www.ignousite.com
Course Code : BCSL-045
Course Title : Introduction to Algorithm design Lab
Assignment Number : BCA(4)/L-045/Assignment/2022-23
Maximum Marks : 50
Weightage : 25% www.ignousite.com
Last date of Submission : 31st October, 2022 (For July Session)
: 30th April, 2023 (For January Session)

Q1. Write a program to implement Binary Search algorithm for an array consisting of at least 15 elements in the range 2 to 85.

Ans.

```
#include <stdio.h>

int recursiveBinarySearch(int array[], int start_index, int end_index, int element)

{

if (end_index >= start_index)

{

int middle = start_index + (end_index - start_index )/2;

if (array[middle] == element)return middle;

if (array[middle] > element)

return recursiveBinarySearch(array, start_index, middle-1, element);

return recursiveBinarySearch(array, middle+1, end_index, element);

}

return -1;

}

int main(void){int array[] = {2, 4, 7, 9, 16, 18, 20, 23, 32, 54, 65, 70, 77, 80, 85};

int n = 15;

int element = 54;

int found_index = recursiveBinarySearch(array, 0, n-1, element);

if(found_index == -1 )

{

printf("Element not found in the array ");

}

else
```

```
{
printf("\n In the given array of 15 elements, 54 found at index : %d\n",found_index);
}
return 0;
}
```

Output:

```
In the given array of 15 elements, 54 found at index : 10

Process exited after 2.009 seconds with return value 0
Press any key to continue .......
```

Q2. Write and test a program to sort the following array of integer numbers using Insertion Sort. Calculate the total no of comparison operations and the number of times the loop will execute.

85	45	70	30	25	35	40	5	10	17

Ans.

```
#include <stdio.h>
void insert(int a[], int n) /* function to sort an array with insertion sort */
{
int i, j, temp; for (i = 1; i < n; i++)
 {
 temp = a[i]; j = i - 1;
while(j>=0 && temp <= a[j]) /* Move the elements greater than temp to one position ahead from their current position*/
{
a[j+1] = a[j]; j = j-1;
 }
a[j+1] = temp;
 }
 }
void printArr(int a[], int n) /* function to print the array */
{
```

```
int i;

for (i = 0; i < n; i++)

printf("%d ", a[i]);

}

int main()

{

int a[] = { 85, 45, 70, 30, 25, 35, 40, 5, 10, 17 };

int n = sizeof(a) / sizeof(a[0]);

printf("Before sorting array elements are - \n");

printArr(a, n);

insert(a, n);

printf("\n\n\nAfter sorting array elements are - \n");

printArr(a, n);

return 0;

}
```

Output:

```
Before sorting array elements are –
85 45 70 30 25 35 40 5 10 17

After sorting array elements are –
5 10 17 25 30 35 40 45 70 85

Process exited after 2.559 seconds with return value 0
Press any key to continue .......
```

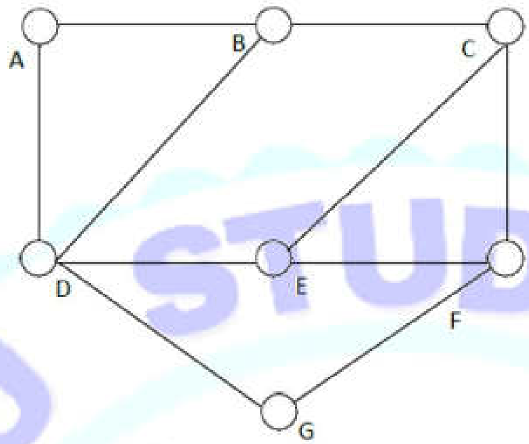
Calculate the total no of comparison operations:

The maximum number of comparisons for an insertion sort is the sum of the first n − 1 integers. Again, this is $O(n^2)$. However, in the best case, only one comparison needs to be done on each pass. In our case, number of comparisons are executed = n − 1 = 10 − 1 = 9

The number of times the loop will execute:

We denote with n the number of elements to be sorted; in our case n = 10. The two nested loops are an indication that we are dealing with quadratic effort, meaning with time complexity of $O(n^2)$*. This is the case if both the outer and the inner loop count up to a value that increases linearly with the number of elements. In our case, number of loops executed = $n^2 = 10^2 = 10*10 = 100$.

Q3. Write a program to traverse a graph using DFS. Apply this algorithm to the following graph and write the sequence of vertices to be travelled. Also calculate the number of times the loop(s) will execute.



Ans.

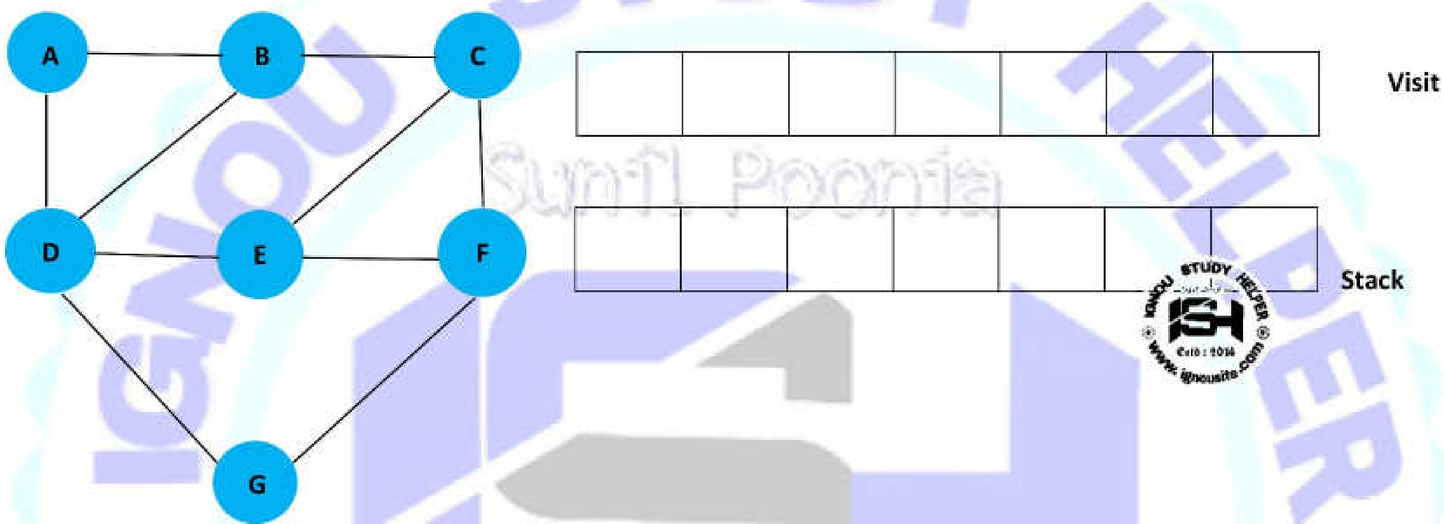
```
#include<stdio.h>
#include<conio.h>

int a[20][20],reach[20],n;

void dfs(int v)

{

int i;

reach[v]=1;

for (i=1;i<=n;i++)

if(a[v][i] && !reach[i])

{

printf("\n %d->%d",v,i);

dfs(i);

}

}

int main()

{

int i,j,count=0;

printf("\n Enter number of vertices:");

scanf("%d",&n);

for (i=1;i<=n;i++)
```

```
{
reach[i]=0;
for
 (j=1;j<=n;j++)a[i][j]=0;
}
printf("\n Enter the adjacency matrix:\n");
for (i=1;i<=n;i++)
for (j=1;j<=n;j++)
scanf("%d",&a[i][j]);
dfs(1);
printf("\n");
for (i=1;i<=n;i++)
{
if(reach[i])
count++;
}
if(count==n)
printf("\n Graph is connected");
elseprintf("\n Graph is not connected");
getch();
}
```

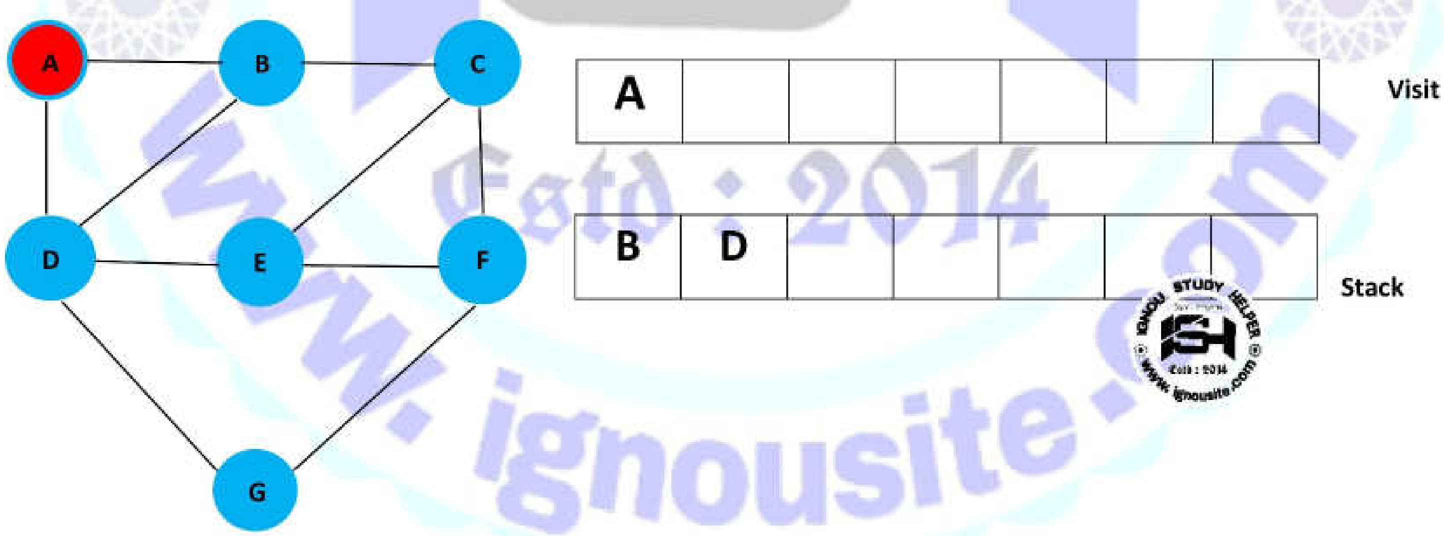
Output:

```
Enter number of vertices:7

Enter the adjacency matrix:
0 1 0 1 0 0 0
1 0 1 1 0 0 0
0 1 0 0 1 1 0
1 1 0 0 1 0 1
0 0 1 1 0 1 0
0 0 1 0 1 0 1
0 0 0 1 0 1 0
```

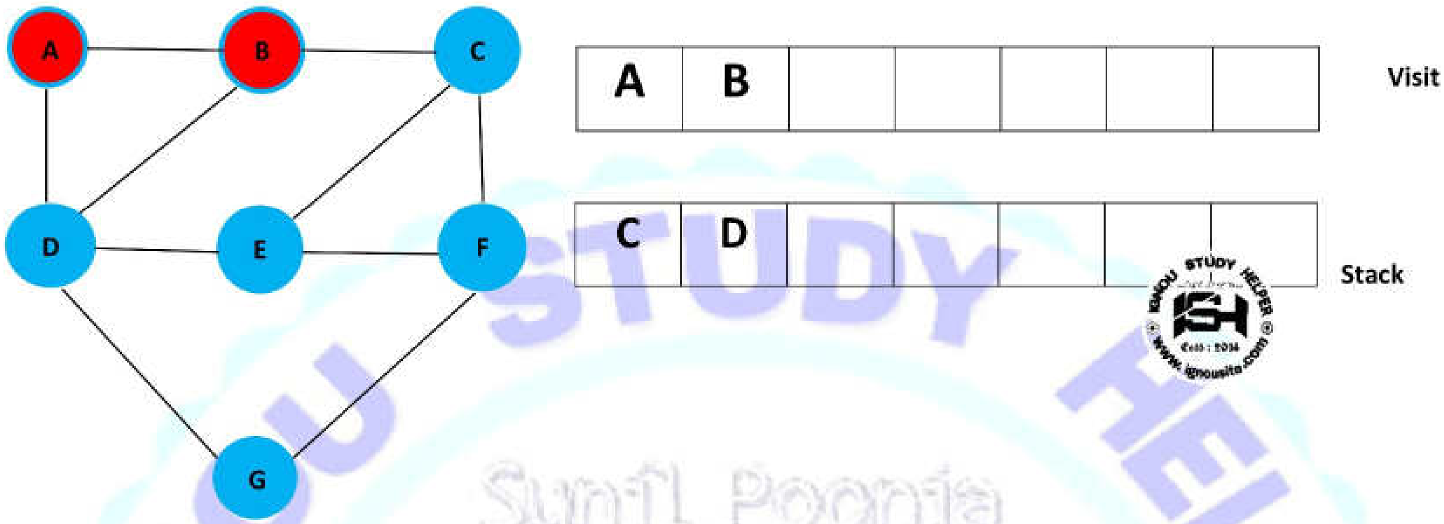
```
1 - >2
2 - >3
3 - >5
5 - >4
4 - >7
7 - >6

Graph is connected_
```



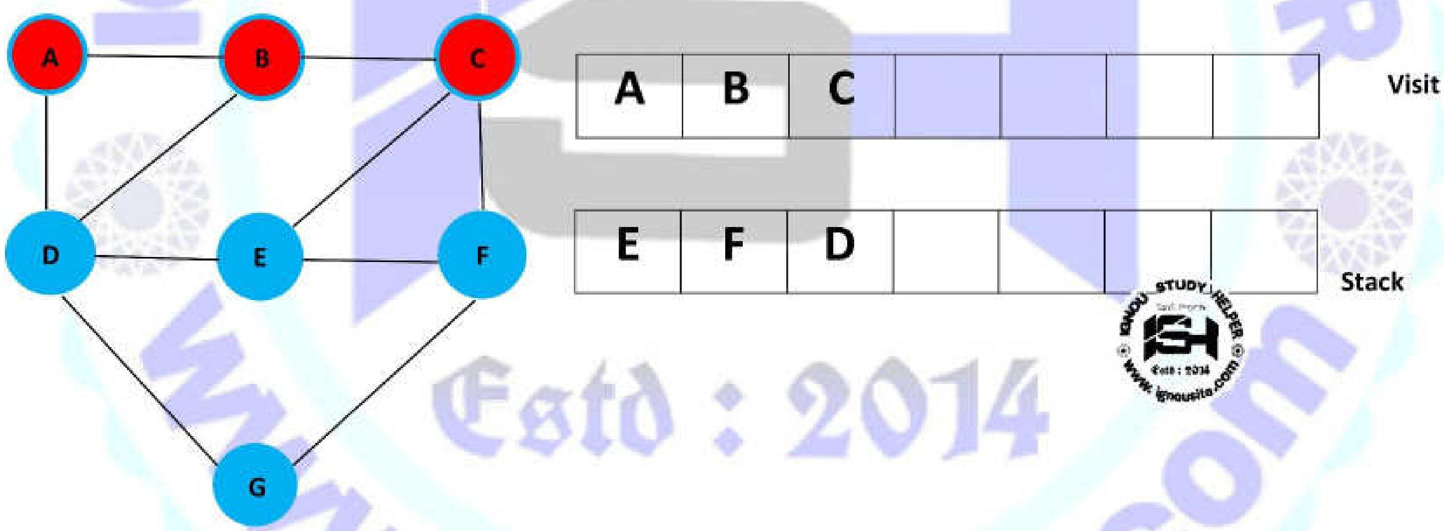
We start from vertex A, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



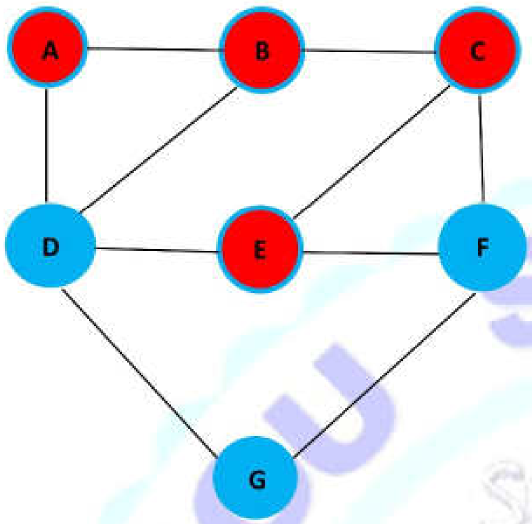
Next, we visit the element at the top of stack i.e. B and go to its adjacent nodes. Since A has already been visited, we visit C instead.



Vertex C has an unvisited adjacent vertex in E and F, so we add that to the top of the stack one after one and visit E or F (in this case we will go with E).



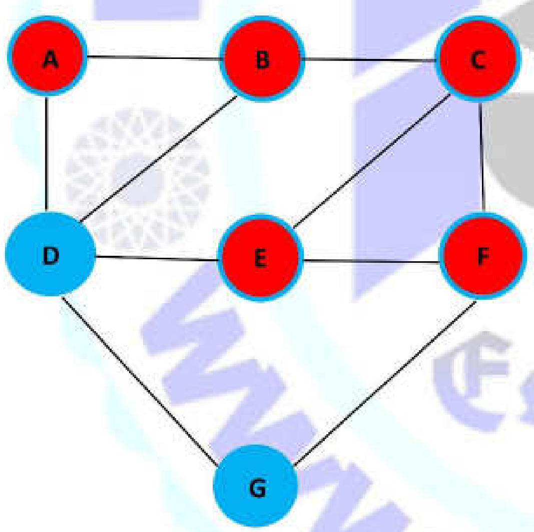
Next, we visit the element at the top of stack i.e. E and go to its adjacent nodes. Vertex E has no exploring adjacent nodes, so we visit F.



A	B	C	E				Visit

F	D						Stack

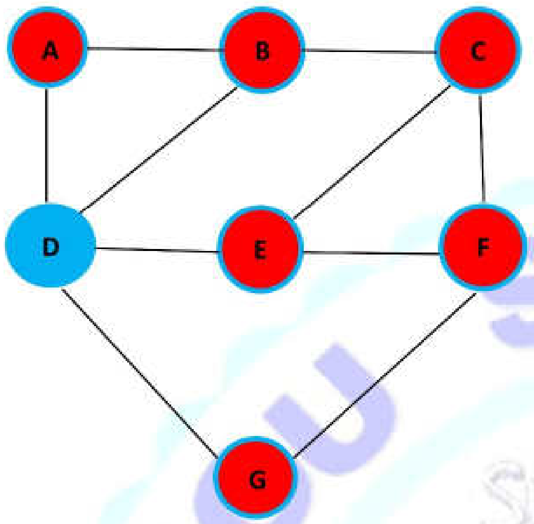
Vertex F has an unvisited adjacent vertex in G, so we add that to the top of the stack and visit it.



A	B	C	E	F			Visit

G	D						Stack

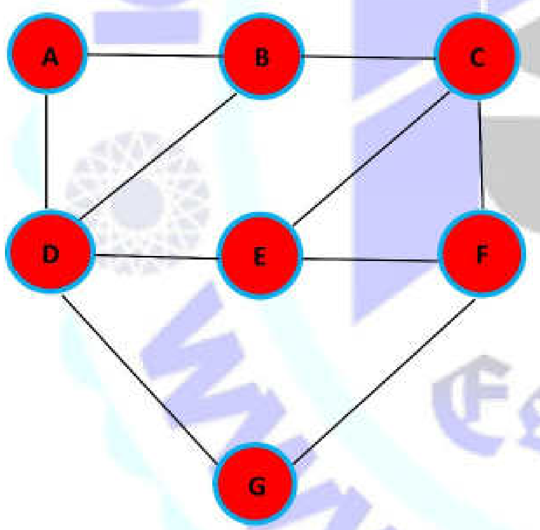
Next, we visit the element at the top of stack i.e. G and go to its adjacent nodes but Vertex G has no exploring adjacent nodes, so we visit D.



A	B	C	E	F	G		Visit

D							Stack

After we visit the last element D, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Search of the graph.



A	B	C	E	F	G	D	Visit

							Stack

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

A = 1, B = 2, C = 3, D = 4, E = 5, F = 6, G = 7 Loop will execute 7 times.

Q4. Implement Horner' rule for evaluating the following polynomial expression at x =5. Calculate the total number of times additions and multiplication operations will occur in this example.

$p(x) = 3x^5 - 4x^4 + 5x^3 - 6x + 9$

Ans. Program in c polynomial expression:

```
#include <stdio.h>

double horner(double *coeffs, int s, double x)

{

int i;

double res = 0.0;

for(i=s-1; i >= 0; i--)

{

res = res * x + coeffs[i];

}

return res;

}

int main()

{

double coeffs[] = { 9, -6, 0, 5, -4, 3 };

printf("\n The value of the polynomial 3x^5 - 4x^4 + 5x^3 - 6x + 9 for x = 5 is = %5.1f\n", horner(coeffs, sizeof(coeffs)/siz

return 0;

}
```

Output:

```
The value of the polynomial 3x^5 - 4x^4 + 5x^3 - 6x + 9 for x = 5 is = 7479.0

Process exited after 4.02 seconds with return value 0
Press any key to continue . . .
```

Addition and multiplication operations:

$p(x) = 3x^5 - 4x^4 + 5x^3 - 6x + 9$

$= 9 + [3x^5 - 4x^4 + 5x^3 - 6x]$

$= 9 + [x(3x^4 - 4x^3 + 5x^2 - 6)]$

$= 9 + [x(-6 + x^2(3x^2 - 4x + 5)]$

$= 9 + [x(-6 + x^2(5 + x(3x - 4)]$

$= 9 + [x.(-6 + x.x.(5 + x.(-4 + 3.x)]$

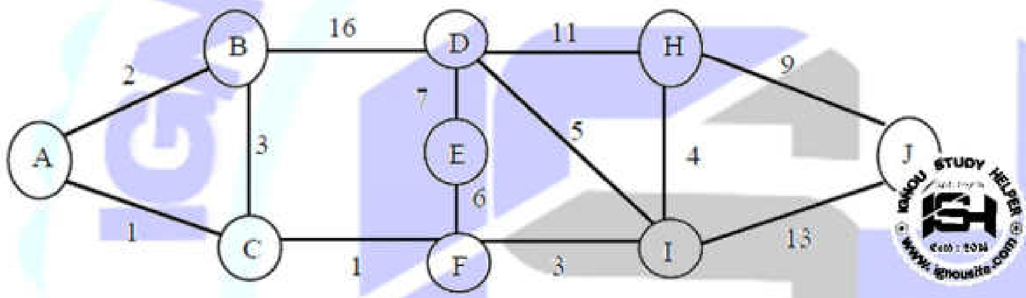
Number of multiplication operation = 5

Number of addition operation = 4

Now, p(5)= 9 + [5.(− 6 + 5.5.(5 + 5.(− 4 + 3.5)]

∴ p(5)= 9 + [5.(− 6 + 5.5.(5 + 5.(− 4 + 3.5)] = 7479

Q5. Implement and apply Kruskal's algorithm to find a minimum cost spanning tree and test the result for the following graph:



Ans. Kruskal's algorithm to find a minimum cost spanning tree:

```
#include <stdio.h>
#define MAX 30
typedef struct edge
{
int u, v, w;
}
edge;
typedef struct edge_list
{
edge data[MAX];
int n;
}
edge_list;
edge_list elist;
int Graph[MAX][MAX], n;
edge_list spanlist;
void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
```

```
void sort();
void print();
// Applying Krushkal Algovoid kruskalAlgo()
{
int belongs[MAX], i, j, cno1, cno2;
elist.n = 0;
for (i = 1; i < n; i++)
for (j = 0; j < i; j++)
 {
if (Graph[i][j] != 0)
{
elist.data[elist.n].u = i;
elist.data[elist.n].v = j;
elist.data[elist.n].w = Graph[i][j];
elist.n++;
} }
sort();
for (i = 0; i < n; i++)
belongs[i] = i;
spanlist.n = 0;
for (i = 0; i < elist.n; i++)
{
cno1 = find(belongs, elist.data[i].u);
cno2 = find(belongs, elist.data[i].v);
if (cno1 != cno2) {spanlist.data[spanlist.n] = elist.data[i];
spanlist.n = spanlist.n + 1;
applyUnion(belongs, cno1, cno2);
} } }
int find(int belongs[], int vertexno)
{
return (belongs[vertexno]);
}
void applyUnion(int belongs[], int c1, int c2)
{
int i;
for (i = 0; i < n; i++)
if (belongs[i] == c2)belongs[i] = c1;
}// Sorting algo
void sort()
{
int i, j;
edge temp;
for (i = 1; i < elist.n; i++)
for (j = 0; j < elist.n - 1; j++)
if (elist.data[j].w > elist.data[j + 1].w)
```

```
{
temp = elist.data[j];
elist.data[j] = elist.data[j + 1];
elist.data[j + 1] = temp;
} }
// Printing the resultvoid print()
{
int i, cost = 0;
for (i = 0; i < spanlist.n; i++)
{
printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
cost = cost + spanlist.data[i].w;
}
printf("\nSpanning Tree Minimum Cost: %d", cost);
}int main()
{
int i, j, total_cost;
n = 9;
// Creating Graph A=0, B=1, C=2, D=3, E=4, F=5, H=6, I=7, J=8Graph[0][0] = 0;
Graph[0][1] = 2;
Graph[0][2] = 1;
Graph[0][3] = 0;
Graph[0][4] = 0;
Graph[0][5] = 0;
Graph[0][6] = 0;
Graph[0][7] = 0;
Graph[0][8] = 0;

Graph[1][0] = 2;
Graph[1][1] = 0;
Graph[1][2] = 3;
Graph[1][3] = 16;
Graph[1][4] = 0;
Graph[1][5] = 0;
Graph[1][6] = 0;
Graph[1][7] = 0;
Graph[1][8] = 0;

Graph[2][0] = 1;
Graph[2][1] = 3;
Graph[2][2] = 0;
Graph[2][3] = 0;
Graph[2][4] = 0;
Graph[2][5] = 1;
Graph[2][6] = 0;
```

Graph[2][7] = 0;
Graph[2][8] = 0;

Graph[3][0] = 0;
Graph[3][1] = 16;
Graph[3][2] = 0;
Graph[3][3] = 0;
Graph[3][4] = 7;
Graph[3][5] = 0;
Graph[3][6] = 11;
Graph[3][7] = 5;
Graph[3][8] = 0;

Graph[4][0] = 0;
Graph[4][1] = 0;
Graph[4][2] = 0;
Graph[4][3] = 7;
Graph[4][4] = 0;
Graph[4][5] = 6;
Graph[4][6] = 0;
Graph[4][7] = 0;
Graph[4][8] = 0;

Graph[5][0] = 0;
Graph[5][1] = 0;
Graph[5][2] = 1;
Graph[5][3] = 0;
Graph[5][4] = 6;
Graph[5][5] = 0;
Graph[5][6] = 0;
Graph[5][7] = 3;
Graph[5][8] = 0;

Graph[6][0] = 0;
Graph[6][1] = 0;
Graph[6][2] = 0;
Graph[6][3] = 11;
Graph[6][4] = 0;
Graph[6][5] = 0;
Graph[6][6] = 0;
Graph[6][7] = 4;
Graph[6][8] = 9;

Graph[7][0] = 0;
Graph[7][1] = 0;

```
Graph[7][2] = 0;
Graph[7][3] = 5;
Graph[7][4] = 0;
Graph[7][5] = 3;
Graph[7][6] = 4;
Graph[7][7] = 0;
Graph[7][8] = 13;

Graph[8][0] = 0;
Graph[8][1] = 0;
Graph[8][2] = 0;
Graph[8][3] = 0;
Graph[8][4] = 0;
Graph[8][5] = 0;
Graph[8][6] = 9;
Graph[8][7] = 13;
Graph[8][8] = 0;
kruskalAlgo();
print();
}
```

Output:

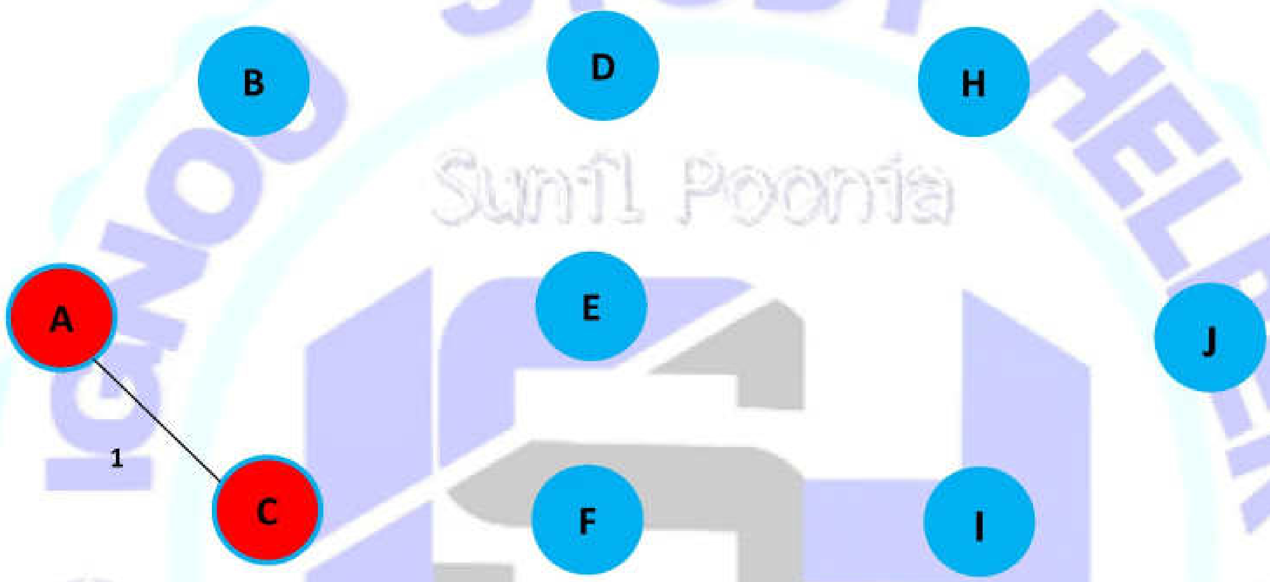
```
2 – 0 : 1
2 – 2 : 1
2 – 0 : 2
2 – 5 : 3
2 – 6 : 4
2 – 3 : 5
2 – 4 : 6
2 – 6 : 9
Spanning Tree Minimum Cost: 31
Process exited after 3.35 seconds with return value 0
Press any key to continue . . .
```

The graph G(V, E) given below contains 9 vertices and 13 edges. And you will create a minimum spanning tree T(V', E') for G(V, E) such that the number of vertices in T will be 9 and edges will be 8 (= 9 – 1). The given graph has no loops or parallel edges, so now we are going to next step. The next step that we will proceed with arranging all edges in a sorted list by their edge weights.

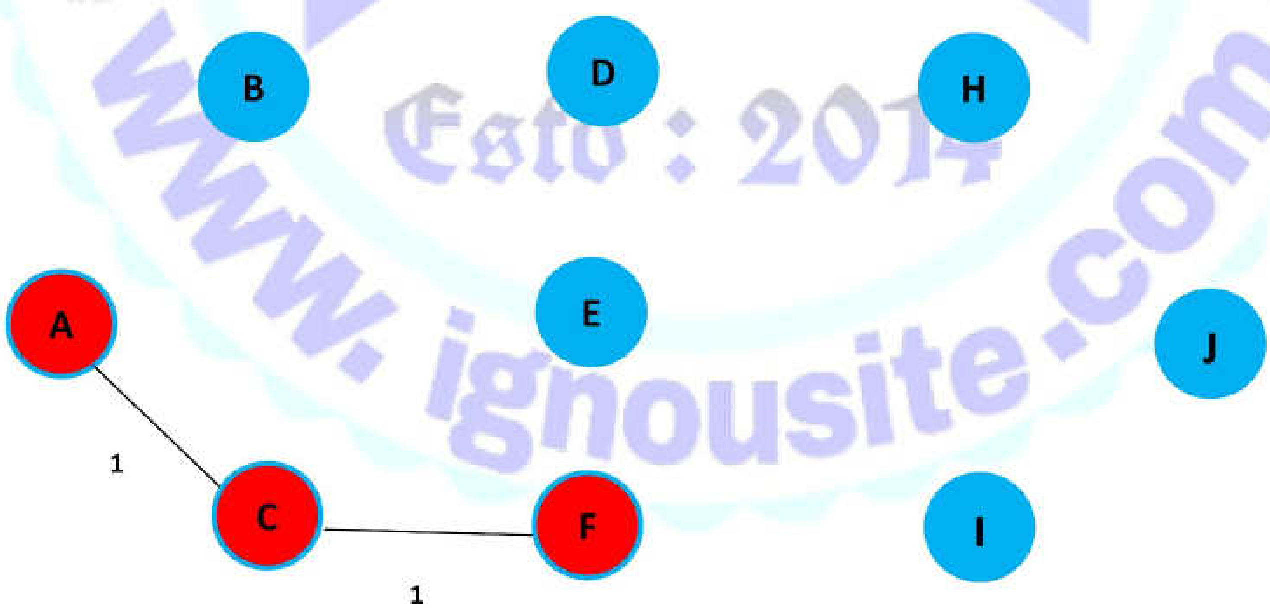
Weight	Source	Destination
1	A	C
1	C	F
2	A	B
3	B	C
3	F	I
4	I	H
5	I	D

6	F	E
7	E	D
9	H	J
11	D	H
13	I	J
16	B	D

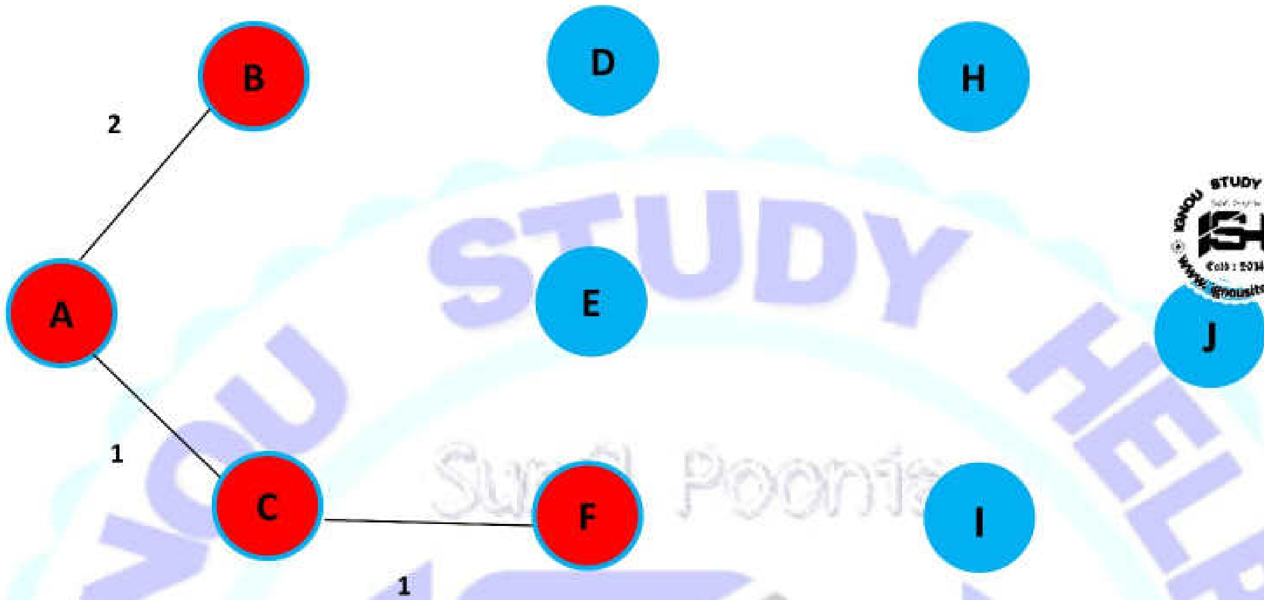
Now we will include edges in the MST such that the included edge would not form a cycle in our tree structure. The first edge that we will pick is edge A C, as it has a minimum edge weight that is 1.



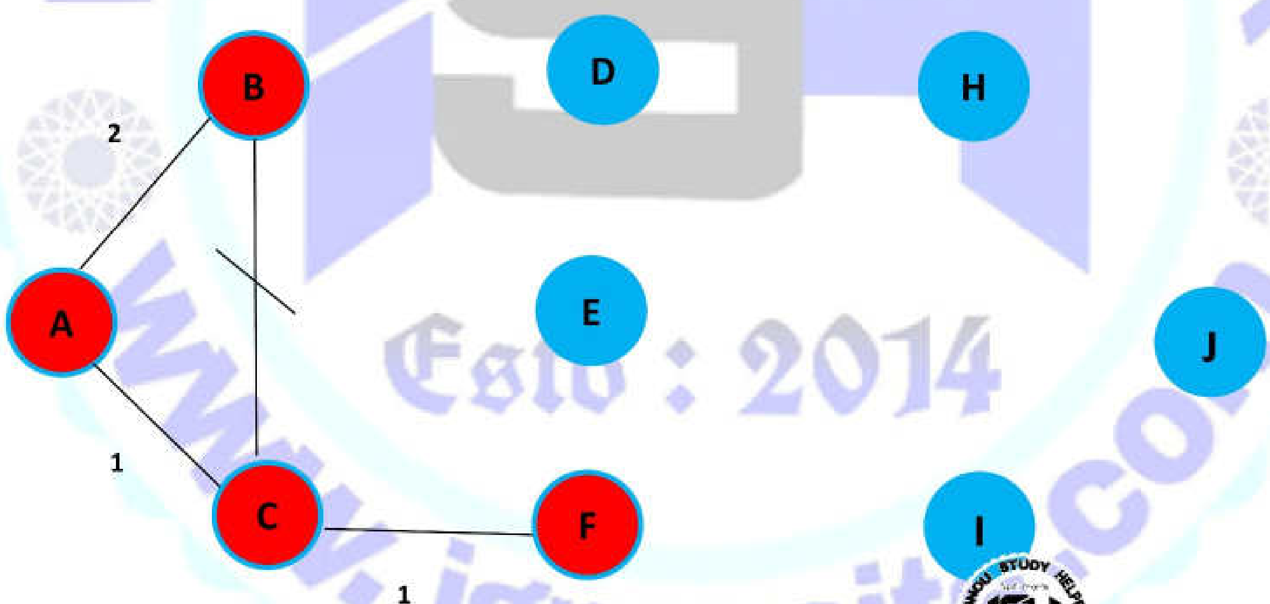
Following edge A C, we have edge C F to the spanning tree.



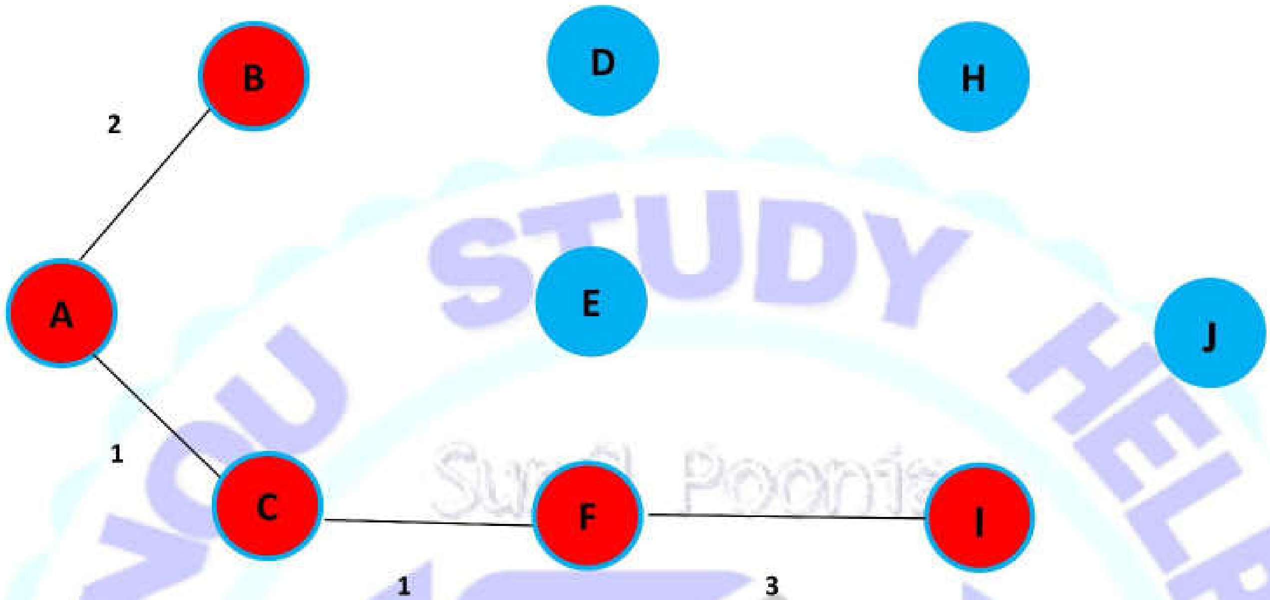
Following edge C F, we have edge A B to the spanning tree.



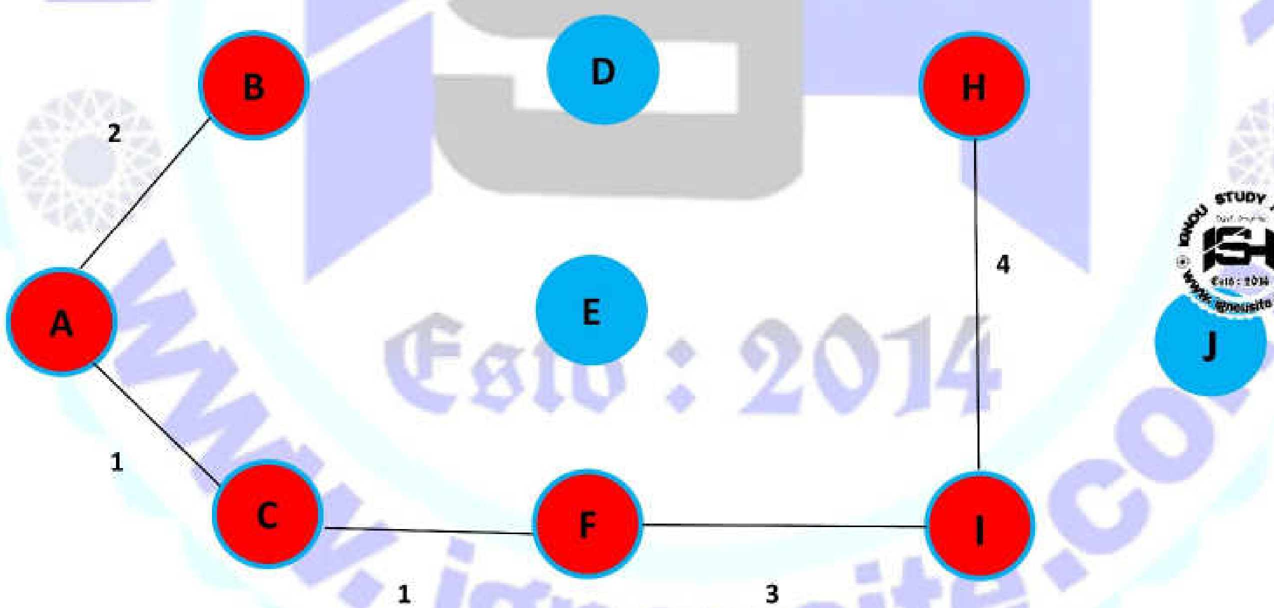
Next up is edge BC. This edge generates the loop in our tree structure. Thus, we will discard this edge.



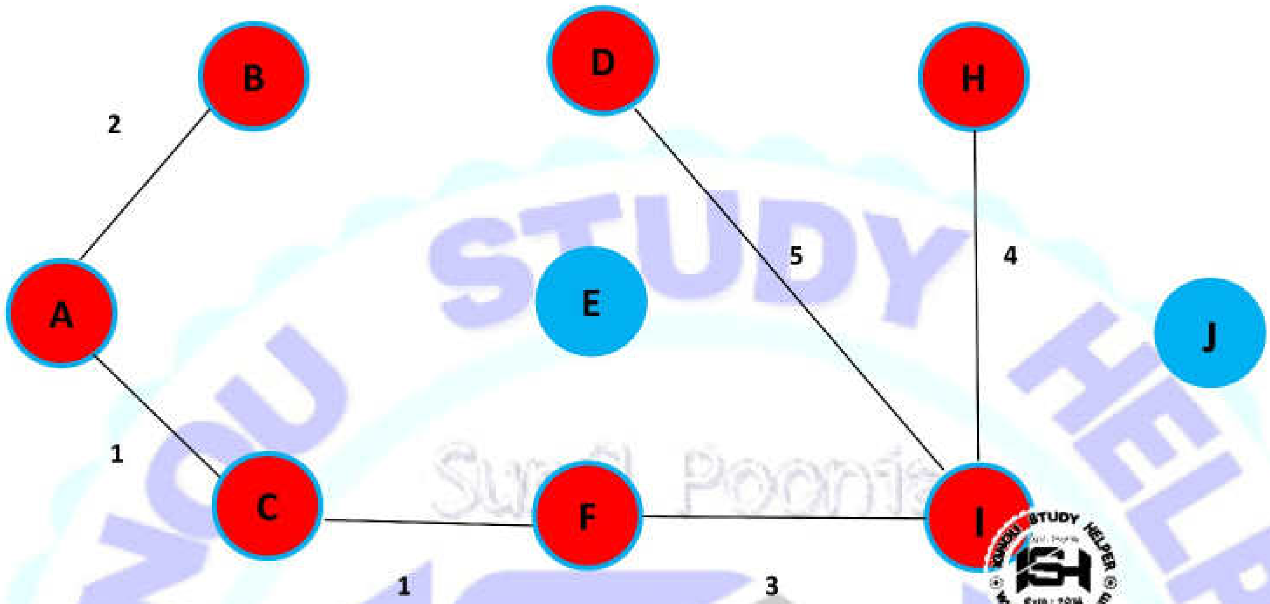
Add edge F I to the spanning tree.



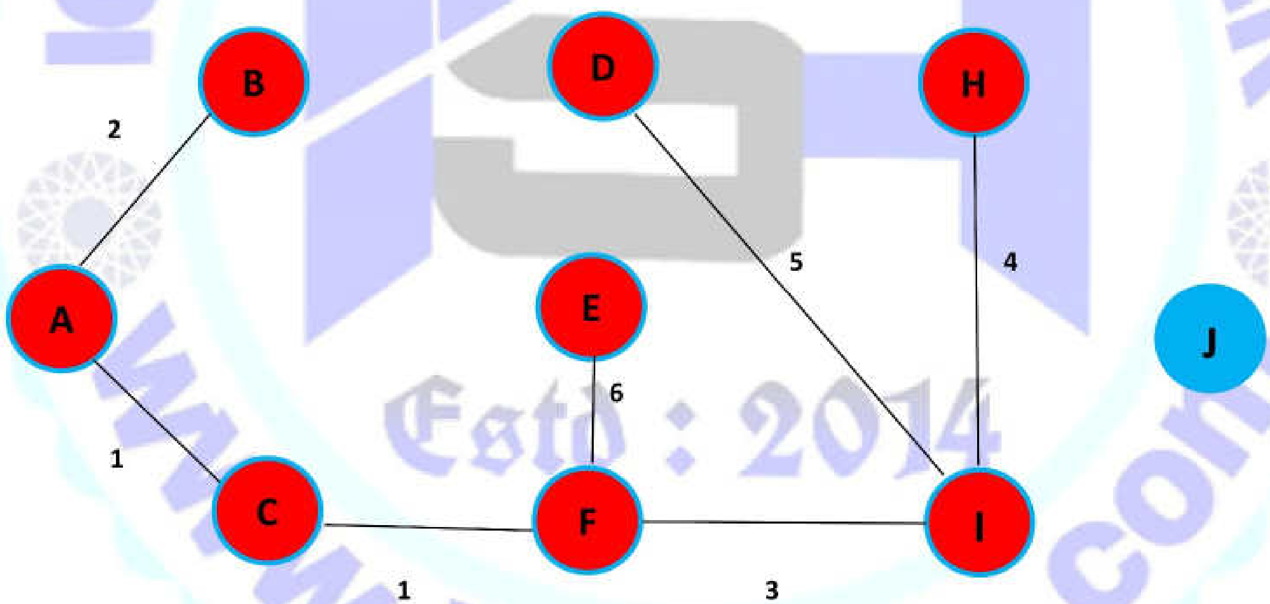
Add edge I H to the spanning tree.



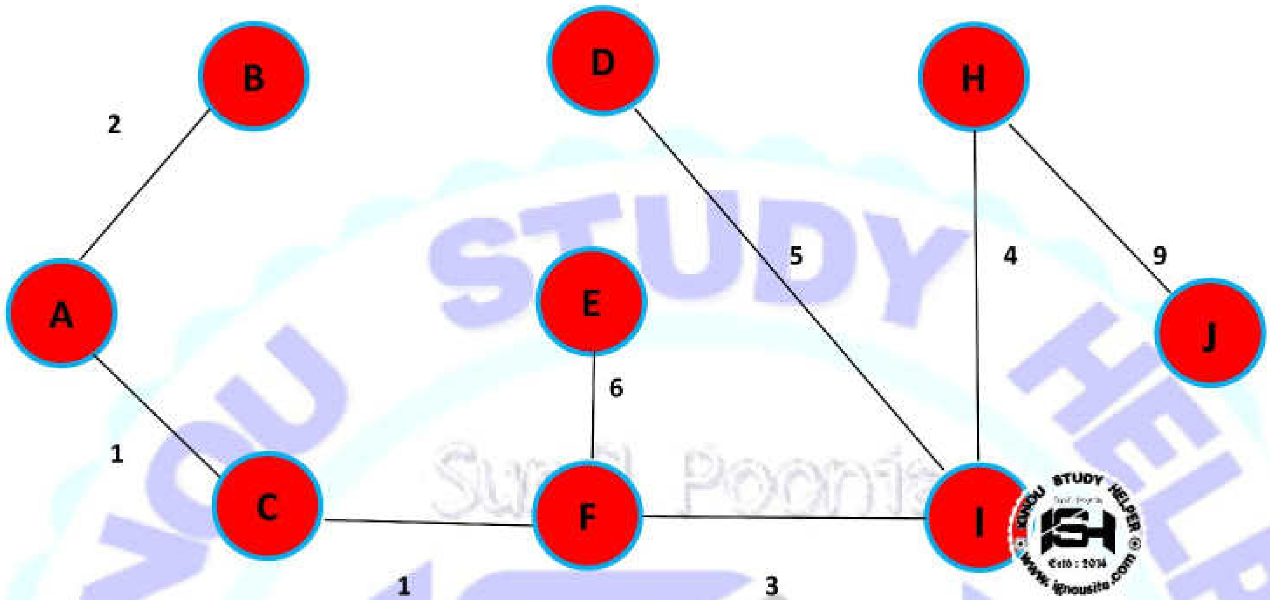
Following edge I H, we have edge I D to the spanning tree.



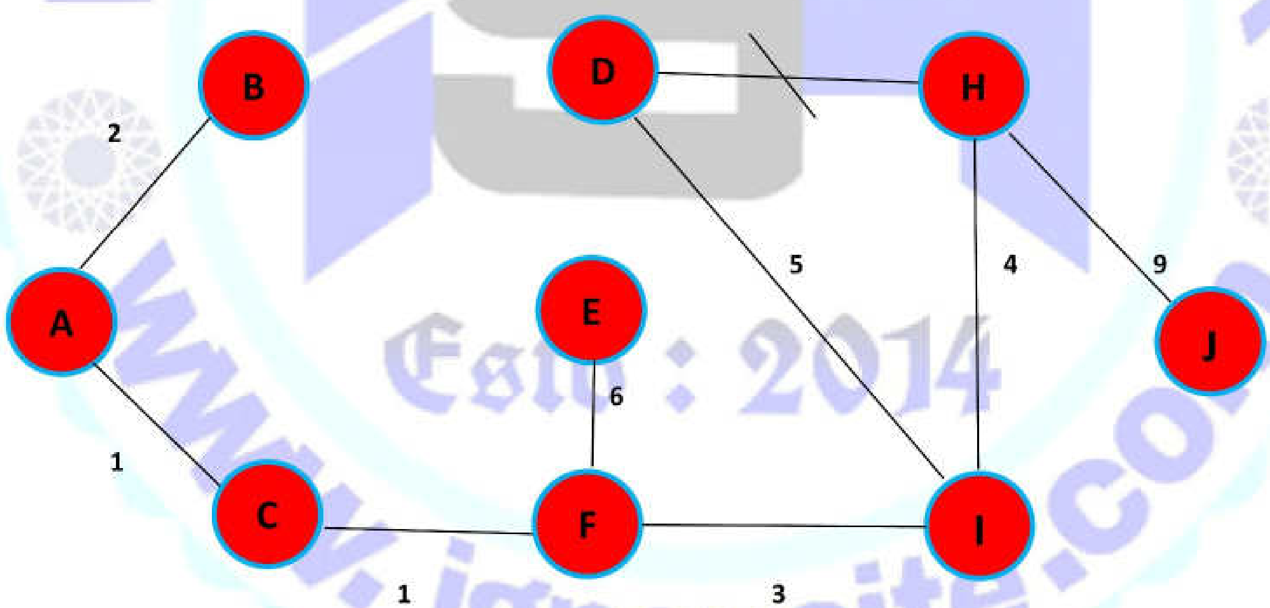
Following edge I D, we have edge F E to the spanning tree.



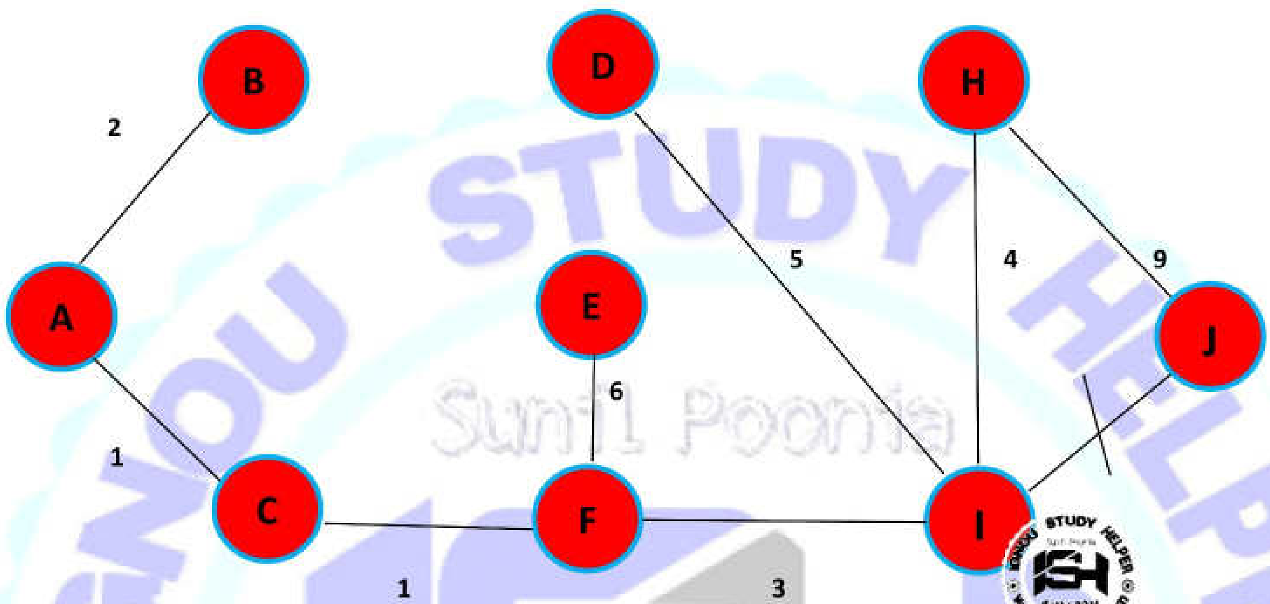
Next up is edge H J to the spanning tree.



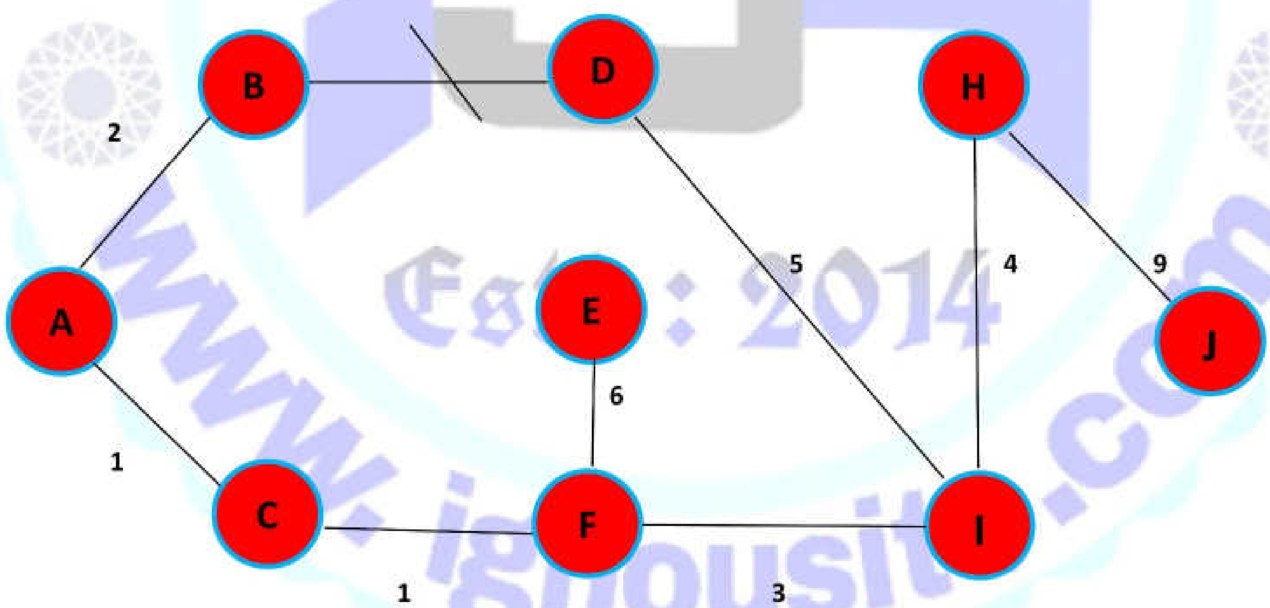
Following edge H J, we have edge D H. This edge generates the loop in our tree structure. Thus, we will discard this edge



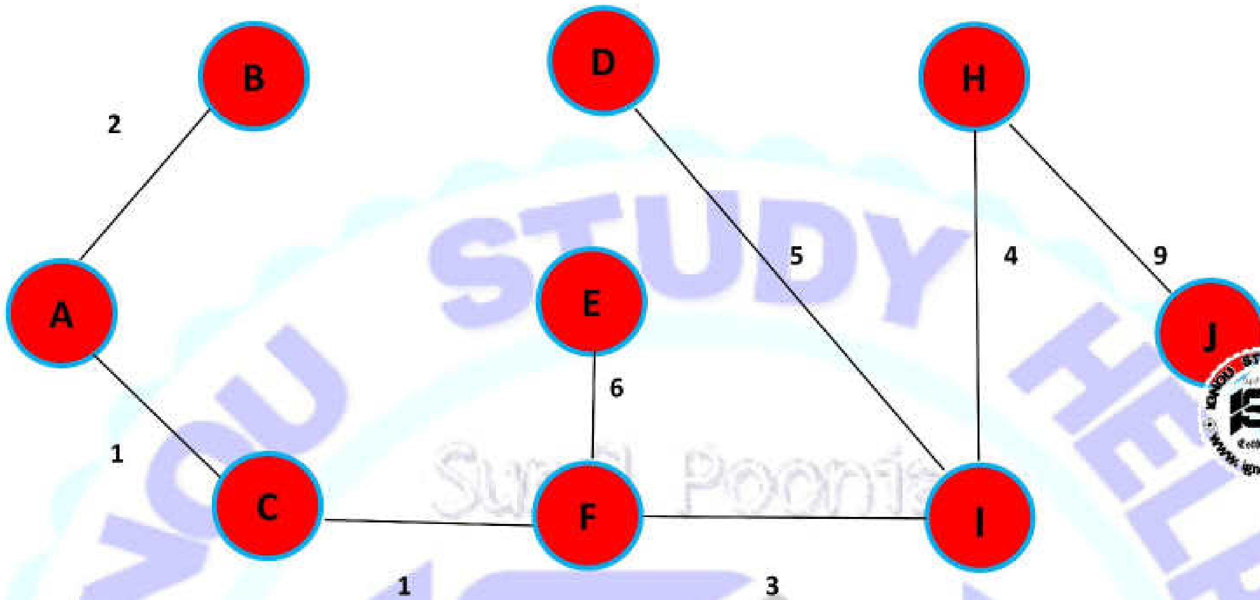
Next up is edge I J. This edge generates the loop in our tree structure. Thus, we will discard this edge.



Next up is edge B D. This edge generates the loop in our tree structure. Thus, we will discard this edge.



The final minimum spanning tree is as follows:



The minimum cost of spanning tree = 1 + 1 + 2 + 3 + 4 + 5 + 6 + 9 = 31.

Q6. Implement Karatsuba's method using Divide & Conquer method to multiply two integer numbers. Test the result in multiplication of the following numbers and count the number of multiplication operations.
5 3 2 6 8 0 * 4 3 2 8 6

Ans. Using Divide and Conquer, we can multiply two integers in less time complexity. We divide the given numbers in two halves. Let the given numbers be X and Y. For simplicity let us assume that n is even

X = $a*10^{n/2} + c$ [a and c contain leftmost and rightmost n/2 bits of X]

Y = $b*10^{n/2} + d$ [b and d contain leftmost and rightmost n/2 bits of Y]

The product X Y can be written as following.

XY = $(a*10n^{/2} + b)(c*10n/2 + d)$

= $10nac + 10^{n/2}(ad + bc) + bd$

Now we are going to multiply two given numbers 532680 and 43286 by Karatsuba's method using Divide and Conquer. In order to apply Karatsuba's method, first we make numbers of digits in the two given numbers equal, by putting zeros on the left of the number having lesser number of digits (in this case 43286).

Thus, the two numbers to be multiplied are written as

X = 532680 and Y = 043286 As, n = no of digits = 6

∴ n/2 = 3, now we can write X = 532680 = $532 \times 10^3 + 680 = a \times 10^3 + b$

Y = 043286 = 043 × 103 + 286 = c × 103 + d

where, a = 532, b = 680, c = 043, d = 286

Now, by the method of Karatsuba's multiplication, we have

XY = 532680 × 043286 = $10^n ac + 10^{n/2}(ad + bc) + bd$

XY = $10^6(532 \times 043) + 10^3(532 \times 286 + 680 \times 043) + 680 \times 286$

XY = 10^6 P + 103(Q + R) + S -------- (1)

Though, the above may be simplified in another simpler way, yet we want to explain Karatsuba's method using Divide and Conquer. Therefore, next we compute the products below

P = 532 × 043, Q = 532 × 286, R = 680 × 043, S = 680 × 286

Again apply above rules, but when it becomes into two digits, Karatsuba's method cannot be applied.

P = 532 × 043 = (53 × 10 + 02) × (04 × 10 + 03) = 10^2×(53 × 04) + 10×(53 × 03 + 02 × 04) + 02 × 03

= 21200 + 1670 + 6

= 22876

Q = 532 × 286

= (53 × 10 + 02) × (28 × 10 + 06)

= 10^2×(53 × 28) + 10×(53 × 06 + 02 × 28) + 02 × 06

=148400 + 3740 + 12

= 152152

R = 680 × 043

= (68 × 10 + 00) × (04 × 10 + 03)

= 10^2×(68 × 04) + 10×(68 × 03 + 00 × 04) + 00 × 03

= 27200 + 2040 + 0

= 29240

S = 680 × 286

= (68 × 10 + 00) × (28 × 10 + 03)

= 102×(68 × 28) + 10×(68 × 03 + 00 × 28) +00 × 03

= 190400 + 4080 + 0

= 194480 Now, on substitution of the value of P, Q, R and S in equation '1', we have XY

= 10^6× 22876 + 103(152152 + 29240) + 194480

= 22876000000 + 194480 + 13920000

= 23057586480

The number of multiplication operation in Karatsuba's method is O($n^{\log 23}$). In this case, n = 6 Therefore, the required number of multiplication process = O(17.114) = 18