GATE 2019

DATA STRUCTURE & ALGORITHM (INCLUDING C)

COMPUTER SCIENCE

ECG Publications

ECG
Publications

A Unit of **ENGINEERS CAREER GROUP**

Head O ce: S.C.O-121-122-123, 2nd oor, Sector-34/A, Chandigarh-160022

Website: www.engineerscareergroup.in **Toll Free:** 1800-270-4242

E-Mail: ecgpublica ons@gmail.com | info@engineerscareergroup.in

GATE-2019: Data Structure & Algorithm (including C)| Detailed theory with GATE previous year papers and detailed solu ons.

First Edi on: 2016

Price of Book: *INR 420/-*

CONTENTS

SECTION-A (PROGRAMMING IN C)

SECTION-B (DATA STRUCTURE)

SECTION-C (ALGORITHM)

SECTION-A
(PROGRAMMING IN C)

CHAPTER - 1
BASICS

1.1 INTRODUCTION

C is a remarkable language. Originally designed by Dennis Ritchie, working at AT & T Bell laboratories in New Jersey.
C is a structured language. It allows verity of programs in small modules. It is easy for debugging, testing and maintenance if a language is a structured one.
C have a low level access to memory, simple set of keywords and clean style. Many later languages have borrowed syntax directly or indirectly from C language. For example Java, PHP, Java Script and many other languages are mainly based on C language. C^{++} is nearly a superset of C language.

1.2 STRUCTURE OF C PROGRAM

```
Include header file section
Global declaration section
Main ( )
{
    Declaration Part
    Executable Part
}
User defined functions
{
    Statements
}
```

Let's write our first C Program (ECG.C)

```
# include < stdio.h >
int main (void)
    {
        Print f (''Engineers Career Group'');
        Return o;
    }
```

Let's Analyze the Program line by line

1. **Include < Stdio.h >**
All the lines starting with # are processed by preprocessor. Preprocessor is a program that is called by complier. In another words we can say preprocessor will take input program given by programmer and will produces output program where, there are no lines starting with #, all those lines processed by preprocessor.
Here, # include <stdio.h> will be replaced by file 'stdio.h' which declares the print f( ) function, by preprocessor.

2. Int Main (void)
In C program, there is a fixed point from where execution of complied c program begins, i.e. main ( ) function int written before main ( ) indicates return type of main ( ) and (void) indicates that main ( ) function doesn't take any parameters.

GATE QUESTIONS

1. Consider the following C program:

```
#include<stdio.h>
int counter = 0;
int calc (in a, int b) {
     int c;
     counter ++;
     if (b =  = 3) return (a*a*a);
     else  {
        c=calc (a, b/3);
        return (c*c*c);
   }
}
int main ( )  {
    calc (4, 81);
    print ("%%d", counter);
}
```

The output of this program is _______.

(GATE - 2018)

Sol. 1. (4)

2. Consider the following C program:

```
#include<stdio.h>
void fun 1 (char *s1, char *s2) {
   char  *temp;
   tmp = S1;
   s1 = s2;
   s2 = temp;
}
void fun2 (char **s1, char**s2) {
   char *tmp;
   tmp = *s1;
   *s1 = *s2;
   *s2= tmp;
}
int main ( ) {
   char *str1 = "Hi", *str2 = "Bye",
   fun1 (str1, str2); print("%s %s", str1, str2);
   fun2 (&str1, &str2); print ("%s %s, str1,
str2);
   return 0;
}
```

The output of the program above is

(GATE - 2018)

(a) Hi Bye Bye Hi (b) Hi Bye Hi Bye
(b) Bye Hi Hi Bye (d) Bye Hi Bye Hi

Sol. 2.

CHAPTER - 2
FUNCTIONS, ARRAYS AND POINTERS

2.1 FUNCTION

In simple terms, we can say function is simply a series of statement that have been grouped together and given a name.

Functions are the building blocks of C programs and each function is essentially a small program with it's own declaration and statements. Every C program can be thought of as a collection of these functions.

Example.

```
# include <stdio.h>
Int average (int a, int b)
{
    return ((a+b)/2);
}
Int Main (void)
{
    int a =2, b = 4;
int C = average (a, b);
Print f ("% d", c); return o;
}
```

Here, average () is a integer function i.e. the value returned by average () is of integer type. Average () taking a and b as input parameters (arguments) and finally returning the average value to the main (), who supplied the arguments or who called the average ().

It is not always compulsory that a function should return something i.e. if a function have data type "void" it will not return anything.

Example.

```
# include <stdio.h>
Void print_value (int n)
{
    Print f ("value is % d", n);
}
Int Main (void)
{
Int i;
For (i=20; i > 0; --i)
{ Print_value (i);}
Return o;
```

"i" will have value 20 initially, as soon as condition check, control goes into body and calls the print_value function.

In general we can define function as

```
              Return-type function (parameter)
{
                       Declarations
```

GATE QUESTIONS

1. Consider the following C program:

```
#include<stdio.h>
int counter = 0;
int calc (in a, int b) {
     int c;
     counter ++;
     if (b =  = 3) return (a*a*a);
     else  {
        c=calc (a, b/3);
        return (c*c*c);
   }
}
int main ( )  {
    calc (4, 81);
    print (“%%d”, counter);
}
```

The output of this program is _______.

(GATE - 2018)

Sol. 1.   (4)

2. Consider the following C program:

```
#include<stdio.h>
void fun 1 (char *s1, char *s2) {
   char  *temp;
   tmp = S1;
   s1 = s2;
   s2 = temp;
}
void fun2 (char **s1, char**s2) {
   char *tmp;
   tmp = *s1;
   *s1 = *s2;
   *s2= tmp;
}
int main ( ) {
   char *str1 = “Hi”, *str2 = “Bye”,
   fun1 (str1, str2); print(“%s %s”, str1, str2);
   fun2 (&str1, &str2); print (“%s %s, str1,
str2);
   return 0;
}
```

The output of the program above is

(GATE - 2018)

(a) Hi Bye Bye Hi   (b) Hi Bye Hi Bye
(b) Bye Hi Hi Bye   (d) Bye Hi Bye Hi

Sol. 2.

CHAPTER - 3
STRINGS, STRUCTURE AND UNION

3.1 STRINGS

Group of characters can be stored in a character array, some times also called as string.
Char a [ ] = {‘G’, ‘A’, ‘T’, ‘E’, ‘’\ o’}
Here, each character will take 1 byte of memory and ‘\ o’ is called as null character i.e. termination of sequence.
The above array declaration is similar to
Char b [ ] – “GATE”;
Here, GATE is a string literal, i.e. characters are enclosed between double quotes. In this ‘\o’ will be added in the end automatically by compiler.

3.1.1. In essence, C treats string literals as character arrays. When a C complier encounters a string literal of length n in a program, it sets aside n+1 bytes of memory for the string. i.e.

G	A	T	E	\0

3.1.2. When C allows char*, then we can use string literals. For example
Char * p = “GATE”
This assignment doesn’t copy the characters in “GATE”, rather makes P a pointer points to first character.

3.2. CHARACTER ARRAY VERSUS CHARACTER POINTERS

1. Char a [ ] = “GATE 2018”
2. Char * b = “GATE 2018”;

In (1), a is an array, while in (2) b is a pointer

Let’s chck some operation on both

a [o] = ‘a’; // valid	P [o] = ‘b’; // invalid
a [3] = ‘r’; // valid	P [2] = ‘r’; // invalid
a [12] = ‘z’ ;// invalid	P [5] = ‘q’ //invalid

3.3 STANDARD LIBRARY STRING FUNCTION

Function	Use
Strlen	Finds the length of string
Strcat	Appends one string at the end of another
Strncat	Appends first n characters of a string at the end of another
Strcpy	Copies a string into another
Strncpy	Copies first n characters of a string into another
Strcmp	Compares two strings
Strcmpi	Compare two strings regardless to case
Strrev	Reverse string
Strdup	Duplicate a string

GATE QUESTIONS

1. Consider the C program fragment below which is meant to divide x by y using repeated subtractions. The variables x, y. q and r are all unsigned int.

```
while (r > = y)
{
        r = r = – y;
        q = q + 1;
}
```

Which of the following conditions on the variable x, y, q and r before the execution of the fragment will ensure will ensure that the loop terminates in a state satisfying the condition x = = (y*q +r)?

[GATE - 2017]

(a) (q = = r) && (r = = 0)
(b) (x > 0) && (r = = x) && (y > 0)
(c) (q = = 0) && (r = = x) && (y > 0)
(d) (q = = 0) && (y > 0)

2. Consider the following C Program.

```
#include <stdio.h>
int main ()
{
   int m = 10;
   int n, nl;
   n = ++m;
   nl = m++;
   n – –;
   – – nl;
   n – = nl;
    printf("%d", n);
   return 0;
}
```

The output the program is _____.

[GATE - 2017]

3. Consider the following C Progarm.

```
#include <stdio.h>
#include <string.h>
int main()
{
     char *c = " GATECSIT2017";
     char *p = c;
     print("%d", (int) strlen (c+2 [p] –6 [p]– 1));
     return 0;
}
```

The output of the program is ____.

[GATE - 2017]

4. Match the following.

A. static char var;
B. m = malloc (10);
   m = NULL;
C. char *ptr [10];
D. register int var1;

(i)Sequence of memory locations to store addresses
(ii)A variable located in data section of memory
(iii)Request to allocate a CPU register to store data
(iv)A lost memory which cannot be freed

[GATE - 2017]

(a) A-ii, B-iv, C-i, D-iii
(b) A-ii, B-I, C-iv, D-iii
(c) A-ii, B-iv, C-iii, D-i
(d) A-iii, B-iv, C-I, D-ii

5. Consider the following function implemented in C.

```
void  printxy (int x, int y)
{
   int *ptr;
   x = 0;
   ptr = &x;
   y = *ptr;
   *ptr = 1;
   print ("%d, %d", x, y);
```

The output of invoking printxy (1, 1) is

[GATE - 2017]

(a) 0, 0 (b) 0, 1
(c) 1, 0 (d) 1, 1

6. Consider the C function foo and bar given below.

```
   int foo (int val)
```

ASSIGNMENT

1. We require a code to print the integer values from 0 to 9. Identify the error?

```
#include<stdio.h>
#include<conio.h>
Void main ( )
{
    static int I,
    clrscr ( ) ;
    for (;i<=9;) ;
    {
        Printf("/n value is: %d", i);
    }
    getch( );
}
```

2. Find the output of the following code?

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int i;
    printf ("Enter the value of i = ");
    sanf ("%d", &i) ; /* input : 5 */
    clrscr ( );
    do
    {
       i++;
       Printf ("\nHello User");
    }
    while (i<4) ;
    getch ( ) ;
}
```

3. Find out the error in the following code.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i=1, j;
    clrscr ( ) ;
    do
    {
       for (j=1 ;; j++)
       {
          if (j>2)
              break ;
          if (i = j)
              continue;
          printf ("\n%d%d", i, j) ;
       }
         i++;
         while (1<3)
    }
    getch ( ) ;
}
```

4. What will be the value of I and j after execution?

```
for (i=0, j=0; i<5, j<25, i++, j++) ;
printf ("i=%d j=%d ", i, j);
```

5. What is the output of the following code?

```
#include<stdio.h>
void main( )
{
    int var = –10;
    for (;var;printf("%d\n", var++));
}
```

6. What is the output of the following code?

```
#include<stdio.h>
main ( )
{
    int var=0;
    for (;var++;printf ("%d", var)) ;
    printf ("%d", var) ;
}
```

7. What is the output of the following code?

```
#include<stdio.h>
main ( )
{
    int y;
    scanf ("%d", &y) ;
    /* input given is 2000 */
```

SECTION-B
(DATA STRUCTURE)

CHAPTER - 1
ARRAYS

1.1 INTRODUCTION
1. Arrays are collection of homogeneous data.
2. Array elements are stored in successive memory locations in the memory.
3. Arrays are broadly categorized as follows
(i) One-Dimensional Array
(ii) Multi-Dimensional Array

1.1.1 One-Dimensional Array
In 1-D array, the elements are stored at consecutive locations.

10	12	14	16	18	20	22	24	26
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Lower Bound (Indices) Upper Bound

Lower Bound (LB): Smallest index of an element in the array.
Upper Bound (UB): Largest index of an element in the array.
Length of the Array = No. of elements in the Array
Length of the Array = UB – LB + 1

Lower Bound of Array is taken as '0' until and unless not mentioned.

1.1.1.1 Address Calculation in One-Dimensional Array
Physical address of any element of array is computed as follows
General Formula
Physical address of a[i] = B.A+ (i – LB of array) Size of the element
Where, B.A (Base address) = Physical address of starting element in the array

1.1.2 Two-Dimensional Array
1. It is an instance of Multi-Dimensional Array.
2. It is collection of 1D arrays.
3. Notation used to represent 2D array is array name[number of 1D arrays][number of elements in each 1D array) **or**
Array name [index vector of rows][index vector of columns]
4. In Mathematics, matrices can be treated as 2D array, where numbers of rows are same as number of 1D arrays and number of columns are same as no: of elements in each 1D array.

1.1.2.1 Graphical Representation

$$\begin{vmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{vmatrix}$$

GATE QUESTIONS

1. Consider the following snippet of a C program. Assume that swap (&x, &y) exchanges the contents of x and y.

```
int main ( )
{
   int array [ ] = {3, 5, 1, 4, 6, 2};
   int done = 0;
   int i;
   while (done == 0)
   {
          done = 1;
          for (i=0; i<= 4; i++)
          {
            if (array [i] < array [i + 1])
            {
               swap (&array[i], &array [i +1]);
               done = 0;
            }

          }
          for (i=5; i>=1; i– –)
          {
            if (array[i] > array[i –1])
            {
            swap(&array[i], &array[i–1];
            done = 0;
            }
          }
   }
   printf("%d", array [3]);
 }
```

The output of the program is ________.

[GATE - 2017]

2. What is the output of the following C code? Assume that the address of x is 2000 (in decimal) and an integer requires four bytes of memory.

```
int main( )
{ unsigned int x [4][3] =
{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};
printf("%u,%u, %u", x+3, *(x+3),*(x+2)+3);
}
```

[GATE - 2015]

(a) 2036, 2036, 2036
(b) 2012, 4, 2204
(c) 2036, 10, 10
(d) 2012, 4, 6

3. Consider the following C program segment.

```
# include <stdio.h>
int main()
{
Char sl[7] = "1234", *p;
p = sl + 2;
*p = '0';
printf("%s", sl)
}
```

What will be printed by the program?

[GATE - 2015]

(a) 12 (b) 120400
(c) 1204 (d) 1034

4. Consider the following C program.

```
# include <stdio.h>
int main ()
{
static int a[ ] = {10, 20, 30, 40, 50};
static unit *p[ ] = {a, a + 3, a + 4, a + 1, a + 2};
int **ptr = p;
ptr++;
printf("%d%d", ptr-p, **ptr);
}
```

The output of the program is ______.

[GATE - 2015]

5. Suppose c =(c[0],……c[k–1]) is an array of length k, where all the entries are form the set {0, 1}. For an positive integers a and n, consider the following pseudo code.

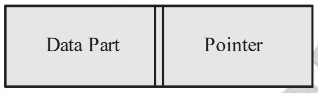
```
DO so METHING (c, a, n)
z ← 1
for I ← 0 to k – 1
do z ← $z^2$ mod n
if c[i] = 1
then z ← (z × a) mod n
return z
```

CHAPTER - 2
LINKED LIST

2.1 INTRODUCTION

1. It is collection of data elements, called nodes. Each node has its data part and pointer to maintain the order of nodes.
2. It is used to store and remove the data dynamically as per requirement.

2.1.1 Representation of Node



Node

2.1.2 Need of Linked List instead of Array

In array, there is wastage of memory whenever the number of stored elements are less than the maximum size of the array. But in linked list, only required number of elements are allocated space in the memory, extra unused space is not allocated. Maximum number of insertions can be done in linked list until memory overflows.

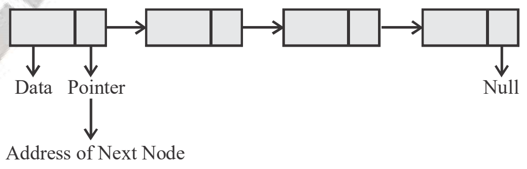
Space utilization is efficient in Linked list.

2.2 TYPES OF LINKED LIST

1. Single Linked List
2. Circular Linked List
3. Double Linked List
4. Circular Double Linked List

2.2.1 Single Linked List

1. It contains nodes having single pointer that contains address of next node in the list.
2. It can be represented as follows



3. We use structure to define a node in linked list as follows:

```
struct node
{
int data;                 // used to store the data elements of the linked list
struct node * link;       // used to store the pointer of next data element in the Linked list.
};
```

GATE QUESTIONS

1. Consider the C code fragment given below.

```
Typedef struct node
{ int data;
Node * next;
} node;
Void joint (node * m , node * n)
{
Node * p = n'
While (p → next ! = NULL)
{
p = p → next;
}
p → next = m;
}
```

Assuming that m and n point to valid NULL-terminated linked lists, invocation of joint will

[GATE - 2017]

(a)Append list m to the end of list n for all inputs.
(b)Either cause a null pointer dereference or append list m to the end of list n.
(c)Cause a null pointer dereference for all inputs.
(d)Append list n to the end of list m for all inputs.

2. N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this order: $\Theta(N)$ delete, $O(\log N)$ insert, $O(\log N)$ find, and $\Theta(N)$ decrease-key. What is the time complexity of all these operations put together?

[GATE - 2016]

(a) $O(\log^2 N)$ (b) $O(N)$
(c) $O(N^2)$ (d) $\Theta(N^2 \log N)$

3. The following C function takes a single-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank.

```
typedef struct node
{
   int value;
   struct node *next;
} node;
Node *move_to-front (Node *head)
{
   Node *p, *q;
   if ((head == NULL) | | (head – > next = = NULL))
     return head;
   q = NULL;
   p = head;
   while (p – >next ! = NULL)
   {
         q = p;
         p = q – > next;
   }
   _________________
   return head;
}
```

Choose the correct alternative to replace the blank line.

[GATE - 2010]

(a) q = NULL; p → next = head; head = p;
(b) q→ next = NULL; head = p; p → next = head;
(c) head = p; p → next = q; q → next = NULL;
(d) q→ next = NULL; p → next = head; head = p;

4. The following C function takes a singly – linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be

CHAPTER - 3
STACKS

3.1 INTRODUCTION

1. It is linear list in which data elements are inserted and deleted at one end.
2. It uses LIFO (Last In First Out) technique to access each element.

Example. Pile of books, pile of coins.
To take the book presents at the bottom of the pile, we firstly, remove all the books above that bottom book in order. And to add new book in the pile, it is added at the top not in the middle. That is called LIFO technique. Similar is the case with pile of coins.

3.2 ABSTRACT DATA TYPE (ADT) OF STACK

Stack Definition contains information about accessing technique used in stack, Standard operations and Status operations, Pointers. It has
Accessing Technique: LIFO(Last In First Out)
TOP: It is a pointer that points to recently pushed element.
Standard Operations : Pop ( ), Push ( )
Status Operations : Isempty ( ), Isfull ( )

3.2.1 PUSH() Operation

Let
S = Name of stack
TOP = Pointer pointing to top of the stack
N = size of stack // maximum number of elements that can be stored
x = element to be pushed

```
Void push(S, Top, N , x)
{
if (Top = = N – 1)
{
printf (``stack is full``);
exit (1) ;
}
else
Top ++;
S[Top] = x;
}
```

N-1 is the index for last element of stack, because stack starts from 0.

3.2.2 POP Operation

```
Pop (S, Top, N)
{
int y ;
```

GATE QUESTIONS

1. The attributes of three arithmetic operators in some programming language are given below.

Operator	Precedence	Associativity	Arity
+	High	Left	Binary
–	Medium	Right	Binary
*	Low	Left	Binary

The value of the expression $2-5+1-7*3$ in this language is _________.

[GATE - 2016]

2. Consider the following C function.

```
int fun (int n){
int x=1, k;
if (n==1) return x;
for  (k=1; k<n; ++k)
x = x + fun (k)  * fun (n–k);
return x;
}
```

The return value of fun (5) is ________

[GATE - 2015]

3. Consider the following recursive C function

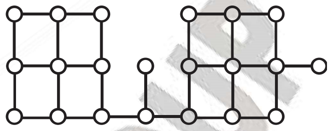
```
void get (int n)
{
If (n < 1) return
get (n – 1);
get (n – 3);
print f ("%d", n);
}
```

If get (6) function is being called in main ( ) the how many times will the get () function be invoked before returning to the main ()?

[GATE - 2015]

(a) 15 (b) 25
(c) 35 (d) 45

4. Suppose depth first search is executed on the graph below starting at some unknown vertex that has not been visited earlier. Then the maximum possible recursion depth (including the initial call) is ________.



[GATE - 2014]

5. Suppose a stack implementation supports an instruction REVERSE, which reverses the order of elements on the stack, in addition to the PUSH and POP instructions. Which one of the following statements is TRUE with respect to this modified stack?

[GATE - 2014]

(a)A queue cannot be implemented using this stack.
(b)A queue can be implemented where ENQUEUE takes a single instruction and DEQUEUE takes a sequence of two instructions.
(c)A queue can be implemented where ENQUEUE takes a sequence of three instructions and DEQUEUE takes a single instruction.
(d)A queue can be implemented where both ENQUEUE and DEQUEUE take a single instruction each.

6. The following postfix expression with single digit operands is evaluated using a stack

8 2 3 ∧ / 2 3 * + 5 1 * –

Note that ^ is the exponentiation operator. The top two elements of the stack after the first* is evaluated are

[GATE - 2007]

(a) 6, 1 (b) 5, 7
(c) 3, 2 (d) 1, 5

7. An implementation of a queue Q, using two stacks S1 and S2, is given below

```
void insert (Q, x) {
    push(S1, x);
}
void delete (Q) {
    if (stack-empty(S2)) then
```

CHAPTER - 4
QUEUES

4.1 INTRODUCTION

It is linear list in which insertion of elements is done at the end and deletion is done at the front of the list.
It uses FIFO (First in First Out) technique to perform any operation.

Example. People waiting in line at a bank form a queue, where the first person in line is the first person to be waited on. New person cannot stand at any position in between queue, it will have to stand at the end of the queue.

4.1.1 Abstract Data Type (ADT) of Queue

Definition: FIFO
1. FRONT = Pointer pointing to the element to be deleted.
2. REAR = Pointer pointing to the recently pushed element

4.1.2 Status Operations

1. PUSH ( )
2. POP ( )
3. Is empty ( )
4. Is full ( )

4.1.3 Graphical Representation of Queue

1. Enqueue
Insert an element in queue
2. Dequeue
Delete an element from the queue

1. If there is not a single element in the queue, then both the pointers point outside the queue.
2. If there is a single element in the queue, then both the pointers point at that element.

It is not possible that one pointer is present inside the queue and other pointing outside the queue.
Either both the pointers are present inside the queue or both will be outside the queue.

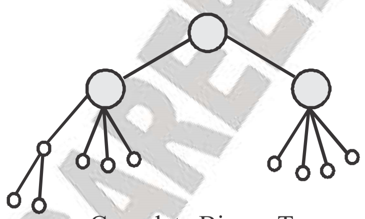
CHAPTER - 5
BINARY TREE

5.1 INTRODUCTION
In Binary tree, each node has at most two children.

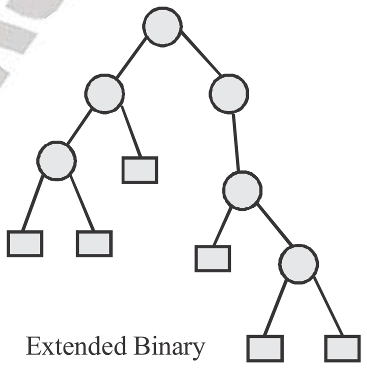
5.1.1 Depth (Height) of Binary Tree
Maximum number of nodes in the longest branch.
Depth is one more than the largest level number of tree.

5.1.2 Types of Binary Tree
1. Complete Binary Tree
2. Extended Binary Tree

5.1.2.1 Complete Binary Tree
Binary Tree is said to be complete if all its levels, except possibly that last, have maximum number of possible nodes.
Depth(d_n) of the complete tree with n nodes is
$d_n = \log_2 n + 1$


Complete Binary Tree

5.1.2.2 Extended Binary Tree: 2-Tree
A Binary Tree is said to be a 2-Tree or an extended binary tree, if each node N has either 0 or 2 children. In such tree, nodes with 2 children are called internal nodes and nodes with 0 children are called external nodes.


Extended Binary

5.2 DEFINITION OF BINARY TREE NODE
1. data will store the information at the node.
2. lptr is the pointer to the left child of node.
3. rptr is the pointer to the right child of node.

CHAPTER - 6
AVL TREES

6.1 INTRODUCTION

AVL trees are known as Balanced Binary Search Trees.

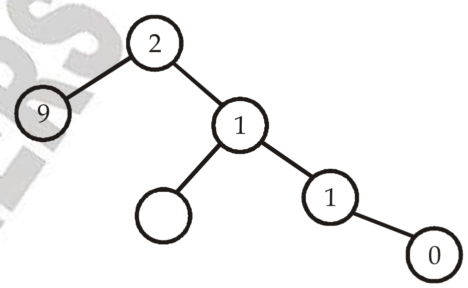
6.1.1 Why AVL?

Insertion of elements Z, X, Y,…………C, B, A in this order results in left skewed binary search tree.
Insertion of elements A, B, C,………X, Y, Z in this order results in right skewed binary search tree.
And disadvantage of a skewed binary search tree is that worst case complexity of a search is O(n). So, to reduce the searching time, we make binary tree of balanced height. And that balanced tree is called AVL tree.
dg

6.1.2 Balanced Factor

B.F. = Height of LST – Height of RST
or
B.F= Height of RST – Height of LST
Where, LST is left subtree and RST is right subtree.
A tree is considered as Balanced Tree, when balanced factor (B.F.) = 0, + 1 or – 1.
If balanced factor (B.F.) is anything else except 0, + 1 or – 1. Values, then it is considered as unbalanced tree.

Example. Unbalanced binary tree
Here, at the root node, the balanced factor comes out to be + 2, it is considered as an unbalanced tree.
(i) Balanced AVL



6.1.3 Searching in an AVL Search Tree

Searching an element in AVL tree is similar to binary search.

6.1.4 Insertion in an AVL Search Tree

Inserting an element into an AVL search tree in its first phase is similar to that of the one used in a binary search tree. However, if insertion of the element disturbs the balanced factor of any node in the tree. We use rotations technique, to restore the balance of the search tree.
To perform rotations it is necessary to identify a specific node A whose BF(A) is neither 0,1,-1, and which is the nearest ancestor to the inserted node on the path from the inserted node to the root.

GATE QUESTIONS

1. B+ Trees are considered **BALANCED** because

[GATE - 2016]

(a)The lengths of the paths from the root to all leaf nodes are all equal.
(b)The lengths of the paths from the root to all leaf nodes differ from each other by at most 1.
(c)The number of children of any two non-leaf sibling nodes differ by at most 1.
(d)The number of records in any two leaf nodes differ by at most 1.

2. What is the maximum height of any AVL-tree with 7 nodes? Assume that the height of a tree with a single node is 0.

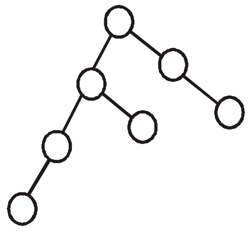
[GATE - 2009]

(a) 2 (b) 3
(c) 4 (d) 5

SOLUTIONS

Sol. 1. (a)
A Tree is balanced, if the length of the paths from the root to all leaf nodes are all equal.

Sol. 2. (b)
Maximum height of any AVL-tree with 7



(There may be different way to draw AVL with 7 nodes)

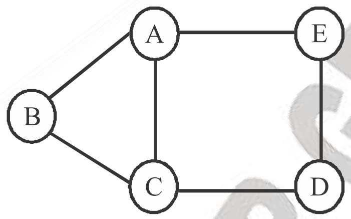
CHAPTER - 7
GRAPHS

7.1 GRAPH

A graph has two properties

1. A set V of elements called nodes (vertices)

2. A set E of edges e in E is identified with a unique (unordered) pair [u, v] of nodes in V, denoted by e[u, v] , where u,v are the endpoints of edge e.



Graph (Figure 1)

7.1.1 Degree of Node

Number of edges containing that node.
In the above graph, vertex set V = {A, B, C, D, E}
Edge set E = {AB, AC, AE, BC, ED, CD, EC}
Degree of A (Degree (A)) = 3
Degree of B (Degree (B)) = 2
Degree of C (Degree (C)) = 3
Similarly we can find the degree of remaining nodes of the above graph.

7.1.2 Path

A sequence of nodes traversed to reach from one point to another.

7.1.2.1 Simple Path

A Path said to be simple if all the nodes are distinct with the exception that starting and ending vertex are distinct.

7.1.2.2 Closed Path

A Path said to be closed if all the nodes are distinct with the exception that starting and ending vertex are same.

7.1.3 Cycle

A cycle is a closed simple path with length 3 or more.
A cycle of length k is called k-cycle.
In the above figure 1,
Cycles: [A, B, C, E, A] and [A, C, D, E, A]
Path: (B, A, E) and (B, C, E) are simple paths of the length 2.

CHAPTER - 8
SORTING

8.1 RADIX SORT

1. It is the technique to sort elements.
2. It uses queue data structure to implement.

8.1.1 Steps for Radix Sort

1. Select the number in the list to be sorted with maximum digits (d).
2. Make all of the numbers having maximum number of digits by adding zeroes before the numbers, if required.
3. Arrange the numbers in the lists in the required order (increasing or decreasing) according
4. To the unit decimal place to $10^{(d-1)th}$ decimal place in each pass.

Example. 101, 1,79, 97, 86, 7, 44, 99, 421, 23, 49, 12
Solution.
Maximum no. of digits = 3
$\therefore$ Make all of them 3-digit
101, 001, 079, 097, 086, 007, 044, 099, 421, 023, 049, 012
1st Pass (Increasing order of digit at unit place)
101, 001, 421, 012,023, 044, 086, 097, 007, 079, 099, 049
2nd Pass (Ten's place digit)
101, 001, 007, 012, 421, 023, 044,049, 079, 086, 097, 099
3rd Pass (Hundred's Place digit)
001, 007, 012, 023, 044, 049, 079, 086, 097, 099, 101, 421
No. of passes = 3
Time complexity = O(m, n)
m = No of passes = Maximum number of digits of a number among the list's numbers.
n= no. of elements in the list to be sorted.
$\approx$ O(n)

Example. 1, 2, 86,421,361,111,6,3,94,55,814,612,522,511
Solution.
Maximum number of digits = 3
$\therefore$ Make all of them = 3-digit

001,002,086,421,361,111,006,003,094,055,814,612,522,511
Pass-I
001,421,361,111,511,002,612,522,003,094,814,055,086,006
Pass-II
001,002,003,006,111,511,612,814,421,522,055,361,086,094
Pass-III
001,002,003,006,055,086,094,111,361,421,511,522,612,814
No. of passes = 3

8.2 BUCKET SORT

It is also a technique to sort the elements.

CHAPTER - 9
HASHING

9.1 DIFFERENT SEARCHING TECHNIQUES AND THEIR EFFICIENCY
1. The sequential (Linear) search algorithm takes time proportional to the data size, i.e, **O(n)**.
2. Binary search improves on linear search reducing the search time to **O(logn)**.
3. With a BST, an **O(logn)** search efficiency can be obtained; but the worst-case complexity is **O(n)**.
4.To guarantee the **O(log n)** search time, BST height balancing is required (i.e., AVL trees)

9.2 HASHING
1. It is the technique to access the records in the file fastly.
2. It has goal of Time complexity O(1).
3. It uses different hashing functions, which are applied on some field of the records. That field is called **hash field**.

9.2.1 Types of Hashing
1. Static Hashing
2. Dynamic Hashing

9.2.1.1 Static Hashing
1. In static hashing, the hash function maps search-key values to a fixed set of locations.
2. It uses single hash function
3. Its searching time is O(1)
4. It does not support range queries

(i) Problems of Static Hashing
1. There is fixed size of hash table due to fixed hash function
2. It may require rehashing of all keys when chains or overflow buckets are full

9.21..2 Dynamic Hashing
In dynamic hashing a hash table can grow to handle more items. The associated hash function must change as the table grows.
1. The load factor of a hash table is the ratio of the number of keys in the table to the size of the hash table.
2. The higher the load factor, the slower the retrieval.
3. With open addressing, the load factor cannot exceed 1. With chaining, the load factor often exceeds 1.

9.3 APPLICATIONS OF HASH TABLES
1. Database Systems
Specifically, it is for those that require efficient random access. Generally, database systems try to optimize between two types of access methods: sequential and random. Hash tables are an important part of efficient random access because they provide a way to locate data in a constant amount of time.

SECTION-C
(ALGORITHM)

CHAPTER - 1
ANALYSIS OF ALGORITHMS

1.1 INTRODUCTION
1. Algorithm is well defined sequence of computational steps that transform the input to output.
2. It can be designed to solve problems such as identifying all genes in human DNA, finding good routes to carry data from one machine to another machine to another machine, sorting, searching on particular data and many more.

1.2 EFFICIENCY
1. There are two valuable resources Memory and processor which are essentially required to solve any problem. Now a day, Processor′s speed is very good but they are not infinitely very fast and memory is also available in good sizes but it is not very cheap. So, processor's speed and memory sizes are two constraints for Algorithm efficiency. Another fact that can solve a given problem efficiently that is designing an algorithm efficiently.
2. Efficiency of algorithm can be defined in terms of time and space. It implies time and space taken by algorithm.
3. Time and space of given algorithm can vary on different input.
4. There are two broad categories of Analysis of algorithm
(i) Priory Analysis
(ii) Postinary Analysis.

(i) Priory Analysis
(a) It is machine independent that does not include processor′s speed and memory configuration of any system in efficiency of Algorithm.
(b) It defines the running time of an algorithm on a particular input is the number of primitive operations.

5. Each line of pseudocode takes constant time. One line of code can take different time (execution) than another line of code.

6. Any algorithm can have different running time on different inputs. So, each algorithm has lower and upper bounds on their running time on particular inputs.

7. Algorithm′s best performance is characterized by lower running time. It is called best case.

8. In worst case, Algorithm gives its worst performance that is upper bound on its running time.

9. Running time at all inputs except that of best and worst case, represents average case.

10. One algorithm is said to be more efficient then another if its worst case running time has a lower order of growth.

11. Each asymptotic running time is specified defined using different asymptotic notations.

12. Asymptotic running time means the running time of an algorithm on large input sizes.

ASSIGNMENT-I

Solve using substitution method

1. $T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n & n > 1 \\ b & n \le 1 \end{cases}$

2. $T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + 2 & n > 2 \\ 1 & n \le 2 \end{cases}$

3. $T(n) = \begin{cases} 8T\left(\frac{n}{2}\right) + n^2 & n > 2 \\ 1 & n \le 2 \end{cases}$

4. $T(n) = \begin{cases} 2T(n-1) + c & n > 1 \\ 1 & n \le 1 \end{cases}$

5. $T(n) = \begin{cases} T(n-1) + n & n > 1 \\ 1 & n \le 1 \end{cases}$

Solve using master method

6. $T(n) = 2T\left(\frac{n}{2}\right) + 2$

7. $T(n) = 5T\left(\frac{n}{2}\right) + n$

8. $T(n) = 7T\left(\frac{n}{2}\right) + n^2$

9. $T(n) = 27T\left(\frac{n}{3}\right) + n^3$

CHAPTER - 2
DIVIDE AND CONQUER

2.1 DIVIDE AND CONQUER

1. It is another way to design algorithms.
2. It firstly divides the problems into number of independent sub problems.
3. Then, it solves the sub problems by solving them recursively
4. It combines the sub problems solutions to give a solution to the original problem.

2.1.1 Control Abstraction of Divide and Conquer

Abstraction is used because some functions are not defined such as small, divide, solution etc.
If P is the problem, DAC works as follows

```
DAC (P)
{
if (small (P))   //checks whether problem is small or big
return (S(P))
else
K = divide (P) into k parts
return (combine (DAC(P1), DAC (P2), DAC (P3)……..DAC (Pk)))
}
```

2.2 APPLICATIONS OF DIVIDE AND CONQUER

1. Finding maximum and minimum
2. Binary search
3. Quick Sort
4. Merge Sort
5. Selection Procedure

2.2.1 Finding Maximum and Minimum

It is the problem to find maximum and minimum number among n numbers.
There are two approaches to solve this problem
1. Straight Method
2. Divide and Conquer

1. Straight Method

(i) This method finds maximum and minimum by comparing all remaining elements with first element of the array linearly.
(ii) Its algorithm is following

```
straight max-min (a, n)
{
max ← a[1]
min ← a[1]
for (i = 2 to n)
if max < a[i] then
do max ← a[i]
else
```

CHAPTER - 3
GREEDY TECHNIQUE, DYNAMIC PROGRAMMING

3.1 INTRODUCTION

1. Greedy and Dynamic Techniques are other approaches to design algorithms.
2. Some algorithms are efficient if they are designed using these approaches.
3. Every problem has its Definition, Solution space, Feasible Solution space, Optimal Solution space.

3.1.1 Problem Definition

Knowing the problem, its inputs and outputs and its conditions clearly.

1. Solution Space: All possible solutions of the problem over given input conditions

2. Feasible Solution Space: It includes solutions from the solution space which satisfies the conditions of the problem.

3. Optimal Solution Space: It includes solutions from the feasible solutions, which satisfy the optimal conditions of the problem.

3.2 GREEDY TECHNIQUE

1. Algorithms that use greedy technique are called greedy algorithm.
2. This technique allows to algorithm to make the choice that always looks the best choice at the moment. It implies it makes locally optimal choice in the hope that this choice will lead to a globally optimal solution.
3. It is used in various optimization problems.
4. But it does not yield optimal solutions for some problems.

3.2.1 Control Abstraction of Greedy Technique

a is an array of n object. Initially solution is ϕ

```
Greedy (a, n)
{
Solution = ϕ
For (i = 1 to n)
{
xi = select (n);
it (feasible (xi)
Add (solution, xi);
}
}
```

3.3 APPLICATION OF GREEDY TECHNIQUE

1. Job sequencing with deadline
2. Real knapsack (fractional knapsack)
3. Optimal merge pattern
4. Minimum cost spanning tree
5. Huffman coding
6. Single Source Shortest Path

CHAPTER - 4
COMPLEXITY CLASSES

4.1 INTRODUCTION

1. There are many problems exist in the world. Some of the problems are very easy and some are difficult. Easy problems are also called solvable and difficult problems are those problems which are not solvable or take more time to solve.
2. Solvable problems are called tractable problems.

4.2 ABSTRACT PROBLEM

1. It is defined as binary relation on a set I of problem instances and a set S of problem solutions.
2. Abstract decision problem is a function that maps the instance set I to the solution set {0, 1}.

For example, decision problem is related to shortest-path is the Problem path.
i = < G, u, v, k > is the instance of the shortest path problem that belongs to set I of shortest path.
If path (i) = yes, it implies there is a path from u to v has almost k edges. Otherwise path (i) = No.

4.3 ENCODING PART

1. It is a mapping of abstract objects from a set to the set of binary strings such as set N = {0, 1, 2, 3, 4, …} $\Rightarrow$ e (3) = 11.
2. Similarly are abstract objects such as polygons, graphs, functions, ordered pairs, programs can be encoded as binary strings.
3. Encoding also exists in shortest part abstract decision problem where every instance from set S can be encoded
4. It transforms abstract problem to concrete problem.
5. The computer algorithm that solves abstract decision problem actually takes on encoding of a problem instance as input.
6. Concrete problem has input instances as binary strings.
7. Polynomial-time solvability of a problem also depends upon encoding but it is assumed that it is independent of encoding procedure.
8. Theory of computation discipline allows us to express the relation between decision problems and algorithms that solve them concisely.
9. If there is an abstract decision problem with instance set I, its encoding set e(I) and solution set S = {0, 1}. Then, if an algorithm/machine model accepts a string x $\in$ e(I) if I given as input then language (L) of machine/Algorithm will be L ={ x $\in$ e (I): S(x) = 1 }. So, it includes all accepted strings but it rejects x $\in$ e (I) and S(x) = 0
10. Language L/problems is said to be decidable if every binary string in L is accepted by machine/algorithm and every binary string into in L is rejected by the machine/algorithm. Therefore, all Turing machine problems/languages are decidable.
11. A language L is said to be decided in polynomial time, if there is an algorithm for which a constant k exist and for strings of any-length n x $\in$ {0, 1}*, the algorithm correctly decides whether x $\in$ L in time O (n^k).
12. Turing machine languages are decided in finite amount of time. It also implies that they are decidable
13. Some algorithm/machine accepts all x $\in$ L, but loop forever. If $x \notin L$. These languages are called recursive enumerable.

GATE QUESTIONS

1. Consider two decision problems Q_1, Q_2 such that Q_1 reduces in polynomial time to 3-SAT and 3-SAT reduces in polynomial time to Q_2. Then which one of the following is consistent with the above statement?

[GATE - 2015]

(a) Q_1 is in NP, Q_2 is NP hard.
(b) Q_2 is in NP, Q_1 is NP hard.
(c) Both Q_1 and Q_2 are in NP.
(d) Both Q_1 and Q_2 are NP hard.

2. Consider the following statements.
I. The complement of every Turing decidable language is Turing decidable
II. There exists some language which is in NP but is not Turing decidable
III. If L is a language in NP, L is Turing decidable
Which of the above statements is/are true?

[GATE - 2015]

(a) Only II (b) Only III
(c) Only I and II (d) Only I and III

3. Language L1 is polynomial time reducible to language L2. Language L3 is polynomial time reducible to L2, which in turn is polynomial time reducible to language L4.
Which of the following is/are true?
I. If $L4 \in P$, $L2 \in P$
II. If $L1 \in P$ or $L3 \in P$, then $L2 \in P$
III. $L1 \in P$, if and only if $L3 \in P$
IV. If $L4 \in P$, then $L1 \in P$ and $L3 \in P$

[GATE - 2015]

(a) II only (b) III only
(c) I and IV only (d) I only

4. Let π_A be a problem that belongs to the class NP. Then which one of the following is TRUE?

[GATE - 2009]

(a) There is no polynomial time algorithm for π_A
(b) If π_A can be solved deterministically in polynomial time, then P = NP
(c) If π_A is NP – hard, then it is NP-complete
(d) π_A may be undecidable

5. The subset sum problem is defined as follows: Given a set S of n positive integers and a positive integer W; determine whether there is a subset of S whose elements sum to W.
An algorithm Q solves this problem in O(nW) time. Which of the following statements is false?

[GATE - 2007]

(a) Q solves the subset sum problem in polynomial time when the input is encoded in unary
(b) Q solves the subset sum problem in polynomial time when the input is encoded in binary
(c) The subset sum problem belongs to the class NP
(d) The subset sum problem is NP-hard

6. Let S be an NP-complete problem Q and R be two other problems which are known to be in NP. Q is polynomial-time reducible to S and S is polynomial-time reducible to R. Which one of the following statements is true?

[GATE - 2006]

(a) R is NP-complete (b) R is NP-hard
(c) Q is NP-complete (d) Q is NP-hard