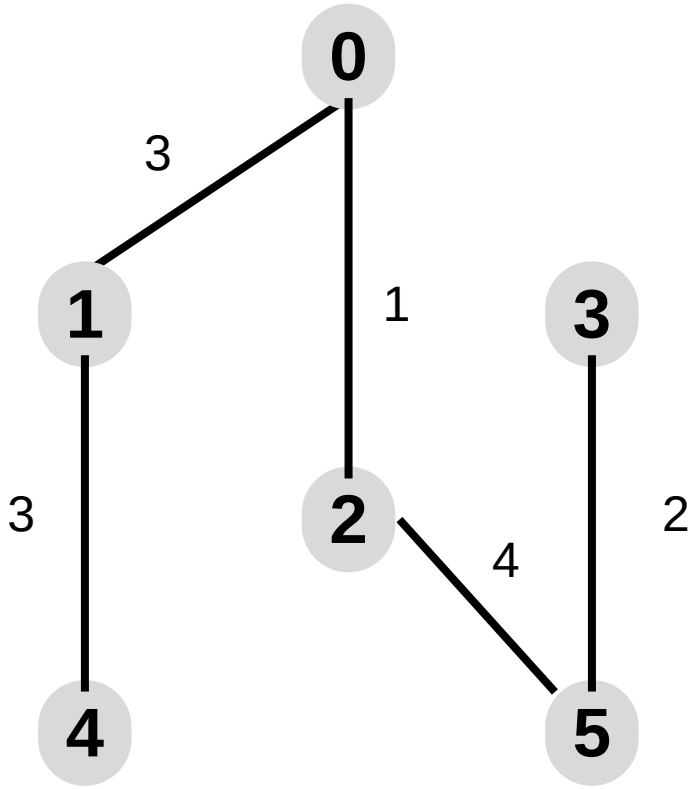


	a	b	c	d	e	f	g
a	0	1	1	1	0	0	0
b	1	0	0	1	1	0	0
c	1	0	0	1	0	0	0
d	1	1	1	0	1	1	0
e	0	1	0	1	0	1	1
f	0	0	0	1	1	0	1
g	0	0	0	0	1	1	0

5.6 Find the minimum spanning tree for the graph using Prims Algorithm



Answer :



Distance = 1 + 3 + 3 + 4 + 2 = 13