Name: Subhapreet Patro
Roll No.: 2211CS010547
Class: Group-7A

In [9]:

```
from sympy import symbols, Eq, solve

# Define the variable and the characteristic equation
r = symbols('r')
characteristic_eq = Eq(r**2 - 5*r + 6, 0)

# Solve for the characteristic roots
characteristic_roots = solve(characteristic_eq, r)

# Print the characteristic roots
print("Characteristic Roots:", characteristic_roots)

# Define the variables for the particular integral
n, C1, C2 = symbols('n C1 C2')

# Extract characteristic roots
r1, r2 = characteristic_roots

# Form the particular integral
particular_integral = C1 * r1**n + C2 * r2**n

# Set up and solve the system of equations for the constants
equations = [particular_integral.subs(n, 0) - 0, particular_integral.subs(n, 1) - 1
constants = solve(equations, (C1, C2))

# Substitute the constants back into the particular integral
particular_integral = particular_integral.subs(constants)

# Print the particular integral and the values of C1 and C2
print("Particular Integral:", particular_integral)
print("Value of C1:", constants[C1])
print("Value of C2:", constants[C2])
```

```
Characteristic Roots: [2, 3]
Particular Integral: -2**n + 3**n
Value of C1: -1
Value of C2: 1
```

In [7]:

```
class DisjointSet:
    def __init__(self, vertices):
        self.parent = {v: v for v in vertices}
        self.rank = {v: 0 for v in vertices}

    def find(self, v):
        if self.parent[v] != v:
            self.parent[v] = self.find(self.parent[v])  # Path compression
        return self.parent[v]

    def union(self, root1, root2):
        if self.rank[root1] < self.rank[root2]:
            self.parent[root1] = root2
        elif self.rank[root1] > self.rank[root2]:
            self.parent[root2] = root1
        else:
            self.parent[root1] = root2
            self.rank[root2] += 1


def kruskal(graph):
    edges = []
    for vertex in graph:
        for neighbor, weight in graph[vertex]:
            edges.append((vertex, neighbor, weight))
```

```python
    edges.sort(key=lambda x: x[2])  # Sort edges by weight

    vertices = set(v for edge in edges for v in edge[:2])
    disjoint_set = DisjointSet(vertices)
    mst = []
    total_cost = 0

    for edge in edges:
        root1 = disjoint_set.find(edge[0])
        root2 = disjoint_set.find(edge[1])

        if root1 != root2:
            mst.append(edge)
            total_cost += edge[2]
            disjoint_set.union(root1, root2)

    return mst, total_cost

# Provided graph
graph = {
    'a': [('b', 15), ('d', 15), ('g', 5)],
    'b': [('a', 15), ('c', 3)],
    'c': [('b', 3), ('d', 5), ('e', 5)],
    'd': [('a', 15), ('c', 5), ('f', 4)],
    'e': [('c', 5), ('f', 15)],
    'f': [('d', 4), ('e', 15), ('g', 18), ('h', 15)],
    'g': [('a', 5), ('f', 18), ('h', 15)],
    'h': [('f', 15), ('g', 15)]
}

minimum_spanning_tree, total_cost = kruskal(graph)
print("Minimum Spanning Tree:", minimum_spanning_tree)
print("Total Cost of MST:", total_cost)
```

```
Minimum Spanning Tree: [('b', 'c', 3), ('d', 'f', 4), ('a', 'g', 5), ('c', 'd',
5), ('c', 'e', 5), ('a', 'b', 15), ('f', 'h', 15)]
Total Cost of MST: 52
```