Sub Code: CS6104
Sub Name: Data structures and Algorithms

Reg No: 2019503519
Name: Hemanth.N

21.) KMP - Knuth Morris Pratt Algorithm

KMP algorithm is a linear time algorithm for string matching problems. Naive algorithm for pattern matching, where by window (window we move has a time complexity $O(m(n-m+1))$ in worst case. But KMP algorithm has $O(n)$ in the worst case. Major disadvantage of Naive algorithm is it doesn't work if many matching characters are followed by a mismatching character.

The above disadvantage is overcome by KMP algorithm. Here, whenever we detect a mismatch after some matches, we already know some of the characters in the text of the next window. So, we avoid matching characters that we know will anyway match.

- Prefix function: $\Pi$ for pattern - how the pattern matches against shift of itself. This is used to avoid useless shift of the pattern.
- KMP matcher: With string, pattern, prefix function $\Pi$ as inputs, the occurance of pattern in string and returns no of shifts in pattern after which occurance is found.

So, the basic problem statement would be given a string S, and P, $(P \leq S)$, find all occurances of P in S.

Date: 25/06/2021



Signature:

Sub code : CS6104

Sub name : Data structures and Algorithms

Reg No : 2019503519

Name : Hemanth. N.

Simple Pseudo code for Prefix function: Takes O(m) time

```
m ← length[P]        // P is pattern to be matched
π[1] ← 0
k ← 0
for q ← 2 to m
do while k > 0 and P[k+1] ≠ P[q]
do k ← π[k]
if P[k+1] = P[q]
  then k ← k+1
  π[q] ← k
Return π
```

For example,

Pattern is a b c d a b c

Prefixes → a, ab, abc, abcd, ...

Suffix → c, bc, abc, dabc

(abc - prefix & suffix match)

Computing π:

Say, P = a b c d a b e a b f

0 0 0 0 1 2 0 1 2 → (π values)

Let us take a proper example,

P: a b a b a c a

0 0 1 2 3 0 1 → [π[3] = 1, π[4] = 2, π[5] = 3]

(This is prefix computed for KMP matching)

Example: T is b a c b a b a b a c a c a

P is a b a b a c a

| P | a | b | a | b | a | c | a |
|---|---|---|---|---|---|---|---|
| π | 0 | 0 | 1 | 2 | 3 | 0 | 1 |

(comparing using KMP matcher algorithm)

T: b a c b a b a b a c a a b

P: a b a b a c

i & i+1 compare

Date : 25/06/2021

Pg No : 2

Signature :

Comparing i and j+1, if it match, move j and i

if b mismatch, move j to the index of b
j only moves
i doesn't move

T: b a c b a b a b a c a a b

P a b a b a c a

Backtrack on P P[1] & T[3]

i=4 => j=0 P[1] doesn't match with T[4]

At i=7 => P[3] = T[7]

a b , carry going on,

a b a b a c a matches from T[3] to T[13]

∴ Total shifts is 6 shifts

Time complexity as analyzed above is linear

Reg No: 2019503519

Name: Hemanth.N

Sub Code: CS6104

Sub Name: Data Structures and Algorithms

13.) Infix to Postfix conversion:

The algorithm is to use a stack to temporarily hold operators and left parenthesis. We begin by pushing a left parenthesis onto stack and adding a right parenthesis at end.

Basic Algorithm: (Infix exp is X)

① Push "(" onto stack and add ")" at end of X.

② Scan X from left to right and Repeat steps 3 and 6 for each element of X until the stack is empty.

③ If an operand is encountered, add it to Y.

④ If a left parenthesis is encountered, add to Y.

⑤ If operator is encountered,

(a) Repeatedly POP from stack and add Y to each operator (on top of stack) which has same precedence as or higher precendence than operator.

(b) Add operator to stack.

⑥ If right parenthesis is encountered,

(a) Repeatedly pop from stack and add to Y each operator (on top of stack) until a left parenthesis is encountered.

(b) Remove the left parenthesis

⑦ END.

Date: 25/06/2021

Pg No: 4

Signature:

Sub code : CS6104

Sub Name : Data structures and Algorithms

Reg No: 2019503519

Name : Hemanth. N.

Example :

A + (B * C - (D / E ↑ F) * G) * H to Postfix

| Symbol scanned | Stack | Expression |
|---|---|---|
| A | ( | A |
| + | (+ | A |
| ( | (+( | A |
| B | (+( | A B |
| * | (+(* | A B |
| C | (+(* | A B C |
| - | (+(- | A B C * |
| ( | (+(-( | A B C * |
| D | (+(-( | A B C * D |
| / | (+(-(/ | A B C * D |
| E | (+(-(/ | A B C * D E |
| ↑ | (+(-(/↑ | A B C * D E |
| F | (+(-(/↑ | A B C * D E |
| ) | (+(- | A B C * D E ↑ / |
| * | (+(-* | A B C * D E ↑ / |
| G | (+(-* | A B C * D E ↑ / G |
| ) | (+ | A B C * D E ↑ / G * - |
| * | (+* | A B C * D E ↑ / G * - |
| H | (+* | A B C * D E ↑ / G * - H |
| ) | - | A B C * D E ↑ / G * - H * + |

ANSwer : ABC * DEF ↑ / G * - H * +

Postfix

Date : 25/04/2021

Pg No : 5

Signature :

Sub code : CS6104

Sub name : Data structures and Algorithms

Reg NO: 2019503519

Name: Hemanth. N.

12.) Algorithm to insert a node X after Y (given), in DLL having Pointer p to first node.

```
Struct Node {
    int data;
    struct Node * prev;
    struct Node * Next;
}
```

Pointer p Pointing to first node (head)

```
Node * head = p;
```

Inserting New_data after Y.

```
Void ins_node_after_y (struct Node * y, int new_data)
```

Algorithm:

1. Start
2. Check whether Node y is NULL (ie, (y == NULL))
3. If yes, Print the given node cant be NULL as error message and exit.
4. Node is created for insertion
5. Put the data in New Node

```
New_node -> data = new_data;
```

Page No. 1

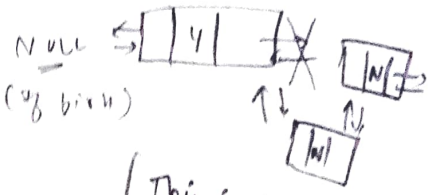
Signature:

(6) Make next of New_node as next of y

New_node -> next = y -> next

(7) Make next of y as new_node

y -> next = New_node




(8) Make y as previous of new_node

new_node -> prev = y ;

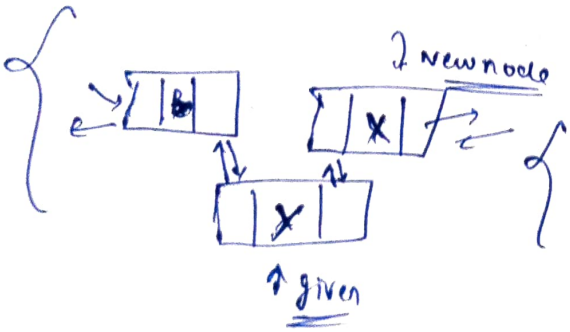
[ This is the main algo ]

(9) Previous of New_node's next node as new_node.

{ if (new_node -> next != NULL)

new_node -> next -> prev = new_node ; }

(10) END.




{ Making y's prev point to new node & Newnode next to y. }

Pseudo code

```
void insert after y (node *n, int d)
{
    node *temp;
    temp = new node();
    temp -> data = d;   temp -> prev = n;
    temp -> next = n -> next;   n -> next = temp;
    if (n -> next != NULL) n -> next -> prev = temp;
}
```

Sub code: CS6104

Sub Name: Data Structures and Algorithms

Reg NO: 2019503519

Name: Hemanth, N.

14.) Inorder and Preorder of Binary Trees:

Inorder:

① Traverse the left subtree
② Visit the root
③ Traverse the right subtree
④ Repeat until all nodes are visited

```
void inorder ( struct node * node )
{
    if (node == NULL)
        return;
    inorder (node -> left); // recurr on left child
    cout << node -> data << " "; // Print data on node
    inorder (node -> right); // recurr on right child nodes
}
```

```
struct Node {
    int data;
    struct node *left, *right;
    Node (int data)
    { this->data = data;
      left = right = NULL; }
};
```

Preorder:

① visit the root ② Traverse Left subtree ③ Traverse Right subtree & repeat till nodes are visited.

```
void Preorder ( struct node * node )
{
    if (node == NULL) return;
    cout << node -> data << " "; // Print data in node
    Preorder (node -> left); // recurr on left childs
    Preorder (node -> right); // recurr on right childs
}
```

Date: 25/08/2021

Pg NO: 8

Signature:

Sub Code : CS6104

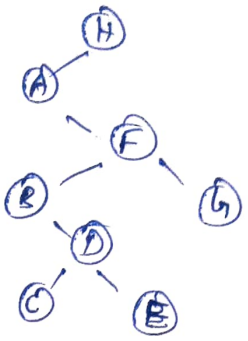
Sub Name : Data Structures and Algorithms

Reg No : 2019503519

Name : Hemanth. N.

Preorder of the tree is

H A F B D C E G




The node H is first visited, Then left subtree & then the Right subtree

Inorder of the tree is

A B C D E F G H

The left child nodes are first visited, then node, and then the right node is visited.

17.) Longest common subsequence is basically the longest subsequence common to the strings given. It must strictly be a increasing sequence.

A B A C D

A BA , AAC, AAD are subsequences but ACA is not a subsequence.

Date : 25/06/2021

PG NO : 9

Signature :

Sub code : CS6104

Sub Name : Data Structures and Algorithms

Reg NO : 2019503579

Name : Hemanth. N.

LCS using Dynamic Programming is a Bottom up approach.

A : a c d b c e  B : a b c b f e

Basic Pseudo code is

If (A[i] = B[j])

LCS [i,j] = 1 + LCS [i-1, j-1]

else

LCS [i,j] = Max ( LCS [i-1, j] , LCS [i, j-1])

| | 0 | a | b | c | b | f | e |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| c | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| d | 0 | 1 | 2 | 2 | 2 | 3 | 3 |
| b | 0 | 1 | 2 | 2 | 3 | 3 | 3 |
| c | 0 | 1 | 2 | 2 | 3 | 3 | 3 |
| e | 0 | 1 | 2 | 2 | 3 | 3 | 4 |

Follow the arrows
if diagonal => include
in sequence

a b c e

Length of LCS is 4

Following the arrows, we get LCS as a b c e

Date : 25/06/2021

Pg No : 10

Signature :

(b.) Greedy Method : Knapsack Problem:

Basically we must,

maximize $\sum_{1 \le i \le n} p_i x_i$ subject to $\sum_{1 \le i \le n} w_i x_i \le M$

and $0 \le x_i \le 1, \ 1 \le i \le n$

Given Problem,

*knapsack capacity = 50

| Item | profit | weight | Profit per weight |
|---|---|---|---|
| 1 | 100 | 20 | 5 (100/20) |
| 2 | 80 | 30 | 2.67 (80/30) |
| 3 | 110 | 10 | 11 (110/10) |
| 4 | 95 | 40 | 2.38 (95/40) |

(i) Add item 3 (weight=10) max profit per weight

Rem: 50-10 = 40

(ii) Add item 1 (20)

Remaining is 40-20 = 20

(iii) Add 2 item . Part of it, $20 - \frac{2}{3}(30) = 0$

$\therefore (1, 2/3, 1, 0)$



Signature :

Subcode : CS6104

Subname : Data Structures and Algorithms

Regno : 2019503579

Name : Hemanth. N.

Solution obtained is $(1, 2/3, 1, 0)$

Total Profit $= \sum p_i x_i$

$= 1 \times 100 + \frac{2}{3} \times 80 + 1 \times 110 + 0 \times 95$

$= 100 + 53.33 + 110 + 0 = 263.33$

Total Profit : 263.33

Total weight filled $= \sum w_i x_i$

$= 20(1) + 30(2/3) + 10(1) + 40(0)$

$= 50$ → (exact capacity of knapsack filled)

Sub Code : CS6104
Sub Name : Data structures and Algorithms

Reg NO : 2019503519
Name : Hemanth N

(9.) Heap sort:

① Build a max heap.

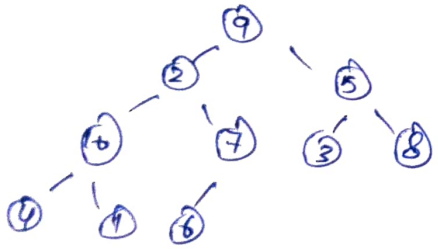
② Largest item is stored at root of heap. Replace it with last item of heap followed by reducing the size of heap by 1. Heapify the root of tree

③ Repeat ② until size of heap is > 1

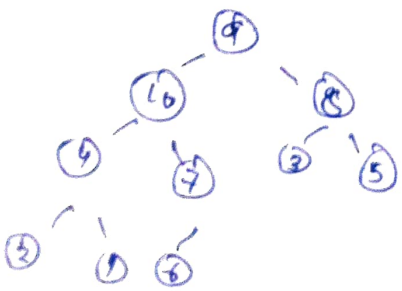
Heapification is performed in Bottom up order.

Time complexity of Heap sort is O(nLogn).

9, 2, 5, 10, 7, 3, 8, 4, 1, 6



(Building Max Heap)



10 is longer than 2 < 4

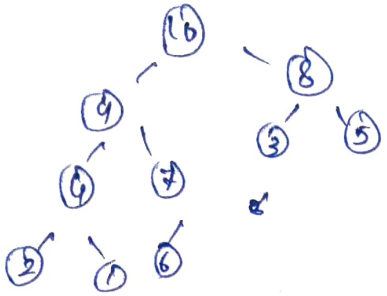
∴ (the following heap)

Sub Code : CS6104

Sub Name : Data structures and Algorithms

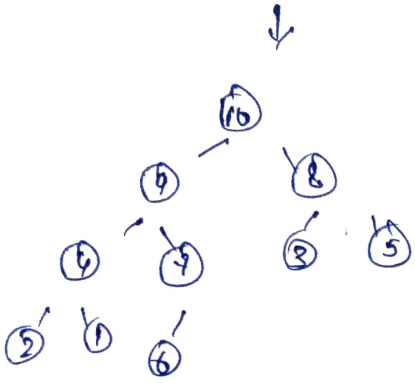
Reg NO: 2019503519

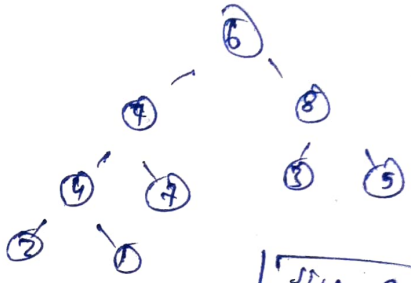
Name : Hemanth N.



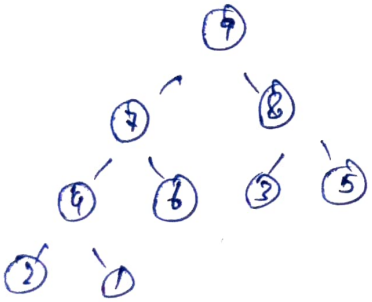
10 > 9

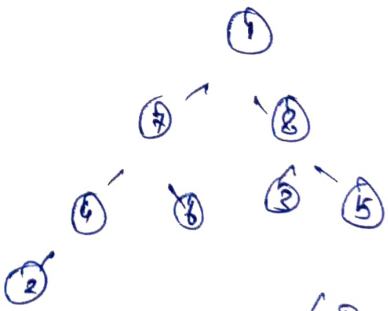
(This is the heap at this step)



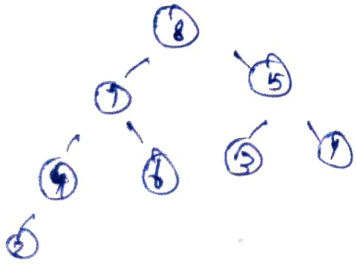


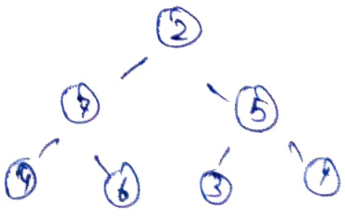
Size of heap reduced by 1





(The least elem is 1)





(Size of heap reduced by 1)

Date : 25/06/2021

Pg NO: 14

S/Signature :

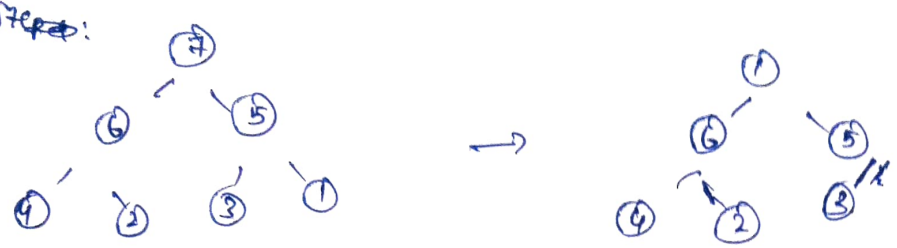
Sub code : CS6104

Sub Name : Data structures and Algorithms

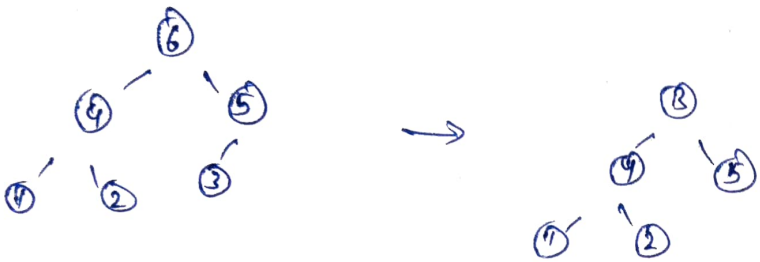
Reg NO : 2019503579

Name : Hemanth.N.

Steps:

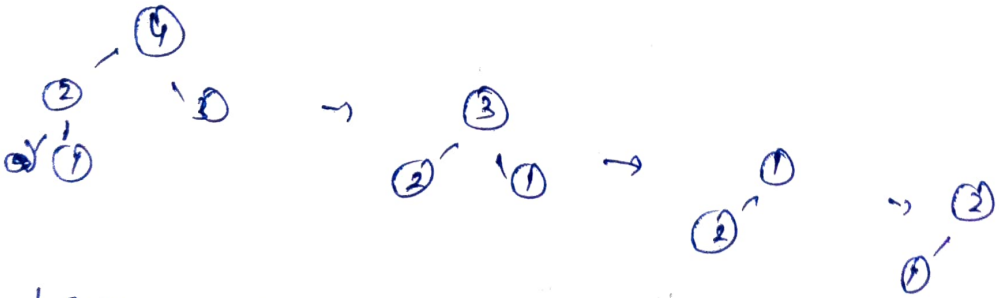


(Heap at each step is being shown)



(According to step 2, the size is reduced)

(Size of heap reduced)



Lastly, (1) will remain,

∴ The sorted Order is (increasing) of Heap sort

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|

Heap at each step each is shown properly

Pg no: 15

Signature :

20.) when there are more number of elements, say n, collision occurs in hash function when it produces the same value for more than one element.

If there is more than one element than no of cells, there might be high probability that 2 values might be placed in same cell of hash table.

There are many methods to Reduce collision.

Open addressing → Linear Probing
Quadratic Probing, etc.,

Collision can be reduced in

$\lambda = N/T$

N = no of elements, T = table size

$\lambda$ = load factor.

Method: Linear Probing → open addressing method.

If a cell is filled then value will be stored in next empty cell.

(This is also called Closed Hashing)

Example,

Consider a hash function, H(x) = x % 10 then 10, 20 occupy same place, alter this as

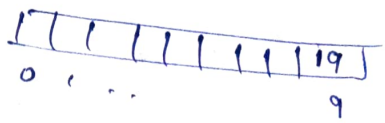
Hi(x) = (Hash(x) + bi)) % 10

b(i) = index of Hash table array

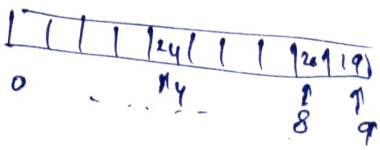
Load factor must be below 5.

Assume series 19, 28, 39, 24, 48, 9

19 is inserted at 9th index. =>

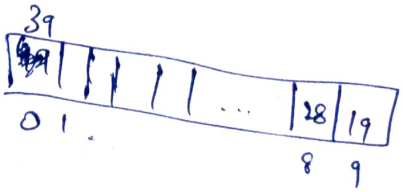



28 is inserted at 8th index =>




39 % 10 is 9, already filled

H(39) = (9+1) % 10 = 0 =>

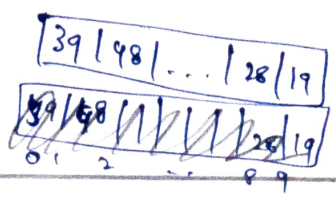



Take 48, again 8th index is filled

8+1 = 9 => also filled

8+2 = 10 % 10 = 0 => also filled

8+3 = 11 % 10 = 1 => not filled =>



Again take 9,

Now, 111y, 9, 0, are filled,

∴ (9+3) % 10 = 2 is where 9 will be filled.

∴

| 39 | 48 | 9 | | | | | 28 | 19 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | ... | | 8 | 9 | |

∴ Total Probe ⇒ 39 → 1

48 → 4 ∴ 9 → 4

Totally 9 Probes

{ Hence, this way, collision is avoided.
Linear Probing method, with a example is explained properly. }

Other Methods, like

Open Hashing, Quadratic Probing, Double Hashing etc. also reduce the collision in Hash Tables.

Sub code : CS6104

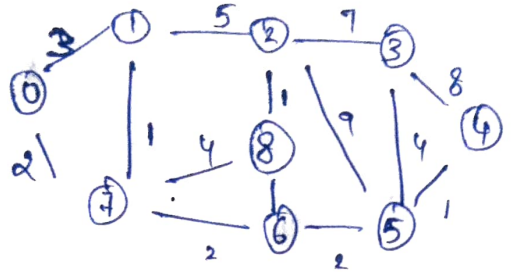
Sub Name : Data structures and Algorithms

Reg No: 2019503519

Name : Hemanth.M.

## PART - C

24.) **Kruskal Method:**




First step is to sort them in decreasing order of weight

| A → | B | Weight |
|---|---|---|
| 1 | 7 | 1 |
| 2 | 8 | 1 |
| 4 | 5 | 1 |
| 0 | 7 | 2 |
| 7 | 6 | 2 |
| 6 | 5 | 2 |
| 0 | 1 | 3 |
| 7 | 8 | 4 |
| 3 | 5 | 4 |
| 1 | 2 | 5 |
| 8 | 6 | 6 |
| 2 | 3 | 7 |
| 3 | 4 | 8 |
| 2 | 5 | 9 |

Basically, a MST will have (no of vertices - 1) edges.

∴ 8 edges in our final MST

Reg No: 2019503519

Name : Hemanth, N.

Sub code : CS6104

Sub Name : Data structures and Algorithms

## Kruskal Algorithm:

① Sort all the edges in non-decreasing order of their weight.

② Pick the smallest edge. Check if cycle is formed.

③ In case, no cycle is formed, include that edge in graph.

④ If cycle is formed, Discard that edge.

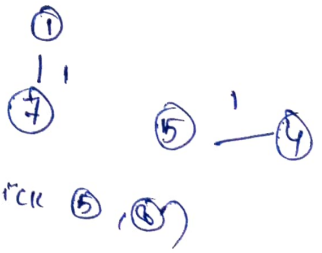
⑤ Repeat the above 2 steps until (V-1) edges are in spanning tree.

## Solving the question,

Step 1:

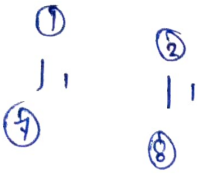

(Pick 1 7)

2:




(Pick 5, 6)

Step 3:




5 — 8 (Pick 2 — 8)

The edges are picked and included, since it doesn't form a cycle.

Date : 25/06/2021

Pg No: 20

Signature:

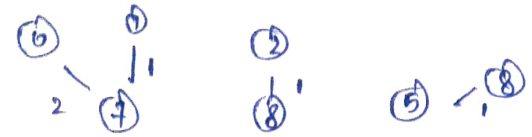
Subject code : CS6104

Subject Name : Data structures and Algorithms

Reg No: 2019503519

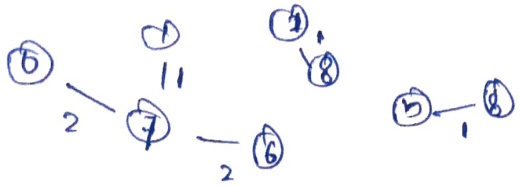
Name : Hemanth.N

Step 4:



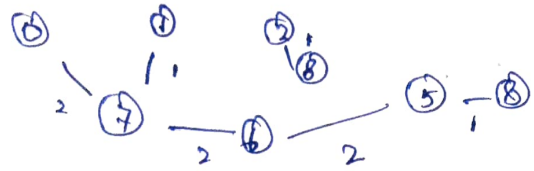
Pick (8)-(2)

Step 5 :



(These edges are all included since it doesn't form a cycle)

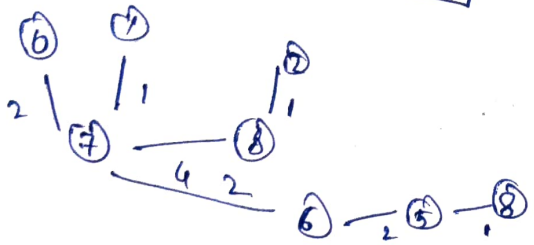
Step 6 :



[Pick (6)-(5)]

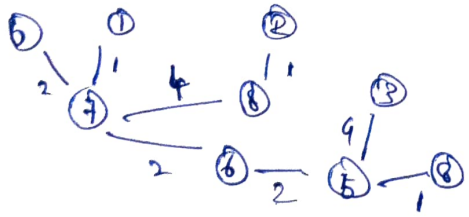
Including 0-1 will form a cycle Thus discarded

7:



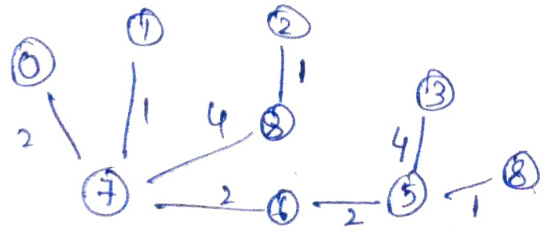
Picking (7)-(8) (not form cycle)

8:



Picking (5)-(3) (Not form cycle)

Thus, the final MST has 8 edges



→ MST using Kruskal Algorithm

Date : 25/06/2021

Pg No: 21

Signature :

Sub Code : CS6104

Sub Name : Data structures and Algorithms

Reg NO : 2019503519

Name : Hemanth.N.

23.) $T(n) = 4T(n/2) + n \quad ; \quad n>1, T(n)=1, n=1$

Substitution method:

$T(n) = 4T(n/2) + n \longrightarrow (1)$

when $n = n/2$ condition,

$\Rightarrow T(n/2) = 4T(n/4) + n/2 \longrightarrow (2)$

$T(n) = 4(4T(n/4) + n/2) + n$ (Sub (2) in (1))

$= 16T(n/4) + 3n \longrightarrow (3)$

$n = n/4$

$T(n/4) = 4T(n/8) + n/4 \longrightarrow (4)$

Sub (3) in (4)

$T(n) = 4(16[4T(n/8) + n/4) + 3n$

$= 64T(n/8) + 4n + 3n$

$= 64T(n/8) + 7n$

$T(n) = 64T(n/8) + n + 2n + 4n$

we observe a series,

$\{4^3 T(n/2^3) + n + 2n + 4n \ldots\}$

Date: 25/06/2021

PgNO: 22

Signature: Hemanth

$$T(n) = 4^k [1 + n(1+2+4\cdots+k)]$$

$$= 2^{2k} + n(1+2+4+\cdots+k) \quad \text{---} (5)$$

$$2^k = n \quad \therefore$$

$$k = \log_2 n \quad \text{---} (6)$$

Sub (6) in (5)

$$T(n) = n^2 + n(1+2+4\cdots k)$$

$$= O(n^2)$$

$$T(n) = n\left(2\frac{\log_2 n e}{2+1} - 1\right) + 4^{\log n}$$

$$= O(n^2)$$

∴ By substitution method, $T(n) = O(n^2)$

**Master Method:**

$f(n) = n$

$a = 4, \; b = 2$

$$n^{\log_b (a-\epsilon)}$$

$$n^{\log_2 4} = n^{2\log_2 2} = n^2$$

$f(n) = n^2$

This is case (iii)

$\therefore$ So, $T(n) = \Theta(n^{\log_b a})$

$= \Theta(n^{\log_2 4})$

$= \Theta(n^2)$

$\therefore$ By Master's Method, $T(n) = \Theta(n^2)$

We observe that time complexity using both substitution & Master's Method is same, (ie) $\Theta(n^2)$.

Code: CS6104

Subname: Data structures and Algorithms

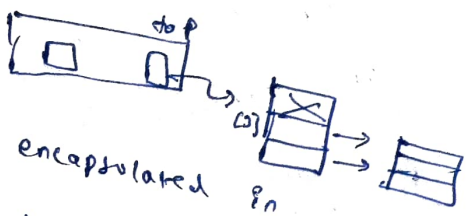
Regno: 2019503519

Name: Hemanth N

# PART-A

1.) Data abstraction basically means providing only the essential and relevant information ~~through~~ to the end user. It separates the specification of a data type from its implementation.

For example, take stack as a Abstract data type. Instead of data being stored in each node, the pointer to Data is stored.




Head node & data nodes are encapsulated in ADT. So, the user only needs to know what a data type is and it can do, need not know how it is implemented.

2.) Queue is a Linear Data structure since elements ~~for~~ are linked in a sequential manner. It is a first In first out data structure.

There can be only one descendant at any node, unlike Trees (more nodes connected to given node) which are Non-Linear.

∴ Since only one node is connected, and forms a sequence, Queues are Linear Data Structures.

Date: 25/06/2021

Page no: 25

Signature:

Subcode : CS6104

Subname : Data structures and Algorithms

Reg No : 2019503579

Name : Hemanth, N.

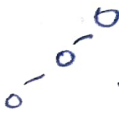
3.) Min & Max no of nodes in a Bin Tree of height k

Minimum = k+1

Maximum = $2^{k+1} - 1$

Max is $2^{k+1} - 1$

$2^{3+1} - 1 = 15$



→ (Left skew would be minimum k+1)

4.) Time complexity of Naive string Matching

Given text → (0 ... n-1)

Given Pattern → (0 ... m-1) n>m

Worst case is $O(m * (n-m+1))$

Avg Time complexity of Naive Algo → $O(m*n)$

(m is size of Pattern
n is size of string)

5) No of different spanning tree with complete graph of n nodes } = $n^{n-2}$

5 nodes => $5^{5-2} = 5^3 =$ 125 Different spanning trees

Sub Code : CS6104

Sub Name : Data Structures and Algorithms

Reg No: 2019503579

Name : Hemanth N

4.) Divide and conquer can't be applied to given problem.

(A) Cannot be divided into sub problems

(B) cannot be recursively solved using sub problems

(C) Is not a combination of subproblem questions.

eg.) Linear search cannot be solved using divide & conquer strategy since it involves traversal of the whole array.

Bubble sort is also not solved using Divide & Conquer since it involves traversing the whole array and bubbling the largest to top.

6.) → Divide & conquer is a top down approach while Dynamic programing is Bottom up approach.

→ Divide & conquer does more on sub-problems, hence more time consumption, DP solves the sub problems only once, hence less time consumption.

→ Divide & conquer → sub problems are independent

DP → sub problems → Interdependent

Divide & conquer → Merge sort, Quick sort | DP → Matrix Chain multiplication

Date : 21/04/2021

PS NO : 27

Signature :

Sub code : CS6104

Sub Name : Data Structures and Algorithms

Reg No: 2019503519

Name : Hemanth, N.

7.) Bounding function is needed to kill live nodes without expanding. do, for optimization, bounding function creates upper bound or lower bound on functions to ~~create~~ tackle minimization or maximization problems.

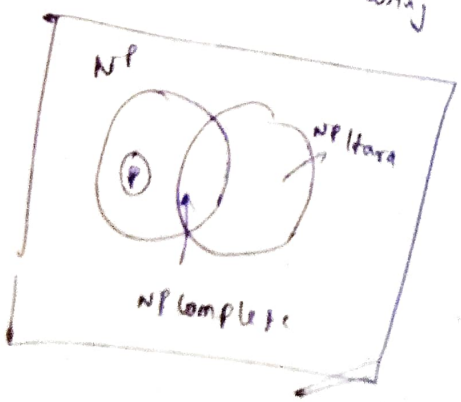
10.) NP is a set of problems for which problem instances where the answer is yes. two types:

NP Hard & NP complete

NP Hard : All problems in NP are polynomial time reducible

NP complete : If Language is NP and is polynomially reducible

NP class contains problems that can't be solved in polynomial time using deterministic Algorithms.




$P \subseteq NP$

If (Sat is in P if $P = N$)



Signature:

Date: 26/06/2021

Sub Code : CS6104

Sub name : Data Structures and Algorithms

Reg No : 2019502519

Name : Hemanth N.

8.)

Heap is a specialized tree-based Data structure which is complete tree satisfying heap property.

Heaps can be implemented as an array, with each element representing a node of the heap.

various types of Heap: Binary Heap, Radix Heap, etc..

Application of Heaps are:

Heap sort, Priority Queue, k-way Merge.

(a) In a Max Heap, for any given node C, if P is parent node of C, then key of P $\geq$ key of C.

(b) In min heap, the key of P $\leq$ key of C.

(c) Node at top of Heap is Root Node.

All the Answers in this Answers Book have been in my own Handwriting. Nobody helped me in writing the answers.

25/06/21