Date 7/12/21
Page 1

Chotai Shagunam Maheshbhai

MG21040

MCA-I (GTA) (2021-22)

PS01CMCA55 : Computer Fundamentals

Submitted to: Dr. Priti S. Sajja

Q.1) Define data and data Structure. Also discuss uses and advantages of using the data structure.

→ Data is raw observation and/or value.

- The data structure is defined as a way of organizing data in such a way so that data can be used efficiently, in terms of space and time.

→ Data structures are needful or useful for:

A) Searching large amounts of data:
- For Retrieval of data, from the large amount of data, data must be stored efficiently.

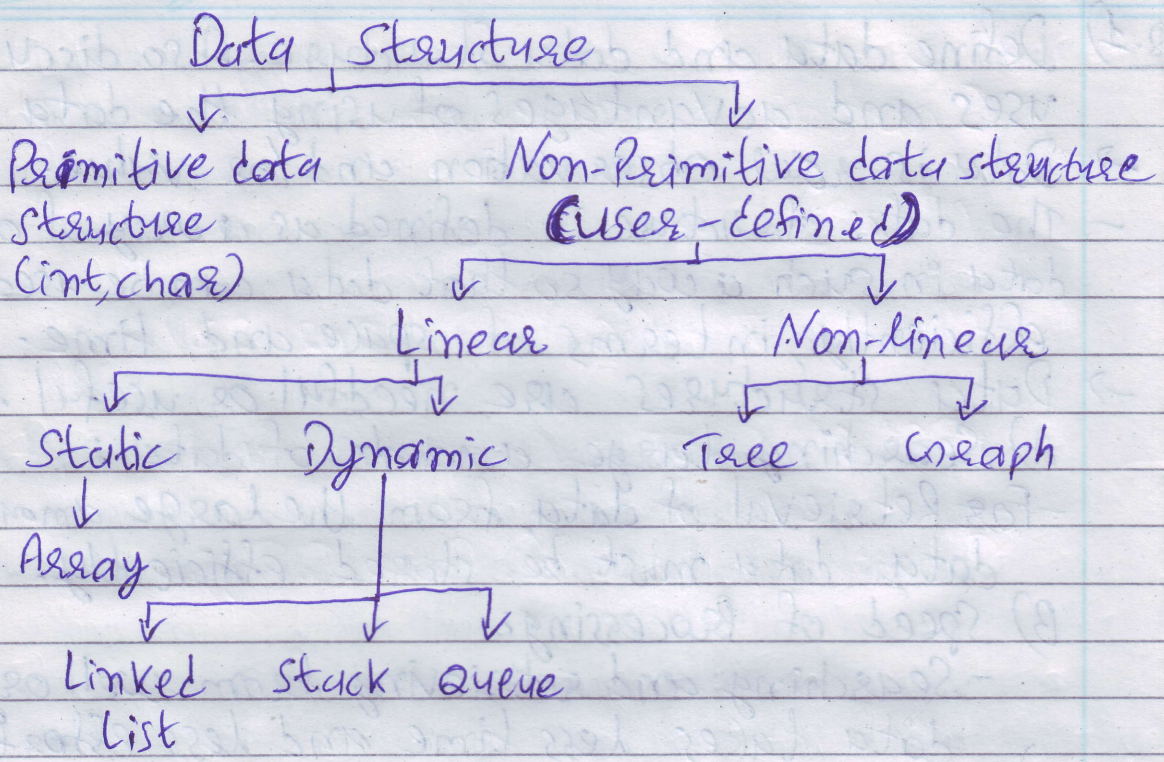
B) Speed of Processing:
- Searching and retrieving from well organized data takes less time and less efforts.

C) Concurrent requests:-
- Many and simultaneous requests can be easily handled due to data structure.

→ Advantages of a data structure:
- Correctness: Using data structure, we can store and manage consistant data.
- Time complexity: Execution time of the operations on data structure is as small/~~little~~ as possible.
- Space complexity: Memory usage of a data structure operation should be as little as possible.

Q.2) Draw a diagram of various types of data structure and explain each type in one line.

→



→ 1) Primitive data structures are basic structures and are implicitly operated upon by machine instructions. They have different representations on different computers. Integers, floats, char and pointers are examples of primitive data structure.

2) Non-Primite data structure is more sophisticated data structure, which is further divided into linear and non-linear data structure.

2.1) Linear data structures are ~~also~~ sub-divided into Array, Linked list, Stack, Queue.

↳ Array: It is group of data with same type and same size stored adjacent to each other. They are always stored in consecutive → memory locations. It can be one or more dimensional.

↳ Stack: It works on 'Last in First out' (LIFO) concept. only top element can be accessed at ~~at~~ any given time in this. For example, we can place or remove a card or plate from the top of stack only. Push and pop are the basic operations on the stack. Meanwhile, peek(), isFull() and empty() are its optional methods.

↳ Queue: It works on First In First Out (FIFO) concept. Basic operations on queue are insert and delete. Calso known as enqueue and dequeue.

↳ Linked list: It is made using ~~lin~~ nodes which are linked to each other via pointers. In each node, data part consists data or payload, meanwhile pointer part consists address of next element in the list. Simple, doubly and circular linked list is type of linked list.

2.2) Non-linear data structures are tree and graph.

↳ Tree represent the nodes connected by edges. If each node is having maximum 2 connected nodes, then it is a binary tree. Insertion, deletion and traversal are the basic operations of tree data structure.

↳ Graph: It is represented as a set of vertices (node or points) connected by edges (arcs or line). They are used to represent network of cities or telephone network or circuit network and also used in social network like facebook.

Q. 3) List and explain ~~in~~ one line various operations on data structure.

→ 1) define or create structure: It means initialization of the data structure.
2) Add an element: After initialization, insertion is the most important part of data structure.
3) Delete an element: Deletion of an element should be allowed in data structure.
4) Traverse / Display: We should see the inserted data into data structure.
5) Sort the list of element: In any data structure sorting is most important feature to manage it, when massive data are stored.
6) Search for a data element: Find out the location of the data item if it exists in data structure.
7) Merging and splitting:- Combining the data items of two sorted file into single file in the sorted form. Combining the data item splitting single data items from multiple data items.
8) Delete complete structure: Destroy the full data structure, existing data items and data ~~element~~ structure, both.

Q-4) Differentiate linear and non-linear data structure:

→ As name suggests linear data structure stores data in linear sequence and ~~it~~ thus it can

be traversed in a single run. Meanwhile, in non-linear data structure, data is not arranged in a linear sequence, hence, it is cannot be traversed in a single run and requires various algorithms for traversal.

→ As every item in linear data structure is related to ~~each other~~ its previous and next item, its implementation is easy. In non-linear data structure, every item is attached with many other items and therefore its implementation is complex.

Q.5) Define linear data structure. Explain static & dynamic linear data structure.

→ If elements of data structure are connected in linear fashion, by means of logically or in sequence memory locations, it is called linear data structure. They can be divided in two categories: Static Memory Allocation and Dynamic Memory Allocation.

→ In static type of linear data structures, memory will be allocated as a whole at the time of initialization and user or operator can perform operations on that ~~of~~ portion of memory only. Those memory locations are most likely to happen in a sequence only.
e.g. Array, in which data with same data type and same size can reside. (It can be one or more dimensional.)

→ - one-dimensional array of integer:

A[0]	A[1]	A[2]	A[3]	} Total 4 elements.
10	20	30	40	

here, A is name of an array and bracket alongside holds index (position of an element).

- Two dimensional array of integers:
  A[Row, column]

	0	1	2	3
0	A[0,0]	A[0,1]	A[0,2]	A[0,3]
1	A[1,0]	A[1,1]	A[1,2]	A[1,3]
2	A[2,0]	A[2,1]	A[2,2]	A[2,3]
3	A[3,0]	A[3,1]	A[3,2]	A[3,3]

→ In dynamic type of linear data structures, elements can be added and/or removed at any given time. Hence, memory locations are not fixed here. Stack, Queue and Linked list are its examples.

- Stack: It is first in last out data structure, from which only top element can be accessed. for eg, we can place or remove a card or plate from the top of the stack only. Basic operations are push and pop.

- Queue: It works on FIFO principle, which can be implemented by linked list, and by also using array. But using (theoretically) concept of linked list, it is easy to allocate and deallocate memory for any given elements as As insert and delete are basic operations which can be done on queue data structure.

- linked lists: It consists of nodes connected to eachother via pointers or links. Each node in this is having pointer part and data part. Each list is having head node, in which pointer points to first node of that list.

Q.6) Write a short note on array.

→ Group of data with same type and same size stored adjacent to eachother. Arrays are always stored in consecutive memory locations. It is a linear data structure with known numbers of elements. Each memory location stores a fixed length of data items. Array can be used in all the programming languages. It can be one dimensional or many dimensional.

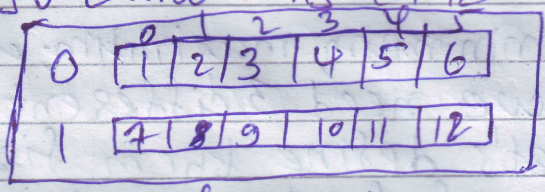
eg. one dimensional array:-

12	14	15	17	19	21	2	← data items.
0	1	2	3	4	5	6	← index

Suppose, name of the ~~ab~~ above array is S_Num, then we can access any of its data item using their respective index-number as given/shown below: S_Num[0] will return 12.

- Two dimensional array:

It is called an array of array. We can ~~dra~~ also draw this kind of array in table-form.



Suppose, name of this whole structure (array) is P_table. It is an array of one dimensional array. We can access ~~eleme~~ inner-most elements by ~~its~~ their row_index and column-index. As shown below:

P-table [Row, Column]

P-table [0, 2] gives 3., P[1,5] = 12.

→ We can define array in various programming languages, as shown below:

In Java, Long arr[] = new Long[5];
In C language, long arr[5];
In Python, arr = [None]*5
In Javascript, let arr = [];

Q.7) Define a one dimensional array called weights of 5 persons in real numbers. Calculate (i) min weight, (ii) maximum weight, and (iii) average weight of the 5 persons.

→ First, to define array, we need to give it a name.
Weight = real_numbers[5]
Above statement is written in 'pseudocode'. In various programming language, syntax and keyword varies.
Now, to perform operations on it and to store results of those operations, we need some other (proper) variables. Let's define them.
Max = 0, Total = 0, Avg = 0
But to find minimum weight, we need to initialize it with any value of weight array, otherwise it will give improper result.
Let say, Min = Weight[0] But, before writing this statement, we need to assign or take 5 values from user to to put them into our one dimensional array.

- Assume, our array looks like,

Weight	40.0	42.3	45.7	48.6	31.6
	0	1	2	3	4

- To check, Max and Min values and to perform sum operation on/from array, we need to iterate through them.

Loop through 'Weight Array:
{
To Assign new value to Total variable as shown here:
Total = Total + weight [i]
where, 'i' is current index number in loop.
We will check Max and Min value (respectively) and assign them accordingly.

```
if Max < Weight [i]:
  {
    Max = Weight [i]
  }
if Min > Weight [i] :
  {
    Min = Weight [i]
  }
```

} Loop ends.

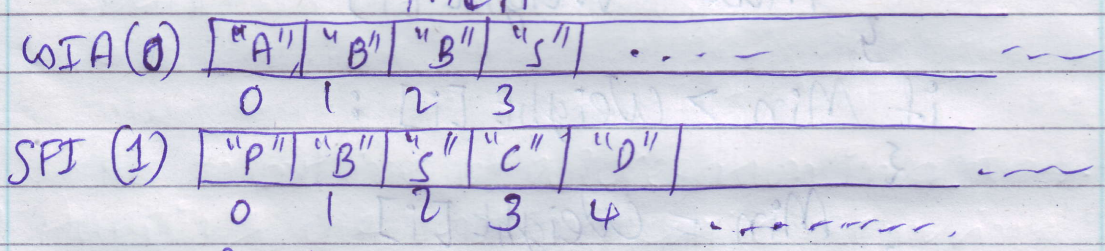
Avg = Total / size of array (which is five).

- Program returns:
  (i) Min value, <- 31.6
  (ii) Max value, <- 48.6
  (iii) Average of 5 persons. <- 41.64
- We can print them in terminal.

Q.8) Define two dimension array of your choice.

→ Two dimensional array means array within another array (or array of array). In this type of array, position of a data element can be accessed using index of inner array of outer array. It looks like rows and columns. By knowing, column's index and row's index we can access any data-element of that 2D array. Arr[Row, Col]

e.g. Name of Students in MCA, which is further divided into GIA and SFI. Here, MCA is an Array ~~of~~ made of two arrays, thus it can be called outer or parent array.



→ ~~As~~ To fetch name of student of GIA batch whose index number is 3, we will write: MCA[0, 3] ← which is "S".

Q.9) Write a short note on stack by explaining operations on the stack.

→ ~~Its name is~~ It is named after real-world stack, for e.g. deck of cards or a pile of plates, etc. It is last in first out data structure.

Only top element can be accessed at any given time. Just like, we can place or remove a card or plate from the top of the stack only. Stack can either be of a fixed size or it ~~have~~ may have a sense of dynamic resizing. It can be implemented by means of array ~~data structure~~ or pointer and linked list.

→ Basic operations on stacks includes-

1) Push() : Pushing / storing an element on the stack.

2) Pop() : Removing / accessing an element from the stack.

→ Other operations or methods are also available for the stack data structure:-

1) Peek() : to get / see the top element of the stack without disturbing the stack.

2) isFull() : check whether stack is full or not.

3) isEmpty() : check whether stack is empty or not.

→ Push operation : Putting a new data element into a stack

Step 1: check if stack is full.

Step 2: If the stack is full, produce an error and exit.

Step 3: Else, increment top's value to point next empty node / cell.

Step 4: Put data into blank cell where top is pointing currently.

Step 5: Return acknowledgement.

→ Pop operation: Taking off the top data element from the stack.
Step-1: check if stack is empty.
Step-2: If stack is empty, then produce an error and exit.
Step-3: Else, access the data element at which top is pointing.
Step-4: Decrease the value of top by 1.
Step-5: Return ~~success~~. accessed data element.

→ Applications of Stack:-
- Parsing expression (infix, prefix and postfix conversion)
- Recursion
- Flow of control and function call.
- Backtracing procedures
- Games.

Q. 10) Write an outline of pop operation in a stack.
→ Accessing the content while removing it from the stack is known as pop operation.
→ If stack is implemented using array data structure, pop operation will just decrement the top's value. But in case of linked list's stack, pop~~ui~~ operation will actually ~~deallo~~ deallocates the memory spaces.
→ This operation can be written in following steps:
Step-1: Check if the stack is empty.
Step-2: If stack is empty, then produce an error

and exit.

Step-3: else, access the data element at which top is pointing.

Step-4: Decrease the value of top by 1 ~~or~~ (and deallocate the memory space.)

Step-5: Return ~~success~~ accessed data element.

→ A simple algorithm for pop operation can be derived as follows:

```
- Begin Procedure Pop:Stack
    if Stack.isEmpty():
        return NULL
    endif
    data <- Stack[top]
    top <- top-1
    return data
  end Procedure
```

Q.11) Write an outline of push operation in a stack.

→ The process of putting a new data element onto stack is known as a push operation. Push operation involves a steps:

Step-1: Check if stack is full.

Step-2: If the stack is full, produce an error and exit.

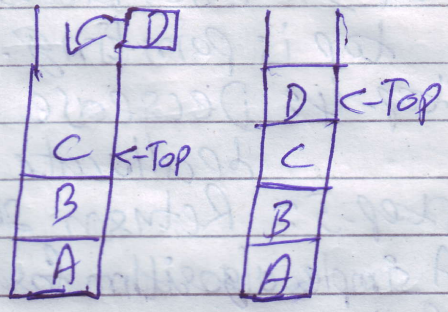
Step-3: Else increment top to point next empty space.

Step-4: Add data element to the stack location where top is pointing.

Step 5: Return success.

→ A simple algorithm for push operation can be derived as follows:

```
Begin Procedure Push: Stack, data
    if stack.isFull():
        return null
    endif
    top ← top+1
    stack[top] ← data
end procedure
```



Q. 12) Write a short note on queue by explaining operations on the queue.

→ Queue is an abstract data structure which follows First In First Out concept - i.e., The data from item stored first will be accessed first. A real world example of queue can be one way road, where the vehicle enter first, exit first. A queue can be implemented by means of Array, linked list, pointers and stacks.

→ Basic operations on queue:
Insert(): always at the end.
delete(): always from the front.

→ other operations in queue:
Peek(): get the first data element from the queue, without removing it.
isFull(): check if the queue is full
isEmpty(): check if the queue is empty.

→ Insert operation: Inserting data in the queue at the end/rear position.
step 1: check if the queue is full.

Step2: If the queue is full, produce overflow error and exit.
Step-3: If the queue isn't full, increment rear pointer to point the next empty space.
Step-4: Add data element to the queue location where the rear is pointing
step-5: Return Success.

→ Delete operation: Deleting data at the beginning or from front position.
Step 1:- Check if ~~stack~~ queue is empty.
step 2:- If the queue is empty, produce underflow error and exit.
Step-3: Else, access the data where front is pointing.
Step-4: Increment front pointer to point to the next available data element.
Step-5: Return that element. (data).

→ Priority queue, and circular queue are variations of queue.

→ Applications of queue:
- Operating system and resource management such as CPU, Memory scheduling, Printer queue, etc.
- call center (~~via~~ operated on voice calls).
- Scheduling Jobs
- Searching.
- In multiplayer games. (at the time of joining)

Q.13) Write an outline of insert operation in a queue.
→ Queue is made of and works using two data

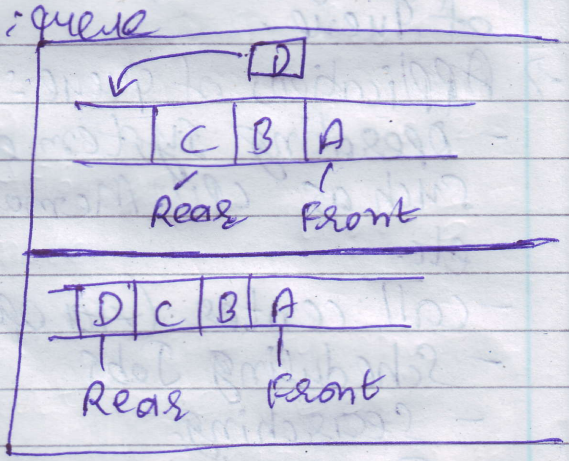
pointers: front and rear. Hence, its operations are complicated relative to operations of stack data structure. Following steps need to be taken to insert data into queue:

Step-1: Check if the queue is full.
Step-2: If the queue is full, produce overflow error and exit.
Step-3: Else, increment rear pointer to point the next empty space.
Step-4: Add data element to the queue location where rear is pointing.
Step-5: Return success.

→ Sometimes, we also check to see if a queue is initialized or not to handle any unforeseen situation.

→ Algorithm for insert operation:

```
Procedure insert(data): queue
    if queue is full
        return overflow
    end if
    rear <- rear+1
    queue[rear] <- data
    return true
end procedure
```



Q.14) Write an outline of delete operation in a queue

→ Accessing data from the queue is a process of two tasks: access the data where front is

pointing and remove the data after access.

→ The following steps taken for delete operation:

Step-1: Check if the queue is empty.

Step-2: If the queue is empty, produce underflow error and exit.

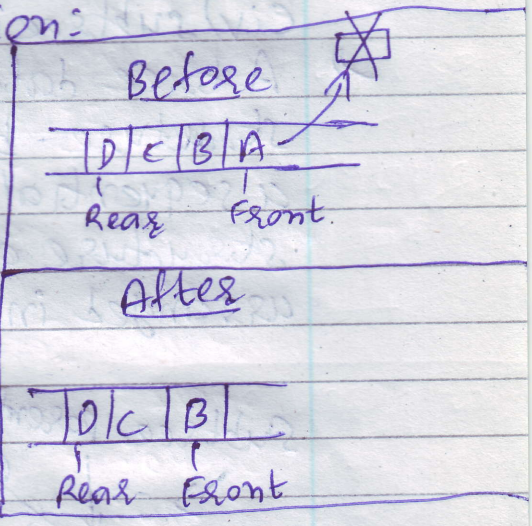
Step-3: If the queue is not empty, access the data where front is pointing.

Step-4: Increment front pointer to point to the next available data element.

Step-5: Return that data element.

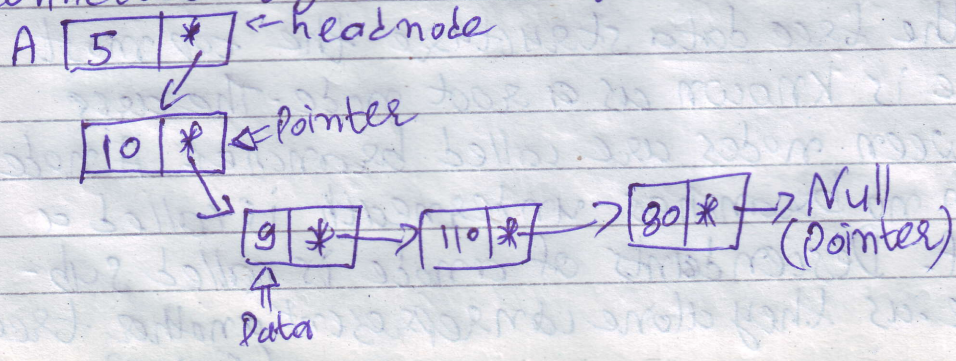
→ Algorithm for delete operation:

```
Procedure dequeue : queue
  if queue is empty
    return underflow
  endif
  data = queue [front]
  front <- front + 1
  return data
end procedure
```



Q.15) Create a linked list of 5 integers called A.

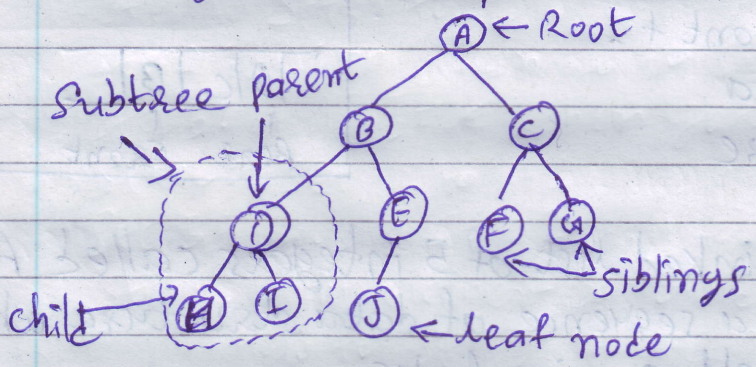
→ Linked list is a sequence of data structure which are connected together via links.



→ In linked list A, called 'Head node'. It consists 5 integers nodes. Each node is having a data and pointer part. The data part stores data (integers here) and pointer part points toward the next element of the list. Last node is pointing toward null pointer.

Q-16) Draw an example of tree data structure and show (i) root, (ii) branches, (iii) leaves and (iv) subtree in the drawing.

→ A tree data structure is a nonlinear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a tree are arranged in multiple levels.



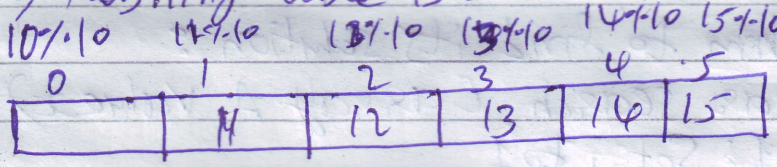
→ In the tree data structure the top most node is known as a root node. The arcs between nodes are called branches. A node with no branches underneath is called a leaf. Descendants of a node is called sub-tree. as they alone can represent another tree. (similar)

Q 17) Explain hashing data structure in detail with its practical applications.

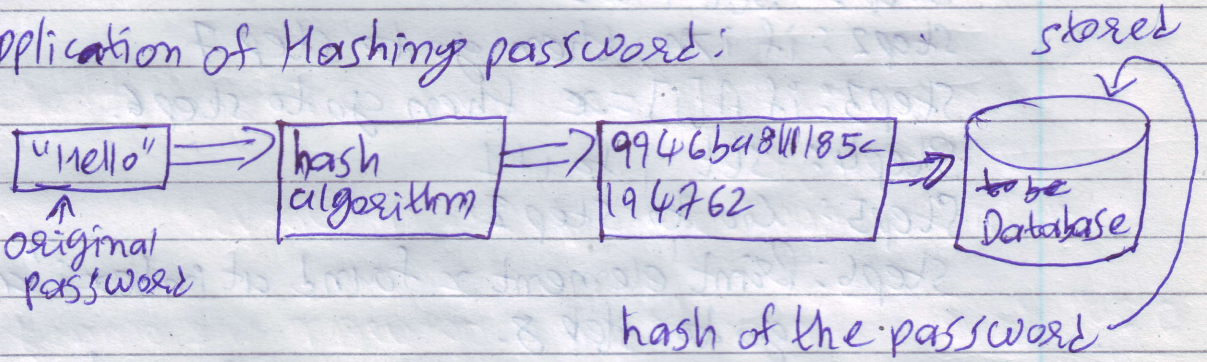
→ Hashing is a technic to convert a range of key values into a range of indices of an array. To apply this data structure, (implement)
we will use modulo operator.

- Let's have a hash function H(x) which maps the value at the index x % 10 in an array.
- For example, if the list of value is [11, 12, 13, 14, 15]. It will be stored at position {1, 2, 3, 4, 5}
  ↳ list: 11, 12, 13, 14, 15
  for each x in list : H(x) = x % 10

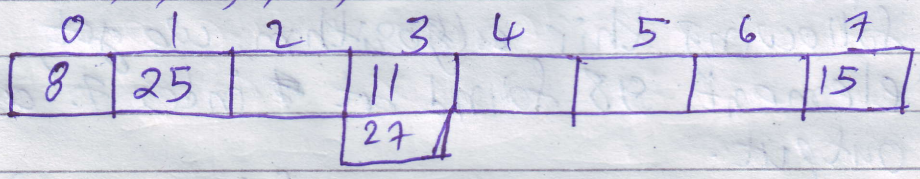
Hence Hashing table is:



→ Application of Hashing password:



- 7 Buckets (0, 1, 2, 3, 4, 5, 6, 7)
  Data (15, 11, 27, 8, 25)



Q.18) Explain linear search in detail by taking an example.

→ Linear search is a very simple search algorithm and most basic searching algorithm. In this type of search, a sequential search is made over all items one by one. Every item is being checked until the match is found or the end of the data collection.

e.g., We have array as shown below, from which we need to search for element with value 98.

10	9	8	27	91	37	88	98	11	19	23	34	array
0	1	2	3	4	5	6	7	8	9	10	11	index

- Algorithm to find its solution :

  Linear search (array A, Value x)
  Step 1: Set i to 1
  Step 2: if i > n, then go to step 7
  Step 3: if A[i] = x then go to step 6.
  Step 4: set i to i+1
  Step 5: Go to step 2
  Step 6: Print 'element x found at index i' and go to step 8.
  Step 7: Print 'element not found'
  Step 8: Exit.

- By following this algorithm, we got 'element 98 found at index 7.' as an output.

- Procedure linear_search (list, value)
  #

```
for each item in the list
    if item == value
        return the item's location
    endif
endfor
end procedure.
```

Q.19) Explain binary search in detail.

→ Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half.

e.g. Let us assume that we need to search the location of value 30 using binary search.

2	5	10	18	21	30	33	35	42	45	55	63
0	1	2	3	4	5	6	7	8	9	10	11

- first, we shall determine half of the array by using below formula:

mid = low + (high - low) / 2

i.e., $0 + \frac{(11-0)}{2} = 11/2 \simeq 5.5 \approx 5$

Now, compare the value stored at location 5. This is the best case where we found the match value 30.

- Let's take another array and value to be searched is 25.

3	25	30	35	46	53	62	70	73
0	1	2	3	4	5	6	7	8

i.e., $mid = \frac{0 + (8-0)}{2} = 4$

Let's check for the match value. 46 is greater than 25. which means match value can only be between

position 0 and 4. Thus, our current array is now like this:-

3	25	30	35	46
0	1	2	3	4

other part is ignored

~~Now, our new mid = low = mid+1~~ Now our

$$\text{newmid} = \frac{\text{low} + (\text{high} - \text{low})}{2}$$

$$= \frac{0 + (4-0)}{2}$$

$$= 2$$

Let's compare the value stored at position 2. It is 30 which is greater than 25. Thus, our new subarray is:

3	25	30
0	1	2

$$\text{new mid} = \frac{2}{2} = 1$$

At position 1, we found 25. And that's our target.

→ This is how binary search works.

Q.20) Give main difference between the binary search and linear search.

→ Linear search is simplest and straightforward to implement unlike binary search which is complex in compare to linear search.

→ Binary search seems faster than linear search.

→ Binary ~~works~~ search works with array ~~and~~ but with linked list it makes things complex. Meanwhile, linear search works fine with array and linked list both.

→ Time complexity of binary search is O(logn), whereas in linear search it is O(n).

→ Binary search works only with sorted array and that array must also be in single dimension only. On other hand, linear search works with any random & ordered one or many dimensional array.

Q.21) Explain Bubble sort by ~~take~~ taking an example.

→ Bubble sort is the simplest algorithm which worked by repeatedly swapping the adjacent element if they are in wrong order.

→ It is majorly used in workspace, where
- complexity doesn't matter.
- simple and short code is preferred.

- e.g. 8 5 3 1 4 7 9 <- we got to sort this array using bubble sort algorithm.

In bubble sort, we will pass this array for (maximum) n-1 times. where, n is length of array.

→ First pass;

8 5 3 1 4 7 9 <- ~~co~~ swapped

5 8 3 1 4 7 9 <- swapped

5 3 8 1 4 7 9 ← swapped

5 3 1 8 4 7 9 ← swapped

5 3 1 4 8 7 9 ← swapped

5 3 1 4 7 8 9 ← correct

5 3 1 4 7 8 9 ← #

→ Second pass:

5 3 1 4 7 8 9 ← swapped

3 5 1 4 7 8 9 ← swapped

3 1 5 4 7 8 9 ← swapped

3 1 4 5 7 8 9 ← ~~swapped~~ correct

3 1 4 5 7 8 9 ← correct

3 1 4 5 7 8 9 ← correct

3 1 4 5 7 8 9 ← #

→ Third Pass:

3 1 4 5 7 8 9 ← swapped

1 3 4 5 7 8 9 ← correct

1 3 4 5 7 8 9 ← correct

1 3 4 5 7 8 9 ← correct

1 3 4 5 7 8 9 ← correct

1 3 4 5 7 8 9 ← correct

1 3 4 5 7 8 9 ← ✱

→ We don't need to pass this array any further as it is sorted now. This will be ensured by a counter variable implicitly. No need to discuss about that here.

→ Algorithm for (improved) bubble sort is as follows:

```
Data: Input Array A[]
Result: Sorted A[]
int i, j, k;
indicator = 1; N = length(A);
for j = 1; j ≤ (N-1) ∧ indicator == 1; j++ do
    indicator = 0;
    for i = 1 to N-1; i++ do
        if A[i] > A[i+1] then
            temp = A[i];
            A[i] = A[i+1];
            A[i+1] = temp;
            indicator = 1;
        end
    end
end
```

—— X ——