23/11

Module 1

Data Structure

- Representation of the logical relationship existing b/w individual element of data.
- Way of organising all data items that considers not only the elements stored but also their relationship to each other.
- It affects both the design of both structural & functional aspects of a program.

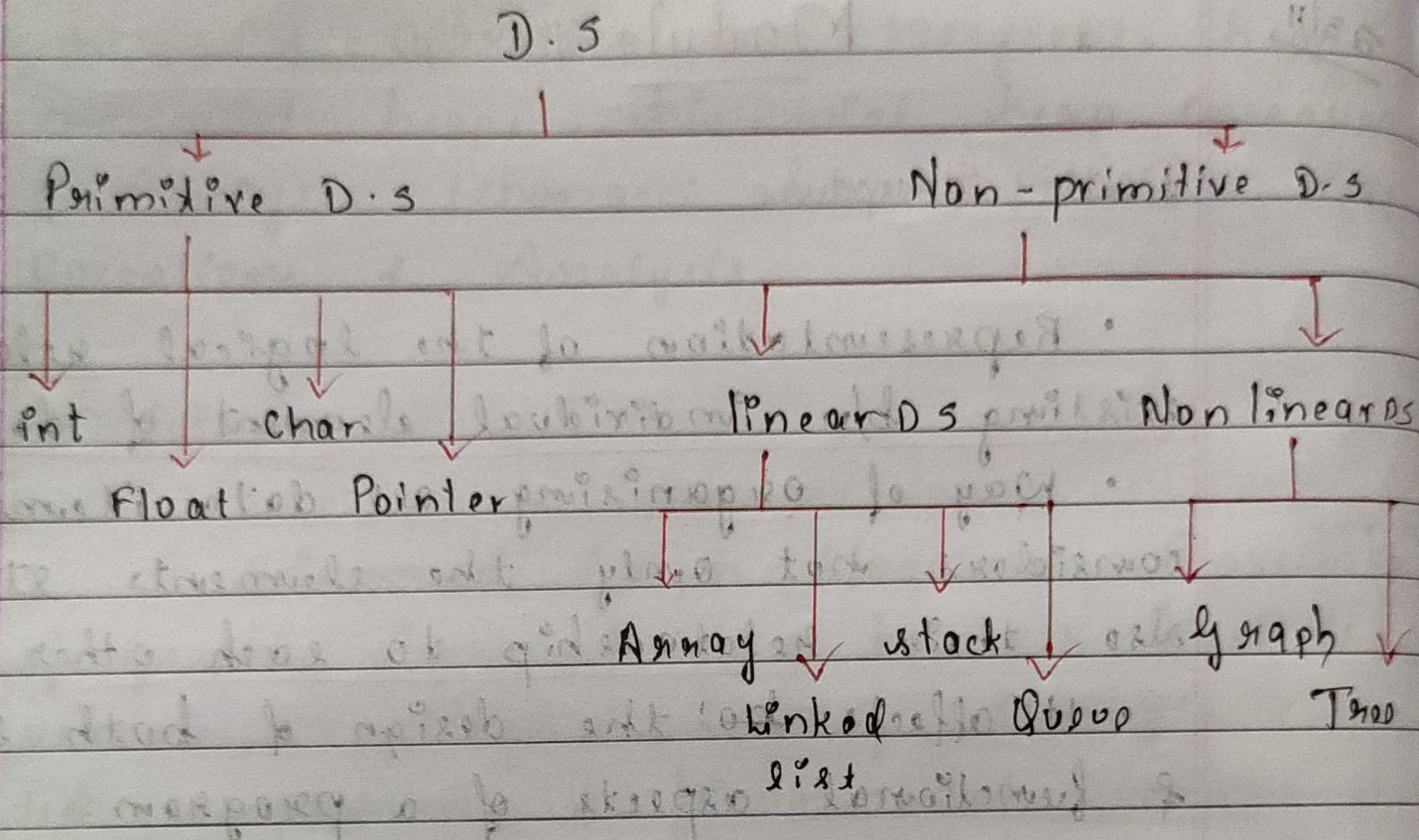
Program = Algorithm + Data Structures

- D.S organises related data in a comp.
- Each D.S has a set of operations that can be performed on the data it stores.

Classification of D.S.

D.S are normally divided into two broad categories:

1) Primitive D.S
2) Non primitive D.S



- Primitive D.S

- These are basic structures & directly operated upon by the machine instruction
- In general, there are dif. representation on dif. computers
  Eg: Int, Float, Char, pointer

- Non primitive D.S.

* More sophiscated data structure.
* These are derived from primitive D.S.
* forms group of homogeneous or hetero-

Page No :
Date :

geneous data items.

Eg: Lists, Stack, Queue, Tree etc.

* Most commonly used Operations on D.S are create, Selection, updating, Searching, Sorting, merging, destroying.

* A primitive D.S. is generally built in to the language whereas non-primitive D.S. is built out of primitive D.S. linked together like linked list, Binary Search tree etc.

24/11/20

ARRAY

1) An array is defined as a set of finite no. of homogeneous elements or same data items.

2) Array can contain one type of data only.

3) It is a non primitive linear D.S that stores a collection of related values that are all the same type.

4) Arrays are useful because they can represent related data with one descriptive name.

* static m/y allocation - before usage we have to declare

Page No :
Date :

5) Eg : int arr[10]

↓ specifies the datatype

↓ Name of Array

↳ Size or length

6) Individual element of an array can be accessed by specifying name & index or subscript inside.

7) first element has index zero.

8) The elements of array will always be stored in the consecutive m/y location.

9) No. of elements that can be stored in an array is given as

(upperbound - lowerbound) + 1

Eg : int arr[10] $\Rightarrow (9-0)+1 = 10$ //

10) Array can always be read or written through loop.

Read : for (i=0; i<=9; i++)
scanf ("%d", &arr[i]);

Write : for (i=0; i<=9; i++)

printf("%d", arr[i])

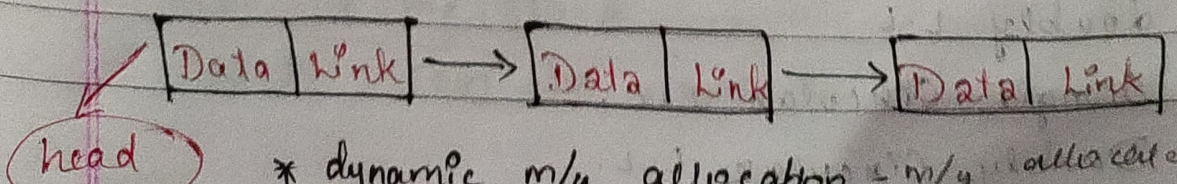
ii) Reading or Writing 2-D Array would require two loops. 'N' dimension require 'n' loops.

Array Operations are :

- Creation
- Deletion
- Traversing
- Modification
- Insertion
- Merging

LINKED LIST

> It can be defined as a collection of variable number of data items.
> Non primitive D.S. :- change during execution
> Element in it contains at least two fields one for storing data & other for storing address of next element
> Used to implement D.S.
> It is a sequence of nodes.
> Each element is referred to as a NODE.



* dynamic m/y allocation = m/y allocated during Execution

> Successive elements are connected by pointer
> Last address to track i.e., must know the pointer to first element of list.

> last item in the list has a link to NULL.

Advantage:

* Linked list is similar to array, but no need to declare number of elements.
* It can insert or remove elements without reallocation or reorganisation of the entire structure as data items need not to be stored contigously.
∴ No wastage of m/y space

Disadvantage:

* Random access not allowed i.e, access modes sequentially. Cannot do a binary search on a linked list.
* Extra m/y space for a link is required for each element in the list.

Types of lists:

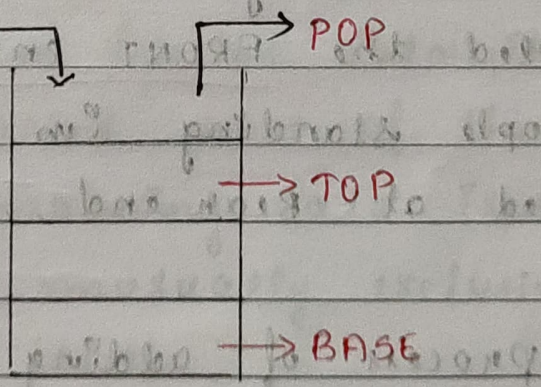
- Single L.L
- Doubly L.L
- Single Circular L.L.
- Doubly Circular L.L.

STACK

- abstract data Type
- Ordered Collection of elements
- deletion & insertion of data done only from one end called top of stack (TOP)
- It is last in first out type (LIFO)

Eg: stack of plates

- It is non-primitive D.S.
- when insertion or deletion is done, its base remain fixed where TOP changes.
- insertion of element into stack is called PUSH & deletion of element from stack is called POP.



- Adds or removes elements in a particular order.
- Every time an element is added, it goes to "TOP" of the stack.
- Stacks can be used to create - redo func-

Page No :
Date :

Normalities, parsing expressions
- Stack can be implemented using an array or a linked list.

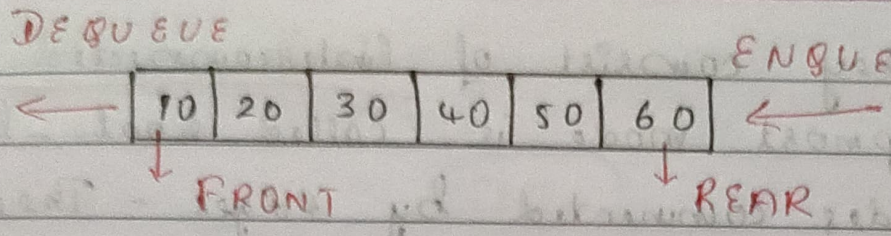
* Array (Static Implem)
* Pointer (Dynamic Impl^n)

QUEUE

- Abstract data type

- FIFO
- In Queue, elements are added to the Queue from one end called REAR end & the element are always removed from other end called the FRONT end.

Eg: people standing in railway reservation
- Inserted at Rear end, Deleted at Front End
- The process of adding new elements into the Queue is called Enqueue.
- The process of Removing element = dequeue
- Queue is used to manage objects in order starting with the first one.
- Queue implemented using
  - Array & pointer

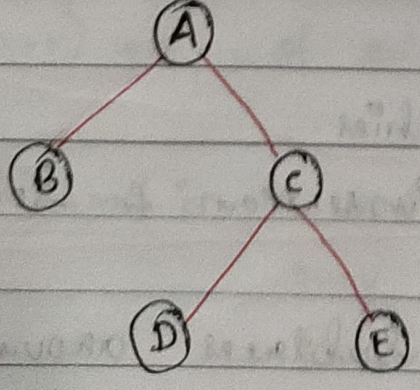


26/11

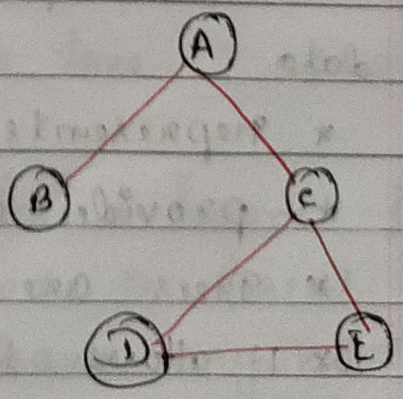
NON-LINEAR D.S

• TREE

- Non linear D.S
- Organises data in a hierarchical structure
- recursive definition
- It is a connected graph without any circuit
- It is a finite set of data
- There is a special data item at the top hierarchy called root of the tree.
- The remaining data items are partitioned into no. of mutually exclusive subset, each of which is itself.
- Multilevel D.S.



(Tree)



(Not a Tree)

- They consist of interconnected nodes
- Topmost node is called root - level 0
- nodes connected by edges - Trees
- Different tree D.S allow Quicker & easier access to the data.
- Each node Contains pointers to the next adjacent Nodes.

Properties

- One & only one path between every pair of vertices in a tree. (A)——(B)
- A tree with n vertices has exactly (n-1) edges
- A graph is a tree & only if it is minimally connected.

Advantages

* Trees reflects structural relationships in the data
* represents hierarchies.
* provide efficient insertion & searching op.
* Trees are flexible.
* It allows to move subtrees around with

4. Child : • The node which is a descendant of some node is called as a child Node.
• All the nodes expt root node are child Node

5. Siblings :- Nodes which belong to the same parent.
- nodes with same parent.

6. Degree : * Degree of a node is the total no. of children of that node.
* Degree of a tree is the highest degree of a node among all the nodes.

7. Internal Node : • The node which has atleast one child is internal Node.
• Also called non-terminal Nodes.
• Every non-leaf node is an internal node.

8. Leaf Node : - The node which doesn't have any child.
- Also called external / terminal node

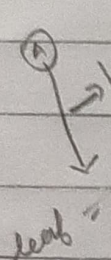
9. Level : * In a tree, each step from top to

Page No :
Date :

bottom is level of a tree.

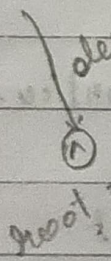
* level counts start with 0 & increments by 1 at each level of or step.

10. Height : - Total no. of edges that lies on the longest path from any leaf node to a particular node is called height of the Node.



- It is the height of root node.
- height of leaf node is 0.

11. Depth : • Total no. of edges from root node to a particular node is depth.

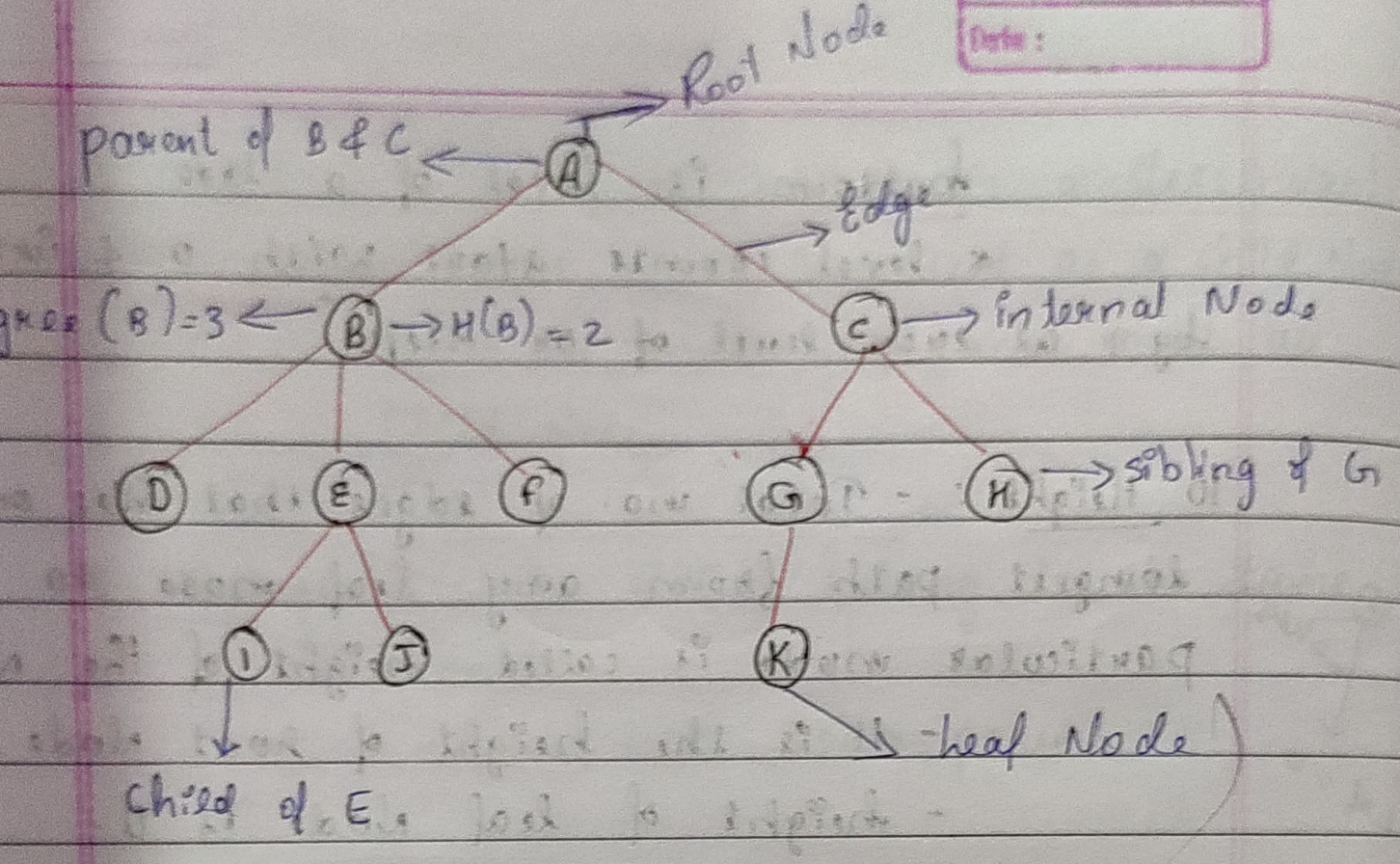


• depth is the total no. of edges from root node to a leaf node in the longest path
• Depth of root Node is 0
• level & 'depth' are used interchangably

12. Sub-tree : * In a tree, each child from a node forms a subtree recursively.

* Every child node forms a Subtree on its parent node.

13. Forest : A forest is a set of disjoint trees.



12020 SET DATA STRUCTURE

- Collection of object need not be in particular ~~object~~ order
- Array elements can repeat, but set elements cannot repeat. (unique elements)
- Set is similar to array

Eg: storing Indian Cricketer Names

* The names should not be repeated, as well the name who is not in a team cant be inserted.
* This is the restriction we found in a Set

Page No :
Date :

1) NULL SET (EMPTY SET)

The set with no element is called null set or Empty set.

Eg: $A = \{x \mid x \text{ is a prime number, where } 24 < x < 29\}$

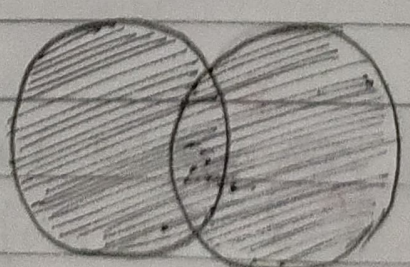
This set could contain a element. We don't have any prime no. within the limit 24.

Basic Set Operations

1) Set Union (Set1, Set2)
2) Set Intersection (Set1, Set2)
3) Set Difference (Set1, Set2)
4) Int Subset (Set1, Set2)

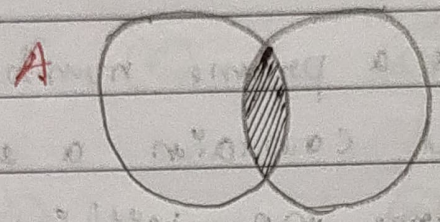
→ Union

Combine two or more sets without violating the rules for set.



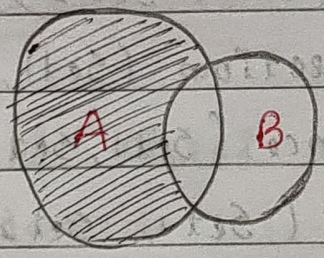
→ Intersection

Gathering common elements in both the sets together as a single set.



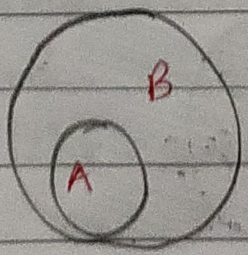
→ Difference

forming a set with elements which are in first set & not in second set.



→ Subset

* int Subset (Set1, Set2)
* Returns 1 if the Set1 is the subset of the set2, Otherwise 0 if not.
* Here A ⊂ B (A contained in B)



Page No :
Date :

Types of Sets

- Immutable (Unchangeable) (static)
- Mutable (Changeable) (dynamic)

→ Operations on Mutable

1) void * create(n)
2) add(set, element);
3) delete(set, element);
4) int capacity(set);

1) CREATE

- void * create(int n)
- It simply create the set that can be used to store 'n' elements & return the starting address of the set in memory
- int * set = create(30);

2) ADD

- add(set set_name, type value)
- This function add one element into the set specified in the function argument.

- The set_name argument is to mention the set which is going to be updated.
- value argument is the value to be inserted into the set that passed to the function.

3) DELETE

- delete(set set_name, type value)
- This func. delete one element from the set specified in the func. argument.
- The set_name argument is to mention the set which is going to be updated.
- Value argument is the value to be deleted from the set that passed to this func.

4) CAPACITY

- int capacity(set set_name).
- This func. is used to find the maximum no. of elements can be stored into the set that passed to the func.
- Find the count & return the integer value.

→ Operations On Immutable

- Int elementof(x, S)
- Int Empty(S)
- int Size(S)
- void * build(x1, x2 ... xn)

A) ELEMENTOF

- int elementof(type x, set set_name)
- This func. check for the element 'x' in the passed set.
- Returns the value whether 0 or 1.
- If the x is there in the set, return 1 other wise 0.

B) EMPTY

- Int Empty(set set_name)
- This func checks for the emptiness of the se that passed to the func.
- If the set is empty, return 1 else 0.

c) SIZE

- Int Size(set set_name)

- Returns the number of elements in set

4) BUILD

- void * build (set_elements)

- This func. create the set with the given elements & return the address of the first element

- Return the void *, we can assign the pointer to the type * which type we declared the set

→ A set, giving the records about the age of cricketer less than or equal to 35 is given as:

- { 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1 }

→ Here 1 indicates the presence of records having the age less than or equal to 35.

→ 0 indicates the absence of records having the age less than or equal to 35.

→ As we have to indicate presence or absence of an element only, so 0 or 1 can be used for indication for saving storage space.

→ A bit array data structure is known for this purpose.

Page No :
Date :

1/12/2020

Operations On Bit Vector Representation

- It is very easy to implement set operation on the bit array data structure.
- The operations are well defined only if the size of the bit arrays representing two sets under operation are of same size.

1) UNION

* To obtain the union of sets S_i & S_j, the bit-wise OR operation can be used.
* S_i & S_j are given below

S_i = 1 0 0 1 0 1 1 0 0 1

S_j = 0 0 1 1 1 0 0 1 0 0

$S_i \cup S_j$ = 1 0 1 1 1 1 1 1 0 1

Algorithm :

> Input : S_i & S_j are two bit array corresponding to two sets.
> Output : A bit array S is the result of $S_i \cup S_j$.

> Data Structure : Bit vector representation of set.

2) INTERSECTION

- To obtain the intersection of sets Si & Sj the bitwise AND operation can be used.
- Si & Sj are given below

Si = 1 0 0 1 0 1 1 0 0 1

Sj = 0 0 1 1 1 0 0 1 0 0

Si ∩ Sj = 0 0 0 1 0 0 0 0 0 0

Algorithm

> Input : Si & Sj are two bit array corresponding to two sets.
> Output : A bit array S is the result of Si ∩ Sj
> Data Structure : Bit Vector Representation of Sets

Steps:

1. li = LENGTH(Si)
2. lj = LENGTH(Sj)

3. If (li != lj) then
   1. Print "Two sets are not compatible for ~~difference~~ Intersection"
   2. Exit
4. End if
5. for i=1 to li do
   ~~1. Sj[i] = NOT~~
   1. S[i] = Si[i] AND Sj[i]

3) DIFFERENCE

> The difference of Si from Sj is the set of values that appear in Si but not in Sj. This can be obtained using bit-wise AND on the inverse of Sj.

> Si & Sj are given below

Si = 1 0 0 1 0 1 1 0 0 1

Sj = 0 0 1 1 1 0 0 1 0 0

Sj' = 1 1 0 0 0 1 1 0 1 1 // 1's complement of Sj

Difference = 1 0 0 0 0 1 1 0 0 1 //

Algorithm :

> Input : Si & Sj are two bit array. Correspond-

ing to two sets

> Output : A bit array s is the result of Si & Sj

> Data Structure : Bit vector Representation of set

Steps

1. li = LENGTH(Si)
2. lj = LENGTH(Sj)
3. if (li != lj) then
   1. Print " Two sets are not compatible for difference"
   2. Exit
4. End if
5. for i=1 to li do
   1. Sj[i] = NOT Sj[i]
6. End for

4) EQUALITY

- The equality operation is used to determine whether 2 sets Si & Sj are equal or not
- This can be achieved by simple comparison b/w the pair-wise bit values in two bit arrays

Date :

Algorithm

> Input: Si & Sj are two bit array corresponding to two sets.
> Output: Return TRUE if they are equal else FALSE
> Data Structure: Bit vector representation of set.

STEPS

1. li = LENGTH(Si)
2. lj = LENGTH(Sj)
3. If (li != lj) then
    1. Return (FALSE)
    2. EXIT
4. End if
5. For i=1 to li do
    1. Si[i] != Sj[i] then
        1. Return(FALSE)
        2. EXIT
    2. END IF
6. END FOR

Page No :
Date :

Application of Set Data Structure

1) Information storing using bit string

→ Technique of storage & retrieval of information using bit strings

→ A bit string is a set of bits that is a string of 0's & 1's.

Eg: 1000110011 is a bit string

→ Length of the bit string = no. of records

→ To store a particular column, we require bit arrays storing a set of bit string.

→ The number of bit arrays will be determined by different attributes that the field may have.

2) Information retrieval using bit string

→ To retrieve this information only bit arrays CS needs to be searched for the number of 1's in it.

3) Performance Issue of the technique