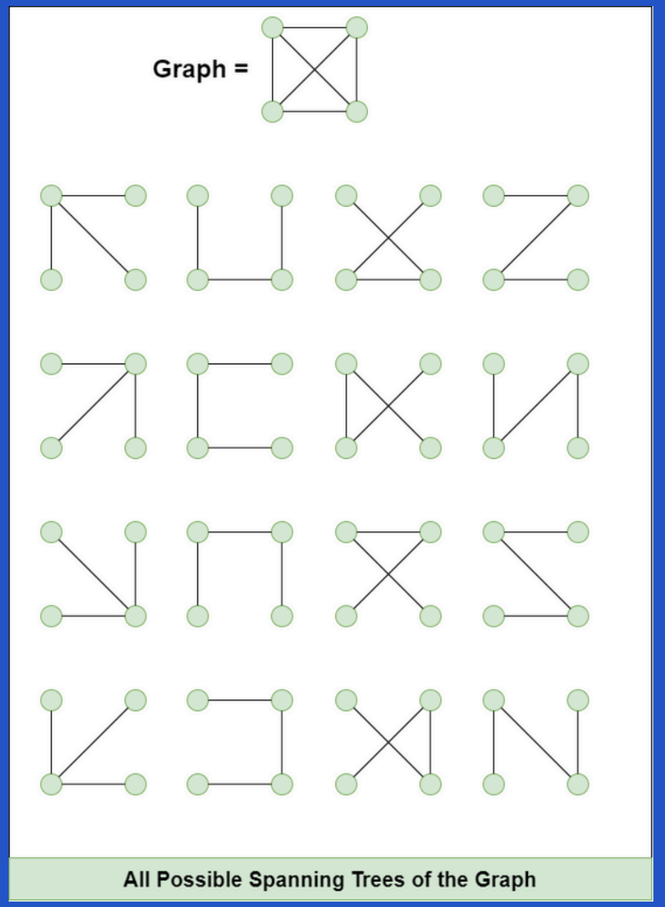
Maximum Leaf Spanning Tree Problem (MLSTP)

Gapaz
Mapute

What is a spanning tree?

- A spanning tree is a subset of a graph that connects all the vertices together with the minimum possible number of edges.
- Key Properties:
  - Contains all the vertices of the original graph.
  - Has no cycles (it’s a tree).
  - For a graph with N vertices, a spanning tree has exactly N-1 edges.
  - There can be multiple different spanning trees for a single graph.



Problem

Given a connected, undirected graph G = (V,E), the goal is to find a spanning tree T of G such that the number of leaf nodes in T is maximized.

Exhaustive approach

- Generate all possible combinations of (n-1) edges (where n = number of nodes).
- For each combination:
  - Check if it forms a spanning tree (connected, no cycles).
  - Count the number of leaf nodes (degree 1).
  - Track the combination with the most leaves.

Key Functions in the Code

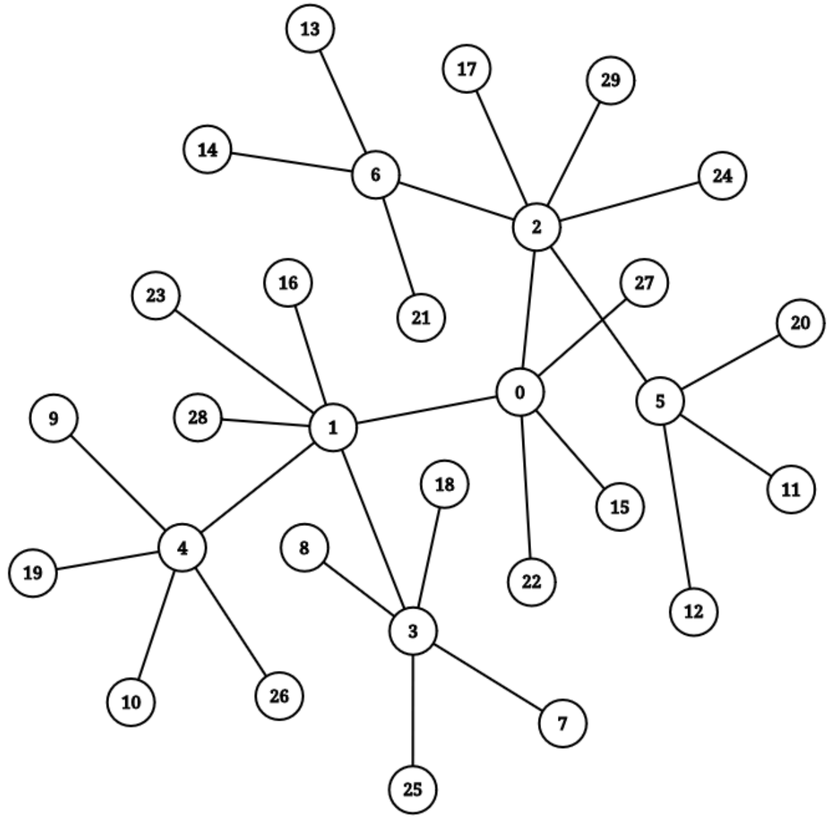
- generate_combinations: Recursively generates all (n-1)-edge combinations.
- is_connected: check if a combination is a spanning tree.
- count_leaves: Counts nodes with degree 1 in the current tree.
- print_best_tree & print_adjacency_matrix: Output the best tree found.

Sample Output

```
Exhaustive Search: Evaluating All Possible Spanning Trees
------------------------------------------------------------
Valid Spanning Tree #1 | Leaves: 23
Combination: (0-1) (0-2) (1-3) (1-4) (2-5) (2-6) (3-7) (3-8) (4-9) (4-10) (5-11) (5-12) (6-13) (6-14) (0-15) (1-16) (2-17) (3-18) (4-19) (5-20) (6-21) (0-22) (1-23) (2-24) (3-25)
6 3:5 4:5 5:4 6:4 7:1 8:1 9:1 10:1 11:1 12:1 13:1 14:1 15:1 16:1 17:1 18:1 19:1 20:1 21:1 22:1 23:1 24:1 25:1 26:1 27:1 28:1 29:1  | Leaves: 23
  [New Best Tree Found]
------------------------------------------------------------
Exhaustive Search Complete: All combinations have been checked.


Final Best Spanning Tree with 23 leaves:
Edges: (0-1) (0-2) (1-3) (1-4) (2-5) (2-6) (3-7) (3-8) (4-9) (4-10) (5-11) (5-12) (6-13) (6-14) (0-15) (1-16) (2-17) (3-18) (4-19) (5-20) (6-21) (0-22) (1-23) (2-24) (3-25) (4-26)
Node Degrees: 0:5 1:6 2:6 3:5 4:5 5:4 6:4 7:1 8:1 9:1 10:1 11:1 12:1 13:1 14:1 15:1 16:1 17:1 18:1 19:1 20:1 21:1 22:1 23:1 24:1 25:1 26:1 27:1 28:1 29:1

Adjacency Matrix of Best Spanning Tree:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
 0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  1  0  0
 1  1  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  1  0
 2  1  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  1
 3  0  1  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0
 4  0  1  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0
 5  0  0  1  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
 6  0  0  1  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
 7  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 8  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 9  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
10  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
11  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
12  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
13  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
14  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
15  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
16  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
17  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
18  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
19  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
20  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
21  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
22  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
23  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
24  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
25  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
26  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
27  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
28  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
29  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Time taken: 2.631422 seconds
```



Time Complexity

The part of the code responsible for this complexity is the generate_combinations function and the functions it calls (is_connected and count_leaves):

```
void generate_combinations(Edge *edges, int m, int k, int n, int start, Edge *curre
    if (cpos == k) {
        if (is_connected(current, k, n)) {
            int leaves = count_leaves(current, k, n);
            // ... print and update best tree ...
        }
        return;
    }
    for (int i = start; i < m; i++) {
        current[cpos] = edges[i];
        generate_combinations(edges, m, k, n, i + 1, current, cpos + 1);
    }
}
```

- Number of combinations: The recursive loop generates all possible combinations of (n-1) edges from m edges, which is C(m, n-1).
- For each combination: It calls is_connected and count_leaves, both of which are O(n).
- Total runtime: O(C(m, n-1) × n).

Non-Exhaustive approach

2-Approximation algorithm by Solis-Oba (1996)

What is an approximation Algorithm?

According to geeksforgeeks.org, an approximation algorithm is a way of dealing with optimization problems that are NP-hard. The goal of this algorithm is to come as close as possible to the optimal solution, usually in polynomial time. However, this technique does not guarantee the best solution.

To get the quality of an approximation algorithm, it must be measured by its approximation ratio/performance ratio.

For a maximization problem:

R = Optimal Solution/Approximate Solution

The closer R is to 1, the better the approximation

4 expansion rules:

- If F contains a vertex v with at least two neighbors in $V(F)^-$, then expand v.
- If F contains a vertex v with only one neighbor w in $V(F)^-$, which in turn has at least three neighbors in $V(F)^-$, then first expand v, and next expand w.
- If F contains a vertex v with only one neighbor w in $V(F)^-$, which in turn has two neighbors in $V(F)^-$, then first expand v, and next expand w.
- If $V(F)^-$ contains a vertex v with at least three neighbors in $V(F)^-$, then expand v.

Key Functions in the Code

- DFS: Create an initial spanning tree using DFS
- computeDegrees: Computes the degree of each vertex
- applyExpansion: Apply the 4 expansion rules by Solis-Oba
- dsu_find: find a direct connection from root to leaf
- dsu_union: connect the result of dsu_find into the spanning tree

Time Complexity

The part of the code responsible for this complexity is the application of the expansion rules

```
//application of the 4 expansion rules
//O(V + E)
void applyExpansion(Graph* original, Graph* tree, int V) {
    for (int u = 0; u < V; u++) {
        if (degree[u] >= 3) {
            Node* temp = original->array[u].head;
            while (temp) {
                int v = temp->vertex;
                if (dsu_find(u) != dsu_find(v)) {
                    addEdge(tree, u, v);
                    dsu_union(u, v);
                }
                temp = temp->next;
            }
        }
    }
}
```

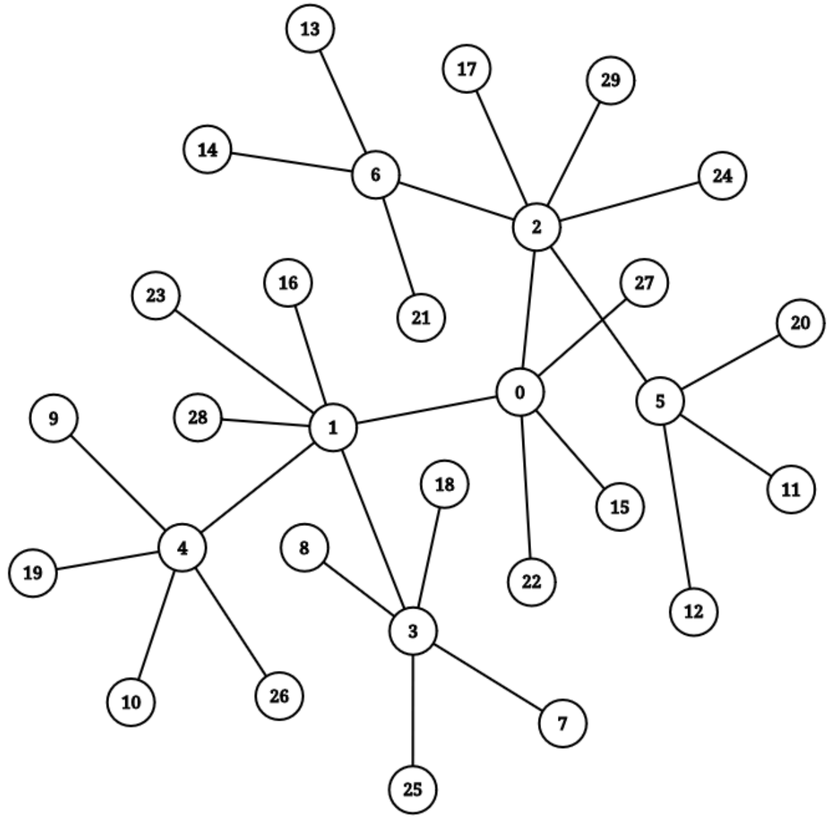
- Iteration over all vertices: O(V)
- Examine the edges of each vertex: O(E)
- DSU Operations: O(α(V))

So the time complexity is: O(V + E·α(V))
But α(V) is nearly constant, so we can just simplify it: O(V + E)

SAMPLE RUN

```
Approximate Spanning Tree:
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
0: 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0
1: 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0
2: 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
3: 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
4: 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0
5: 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
6: 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0
7: 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
8: 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
9: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
10: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
11: 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
12: 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
13: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
14: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
15: 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
16: 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
17: 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
18: 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
19: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
20: 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
21: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
22: 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23: 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24: 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
25: 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
26: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
27: 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
28: 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
29: 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Number of Leaves: 23
Time taken: 0.200000 seconds
```



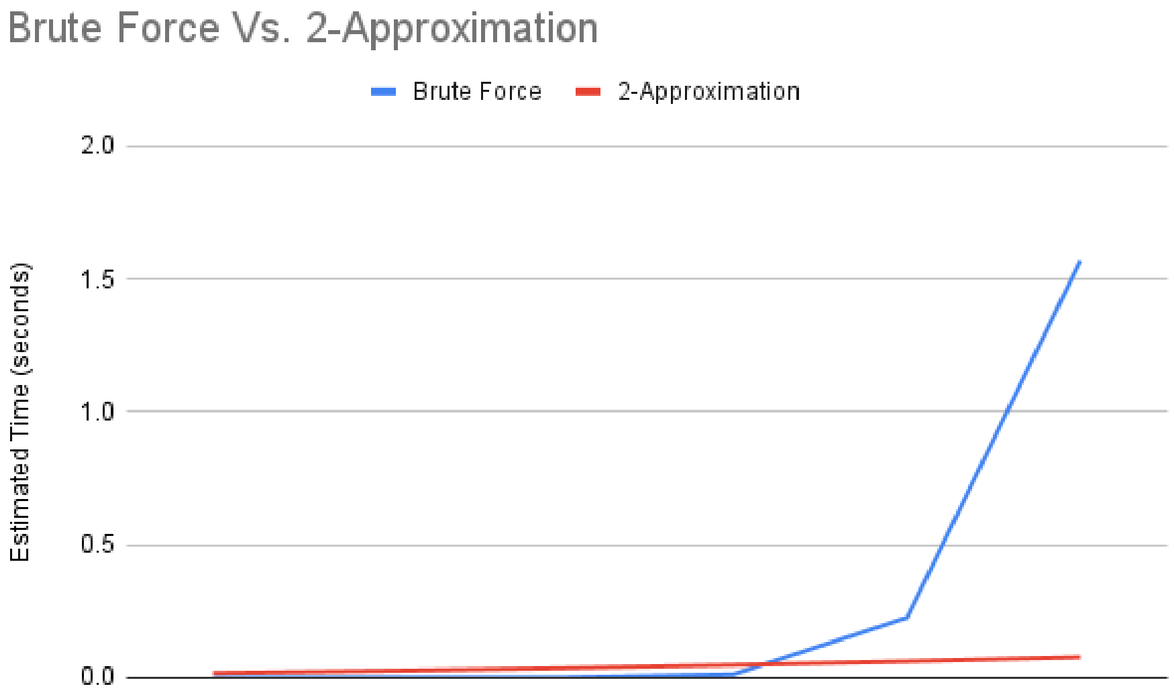
Comparison

Number of Vertices	Number of Edge	Num Leaves of Generated Spanning Tree Using Brute Force	Num Leaves of Generated Spanning Tree Using Brute Force
4	3	2	2
4	6	4	2
5	8	4	4

Comparison

Approach	Time Complexity	Optimal?	Practical for Large Graphs?	Notes
Exhaustive	O(C(m, n-1) × n)	Yes	No	Only for small graphs
2-approximation	O(V + E)	No	Yes	Fast, 2-approximation guarantee

Comparison


Brute Force Vs. 2-Approximation
Brute Force
2-Approximation
Estimated Time (seconds)
2.0
1.5
1.0
0.5
0.0

Brute Force vs 2-approximation

Brute Force: Generates the Optimal Solution, but slow

2-appxorimation: Faster but can only guarantee a solution that is >= 50% of the optimal solution

Real-Life Use Cases of the Maximum Leaf Spanning Tree Algorithm

1. Network Design

- Goal: Connect all computers/routers with the fewest connections to the "core" and as many "endpoints" (leaves) as possible.
- Why: More leaves mean more devices can be endpoints, making the network easier to expand and maintain.

2. Telecommunications

- Goal: Design phone or internet cable layouts where most houses are at the ends (leaves) of the network.
- Why: Reduces the number of cables running through each house, making installation and repairs simpler.

3. Power Distribution

- Goal: Lay out power lines so that most houses are at the ends of the network.
- Why: Minimizes the number of houses affected if a line breaks, and makes the system easier to manage.

Thank You

REFERENCES

- Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). A survey on sensor networks. IEEE Communications Magazine, 40(8), 102–114. https://doi.org/10.1109/MCOM.2002.1024422
- Chowdhury, A. A., & Koval, D. O. (2009). Power distribution system reliability: Practical methods and applications. IEEE Transactions on Industry Applications, 45(4), 1307–1313. https://doi.org/10.1109/TIA.2009.2023480
- Diestel, R. (2017). Graph theory (5th ed.). Springer.
- Galbiati, G., Morzenti, A., & Maffioli, F. (2014). On the approximability of the maximum leaf spanning tree problem. Theoretical Computer Science, 513, 98–107. https://doi.org/10.1016/j.tcs.2013.10.012
- GeeksforGeeks. (n.d.). Approximation algorithms. Retrieved May 16, 2025, from https://www.geeksforgeeks.org/approximation-algorithms/
- GeeksforGeeks. (n.d.). Spanning Tree. https://www.geeksforgeeks.org/spanning-tree/
- Pióro, M., & Medhi, D. (2004). Routing, flow, and capacity design in communication and computer networks. Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-557189-0.X5000-4