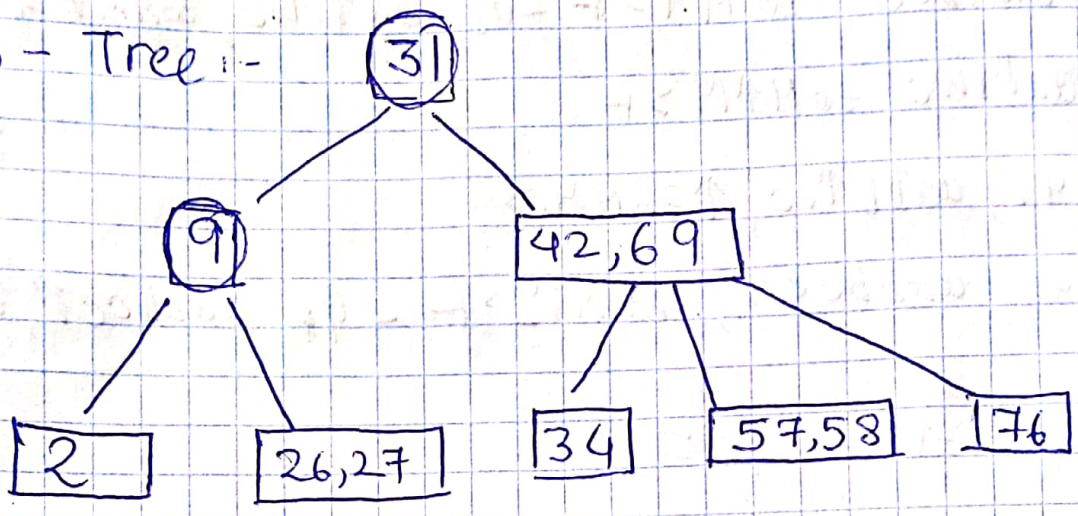
Theoretical part.

1.) B - Tree :-



2. Least square graph :-

Let us denote our graph by $G = (V, E)$ - this is the input graph.

$G_2 = (V, E_2)$ - this the square graph.

We assume that the nodes of the graph are indexed from 1 to N.

2.1 Using adjacency list.

Let adj(v) be the adjacency list of the vertex v.

2.1.1. Algorithm.

- For each 'v' we look for paths with exactly 2 edges such that they start from 'v'.
- We check the neighbours of vertices neighbouring 'v'.
- For each node 'n' $\in$ adj(v) - adding an edge in G_2 from 'v' to every vertex in the adjacency list i.e adj(n).

2.1.2 Pseudocode.

```
List<int>[] squareGraph(List<int>[] adjList, int N)

{ List<int> result[N]; // empty at start
  for (int i=1; i≤N; i++)
  { for (int v: adjList[u])
    { for (int w: adjList[v]))
      { result[u].add(w);
        {// There exists 2-edge path.
        } // Therefore we add an edge
      } // This added edge in square-
    }   //-graph is from u to w.
    return result;
}.
```

2.1.3 Time complexity:-

Algorithm has 3 nested loops. In compute graph, all loops perform N-Iterations

This results that the complexity of the algorithm is $O(N^3)$.

This is the worst case.

2.2. Using adjacency matrix.

Let M be the matrix for G_1 and M_2 the matrix for G_2.

2.2.1. Algorithm:-

$M^2 = M \cdot M$ { property of matrix }.

The above equation has non-zero entry if there is atleast one path of length 2 between the corresponding vertices.

Let a_{ij} and b_{ij} be components of M_2 and M^2 respectively $1 \le i, j \le N$.

$$a_{ij} = \begin{cases} 1 & \text{if } b_{ij} \neq 0 \\ 0 & \text{if } b_{ij} = 0 \end{cases}$$

Hence, our algorithm includes calculation of $(A)^2 = A \cdot A)$ and then setting the elements of M_2.

Simply, $M \cdot M$ first and M_2 later.

2.2.2. pseudocode

```
int [][] squaregraph(int [][] adjmatrix,
                     int N)
{ int squared_M[N][N] // matrix M.M
  for(int i=1, i<=N, i++)
  { for(int j=1; j<=N; j++)
    { squared_M[i][j] = 0.
      for(int k=1; k<=N; k++)
      { squared_M[i][j] += adjmatrix[i]
                           * adj matrix[k][j
      }// Above code is matrix-multipli-
       // --cation
  }.
}. int result[N][N] // adjacency matrix
    // result is for square graph.
  for(int i=1; i<=N; i++) // assigning val
  { for(int j=1; j<=N; j++)
    { if squared_M[i][j]==0 result[i][j]=
      else result[i][j] = 1;
    }
  }return result;
}.
```

2.2.3. Time complexity:-

Step 1: consist of 3 nested loops: $O(N^3)$

Step 2: consist of 2 nested loops: $O(N^2)$

Therefore, overall $O(N^3+N^2) = O(N^3)$.