Imp I → no. of internal nodes
L → no. of leaf nodes

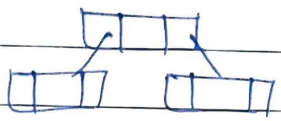
$I(n-1)+1=L$

n → type of n-ary tree

Date
Page No.

Various tree representations

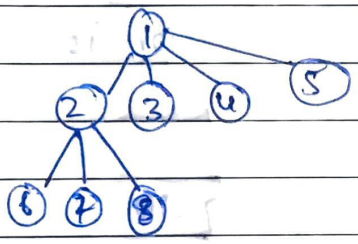
① Structure Representation using pointers.



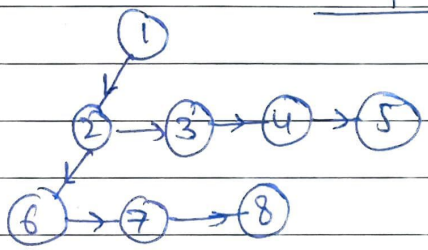
→ It works for binary tree.

But if children > 2 then, Assume 10, then giving 10 pointers is just a waste of space. So we use -1

② LCRSR → Left child Right Sibling Representation using pointers.

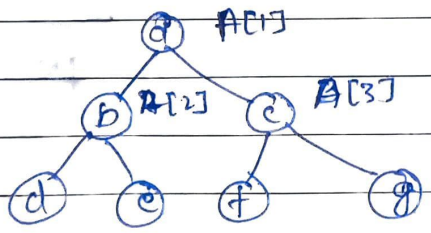


→



for very high no. of children

③ Array representation for Binary tree



	1	2	3	4	5	6	7
A	a	b	c	d	e	f	g

i → node root (Parent)

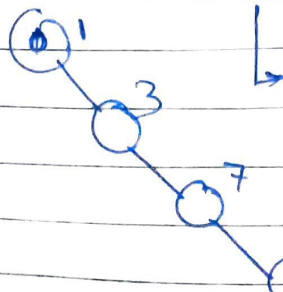
LC = 2i

RC = 2i+1

Almost complete binary tree or heap

Advantage → Storing a heap kind of tree

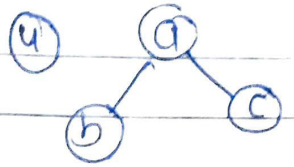
Disadvantage → Skew tree representation.



↳ for n nodes You need 2^n-1 space

$O(2^n)$.

Date :
Page No.

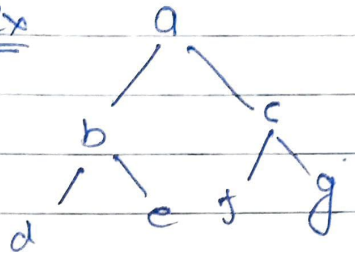


$\Rightarrow$ (b a c) = (a . b c)

Root L R — Root L R

Nested form

Ex



a(bde)(cfg) — [Root Left Right form]

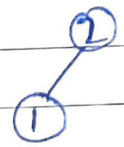
Not generally used.

Gate-2000 Nested Tree representation

Inside Paranthesis ~~atleast~~ only three things required

If there is no left or child then represent as NULL

Ex

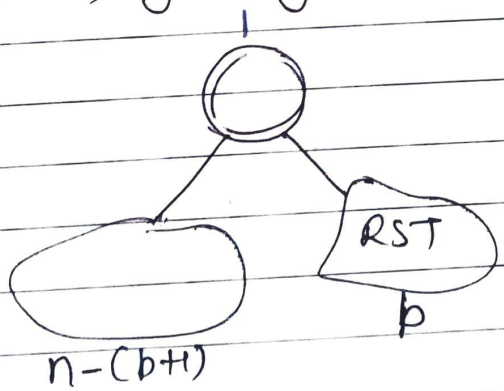


(2 1 NULL)

((2,3,4) 5 6) — Not possible because Root have only single node not a paranthesis.

Root 2 R

Question on guessing the root node :-



All the n-(p+1) are less than Root so Root will we the next element after n-(p+1) so

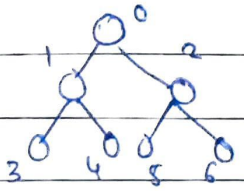
Root = n-p-1+1

= n-p

[n-(p+1)] (n-p) [n-p+1 ... n]

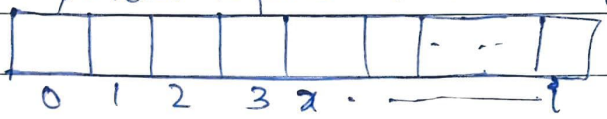
LST Root p

Gate IT 06 Array representation of binary tree



0	1	2	3	4	5	6

Let the element present at i^{th} position and parent is present at x^{th} position then



for left child (ith) $i - x - 1 = x$ → No. of nodes btw x and i

$\Rightarrow x = \left(\frac{i-1}{2}\right) = \left\lfloor \frac{i-2}{2} \right\rfloor$ OR $\left\lfloor \frac{i-1}{2} \right\rfloor$

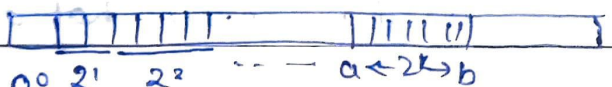
for right child (ith) $i - x - 1 = x + 1$

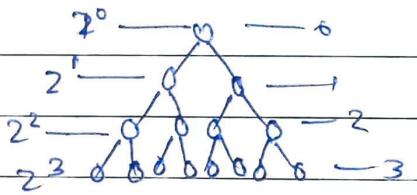
$\Rightarrow x = \left(\frac{i-2}{2}\right) = \left\lfloor \frac{i-2}{2} \right\rfloor$ OR $\left\lfloor \frac{i-1}{2} \right\rfloor$

→ Both for left child and right child

Q Next problem (Level of element x[i])

Let 'i' is at level 'k'





$a \le i \le b$

$a = 2^0 + 2^1 + \dots + 2^{k-1}$

$= 2^k - 1$

$b = a + 2^k - 1$

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

$a \le i$

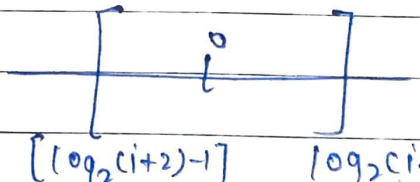
$2^k - 1 < i$

$\Rightarrow \boxed{k \le \log_2(i+1)}$

$b \ge i$

$\Rightarrow 2^{k+1} \ge i + 2$

$\Rightarrow k \ge (\log_2(i+2) - 1)$



So, Ans → $\lfloor \log_2(i+1) \rfloor$ or $\lceil \log_2(i+2) - 1 \rceil$

Date :
Page No.

Gate-2014 Leftmost child and right siblings representation of tree.

Defaulted Question - All the options were incorrect

Additional Information regarding Tree

Spanning Tree → In Algorithms Notebook (Greedy Algorithms)

(4) Tree search (x, k)
if x ≠ NIL or k = key[x]
then return x
if k < key[x]
then return Treesearch (left[x], k)
else ——————— (right[x], k)

(5) Tree min(x)
while left[x] ≠ NIL
do x ← left[x]
return x

(6) Tree max(x)
while right[x] ≠ NIL
do x ← right[x]
return x

(7) Tree successor(x)
if right[x] ≠ NIL.

(8) Tree predecessor

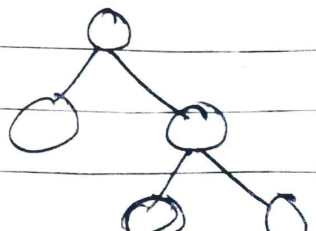
(9) Tree Insert

y ← NIL
x ← root[T]
while x ≠ NIL
do y ← x

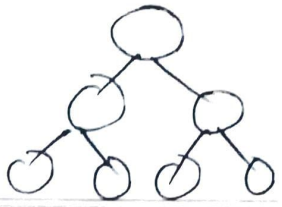
(10) Tree delete

Types of Binary tree

(1) Strict binary tree: if each node has exactly two or no children.

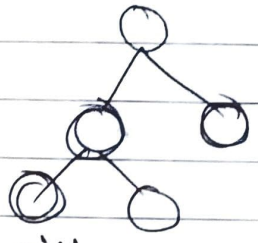


(2) full binary tree



exactly two children and all leaves at same level.

(3) Complete binary tree → On traversing top to bottom, left to right we found all the no. without any missing node and all leaves found at either height 'h' or 'h-1'.



→ In full binary tree with 'h' levels no. of nodes are $2^{h+1}-1$.

→ In complete binary trees of level 'h' min. nodes (2^h), max. nodes $(2^{h+1}-1)$.

→ No. of leaves in full binary tree = 2^h.

→ No. of Null links in a complete binary tree is 'n+1'.

→ No. of min. heaps formed by n-distinct nodes

$$= {}^{N-1}C_r \; T(r) \; (T(N-r-1))$$

$$r = \left\lceil \frac{N-1}{2} \right\rceil$$

Day 1 — C Programming and Data Structures

HLL → Preprocessor → Compiler → Assembler → Loader/Linker

→ long long double doesn't exist and unsigned can be used with double.

* signed, unsigned generally used with int, char. char is generally unsigned only.

* Array, Pointer, structure, Union, Enum are user define datatype.

* Const float pi = 3.14 — ① ; # define pi 3.14 — ② ← If you try to change value in ① it will show an error.

* Left Shift = $A \times 2^n$, Right Shift = $A / 2^n$

* "a" : return string, 'a'; return ASCII, "a"+1 : error, 'a'+1; ASCII

Precedence: [( ), [ ], → , .] : have highest precedence. [++, --]

Order U A, S C B L T A_2 C → Assignment

U → Unary, Arithmatic
A, S → Arithmatic, Shift
C → Conditional
B → Bitwise
L → Logical
A_2 → Assignment
C → Comma

U, T, A_2 have Right associativity.

→ Tab means 4 spaces, remember the short circuit rules.

→ Post increment / decrement have more priority then pre increment / decrement.

* Take care of break instruction in switch statements.

* Armstrong no: $373 = 3^3 + 7^3 + 3^3$

→ Parameter passed: actual, Parameter received: formal.

→ Pass by value / reference can change swap results

* We can't assign expression to pointer variables.

* We can't perform ~~over~~ arithmatic over array mnemonics

* char a[] = "aryan"; — ① ; char *p = "aryan"; — ② ← we can perform changes only in ① but we can access both.
→ changing value gives segmentation fault.

Specifier	Storage	Initial V.	Scope	Life
auto	stack	GV	within block	End of block
extern	DS	0	Global	end of prog.
static	DS	0	within block	,,
register	Register	GV	,,	end of block.

→ A[i] = i[a], A[][][] ↳plane, declaration of col is compulsory.

List of string functions :-

① Strcat (s,t), ② Strncat (s,t,n), ③ Strcmp (s,t), Strncmp (s,t,n)

④ Strcpy (s,t) ⑤ Strncpy (s,t,n) ⑥ Strrev (s), ⑦ Strlen (s)

⑧ Strchr (s,c): return pointer to location where 'c' appears first otherwise NULL

⑨ Strrchr (s,c) ⑩ Strstr ("d", "b") ⑪ Strset (str, "c")

→ If b is pointer variable then b++ means next address.

→ Scanf returns no. of format specifier and printf returns no of character in string).

File Handling function

fopen(), getc(), putc(), fscanf() → read set of data, fprintf()
getw(), putw(), fseek(): set position to desire pointer position, ftell()

Tower of Hanoi

Equation:

$T(n) = 2T(n-1) + 1$

$T(0) = 1$.

Results

No. of Invocation: $2^{n+1} - 1$

No. of movements = $2^n - 1$

$T(n) = O(2^n)$

$S(n) = O(n)$

Linked List

→ Insertion of a new node always occur at begging if nothing is given.

→ Insertion O(1), Searching O(n), Deletion O(n)

→ To reverse a SLL we need two more pointers.

Reverse

```
*prev = *nextnode = 0
while(curr){
nextnode = curr->next;
curr->next = prev;
    prev = curr;
    curr = nextnode;
}
```

→ CLL manages computer resources in round robin.

→ Backup opn done by DLL in constant time

In DLL: Deletion O(1), Insert O(n), find O(n)

→ M/m efficient DLL used only one pointer by doing X-OR of prev and next.

In Stack MAX or any pointer pointing top of stack Initialize to −1.

T(n) for Push() or Pop() = O(1)

In queue (circular) : front == (rear+1) mod n : full

front == rear : empty.

Initially front = rear = 0, on inserting first element increment both by 1 to (0 → 1) then for insertion increment rear only.

Graph

① In Matrix representation S(n) = O(V²) = O(E) : dense graph

② In List representation S(n) = O(V+E) : O(E) = O(V) ↳ sparse graph

In BFS : we use Queue for explored O(V)
we use array for visited O(V)] → O(V)

T(n) → O(E+V) for List
→ O(V²) for Matrix

In DFS : for both directed or undirected

S(n) = O(V)

T(n) = O(E+V) or O(V²)

Topological Sort : for directed acyclic graph only

T(n) = O(V+E)

→ In undirected graph there can be maximum of n(n−1)/2 edges.

Hashing : Performed searching in O(1).

→ Probe the key into any Data structure using hash function, but there is a problem of collision.

→ To resolve collision.

① Choose suitable Hash fn.

② Chaining

③ Open Addressing [Close Hashing]

→ In chaining worst case O(n) but if uniformly distributed

$O(1+\frac{n}{m}) = O(1)$ where $\frac{n}{m}$ is load factor.

n = no. of elements

m = no. of slots

① **Linear Probing** : Primary clustering

$h(k) = k \bmod n$ or $h'(k) = (k+i) \bmod n$

where, i = no. of collisions.

② **Quadratic Probing** : Secondary clustering

$h'(k) = (k+i^2) \bmod n$

③ **Double hashing**

$h^1(k) = k \bmod n$ $\quad h^2(k) = k' \bmod n$

→ ① $h(k) = k \bmod n$ ② $h(k) = k \bmod n + k' \bmod n$

③ $h(k) = k \bmod n + 2 * k' \bmod n$ ----

Trees : Traversal tech. : In, Pre, Post, Level [$T(n) = S(n) = O(n)$]

→ Space complexity depends on height of tree, it can be $O(n)$ for skew trees.

No. of binary tree possible with n nodes → Unlabelled = $^{2n}C_n/n+1$ → Labelled = $(^{2n}C_n/n+1)\,n!$

→ There is only one tree satisfying all 3 traversal techniques

No. of nodes

```
if (t)
{
int l,r;
l=NN(t->left);
r=NN(t->right);
return(1+l+r);
}
```

No. of leaves.

```
if(t==Null)
return 0;
if (!t->left && !t->right)
return 1;
else
return (NL(t->left)+NL(t->right))
```

for height

```
return (1+max(l,r));
```

→ Inorder always in ascending order

→ No. of BST with n distinct keys = $^{2n}C_n/n+1$

→ In n-ary tree with X internal nodes : $X(n-1)+1 = L$ [Leaf]

BST	BBST	opn
$O(n)$	$O(\log n)$	Search
$O(n)$	$(\log n)$	Height
$O(n)$	$(\log n)$	Insertion

→ Max. height of AVL = $\lceil \log_2 n \rceil$

→ In AVL of height h max. no. of nodes = $2^{h+1} - 1$

→ Min. no. of nodes = $2^{h-1} + 1$

→ for max. nodes we get min. height.

Strict binary tree: If each node has exactly two or zero children.

full binary tree: Each will have exactly two children and all leaves are at same level.

Complete binary tree: Same as full binary tree but leaves can found at (h) or (h-1) levels

→ In full binary tree with h-levels : $2^{h+1}-1$ (max) (no. of nodes)

→ In complete binary trees of h-level min: 2^h, max: $2^{h+1}-1$

→ No. of leaves in full binary tree = 2^h (max)

→ No. of Null links in a complete binary tree = $n+1$ (max)

→ No. of min. heaps formed by n-distinct nodes;

$T(1)=1$, $T(2)=1$, $T(3)=2$

$N-1C_r \; T(r)\, T(N-r-1)$; $r = \lfloor \frac{N-1}{2} \rfloor$ no. of nodes in Left subtree of root

→ In a tree of height (h) and no. of nodes (n) then, max nodes present at height $h = \left\lceil \frac{n}{2^{h+1}} \right\rceil$

Ex → `char *str = "abc";`
`char str2[] = "abc";`
`sizeof(str) = 2` byte size of pointer
`sizeof(str2) = 4`
`sizeof("abc") = 4`

→ In a hash table if no. of collisions are n then no. of comparisons = $n+1$

→ $(x++)+y$ $\boxed{x++ + y}$

→ `strset()` is used to set all characters of a string with single char.

Imp. Questions

① **Programming:** 2, 4, 6, 7, 11, 14, 17, 19, 21, 24, 29, 31, 37, 45, 52, 54, 55, 56, 58, 64, 65, 72, 84, 85, 89, 94, 97, 98, 99, 105, 108, 116, 118, 123, 131, 133, 134

② **Arrays:** 1, 6, 7, 9, 11, 12

③ **S&Q:** 4, 6, 9, 10, 15, 16, 21

⑤ **Trees:** 2, 8, 11, 13, 17, 34, 39, 46, 47, 48, 55, 62, 63, 65, 76, 78, 79

⑥ **Graph:** 2, 5, 7, 9, 10, 11, 12, 17, 21, 22

⑦ **Hashing:** 1, 6, 11.

→ In complete binary tree no. of leaf nodes = $\left\lceil \frac{n+1}{2} \right\rceil$

→ void pointer;
`void *p;`

→ In 2-D array, (x+k) and *(x+k) are same and denoted the address.

→ goto statement has no effect on memory and performance

char a[20] = "Aryan";
char b[20] = {'a','r','y','\0'};
char c[20];

] strlen(a) = 5
strlen(b) = 3

→ Data structure used to store data on HD is tree.

→ C is procedural, strictly typed, middle level language.

→ void pointer needs to typecast to particular data type to dereference or perform arithmetic operation

→ Const int x; it stores 0 by default otherwise initilize while define

extern int v;
main(){
return 0;
}
(Right)

no address allotted

extern int v;
main(){
v = 10;
return 0;
}
(wrong)

#include <somefile.h>
extern int v;
main(){
v = 10
retur 0;
}
(Right)

ptr = realloc (ptr, 10 * sizeof(int));
ptr = (int *) malloc (20 * sizeof());
ptr = (int *) calloc (20, sizeof());

] → Dynamic m/m allocation

→ malloc, calloc initiated as void pointer, so we need to typecast it to

→ In a binary tree with 'n' nodes, time require to find no of subtrees with 'k' nodes is O(n).

→ In Postfix → Infix : Scan L → R and if operator found after two operands as AB+, do (A+B) and again scan L → R.

→ In Prefix → Infix : Scan R → L and if operator preceded by two operand as /AB, do A+B at that instant and again start R → L.

→ Using DLL we can perform both Push & Pop() in O(1) time

→ $H(h) = 1 + H(h-1) + H(h-2)$: AVL tree

→ In doubly LL organization insertion at end involves modification of only 1 pointer in existing list.

→ No. of BST possible with only 1 leaf node such that all nodes are numbered distinctly = 2^{n-1} for n-nodes.

→ The max. height of a BFS Tree if BFS is running on a $(K_{m,n})$ complete Bipartite graph is 2.

→ In structure a pointer to structure access through '→' operator and a normal variable access its member through '.'.

→ Rotation in AVL always preserves in-order and the median of elements in AVL tree doesn't guarantee present at root or one of the children bcoz LST and RST doesn't have equal no. of elements always.

→ (POST, IN), (PRE, IN), (LVL, IN) are uniquely sufficient to construct the binary tree.

Time taken to find a non-existence element in best case-
① BST → O(1) ② AVL → O(log n) ③ Binary Tree → O(n).
④ Minheap → O(n).

→ Calling same fn within causes stack would overflow and eventually the program will be killed by OS (infinite loop)

```
fun(3){
fun(3);
}
```

→ Representation of 2D array A[10][10] can be show as
int *A[], int A[][10], int [10][5]. [└──┘]
int A[], int A[10][] (X) Wrong.

→ Void pointer can't be use for dereferencing bcoz each pointer can take different size without typecasting.

→ A particular extern variable can be declared many times but initialized once.

→ Arithmatic operation doesn't work on void pointers.

→ Stack implementation using DLL takes O(1) for both push and pop.

→ RR, LL → 1 Rotation, LR, RL → 2 rotations.

→ If base add = 10K then compute the total no. of elements before the number and multiply with size and add to base add.

→ static int y=10(└──┘)

```
int x=10;
static int y = x; (X)
```

→ compile error
Only with constant literals

→ Worst case T(n) to find path btw two vertices of a graph with 'm' edges and 'n' vertices = O(m+n) : BFS.

→ DFS takes low space as compare to BFS but BFS is faster than DFS.

The depth of the DFS tree rooted at a vertex is at least as much as the depth of the BFS tree rooted at same vertex.

→ In G=(V,E) if all edge cost are distinct then,
① Cost of MST and MST itself should be unique
② If it contains two parts join by a edge of max. cost then we can hold max. cost in MST.

We can make an unique BST using (POST, IN), (PRE, IN) in O(n log n) time.

→ If the TOEPLITZ (nxn) matrix needs to store in 1-D array optimally then size of array = 2n-1.

→ In printf( ), expression evaluates R→L and prints L→R.

→ In expression tree Preorder, Postorder, Inorder represents the Prefix, Postfix and Infix of expression.

→ Extern storage class have global visibility

→ If str = "Aryan" then sizeof(str) = 6, strlen(str) = 5.

→ for strcpy(str1, str2) and strcpy(str1, "Well") both are correct).

→ ASCII values
'0' → 0, 'a' → 97, 0 → 48
A → 65, '\n' → 10, space → 32

→ by default sizeof() returns sizeof integer.

Rank / Index of a node in BST = no. of nodes in LST + 1.

→ If topological sort is applicable on a graph (directed) then its DFS will produce no back edges.

→ If (lower triangular matrix) represent as 1-D array (index start from 0) and index of matrix start from 1 then
array index = $(j-1) + (i-1)i/2$

→ Node(LST) + Node(RST) + 1 = Total no of node
→ Internal node + Leaf node = ,,

→ A queue is implemented using 2 stack then in worst case one opn from enque and deque will take O(n) and other will take O(1).

→ int main (int argc , char *+ argv)

1. program hello coolol int argc = 3 ;
   argv[0] argv[1] argv[2]

%d → int , %u → unsigned int , %hu → short unsigned int ,
%hd → short int , %ld → long int , %lld → long long int ,
%llu → unsigned long long int , %c → both signed or unsigned char.
%f → float , %lf → double , %Lf → long double.

→ long long double doesn't exist.

(), [], →, ., ++, --
+, -, !, ~, ++, --, (type)*, sizeof()
*, /, %
+, -
<<, >>
<, <=, >, >=
==, !=
& (AND)
^ (X-OR)
| (OR)
AND &&
OR ||
?: (Ternary)
Assignment
comma.

→ **full binary tree**: Each node has either '0' or '2' children.
→ No. of leafs = (No. of Internal Node) + 1

→ **Complete binary tree**: Tree with all levels are filled except the last level has all the keys as left as possible.

→ **Perfect Binary tree**: All internal nodes have two children and all leaves are at same level.

→ **Degenerate / pathological tree**: Each internal node has only one children. Working is same as LL.

→ Void pointer can store the address of any data type.

→ $n^{\log_2 2^2} = 2^{\log_2 n}$

or $a^{\log_c b} = b^{\log_c a}$