CHHATTISGARH SWAMI VIVEKANAND TECHNICAL UNIVERSITY

छत्तीसगढ़ स्वामी विवेकानंद तकनीकी विश्वविद्यालय

To be filled, scanned and kept at 1st page of Answer Booklet.

Nov-Dec 2021 Examination

Student Name: V. Om Sai Nageshwar sharma

Mobile No.: 8602727389

Email ID: Nageshwar.sharma@gmail.com

Enrollment No.: | B | J | 4 | 5 | 9 | 9 |

Roll No.: | 3 | 0 | 3 | 3 | 0 | 2 | 2 | 2 | 0 | 0 | 2 | 0 |

Course: B.Tech Semester: 3rd

Branch/Specialization: CSE

Subject Code: | B | 0 | 2 | 2 | 3 | 1 | 2 | (| 0 | 2 | 2 |) |

Subject Name: Data Structure & Algorithm

Regular/Backlog: Regular

Date of Exam: 23/03/2022

Note:

1) Only above format is to be used for Nov-Dec 2021 Exams. Older/earlier format will not be accepted.
2) Nomenclature to be mentioned in the Answer Booklet should be Subject code_Roll No. only.
3) Only Roll No. generated in Admit Card must be filled (College Transfer students must take care in filling their Roll Nos,).

I certify that above information given there in is correct and I shall be personally responsible for the same if proved wrong/false later on.

Signature: [signature]

CHHATTISGARH SWAMI VIVEKANAND
TECHNICAL UNIVERSITY
Bhilai, Durg, Chhattisgarh
Admit Card for Session Nov-Dec 2021
STUDENT DETAILS
Roll No. 303302220020
Registration No. BJ4599
College Name 033 - SHRI SHANKARACHARYA INSTITUTE OF PROFESSIONAL MANAGEMENT & TECHNOLOGY,RAIPUR
Student's Name V OM SAI NAGESHWAR SHARMA
Date of Birth 17-02-2001
Gender Male
Current Semester 3 SEMESTER
Father's Name V Somshekhar Sharma
Course B.Tech
Program B Tech Computer Science Engineering
SUBJECT DETAILS
Sl | Semester | Subject Type | Subject Code | Subject Name | Exam Session | Exam Type | ToAppear | Date & Time of Exam
1 | 3 SEMESTER | Sessional | B000308(046) | Personality Development | Nov-Dec 2021 | Regular | Y
2 | 3 SEMESTER | Theory | B000311(022) | Data structure & Algorithms | Nov-Dec 2021 | Regular | Y | 23/03/22 10:00 am
3 | 3 SEMESTER | Theory | B000312(014) | Mathematics - III | Nov-Dec 2021 | Regular | Y | 21/03/22 10:00 am
4 | 3 SEMESTER | Theory | B000313(022) | Principles of Programming Languages | Nov-Dec 2021 | Regular | Y
5 | 3 SEMESTER | Theory | B000314(022) | Digital Electronics& Logic Design | Nov-Dec 2021 | Regular | Y
6 | 3 SEMESTER | Theory | B000315(022) | Operating Systems | Nov-Dec 2021 | Regular | Y
7 | 3 SEMESTER | Practical | B000321(022) | Data structure & Algorithms Laboratory | Nov-Dec 2021 | Regular | Y
8 | 3 SEMESTER | Practical | B000322(022) | Digital Electronics& Logic Design Laboratory | Nov-Dec 2021 | Regular | Y
9 | 3 SEMESTER | Practical | B000323(022) | Operating Systems Laboratory (UNIX) | Nov-Dec 2021 | Regular | Y
10 | 3 SEMESTER | Practical | B000324(022) | Software Laboratory (Sci Lab/MATLAB) | Nov-Dec 2021 | Regular | Y
Signature should match with uploaded signature of candidate
Signature of the Candidate (while receiving)
Signature of the Principal
Signature of COE
INSTRUCTION FOR WRITTEN EXAMINATION

Unit - 1

(a)

Ans => Data structure is a collection of elements organised in a specified manner and accessing functions are defined to store and retrieve individual data elements.

→ **Time Complexity of an algorithm:**

Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.

→ **Space Complexity of an algorithm:**

Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.

(C)

Ans => Let p and q be the two polynomial equations represented by 2 seperate linked list.

① while p and q are not null, repeat step ②

② If ~~op~~. powers of the two term are equal then, & if the terms do not cancel then, insert the sum of the term into sum polynomial.

Advance p and Advance q

else if, the power of first polynomial is greater than power of second polynomial.

then insert the turn ~~first~~ from first polynomial to the sum polynomial Advance p.

else insert the term from second polynomial onto sum polynomial

Advance q

③ Copy the remaining terms from the non-empty polynomial into sum polynomial.
repeat ③ till the polynomial is empty.

④ Exit.

$$p = 5x^4 + 2x^3 + 7x + 1 = 0$$
$$q = 2x^3 + 5x^2 - 3x + 1 = 0$$

p = * L → (5, 4) → (2,3) → (0,2) → (7,1) → Null

q = * L → (0, 4) → (2,3) → (5,2) → (-3,1) → (1,0) → Null

#1 P_1 = (5,4) q_1 = (2,3)

Sum = * L → (5,4) → (4,3)

#2 P2 = (2,3) ; q_1 = (2,3)

Sum = * L → (5,4) → (4,3)

#3 P_3 = (7,1) q2 = (5,2)

Sum = * L → (5,4) → (4,3) → (5,2)

$P_3 = (7, 1)$ $\quad q_3 = (-3, 1)$

Sum = $\overset{*}{L} \rightarrow (5, 4) \rightarrow (4, 3) \rightarrow (5, 2) \rightarrow (4, 1)$

(D)

Ans => <u>Sparse matrix</u>: Matrix with many zero entities is called a sparse matrix. For effective memory utilization we store only the non-zero elements. Each element is uniquely identify by its row and column position. (row, column, element). It is stored in the row major form i.e. in the ascending order of rows. Inside each row, store elements in the ascending order of columns.

Program:

```
function Sparse (int a[][], int m, int n)
{
  int i, j, k;
  s[0][0] = m;  s[0][1] = n;  k = 1;
  for (i = 0; i < n; i++)
      for (i = 0; i < n; i++)
       if (a[i][j] ! = 0)
       {
          s[k][0] = i;
          s[k][1] = j;
           s[k][2] = s[i][j];
          k++;
       }

  s[0][2] = k-1;
}
```

Pros and cons of using sparse matrix over 2-D array.

Pros: It has many numbers of zeroes so easy to implement.

cons: slow as compared to two dimensional array.

Types of sparse matrix:

1. Upper triangular matrix,
2. Lower triangular matrix.
3. Diagonal matrix,
4. Triangular matrix.

Unit - 2

(a)

Ans => ☆ **Stack Underflow** => Stack underflow happens when we try to pop (remove) an item from the stack, when nothing is actually there to remove i.e. stack is empty.

Stack Underflow cond^n =>

~~overflow~~

when stack is empty (i.e. Top == -1 or Top == Null) and we try to delete more element from it, ~~it~~ this condition is called underflow condition.

☆ **Stack Overflow** => Stack overflow happens when we try to push (add) one or more item onto our stack than it can actually hold.

Stack Overflow condition:

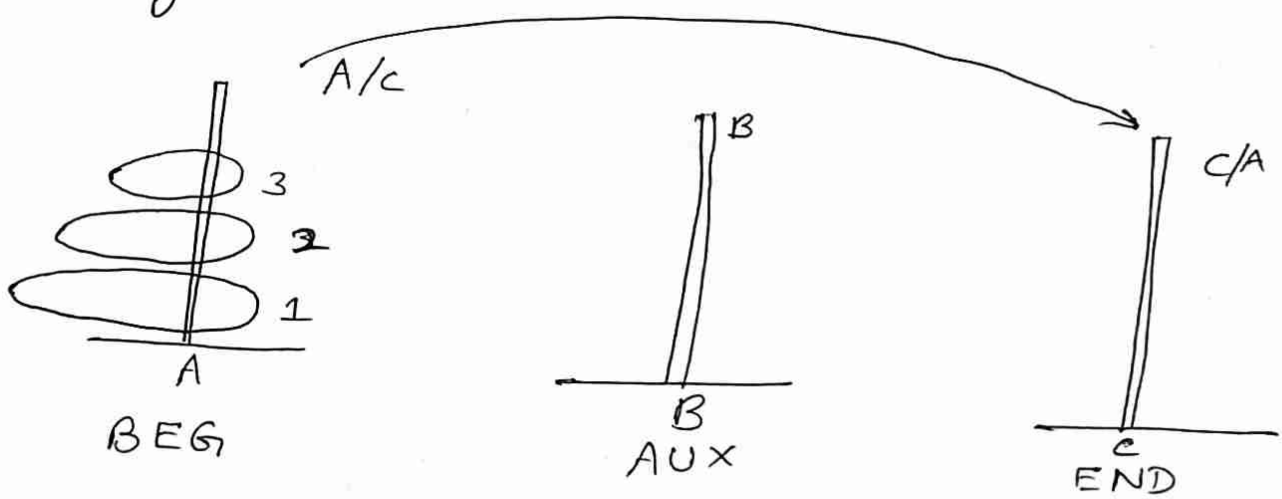
If MAXSTK is the size of the stack, then if, TOP = MAXSTK then this condition is called overflow in stack.

(b)

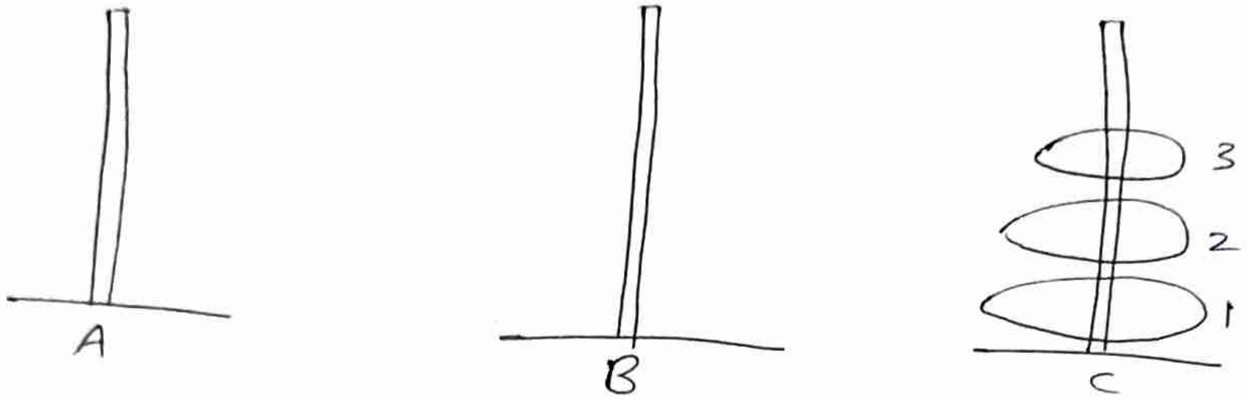
Ay => Tower of Hanoi :

→ In this problem three no. of pegs will be given and we have to use the stack holders to solve the problem.

→ The pegs are initiated by 1, 2, 3 and stack holders are indicated by A, B, C.



→ we have to move all the pegs from stack A to C and the final output is displayed as given.



Algorithm to solve tower of Hanoi problem:

1.) If N = 1 then,
   a) Write : BEG → END
   b) Return
   (End of if structure)

2.) [MOVE N-1 disk from peg BEG to peg AUX] call Tower [N-1, BEG, ENG, AUX]

3.) Write : BEG → END

4.) [MOVE N-1 disks from peg AUX to peg END]
   call Tower (N-1, AUX, BEG, END)

5.) Return.

(C)

Ans → A stack data structure can be implemented using a 1-D array. But stack implementation using array stores only a fixed number of data values.

⇒ Algorithm to push value.

Push (STACK, TOP, MAXSTK, ITEM)

① If TOP = MAXSTK, then write overflow & exit.
// checking if stack is already full.

② Set TOP = TOP + 1
// making the above node as top.

③ Set stack [TOP] = ITEM]
// Inserting the data to the Top node.

④ Exit.

2.) Algorithm for POP() operation in stack using array:-

POP (stack, TOP, ITEM)

① If TOP = 0 or TOP = NULL then write underflow & EXIT.

// checking if the stack is empty.

else

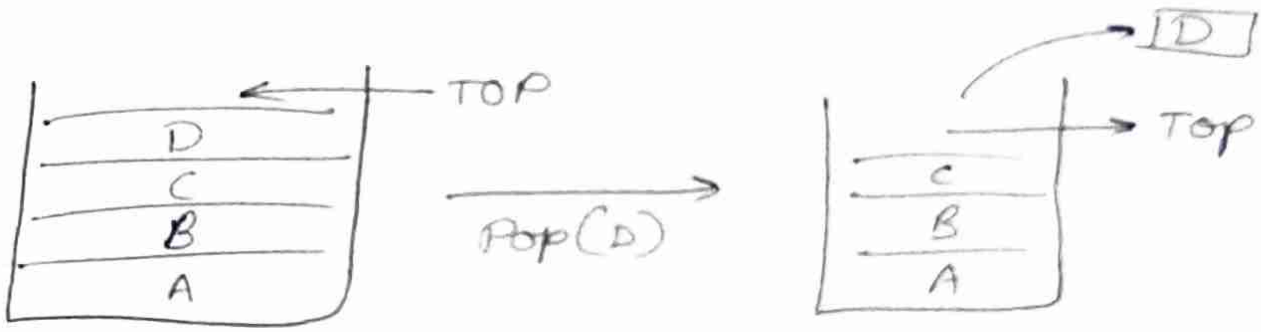
② Set ITEM = STACK [TOP]

// Assigining top value to item

③ Set TOP := TOP - 1

// Deleting the top element and shifting top below.

④ Exit



Algorithm for travering a stack.

① If TOP=0 or TOP =NULL write underflow and Exit.

else

② Print STACK[TOP]

③ Set TOP --;

④ Repeat step ② and ③ while (TOP>0)

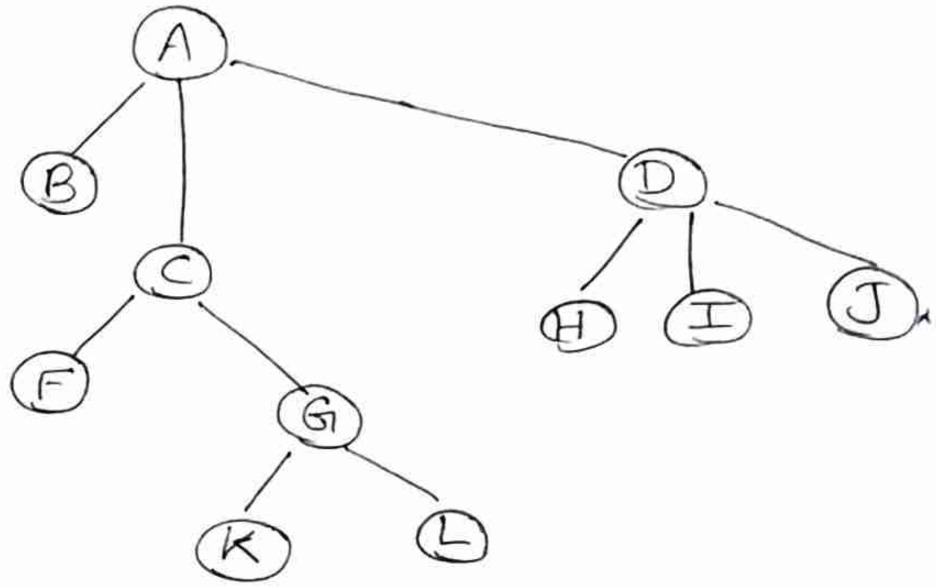
⑤ Exit.

Unit-3

(a)

Ans-> <u>Tree</u> → A tree is a collection of elements called 'nodes'.
Each node contains some values or element. It stores the actual data along with links to other nodes.

<u>Example:</u>



<u>linked representation of binary trees:</u>

consider a binary tree T. It uses three parallel arrays. Info, LEFT & Right and a pointer variable ROOT as followers.

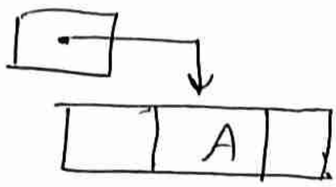
1.) Info [K] contain the data at the node.

2.) LEFT [K] contains the location of the left child of node N.

3.) RIGHT [K] contains the location of the right child of node N.

ROOT will contain the location of the root R of T.

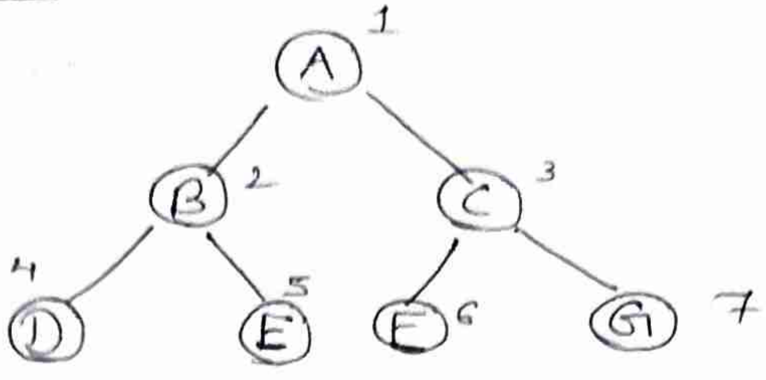
Example:



Array representation of binary Tree

In order to represent a tree in a single one-dimensional array, the nodes are numbered sequentially level by level from left to right. Even empty nodes are numbered.

Example:



Array

0	1	2	3	4	5	6	7
7	A	B	C	D	E	F	G

(C)

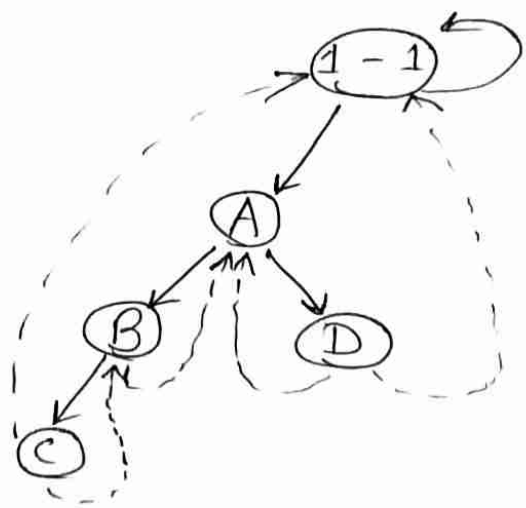
Ans=> Threaded binary trees:

- In usual tree traversal, using recursive funct~~ing~~ions, the run time stack is utilized. In the case of non-recursive variants, an explicity defined and user maintained stack (or queue) is used. It is more efficient to incorporate the stack as part of the tree. This is done by incorporating threads in a given node.

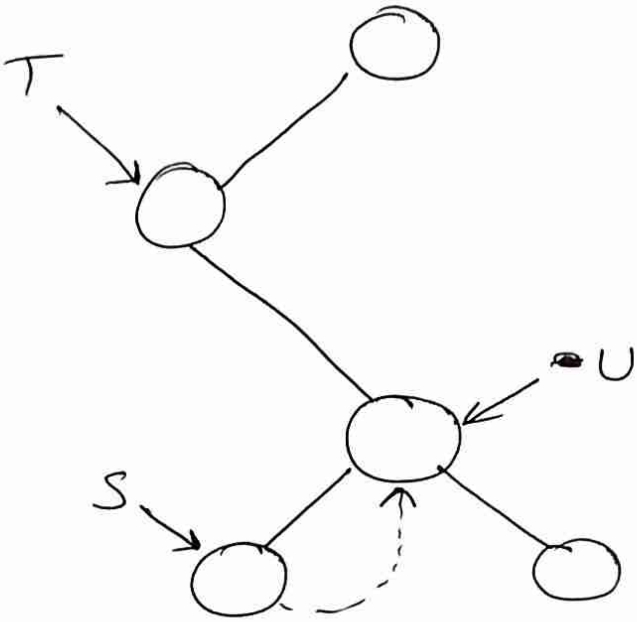
Threads are pointers to the predecessor and successor of the node according to an inorder traversal, and the tree whose nodes use threads are called threaded trees.

Traversing in threaded Binary Tree:

1.) Preorder traversal (TBT): A non-recursive preorder traversal on TBT can be implemented without sting a stack. In a TBT, it is easy to find the preorder successor.



2.) Inorder Traversal (TBT): If the right child of a node is not a thread then the left most child in the right subtree will be the inorder sucessor. If the right link is a thread, then the thread itself points to the successor.



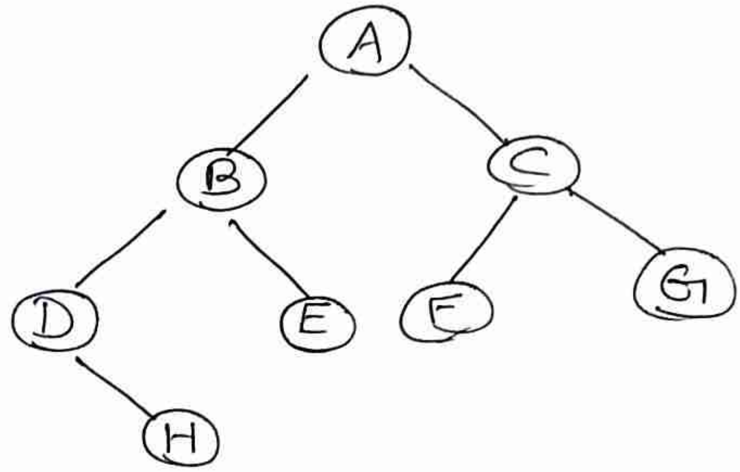
3.) Postorder traversal (TBT): Comparatively difficult. Inorder ~~to~~ to locate ~~to~~ the postorder successor of any arbitrary node (say t), there should be a method of finding the parent of node "t".

(b)

Ay => To construct a Binary tree from inorder and Preorder Traversal-

Preorder : A B D H E C F G

Inorder : D H B E A F C G



Hence, the binary tree from the given traversal.

→ Start with the root node, which would be the first item in the preorder sequence and find the boundary of its left and right subtree in the inorder sequence.

→ To find the boundary, search for the index of the react node in the inorder sequence.

→ All keys before the roots node becomes part of the right subtree.

→ Repeat this recursively for all nodes in the tree and construct the tree.

Unit - IV

(a)

Ay-> A graph is a pictorial representation of set of objects where some pairs of objects are connected by links.

In a sequential representation, there is a use of an adjacency matrix to representation, there is a use of an adjacency matrix to represent the mapping between vertices and edges of the grap.

An adjacency list represents a graph as an array of linked lists. The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.

(C)

A) => <u>Spanning Tree:</u>

Spanning tree is defined as an undirected graph $G = (V, E)$ where V indicates the no. of vertices and E indicates edges and near to create subgraph that is called $T = (V', E')$

=> <u>Kruskal's Algorithm:</u>

1. In step 1 Find out the total no. of vertices and total no. of edges in a given graph.

2. Arrange all the weighted edges in non-decreasing order (increasing order).

3. Create a set E and initially it is empty $E = \phi$.

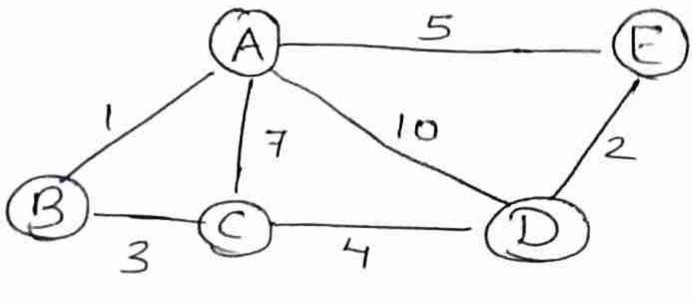
~~4~~

(4) Select an edge that having minimum cost and check that by using that edge a cycle is created or not.

(5) If the cycle is created then we have to discard that edge into set E.

(6) Continue the process until you obtain a minimum spanning tree.

Given,



(1) By using Kruskal's algorithm total no. of vertices = 5 and edges = 7.

(2) A - B = 1

E - D = 2

B - C = 3

C - D = 4

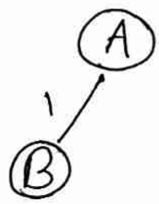
A - E = 5

A - C = 7

A - D = 10

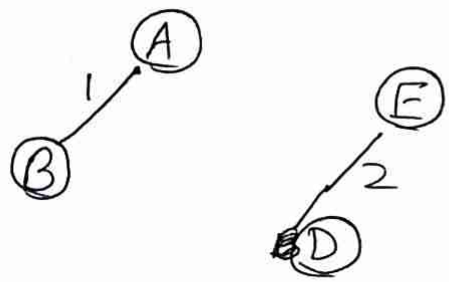
(3) Select edge A - B and this edge doesn't create any cycle so that we can add that edge in E.

E = [(A - B)]



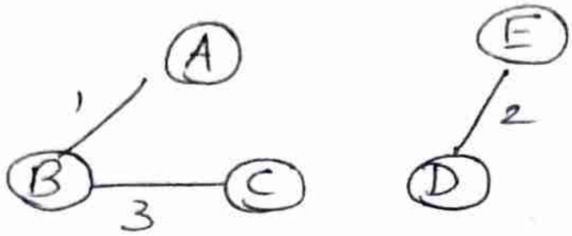
Now selecting D - E

E = [(A - B)(E - D)]



Now selecting B - C

E = [(A - B), (E - D), (B - C)]

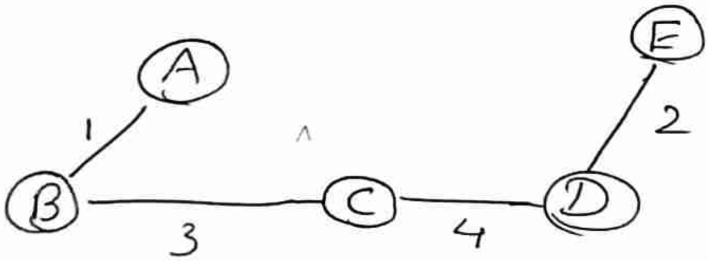


Now selecting C - D

E = [(A - B), (E - D), (B - C)]

Now selecting C - D

E = [(A - B), (E - D), (B - C), (C - D)]



Now selecting (A - E)

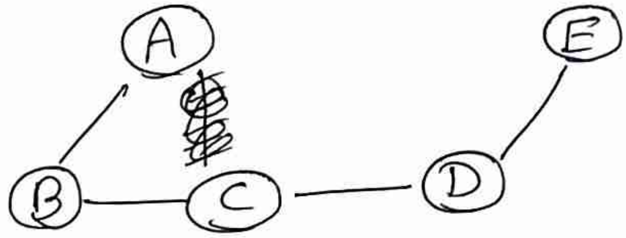
A-E we create a cycle / loop so we need to discard (A - E)

Similarly,

(A - C) & (A - D) will be discarded as they will form a cycle is used.

so,

E = [(A - B), (E - D), (B - C), (C - D)]



= This is the required minimum Spanning tree of the given graph.

(D)

Ans=> ~~Let~~ Warshall's Algorithm:

Suppose we want to find the path matrix P of the graph G. Warshall gave an algorithm for this purpose that is much more efficient than calculating the powers of the ~~ado~~ adjacency matrix. First we define m-square Boolean matrices $P_0, P_1, \ldots P$ as follows.

Algorithm:

1.) Repeat for I, J = 1 to M [Initializes P]
if A[I, J] = 0, then: set P[I, J] := 0
else: set P[I, J] := 1;
[End of loop].

2.) Repeat steps ③ & ④ for K = 1 to M [Update P]
3.) Repeat step 4 for I = 1 to M.
4.) Repeat step 4 for J = 1 to M.
set P[I, J] := P[I, J] ∨ (P[I, K] ∧ P[K, J])
[End of loop].
[End of steps 3 loop].
[End of step 2 loop]

Unit - V

Ans :> (a) > Linear Search (Sequential Search)

In sequential search we search for an item in a sequential manner also the linear and sequential both are same. This is done by ~~a~~ accessing each element only once from the beginning of the list.

The algorithm for linear search is a follows:-

★ Algorithm : LINEAR (A, N, VALUE, LOC)

Here A is a linear array with ~~st~~ N elements and VALUE is a given item of information. This algorithm finds the ~~tog~~ location LOC of VALUE in A, or sets LOC=0 if the search in unsuccessful.

Step-1: [Insert VALUE at the end of A]. set A [N+1] := VALUE.

Step-2: [Initialise counter] set LOC:=1

Step-3: [Search for VALUE]

Repeat while A[LOC] ≠ VALUE:

set LOC := LOC + 1

[End of loop].

Step-4: [Successful?] if LOC := N+1,

then : set LOC := 0

Step 5: Exit.

(b)

Ans:=> Inserting the key

$K = [29, 46, 18, 36, 48, 21, 24, 54]$

using the hash function $h'(k) = k \bmod m$

where $m = 11$

~~Empty Table After 29 After 46 After 18 After 36 After 43 After 21~~

	Empty Table	After 29	After 46	After 18	After 36	After 43	After 21	After 24	After 54
0						43	43	43	43
1							21	21	21
2			46	46	46	46	46	46	46
3					36	36	36	36	36
4								24	24
5.									54
6.									
7.		29	29	29	29	29	29	29	29
8.				18	18	18	18	18	18
9.									

(D)

Ay => Given numbers;
348, 143, 361, 423, 538, 128, 321, 543, 366

In Radix sorting, these numbers would be sorted in three phases;

★ <u>First Pass:</u>

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361		361								
423				423						
538									538	
128									128	
321		321								
543				543						
366							366			

✱ Second Pass:

Input	0	1	2	3	4	5	6	7	8	9
361							361			
321			321							
143					143					
423			423							
543					543					
366							366			
348					348					
538				538						
128			128							

✱ Third Pass:

Input	0	1	2	3	4	5	6	8	9
321				321					
423					423				
128		128							
538						538			
143		143			~~143~~				
543						543			
348				348					
361				361					
366				366					

Thus sorted nos. would be :-
128, 143, 321, 348, 361, 366, 423, 538, 543.