Abstract: The median path problem (min-sum criterion) is a common problem in graph theory and tree networks. This problem is open to study because its applications are growing and extending in different fields, such as providing insight for decision-makers when selecting the optimal location for non-emergency services, including railroad lines, highways, pipelines, and transit routes. Also, the min-sum criterion can deal with several networks in different applications. The location problem has traditionally been concerned with the optimal location of a single-point facility at either a vertex or along an edge in a network. Recently, numerous investigators have investigated this classic problem and have studied the location of many facilities, such as paths, trees, and cycles. The concept of the median, which measures the centrality of a vertex in a graph, is extended to the paths in a graph. In this paper, we consider the problem of locating path-shaped facilities on a tree network. A new modified and improved algorithm for finding a median single path facility of a specified length in a tree network is proposed. The median criterion for optimality considers the sum of the distances from all vertices of the tree to the path facility. This problem under the median criterion is called the ℓ-core problem. The distance between any two vertices in the tree is equal to the length of the unique path connecting them. This location problem usually has applications in distributed database systems, pipelines, the design of public transportation routes, and communication networks.

Keywords: core; facility location; median; central diameter of a tree; tree network

MSC: 05C90; 05C05

1. Introduction

Graph theory began in 1736 with Euler’s work, in which he solved the Konigsberg bridge problem [1]. It is noticeable that in the last half of the twentieth century, research improved in terms of both operational research and discrete mathematics thanks to the computer revolution and the development of computer uses [2].

The terms graph, network, and tree are closely related to algebra. The term network is used in concrete settings, for example, a computer network, a network of friends, social networks, and social networking sites [3,4].

Trees will certainly make us think about what we see in our backyards, parks, and forests, but also about genealogical trees.

In fact, graphs, networks, and trees are very important concepts in computer science. Also, they are important branches of mathematics in the study of graph theory.

The expressions network and graph are mostly used mutually. Work networks are used more often in concrete applications, and a graph is often seen as the abstract representation of a network. For example, in computer networks and social networks, we can see how they share some common notions.
Modern mathematical techniques and powerful computers have played an important role in spreading the use of social networks in many disciplines, such as economics, biology, engineering, and informatics [1,5,6].

Location theory is considered an important domain in operations research. Classical location theory is concerned with finding the optimal location of a set of points on a given network. One of the important location problems for more than one facility is the p-median problem.

A well-known facility location problem in a general network is the p-median problem. Consider a connected and directed network. This median-path problem is relevant to many applications, as studied by several researchers. For example, Avella et al. [7] presented an algorithm to deal with the median-path problem that has arisen in the design of bus and underground train routes in a mass transportation system.

The target of this study was to find the optimal location of a subset of nodes, i.e., facilities, on a network, e.g., the optimal paths of underground trains, overground trains, and trams.

In this paper, we consider that the network is an unweighted tree, and the relationships between its nodes are unidirectional, either up or down, from the root to the leaves.

We assume that this tree represents a graph of a city and its vertices are centers (facilities) in the city, i.e., the area of application of this study is how a public transportation path can be specified to serve the maximum possible number of facilities in a huge city.

In general, this paper concerns the problem of locating a path- or tree-shaped facility on a network. Namely, we aim to find the best path \( P \) in the tree \( T \), that minimizes the sum of the distance from the path \( P \) to all vertices in \( T \).

Network models are a very important type of mathematical program that have several practical applications.

Path planning has attracted the attention of many authors, who have proposed several algorithms depending on this concept, e.g., for unmanned aerial vehicles (UAVs). See [8–11].

Many network problems can be solved by linear programming, and, in fact, special extremely fast variants of linear programming can be applied. This study focuses on problems in which the facilities are path-shaped and the minisum criterion is applied. This field of study has been interesting to researchers past and present. Thus, we provide a comprehensive survey of some previous literature. Several applications require the location of a path-shaped facility instead of a single point or a set of points. For example, the location of pipelines and express lanes in a highway and the design of public transportation routes can be regarded as the location of path-shaped facilities. These problems are NP-hard on general networks, while they are polynomially solvable on tree networks [12].

The core of a tree is defined as a path that is optimal with respect to the property of minimizing the sum of the distances from each vertex in the tree to the path. Richey [13] presented a linear time algorithm for finding the core of a tree network of equal arc lengths. Also, he showed how this algorithm could be extended when there are different arc lengths.

Becker [14] presented a linear time algorithm for finding a minisum path of a tree. He also presented a general bottom-up and top-down procedure that was outlined with several examples. Motevalli et al. [15] presented a study for finding the \((k, \ell)\)-core problem on a tree network whose vertices have positive or negative weights. The \((k, \ell)\)-core of the \( T \) is a subtree with at most \( k \) leaves and with a diameter of at most \( \ell \), where the sum of the weighted distances from all vertices to this subtree is minimized. They showed that, when the sum of the weights of the vertices is negative, the \((k, \ell)\)-core must be a single vertex. They proposed an algorithm with a time complexity of \( O(n^2 \log n) \) for finding the \((k, \ell)\)-core of a tree with positive or negative weight.

Ashkezari and Fathali [16] presented an algorithm for finding the \(2-((k, \ell))\)-core of a tree with a pos-neg weight.

Also, they indicated that in the case when the sum of weights of a tree is negative, the solution of \(2-(k, \ell)\)-core is also \( \ell \)-median. Therefore, the authors of [16] have stated some properties in a case a tree has a positive weight. Moreover, they presented a polynomial algorithm to find a solution to this problem.
Tamir [17] improved the complexity bound of the p-median problem on the tree networks by showing that the total running time of the “leaves to root” dynamic programming algorithm is $O(pn^2)$.

Dankelmann and Osaye [18] presented upper bounds on the average eccentricity of the graphs in terms of order and either k-packing number, k-domination number, or connected k-domination number.

Wang et al. [19] presented two efficient algorithms for finding a $(k, \ell)$-tree core of the $T$.

The time complexity of the first algorithm is $O(n^2)$ when the edge is an arbitrary length, and the second algorithm has $O(\ell kn)$ time complexity for the case that the lengths of all edges are equal to 1.

Moshtagh et al. [20] developed two algorithms for finding the optimal path on a tree network along with the queues.

The above brief survey of the previous literature motivates us to present the following contributions in this paper.

- Suggesting a new approach to re-orientate the directed path on a rooted tree network ($T_r$).
- Adaptation and modification of Dijkstra’s algorithm for computing the distances between the vertices on the tree network and the number of vertices on each subtree.
- Design two algorithms that will be applied to compute the optimal distances in both upper and lower subtrees.
- Combining all presented algorithms in this study to obtain a new modified and improved algorithm for finding a core of length $\ell$ of a tree network.

The rest of this paper is organized as follows. In the next section, some of the notations and definitions with a problem description are presented. Section 3 gives a new approach to re-orientate the directed path on a rooted tree network. Section 4 presents some parameters, definitions and functions to obtain a modified and improved algorithm. Numerical results, complexity of time and some examples of practical networks are presented in Section 5. In Section 6, the conclusion and further work are presented.

2. Notation and Definitions with Problem Description

According to the approach mentioned in [13,14,17,21–24], we present a new similar one with some major modifications and improvements that make the new approach more clear and easy to use. This approach depends on the idea of deleting each edge of a tree network to obtain two subtrees. The lower subtree is rooted at the head vertex of the deleted edge and the upper subtree is rooted at the tail vertex of the deleted edge and then a path with minisum in both subtrees is found.

Therefore, the median path that we intend to find is a combined path of the optimal piece path in the upper subtree with another piece path in the lower subtree so as to make the total length $\ell$.

Hence the length of the optimal path that is obtained in the upper subtree is $< \ell - 1$. Also, the length of the optimal path that is obtained in the lower subtree is $< \ell - 1$.

Thus, the median path is a mingle of the optimal edges in both subtrees beginning at the deleted edge [21].

Definition 1. Let $T = (V, E)$ be a tree network rooted at a vertex $r = v \in V$, $n = |V(T_r)|$ is the number of the vertices in the tree $T$, and the number of the edges of the tree $T_r$ is $|E| = n - 1$, and let $P$ be a path in the tree $T_r$. The path $P$ consists of sequences of vertices where $P \subset T_r$. The objective is to find the path $P$, which has the property of minimizing the sum of the distances from all vertices in the tree $T_r$ to the nearest vertex in the path $P$ (minisum criterion).

The mathematical formulation of this problem can be described as follows.

$$\min_{P} \ d(P) = \sum_{v \in V \setminus P} d(v, P). \quad (1)$$
The lengths of all edges are equal to one.

Solving Problem (1) is to minimize the sum of the distances from the path \( P \) to all the vertices of the tree \( T \). Hence, the optimal maximum saved distance is denoted by the function \( \delta(d(r), \alpha) \), i.e., it is obtained by the path \( P \). Where \( d(r) = \sum_{v \in V} d(v, r) \) is the sum of the distances from a vertex \( r \) to all vertices in the tree \( T_r \), \( \alpha = \min(\psi, \mu) \), the parameters \( \psi \) and \( \mu \) are defined by Definition 14 and Formulas (26) and (27), respectively.

In the next sections, we show how the optimal path is selected by implementing \( n - 1 \) of operations on the entire tree network.

There are several different ways to represent a graph on a computer.

In this paper, a tree network \( T_r \) is represented in a computer by using two ways, as follows.

**Definition 2 (An adjacency matrix).** Let \( T_r \) be a tree with \( n \) vertices that are assumed to be ordered from \( v_1 = r \) to \( v_n \). The \( n \times n \) matrix \( D \), which is called an adjacency matrix and defined by

\[
d_{ij} = \begin{cases} 
1 & \text{if a vertex } v_i \text{ parent of a vertex } v_j, \\
0 & \text{otherwise.} 
\end{cases}
\]  

(2)

**Definition 3 (Edge lists “Family Matrix”).** It is a simple way to represent a tree \( T_r \) as a list, or array, of \( n - 1 \) edges and it is called an edge list (FATH). The edge lists could be stored in the computer as

\[
\text{FATH} = \begin{bmatrix} f_{a_i} & s_{n_{i+1}} \end{bmatrix}_{n-1 \times 2},
\]

(3)

where a vertex \( f_{a_i} \) is a parent of vertex \( s_{n_{i+1}} \), \( i = 1, 2, \ldots, n - 1 \), when \( i = 1 \), then \( f_{a_1} = r \).

In general, an edge of the matrix FATH is denoted by \( fs = (f_{a_i}, s_{n_{i+1}}) \).

### 3. Re-Orientating a Path on a Rooted Tree Network

This approach is used to re-orient a directed path on a rooted tree network \( T_r \).

By deleting an edge \( fs = (f_{a_i}, s_{n_{i+1}}) \) of the tree \( T_r \), the tree \( T_r \) is divided into two subtrees, where the first one is rooted at the vertex “\( s_{n_{i+1}} \)” and denoted by

\[
S_{n_{i+1}} \subset T_r.
\]

(4)

The second subtree is rooted at the vertex “\( f_{a_i} \)” and denoted by

\[
F_{a_i} = T_r \setminus S_{n_{i+1}}.
\]

(5)

In the following, we show how the tree \( T_r \) is divided into two subtrees \( F_{a_i} \) and \( S_{n_{i+1}} \) namely upper and lower subtrees, respectively.

**Definition 4.** Let \( P_r = P\{v_1 = r, \ldots, v_k\} \) be a path starting from the root of the tree \( T_r \) and ending at the vertex \( v_k \), where \( 1 \leq k \leq \ell - 2 \), \( \ell = L(P_r) \) is the length of the diameter path \( P_d \) of the tree \( T_r \). Then, the orientation of the directed path \( P_r \) is reversed to be \( P'_{r} = P\{v_k, \ldots, v_1, r\} \), i.e., we find for the predecessors of vertex “\( f_{a_i} \)” and then we reverse the orientation of their direction. Algorithm 1 implements this task as follows.

**Algorithm 1.** Reverses the orientation of the directed path \( P_r \) as follows.
Algorithm 1 For reversing the orientation of the directed path $P_i$ in the tree $T$.

Input: fath, $\overline{\text{fath}}$, set $N_i = |\overline{\text{fath}}|$, $N = 1$, and $d$. $\triangleright$ Where $d$ is a number of levels in $T$, the matrix fath contains the levels from $L_1$ to $L_d$, while the matrix $\overline{\text{fath}}$ contains all levels $L_1, L_2, \ldots, L_d$.

1: Set $\eta = 1$.
2: for $i = 1 : N_i$ do $\triangleright$ algorithm is running until the levels of $\overline{\text{fath}}$ are scanned once.
3: \hspace{0.5em} \text{pick } \overline{\text{fs}}_i = (\overline{\text{fa}}_i, \text{sn}_i) \in L_j \subset \overline{\text{fath}}
4: \hspace{0.5em} \text{As long as } \overline{\text{fs}}_i \in L_j \subset \overline{\text{fath}} \text{, then we are looking for the predecessor of the vertex } \overline{\text{fa}}_i \text{ in the levels } L_{j-\eta} \text{ of the matrix fath respectively as follows: } \triangleright \text{ where } j = 1, \ldots, d \text{ and } \eta = 1, \ldots, j.
5: \hspace{0.5em} \text{Set } N_2 = |L_{j-\eta}|.
6: \hspace{0.5em} for $k = 1 : N_2$ do $\triangleright$ where $N_2 = |L_{j-\eta}|$
7: \hspace{1em} \text{pick } \text{fs}_k = (\text{fa}_k, \text{sn}_k) \in L_{j-\eta} \subset \text{fath}. \hspace{0.5em} \triangleright \text{ where the edge fs belongs to one of } \{L_{j-1}, L_{j-2}, \ldots, L_0\}.
8: \hspace{1em} \text{if } \text{fa}_k = = \text{sn}_k \text{ then}
9: \hspace{1.5em} \text{Set } N_2 = \text{sn}_k \text{ and stored into } P_r = \{u_N\}.
10: \hspace{1em} \text{Set } \overline{\text{fa}}_k = \text{fa}_k \text{ and } N = N + 1.
11: \hspace{1em} \text{if } j \geq 2 \text{ & } \overline{\text{fa}}_k \neq r \text{ then}
12: \hspace{1.5em} \text{Set } \eta = \eta + 1 \text{ and go to Line 5.}
13: \hspace{1em} \text{end if}
14: \text{end if}
15: \hspace{0.5em} \text{if } \overline{\text{fa}}_k = = r \text{ then}
16: \hspace{1em} \text{return re-orientated path } P_r = \{u_1, \ldots, u_N, r\}. \hspace{0.5em} \triangleright \text{i.e., } u_1 \rightarrow \ldots \rightarrow u_N \rightarrow r.
17: \hspace{1em} \text{Set } N = 1, \eta = 1 \text{ and go to Line 2.}
18: \hspace{1em} \text{else}
19: \hspace{1.5em} \text{go to Line 6.}
20: \hspace{1em} \text{end if}
21: \text{end for}
22: \text{end for}

The Mechanism of a Run of Algorithm 1

Algorithm 1 is run according to the following scenario: A given tree network $T = (V, E)$ is represented as a list of edges denoted by “a family matrix” defined by (3).

The set of edges $E$ is stored in the matrix FATH as levels $L_1, \ldots, L_d$. For example, the $L_i$ and $L_j$ denote the first and the second levels of the matrix FATH, respectively. Where the matrix FATH represents the edges that connect to the root $r$ of the tree $T$.

Note: the edges of the first level $L_1$ connect to the root vertex $r$, while the edges of the last level $L_d$ are the last generation of descendants of the root $r$ with leaf vertices, i.e., each edge of the last level $L_d$ ends by leaf vertex “sn”, where $d$ is the number of levels of the tree $T$.

Thus, the first level $L_1$ of the matrix FATH will be removed from the FATH and then we obtain a new matrix denoted by $\overline{\text{fath}} = \text{FATH} \setminus L_1$.

Similarly, the last level $L_d$ of the matrix FATH will be deleted, and then we obtain a new matrix denoted by $\text{fath} = \text{FATH} \setminus L_d$. We set $N_1 = |\overline{\text{fath}}| = n - \left(1 + |L_1|\right)$ is a number of edges of the matrix $\overline{\text{fath}}$, and $N_2 = |\text{fath}| = n - \left(1 + |L_d|\right)$ is a number of edges of the matrix fath.

Thus, Algorithm 1 visits each edge of the matrix $\overline{\text{fath}}$ only once.

Then, for each iteration $i = 1 : N_i$, Algorithm 1 visits $1 \leq k \leq |L_{j-\eta}|$ edges of the matrix fath, where $j = 1, 2, \ldots, d$ denotes the set of the indices of the levels, which belong to the matrix fath and $\eta = 1, 2, \ldots, j$ denotes the set of indices of the levels, which belong to the matrix fath.

For example, when we want to delete the first edge of the first level of the matrix $\overline{\text{fath}}$, then the first edge is $\overline{\text{fs}}_1 = (\overline{\text{fa}}_1, \text{sn}_1) \in L_1 \subset \overline{\text{fath}}$. 
Hence, we look for the predecessor of the vertex $f_{a_1}$ in the level $L_{1-1} = L_0$ of the matrix $fath$, i.e., in the worst case this operation needs $|L_0|$ iterations and the second edge of the matrix $fath$ needs $|L_0| - 1$ iterations.

Thus, the operation continues such that one or more edges of the first level of the matrix $fath$ need only one iteration. The stopping criterion of this operation is listed in Line 8 of Algorithm 1.

Accordingly, Algorithm 1 has two loops, which are the outer loop that is denoted by $i = 1 : N_1$ and the inner loop that is denoted by $k = 1 : N_2$.

Thus, the inputs of Algorithm 1 contain the following parameters: $fath$, $N_1 = |fath|$, $N = 1$ is the initial index of the targeted path $P_\pi$ and $d$ is the number of possible levels in the tree $T_S$.

In Line 1 of Algorithm 1, $\eta = 1$ is a counter of the levels of the matrix $fath$, which will be visited. The outer loop of Algorithm 1 begins at Line 2.

Each edge of the matrix $fath$ is deleted in Line 3 of Algorithm 1 by which two subtrees lower and upper subtree are obtained. Therefore, the upper subtree $F_{a_i}$ is created and rooted at the vertex $f_{a_i}$.

In Line 4, the level in which the edge $f_{s_i}$ belongs is selected, namely that $f_{s_i} \in L_j \subset fath$. Hence, we look for the predecessor of the vertex $f_{a_i}$ in the levels $L_{j-\eta}$ of the matrix $fath$, respectively, where $j = 1, \ldots, d$ and $\eta = 1, \ldots, j$, i.e., we pick the levels $L_{j-\eta}$ of the matrix $fath$ in which the vertex $f_{a_i}$ was a son.

In Line 5 of Algorithm 1, we set $N_2 = |L_{j-\eta}|$ that denotes the number of the edges in the level $L_{j-\eta}$.

In Line 6 of Algorithm 1, the inner loop runs. In Line 7 of Algorithm 1, we pick the edge $f_{s_k} = (f_{a_k}, s_{n_k}) \in L_{j-\eta} \subset fath$ in which it is possible the vertex $f_{a_i}$ is found.

In Lines 8–9 of Algorithm 1, the path $P_\pi = \{u_r\}$ is created. In Lines 11–20 of Algorithm 1, some conditions and operations are tested and implemented, respectively. Therefore, if the condition in Line 15 of Algorithm 1 satisfies, the targeted path $P_\pi = \{u_1, \ldots, u_r\}$ is completed.

To show how Algorithm 1 is run, by using the tree network which is depicted in Figure 1 as follows. The edge $f_{s} = (d,v)$ of the tree network $T_s$ of Figure 2 is deleted. The subtrees $S_v$ and $F_a$ are created. The subtree $S_v$ is obtained in a very easy way due to the direction of the edges in the subtree $S_v$ are the same direction of the edges in the original tree $T_s$. The little difficult is to create the subtree $F_a$; because the orientation of the directed path $P_\pi = P\{s, a, b, h, d\}$ must reverse to be $P_d = P\{d, h, b, a, s\}$ and then the subtree $F_a = T_s \setminus T_a$ is obtained, i.e, Algorithm 1 is applied to make the path $s \rightarrow a \rightarrow b \rightarrow h \rightarrow d$ become $d \rightarrow h \rightarrow b \rightarrow a \rightarrow s$. See Figures 2 and 3. So, we obtain the upper subtree $F_a$, which is rooted at the vertex $d$. See Figure 3.
4. Median Path Algorithm

To design the algorithm that finds the median path of a tree network $T_r$, therefore, we give some definitions, parameters, and formulas. Then, a new improved and modified algorithm is presented according to the following subsection.

4.1. Definition of Parameters

The parameters “$|Sn_v|$, $d(sn)(fs, 0)$, $d(fa)(fs, 0)$, “msdsn”, “msdfa”, “d(sn)”, “d(fa)”, 
“k(d(sn))”, “k(d(fa))”, “$\psi(k(d(sn));d(fa))$”, “$\mu(k(d(fa));d(sn))$”, “$\alpha(\psi, \mu)$” and
$\delta = d(r) - \alpha$, will be defined in the following:

**Definition 5.** Let $|Sn_v|$ be the number of the vertices in the subtree $T_v$, rooted at $v$, the $|Sn_v| = |T_v|$ is computed as follows for each $v \in V$, then.

$$|Sn_v| = \begin{cases} 
1 & \text{if } v \in S_d, \\
1 + m & \text{otherwise},
\end{cases} \quad (6)$$

where $S_d$ is the set of leaves, i.e., $S_d$ is the set of the vertices in the tree $T_r$ of degree 1 and $m$ is the number of the vertices in the subtree $T_v$ that are descendants of the vertex $v$.

**Definition 6.** Let $V_{sn}$ be a set of the vertices in the subtree $Sn_m$ and $E_{sn}$ be a set of the edges in the subtree $Sn_m$, and $d(sn)(fs, 0)$ be the sum of the distance from the vertex $sn$ to all vertices in the
subtree Sn which it is called “distancesum” [25].

It is defined by

\[ d(sn) (fs, 0) = \sum_{v \in V_{sn}} d(v, sn), fs \in E_{sn}. \]  \hspace{1cm} (7)

**Definition 7.** Let \( V_{fa} \) be a set of vertices in the subtree Fa, \( E_{fa} \) be the set of the edges in the subtree Fa and \( d(fa) (fs, 0) \) be the sum of the distances from the vertex fa to all vertices in the subtree Fa, and it is called “distancesum” [25]. The “distancesum” is defined by

\[ d(fa) (fs, 0) = \sum_{v \in V_{fa}} d(v, fa). \]  \hspace{1cm} (8)

Dijkstra’s algorithm solves the single-source shortest-paths problem on a weighted, directed graph \( G = (V, E) \) for the case in which all edge weights are non-negative [26].

The strategy of Dijkstra’s algorithm depends on visiting all reachable vertices from the source. Then we adapt Dijkstra’s algorithm for calculating the values of the \( |Sn| \), \( d(sn) (fs, 0) \) and \( d(fa) (fs, 0) \), which are defined by Formulas (6)–(8). Consequently, Algorithms 2 and 3 are designed to calculate the values of the \( |Sn| \), \( d(sn) (fs, 0) \) and \( d(fa) (fs, 0) \) in the two subtrees Sn and Fa, respectively.

**Algorithm 2** Adaptation and modification of Dijkstra’s algorithm to compute \( |Sn| \) and \( d(fa)(fs, 0) \)

**Input:** \( i = 0, T_{r} \) and fath.

**Output:** \( DS_{n} = [fa \ sn \ |Sn_{n}| \ d(sn)]_{(n-1)\times 4} \)

1. **for each** \( fs \in \text{fath}; \ fs = (fa, sn) \), set \( sn = fs(2) \), **do**
2. **delete** the edge \( fs \) from \( T_{r} \) to create the subtree \( Sn_{n} \).
3. **end for**
4. **for each** \( u \in V(Sn_{n}) \) **do**
5. \( d(u) \leftarrow \infty \).
6. \( \text{color}(u) = \text{white} \).
7. \( \text{set } Q_{sn} = \text{queue with all vertices} \) is defined in (9).
8. **end for**
9. **we find** for the vertex \( u \) in \( Q_{sn} \) that represents the \( sn \) and \( \text{we set } u = sn \) and \( d[u] = 0 \).
10. **sum \leftarrow 0, the sum of the distances from the vertex “sn” to all reachable vertices in Sn**.
11. \( \text{Num}_{Node} \leftarrow 0, \) counter of the vertices that are accessible in \( Sn_{n} \).
12. **while** \( Q_{sn} \neq \emptyset \) **do**
13. \( u = \text{Extract} - \min(Q_{sn}), \) select the element of \( Q_{sn} \) with the min distance \( (d(u)) \).
14. \( \text{color}(u) = \text{green} \).
15. \( \text{Num}_{Node} = \text{Num}_{Node} + 1 \)
16. **for each** \( v \in Adj(u) \) **do**
17. \( \text{colorAdj}(u) = \text{yellow} \).
18. **if** \( d(v) > d(u) + w(u,v) \) **then**
19. \( d(v) \leftarrow d(u) + w(u,v), \) relaxation step.
20. \( \text{sum} = \text{sum} + d(v) \).
21. **end if**
22. **end for**
23. \( \text{color}(u) = \text{black}, \) the vertex \( u \) has been visited.
24. **delete** the vertex \( u \) with its distance \( d(u) \) from \( Q_{sn} \).
25. **replace** the old value of the \( d[v] \) by the new value into the \( Q_{sn} \) for each \( v \in Adj(u) \).
26. **end while**
27. \( \text{set } d(sn)(fs, 0) = \text{sum} \).
28. **set** \( i = i + 1 \) and \( |Sn_{n}| = \text{Num}_{Node} \).
29. **store** the values of \( d(sn)(fs, 0) \) and \( |Sn_{n}| \) into \( DS_{n} = [fa \ sn \ |Sn_{n}| \ d(sn)]_{(n-1)\times 4} \)
30. **return** \( DS_{n} = [fa \ sn \ |Sn_{n}| \ d(sn)]_{(n-1)\times 4} \)
Algorithm 3 Adaptation and modification of Dijkstra’s algorithm to compute $|Fa_{fa}|$ and $d(fa)(fs,0)$

**Input:** $i = 0$, $T$, and fath

**Output:** $DFa = [fa \hspace{1em} sn \hspace{1em} |Fa_{fa}| \hspace{1em} d(fa)]_{(n-1)\times 4}$

1. for each $fs \in$ fath; $fs = (fa, sn)$, set $sn = fs(2)$.
2. delete the edge $fs$ from $T$, to create the subtree $Fa_{fa}$.
3. **end for**
4. for each $u \in V(Fa_{fa})$ do
5. \hspace{1em} $d(u) \leftarrow \infty$.
6. \hspace{1em} color($u$) = white.
7. \hspace{1em} set $Q_{fa} = \{\text{queue with all vertices}\}$ is defined in (9).
8. **end for**
9. we find the vertex $u$ in $Q_{fa}$ that represents the “fa” and we set $u = fa$ and $d[u] = 0$.
10. sum $\leftarrow 0$, the sum of the distances from the vertex “fa” to all reachable vertices in $Fa_{fa}$.
11. Num\_Node $\leftarrow 0$, counter of the vertices that are accessible in $Fa_{fa}$.
12. while $Q_{fa} \neq \emptyset$ do
13. \hspace{1em} $u = \text{Extract} - \min(Q_{fa})$, select the element of $Q_{fa}$ with the min. distance ($d(u)$).
14. \hspace{1em} color($u$) = green.
15. \hspace{1em} Num\_Node = Num\_Node + 1
16. for each $v \in Adj(u)$ do
17. \hspace{2em} color$Adj(u)$ = yellow.
18. \hspace{2em} if $d(v) > d(u) + w(u,v)$ then
19. \hspace{3em} $d(v) \leftarrow d(u) + w(u,v)$, relaxation step.
20. \hspace{2em} sum $\leftarrow$ sum $+$ $d(v)$.
21. **end if**
22. **end for**
23. \hspace{1em} color($u$) = black, the vertex $u$ has been visited.
24. deleted the vertex $u$ with its distance ($d(u)$) from $Q_{fa}$.
25. replace the old value of the $d[v]$ by the new value into the $Q_{fa}$ for each $v \in Adj(u)$.
26. **end while**
27. set $d(fa)(fs,0) = \text{sum}$.
28. set $i = i + 1$ and $|Fa_{fa}| = \text{Num\_Node}$.
29. store the values of $d(fa)(fs,0)$ and $|Fa_{fa}|$ into $DFa = [fa \hspace{1em} sn \hspace{1em} |Fa_{fa}| \hspace{1em} d(fa)]_{i\times 4}$
30. **return** $DFa = [fa \hspace{1em} sn \hspace{1em} |Fa_{fa}| \hspace{1em} d(fa)]_{(n-1)\times 4}$

In both Algorithms 2 and 3, the process mechanism of selecting a vertex $u$ depends on the distance $d(u)$ of a current vertex to the source. In our case, the source is the vertex “sn” or the vertex “fa”.

We organize this process into three rows at the priority queue $Q_{sn}$ in Matrix (9) and $Q_{fa}$ in Matrix (10). The two matrices $Q_{sn}$ and $Q_{fa}$ are used in Algorithms 2 and 3, respectively. We store all vertices of the two subtrees $Sn_{sn}$ and $Fa_{fa}$ in the first row of the $Q_{sn}$ in (9) and the $Q_{fa}$ in (10), respectively. We list the sum of the distances from each vertex to source in the second row of both Matrices (9) and (10).

Similarly, the third row of Matrices (9) and (10) contains the color of the current vertex, i.e., a current vertex is the vertex that we want to process; it is the vertex $u$ that we will extract from the first row of the $Q_{sn}$ and $Q_{fa}$ according to the value of the distance $d(u)$.

At the beginning, we set $d(u) = \infty$ and color($u$) = white for all vertices in the subtrees $Sn_{sn}$ and $Fa_{fa}$. These procedures are implemented by Lines 4-8 in both algorithms.
In Line 9 of Algorithm 2, we find the vertex \( u \) in Q represents the source “\( sn \)” and we set \( u = sn \) and \( d(u) = 0 \). Similarly, in Line 9 of Algorithm 3, we find the vertex \( u \) in Q represents the source “\( fa \)” and we set \( u = fa \) and \( d(u) = 0 \).

\[
Q_{sn} = \begin{bmatrix}
\frac{\overrightarrow{u}}{d(\overrightarrow{u})}
\end{bmatrix}_{3 \times |Sn_{sn}|}, \quad (9)
\]

\[
Q_{fa} = \begin{bmatrix}
\frac{\overrightarrow{u}}{d(\overrightarrow{u})}
\end{bmatrix}_{3 \times |Fa_{fa}|}. \quad (10)
\]

**Definition 8.** Let \( msdsn \) be the maximum distance saving of a path in the subtree \( Sn_{sn} \) starting from the vertex “\( sn \)”.

To compute the values of the \( msdsn \), we use two different types of recursive principles in their operation. See [14,21,24,25,27,28].

In the following scenario, we explain how the values of the “\( msdsn \)” are calculated. For each edge \( fs = (fa, sn) \) in \( T_r \), where “\( sn \)” is the root of the subtree \( Sn_{sn} \) as we have illustrated in Section 3 and Figure 2.

Let \( Lsn = \{l_{sn_1}, \ldots, l_{sn_{\varphi}}\} \) be the set of the sons of the vertex \( sn \), where the \( \varphi = |Lsn| \) is the number of the elements in the set \( Lsn \).

We set \( z_i = (sn, l_{sn_i}) \) and \( i = 1, \ldots, \varphi \), where the \( l_{sn_i} \) is the \( i \)th son for the vertex \( sn \). We compute the values of the “\( msdsn \)” using the following recursive formulas.

\[
msdsn(fs, \rho) = \max_{1 \leq i \leq \varphi} \left( |Sn_{l_{sn_i}}| + msdsn(z_i, \rho - 1) \right) = msdsn(P_{sn_{w_1}},), \quad (11)
\]

where the \( P_{sn_{w_1}} \) is the path connecting \( sn \) and \( w_1 \) with length \( \leq \ell - 1 \) and maximum distance saving.

If \( \deg(sn) \geq 2 \), calculated a path with length \( \leq \ell - 1 \) and the second maximum distance saving using as follows.

\[
msdsn_2(fs, \rho) = \max_{1 \leq i \leq \varphi} \left( |Sn_{l_{sn_i}}| + msdsn(z_i, \rho - 1) \right) = msdsn_2(P_{sn_{w_2}}), \quad (12)
\]

and \( P_{sn_{w_1}} \cap P_{sn_{w_2}} = \{sn\} \), where \( \rho = 1, \ldots, \ell - 1 \), \( msdsn(fs, 0) = 0 \) and the vertices \( w_1 \) and \( w_2 \) belong to the subtree \( Sn_{sn} \).

The value of the \( msdsn(fs, \rho) \) represents the largest distance saved by the path \( P_{sn_{w_1}} \), calculated by (11).

Let \( c_i = z_\eta = (sn, l_{sn_\eta}) \) be the first edge in the path \( P_{sn_{w_1}} \), where \( \eta \in \{1, 2, \ldots, \varphi\} \).

The value of the \( msdsn_2(fs, \rho) \) represents the second largest distance saved by the path \( P_{sn_{w_2}} \).

Let \( c_\tau = z_\tau = (sn, l_{sn_\tau}) \) be the first edge in the path \( P_{sn_{w_2}} \), where \( \tau \in \{1, 2, \ldots, \varphi\} \\setminus \{\eta\} \).

Hence, the two paths \( P_{sn_{w_1}} \) and \( P_{sn_{w_2}} \), of the length \( \rho \leq \ell - 1 \), represent the paths with the largest and the second largest distance saving in the subtree \( Sn_{sn} \) rooted at the vertex \( sn \).

**Note 1:** The edge \( c_i \) and the value of the \( msdsn_2 \) will be used in the first branch of Formula (16). The computations in Formula (11) are repeated \( \ell - 1 \) times at \( \rho = 1 \).

**Note 2:** When \( \varphi = 0 \), i.e., \( Lsn = \emptyset \), then \( msdsn(fs, \rho) = -\infty \), i.e., \( sn \in Sd \), also, if \( \varphi = 1 \), then \( msdsn_2(fs, \rho) = -\infty \).

In the following example, we show how the above scenario, Formula (11), and Algorithm 4, are used.
**Algorithm 4** For finding the maximum distance saving “msdsn” in the subtree Sn

**Input:** fath, n and ρ.

**Output:** MsdSn.

1. for each fs ∈ fath, fs = (fa, sn) do
2. set sn = fs(2), Lsn = {lsn1, · · · , lsn_ρ} is the set of sons of sn.
3. calculate \[ ϕ = |Lsn| \]
4. set \[ z_i = (sn, lsn_i), i = 1, \ldots, ϕ \]
5. if \[ ϕ = 0 \] then
6. set msdsn(fs, ρ) = −∞.
7. else
8. for each \[ lsn_i \in Lsn \] and \[ i = 1, \ldots, ϕ \] do
9. go to Formulas (11) and (12) to calculate the msdsn(fs, ρ) and msdsn_2(fs, ρ) respectively.
10. end for
11. if \[ ϕ = 1 \] then
12. set msdsn_2(fs, ρ) = −∞.
13. end if
14. end if
15. end for
16. return \[ MsdSn = [fa, sn, msdsn(fs, ρ), msdsn_2(fs, ρ)]_{(n−1)×4} \].

Given the tree \( T \) in Figure 4, which is rooted at a vertex b, by deleting the edge \( fs = (W, H) \) the tree \( T_b \) is divided into two subtrees, in this case, the first one is the subtree \( Sn_H, sn = H \), and then \( Lsn = \{h_1, h_2, h_3, h_4\} \), i.e, \( lsn_1 = h_1, lsn_2 = h_2, lsn_3 = h_3, lsn_4 = h_4 \), are the sons of the vertex H and \( ϕ = 4 \).

![Figure 4](image-url)

**Figure 4.** Example of showing the computing of the “msdsn” and “msdfa”.

\[ \text{Algorithm 4} \quad \text{For finding the maximum distance saving “msdsn” in the subtree } Sn \]

\[ \text{Input: } \text{fath, } n \text{ and } ρ. \]

\[ \text{Output: } \text{MsdSn.} \]

1. for each \( fs \in \text{fath}, fs = (fa, sn) \) do
2. set \( sn = fs(2), Lsn = \{lsn_1, \ldots, lsn_ρ\} \) is the set of sons of \( sn \).
3. calculate \( ϕ = |Lsn| \) which is the number of elements in the set \( Lsn \).
4. set \( z_i = (sn, lsn_i), i = 1, \ldots, ϕ \)
5. if \( ϕ == 0 \) then
6. set \( msdsn(fs, ρ) = −∞ \).
7. else
8. for each \( lsn_i \in Lsn \) and \( i = 1, \ldots, ϕ \) do
9. go to Formulas (11) and (12) to calculate the \( msdsn(fs, ρ) \) and \( msdsn_2(fs, ρ) \) respectively.
10. end for
11. if \( ϕ == 1 \) then
12. set \( msdsn_2(fs, ρ) = −∞ \).
13. end if
14. end if
15. end for
16. return \( MsdSn = [fa, sn, msdsn(fs, ρ), msdsn_2(fs, ρ)]_{(n−1)×4} \).
3–4. The values of the msdsn\(_{(fs, \rho)}\) and msdsn\(_2\) (fs, \(\rho\)) of the edge fs= (fa, sn) are stored into the matrix MsdSn that is defined by the following.

\[
\text{MsdSn} = \begin{bmatrix}
fa & sn & \text{msdsn}(fs, \rho) & \text{msdsn}_2(fs, \rho)
\end{bmatrix}
\]  \(n-1\times 4\)  \(13\)

**Table 1.** The values of “msdsn” for paths of different lengths to b, (D, W), (W, H) in Figure 4.

<table>
<thead>
<tr>
<th>fs</th>
<th>Sn_{sn}</th>
<th>msdsn(fs,1)</th>
<th>msdsn_2</th>
<th>c,</th>
<th>msdfa(fs,2)</th>
<th>msdsn_2</th>
<th>msdsn(fs,3)</th>
<th>msdsn_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b,A)</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>(A,a)</td>
<td>−∞</td>
<td>−∞</td>
<td>−∞</td>
<td>−∞</td>
</tr>
<tr>
<td>(b,Q)</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>(Q,q)</td>
<td>−∞</td>
<td>−∞</td>
<td>−∞</td>
<td>−∞</td>
</tr>
<tr>
<td>(b,D)</td>
<td>21</td>
<td>18</td>
<td>−∞</td>
<td>(D,W)</td>
<td>35</td>
<td>−∞</td>
<td>40</td>
<td>39</td>
</tr>
<tr>
<td>(D,W)</td>
<td>18</td>
<td>17</td>
<td>−∞</td>
<td>(W,H)</td>
<td>22</td>
<td>21</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>(W,H)</td>
<td>17</td>
<td>5</td>
<td>4</td>
<td>(H,h)</td>
<td>6</td>
<td>5</td>
<td>−∞</td>
<td>−∞</td>
</tr>
</tbody>
</table>

**Table 2.** The values of “msdfa” for paths of different lengths to b, (D, W), (W, H) in Figure 4.

<table>
<thead>
<tr>
<th>fs</th>
<th>msdfa(fs,1)</th>
<th>msdfa(fs,2)</th>
<th>msdfa(fs,3)</th>
<th>msdfa(fs,4)</th>
<th>msdfa(fs,5)</th>
<th>msdfa(fs,6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b,A)</td>
<td>21</td>
<td>39</td>
<td>56</td>
<td>61</td>
<td>62</td>
<td>−∞</td>
</tr>
<tr>
<td>(b,Q)</td>
<td>21</td>
<td>39</td>
<td>56</td>
<td>61</td>
<td>62</td>
<td>−∞</td>
</tr>
<tr>
<td>(b,D)</td>
<td>11</td>
<td>12</td>
<td>−∞</td>
<td>−∞</td>
<td>−∞</td>
<td>−∞</td>
</tr>
<tr>
<td>(D,W)</td>
<td>18</td>
<td>29</td>
<td>30</td>
<td>−∞</td>
<td>−∞</td>
<td>−∞</td>
</tr>
<tr>
<td>(W,H)</td>
<td>21</td>
<td>39</td>
<td>50</td>
<td>51</td>
<td>−∞</td>
<td>−∞</td>
</tr>
</tbody>
</table>

**Table 3.** The distancesum of d(sn) for paths of different lengths to b, (D, W), (W, H) in Figure 4.

<table>
<thead>
<tr>
<th>fs</th>
<th>d(sn)_0</th>
<th>d(sn)_1</th>
<th>d(sn)_2</th>
<th>d(sn)_3</th>
<th>d(sn)_4</th>
<th>d(sn)_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b,A)</td>
<td>5</td>
<td>4</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>(b,Q)</td>
<td>10</td>
<td>9</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>(b,D)</td>
<td>65</td>
<td>47</td>
<td>30</td>
<td>25</td>
<td>24</td>
<td>∞</td>
</tr>
<tr>
<td>(D,W)</td>
<td>45</td>
<td>28</td>
<td>23</td>
<td>22</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>(W,H)</td>
<td>28</td>
<td>23</td>
<td>22</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

**Table 4.** The distancesum of d(fa) for paths of different lengths to b, (D, W), (W, H) in Figure 4.

<table>
<thead>
<tr>
<th>fs</th>
<th>d(fa)_0</th>
<th>d(fa)_1</th>
<th>d(fa)_2</th>
<th>d(fa)_3</th>
<th>d(fa)_4</th>
<th>d(fa)_5</th>
<th>d(fa)_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b,A)</td>
<td>107</td>
<td>86</td>
<td>68</td>
<td>51</td>
<td>46</td>
<td>45</td>
<td>∞</td>
</tr>
<tr>
<td>(b,Q)</td>
<td>97</td>
<td>76</td>
<td>58</td>
<td>41</td>
<td>36</td>
<td>35</td>
<td>∞</td>
</tr>
<tr>
<td>(b,D)</td>
<td>32</td>
<td>21</td>
<td>20</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>(D,W)</td>
<td>52</td>
<td>34</td>
<td>23</td>
<td>22</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>(W,H)</td>
<td>73</td>
<td>52</td>
<td>34</td>
<td>23</td>
<td>22</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
Table 5. Keeping the d(sn) at minimum at locations connected to b and (W;H) in Figure 4.

<table>
<thead>
<tr>
<th>fs</th>
<th>kd(sn)₀</th>
<th>kd(sn)₁</th>
<th>kd(sn)₂</th>
<th>kd(sn)₃</th>
<th>kd(sn)₄</th>
<th>kd(sn)₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b,A)</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>(b,Q)</td>
<td>10</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>(b,D)</td>
<td>65</td>
<td>47</td>
<td>30</td>
<td>25</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>(D,W)</td>
<td>45</td>
<td>28</td>
<td>23</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>(W,H)</td>
<td>28</td>
<td>23</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 6. Keeping the d(fa) at minimum at locations connected to b and (W;H) in Figure 4.

<table>
<thead>
<tr>
<th>fs</th>
<th>kd(fa)₀</th>
<th>kd(fa)₁</th>
<th>kd(fa)₂</th>
<th>kd(fa)₃</th>
<th>kd(fa)₄</th>
<th>kd(fa)₅</th>
<th>kd(fa)₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b,A)</td>
<td>107</td>
<td>86</td>
<td>68</td>
<td>51</td>
<td>46</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>(b,Q)</td>
<td>97</td>
<td>76</td>
<td>58</td>
<td>41</td>
<td>36</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>(b,D)</td>
<td>32</td>
<td>21</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>(D,W)</td>
<td>52</td>
<td>34</td>
<td>23</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>(W,H)</td>
<td>73</td>
<td>52</td>
<td>34</td>
<td>23</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 7. Picking the best edge in the subtree (lower) Sn in Figure 4.

<table>
<thead>
<tr>
<th>ψ(fs)₁</th>
<th>ψ(fs)₂</th>
<th>ψ(fs)₃</th>
<th>ψ(fs)₄</th>
<th>ψ(fs)₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b,A)</td>
<td>112</td>
<td>111</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>(b,Q)</td>
<td>107</td>
<td>106</td>
<td>106</td>
<td>106</td>
</tr>
<tr>
<td>(b,D)</td>
<td>97</td>
<td>79(D,W)</td>
<td>62(W,H)</td>
<td>57</td>
</tr>
<tr>
<td>(D,W)</td>
<td>97</td>
<td>80</td>
<td>75</td>
<td>74</td>
</tr>
<tr>
<td>(W,H)</td>
<td>101</td>
<td>96</td>
<td>95</td>
<td>95</td>
</tr>
</tbody>
</table>

Table 8. Picking the best edge in the subtree (upper) Fa in Figure 4.

<table>
<thead>
<tr>
<th>µ(fs)₁</th>
<th>µ(fs)₂</th>
<th>µ(fs)₃</th>
<th>µ(fs)₄</th>
<th>µ(fs)₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b,A)</td>
<td>112</td>
<td>91</td>
<td>73</td>
<td>56</td>
</tr>
<tr>
<td>(b,Q)</td>
<td>107</td>
<td>86</td>
<td>68</td>
<td>51(b,Q)</td>
</tr>
<tr>
<td>(b,D)</td>
<td>97</td>
<td>86</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>(D,W)</td>
<td>97</td>
<td>79</td>
<td>68</td>
<td>67</td>
</tr>
<tr>
<td>(W,H)</td>
<td>101</td>
<td>80</td>
<td>62</td>
<td>51</td>
</tr>
</tbody>
</table>

Definition 9. Let msdfa be the maximum distance saving of a path of length $\rho \leq \ell - 1$; $\rho = 1, 2, \ldots, \ell - 1$ in the subtree Fa_r.

To compute the maximum distance saving “msdfa” in the subtree Fa_r, we designate two cases as follows.

Case 1: For any edge fs = (fa, sn) of a family matrix “fath”, if fa = r, then the values of the msdfa = (fs, $\rho$) can be calculated by Formula (15) below. Since the root r has no father “fa”, i.e, it has no predecessors, but it has one or more sons. So, we cannot use our approach that we have presented in Algorithm 1 which used to create the subtree Fa_r of $T_r$ for computing the “msdfa”, so, instead of that, we present the following approach to
compute the values of the "msdfa" regorging the root \( r \) with its sons and descendants to the leaves of a tree network. This case represents a special case for calculating the values of the \( \text{msdfa}(fs, \rho) \).

Let \( L_r = \{ \text{son}_r, \ldots, \text{son}_N \} \) be the set of the sons of the root \( r \), \( N = |L_r| \) is the number of the elements in the set \( L_r \), we set \( E_k = (r, \text{son}_k) \) and \( k = 1, \ldots, N \) is the set of edges linked directly to the root.

\[
M_r(r, E_k) = \begin{cases} 
-\infty & \text{if } N = 1, \\
|Sn_{son_k}| + \text{msdsn}(E_k, \rho - 1) & \text{otherwise},
\end{cases}
\]  
(14)

where \( \rho = 1, 2, \ldots, \ell - 1 \), the values of the \( \text{msdsn}(E_k, \rho) \) are calculated by Formula (11) and \( \text{msdsn}(E_k, 0) = 0 \).

If \( \text{deg}(r) = 1 \) and fs = \( (r, \text{son}_r) \), then set \( M_r(r, E_k) = -\infty \). Otherwise, \( \text{deg}(r) \geq 2 \), then find \( M_r(r, E_k) \) for each edge with the root \( r \).

Let \( \gamma_{\rho} = \max_{1 \leq k \leq N} \{ M_r(r, E_k) \} \) be the maximum distance saving obtained by attaching a path of the length \( \rho \) through the edge \( E_k \) in the tree \( Fa_r = T_r \) rooted at \( r \). Also, let \( \xi_{\rho} = \max_{1 \leq k \leq N, k \neq k^*} \{ M_r(r, E_k) \} \) be the second maximum distance saving obtained by attaching a path of the length \( \rho \) through one of the edges of \( E_k \setminus E_{k^*} \) in the tree \( Fa_r = T_r \) rooted at \( r \).

Therefore, if \( fs = (r, sn) \), the values of the \( \text{msdfa}(fs, \rho) \) can be computed as follows.

\[
\text{msdfa}(fs, \rho) = \begin{cases} 
\xi_{\rho} & \text{if } fs = E_{\rho}^*, \\
\gamma_{\rho} & \text{if } fs = E_{\rho}^*, \\
-\infty & \text{if } N = 1,
\end{cases}
\]  
(15)

For more clarification on how Formulas (14) and (15) can be applied, we use the tree \( T_3 \) in Figure 4 as follows.

Suppose we are at the vertex \( r = b \) on the tree network of Figure 4, and we want to compute the value of the \( \text{msdfa}(fs, \rho) \) as we have shown above.

Since \( L_r = \{ \text{son}_r, \ldots, \text{son}_N \} = \{ Q, D, A \} \), \( N = 3 \), this means the \( \text{son}_1 = Q, \text{son}_2 = D, \text{son}_3 = A \). Accordingly, we have

\[
M_r(r, E_k) = \{ |Sn_Q| + \text{msdsn}(E_k, \rho), |Sn_D| + \text{msdsn}(E_k, \rho), |Sn_A| + \text{msdsn}(E_k, \rho) \}.
\]

Because \( \text{msdsn}(E_k, 0) = 0 \) at \( \rho = 1, k = 1, 2, 3, |Sn_Q| = 11, |Sn_D| = 21 \) and \( |Sn_A| = 6 \), therefore, \( M_r = \{ 11, 21, 6 \} \).

Hence, \( \gamma_1 = 21, \xi_1 = 11, E_1 = E_2 = (b, D), E_3 = (b, \text{son}_3) \).

Therefore, \( \text{msdfa}(fs, 1) = 21 \) for \( fs = (b, Q) \) and \( fs = (b, A) \) and \( \text{msdfa}(fs, 1) = 11 \) for \( fs = (b, D) \) at \( \rho = 1 \).

Similarly, the rest of the results are obtained from the root “b” to the end of the vertices in the tree \( T_3 \) listed in Table 2. The above approach finds two paths of the length \( \rho \) of maximum and second maximum distance saving, which are attached at the root node \( r \).

Case 2: For all \( fs \in \text{fath}, fa = fs(1) \) and if \( fs = r \), the “msdfa” is computed by the following.

First of all, we go to Algorithm 1 for creating the subtree \( Fa_{fa} \). Let \( gf \) be a father of the vertex \( fa \) we set \( g = (gf, fa) \) and \( fs = (fa, sn) \), and we compute the values of the “msdfa” by Formula (16).

\[
\text{msdfa}(fs, \rho) = \begin{cases} 
\max \{ \pi(fa) + \text{msdfa}(g, \rho - 1), \text{msdsn}_1(g, \rho) \} & \text{if } fs = c_i, \\
\max \{ \pi(fa) + \text{msdfa}(g, \rho - 1), \text{msdsn}(g, \rho) \} & \text{otherwise},
\end{cases}
\]  
(16)

where the value of the \( \pi(fa) \) is calculated by the following equation:

\[
\pi(fa) = n - |V(Sn_{fa})|,
\]  
(17)
where \( n = |V(T_r)| \) is the number of vertices of the tree \( T_r \), \( |V(S_{n_{fa}})| \) is the number of vertices in the subtree \( S_{n_{fa}} = T_{fa} \), it is calculated by Formula (6) and the value of the msdsn \( (g, \rho) \) is computed by Formula (12).

For example, assume we choose the edge \( fs = (W, H) \) of the tree network in Figure 4, as we have done above where \( fa = W \) and \( sn = H \). So, the subtree \( Fa_w \) is created by Algorithm 1.

We set \( fg = D \) as the father of the vertex \( W \), therefore, \( g = (D, W) \).

Now, we compare \( fs = (W, H) \) with the edges \( c_1 \) that are listed in Column 5 of Table 1, i.e., the edges \( c_i \) that we have obtained from Formula (11), when we computed the value of the msdsn \((D, W), 1\) = 17, it is obtained by the edge \( c_i = (W, H) \).

Since, \( fs = (W, H) = c_1 \) is listed in Line 6 of Column 5 in Table 1 at \( \rho = 1 \), where \( c_1 \) is the edge that corresponds the value of the msdsn \((g, 1)\) = 17 and msdsn \((g, 1)\) = \(-\infty\).

Therefore, we apply the first branch of Formula (16) as follows: msdfa \((fs, 1)\) = \( \max \{\pi(W) + \text{msdfa}(g, 0), \text{msdsn}_1(g, 1)\} \), \( |V(S_{n_{fa}})| \) = 18, \( \pi(W) = 39 - 18 = 21 \), msdfa \((g, 0)\) = 0, msdsn \((g, 1)\) = \(-\infty\).

Therefore, msdfa \((fs, 1)\) = \( \max \{21, -\infty\} = 21 \) at \( \rho = 1 \).

The rest of the results are in Table 2 below. Columns 2 – 7 contain the values of the msdfa \((fs, \rho)\), with \( \rho = 1, \cdots, \ell - 1 \), that we have obtained from Formulas (15) and (16) and Algorithms 5 and 6.

The results of the “msdfa” that we have obtained from Cases 1 and 2 of Definition 9 are stored in the matrix MsdFa in the following.

\[
\text{MsdFa} = \begin{bmatrix} \text{fa} & \text{sn} & \text{msdfa}(fs, \rho) \end{bmatrix}_{(n-1) \times 3} \tag{18}
\]

**Algorithm 5** For finding the maximum distance saving (msdfa) in the subtree \( fa \),

**Input:** “fath” and \( n \).

**Output:** msdfa \((fs, \rho)\) = \( \bar{c}_{\rho} \), if \( fs = E_1 \) and msdfa \((fs, \rho)\) = \( \gamma_{\rho} \) if \( fs \neq E_1 \).

1. Let \( L_r = \{sonr_1, \cdots, sonr_N\} \) be the set of sons of the root \( r \).
2. compute \( N = |L_r| \) is the number of elements in the set \( L_r \).
3. set \( E_k = (r, sonr_k) \) where \( k = 1, \cdots, N \).
4. if \( N = 1 \) then
   5. set msdfa \((fs, \rho)\) = \(-\infty\).
   6. else
   7. for each \( sonr_k \in L_r \) and \( k = 1, \cdots, N \) do
   8. go to Formula (15) to compute the values of the \( \gamma_{\rho} \) and \( \bar{c}_{\rho} \).
   9. end for
10. end if
11. return msdfa \((fs, \rho)\) = \( \bar{c}_{\rho} \), if \( fs = E_k \) and msdfa \((fs, \rho)\) = \( \gamma_{\rho} \) if \( fs \neq E_k \).

**Algorithm 6** For finding the maximum distance saving “msdfa” in subtree \( Fa_{fa} \)

**Input:** fath, \( n, T_r \), and \( \rho = 1, \cdots, \ell - 1 \).

**Output:** MsdFa.

1. for each \( fs \in \text{fath} \), \( fs = (fa, sn) \). do
2. set \( fa = fs(1) \) and go to Algorithm 1 to create the subtree \( Fa_{fa} \).
3. if \( fa \neq r \) then
4. return to Formula (16) to compute the value of the msdfa \((fs, \rho)\).
5. else
6. go to Algorithm 5 to compute the value of the msdfa \((fs, \rho)\), where \( fa = r \).
7. end if
8. end for
9. return MsdFa = \( \begin{bmatrix} \text{fa} & \text{sn} & \text{msdfa}(fs, \rho) \end{bmatrix}_{(n-1) \times 3} \)

**Definition 10.** Suppose, the best path in the subtree \( S_{n_{fa}} \) is defined by the following.
where \( \sigma = 0, 1, \ldots, \ell - 1 \), for example, when \( \sigma = 0 \), \( \text{msdsn}(fs, 0) = d(sn) \), hence the sum of the distances from the root "sn" of the subtree \( Sn \) to all vertices in the subtree \( Sn \), it is defined by (7).

This means the beginning of the targeted path is attached to the head of the edge \( fs = (fa, sn) \).

Hence, Formula (11) is used to find the maximum distance saving obtained by attaching a path of length \( \rho \) to the vertex \( sn \). For example, when the edge \( fs = (d, v) \) is deleted from the tree network \( T_v \) of Figure 1, we obtain two subtrees; the first one is the subtree \( Sn_v = T_v \). Hence, the number of adjacent edges of the vertex \( sn = v \) are three edges, the optimal edge of them is the edge \( c_1 = (v, p) \). See Figure 2. The edge \( c_1 = (v, p) \) is obtained by Formula (11), in this case, when \( \rho = 1 \) and \( \sigma = 1 \), so the targeted path is \( p_1 = p_0 \cup c_1 \), hence \( p_1 = c_1 \). Therefore, the sum of the distance from the path \( p_1 \) to all vertices in the subtree \( Sn_v \) equals to \( d(sn)(fs, 1) = d(sn)(fs, 0) - \text{msdsn}(fs, 1) \).

Similarly, at \( \rho = 2 \) and \( \sigma = 2 \), \( p_2 = p_1 \cup c_1 \), then the "distancesum" of the path \( p_2 \) equals to the \( d(sn)(fs, 2) = d(sn)(fs, 2) \).

For further illustration, we use the tree network depicted in Figure 4 as follows. Assume we pick the edge \( fs = (W, H) \) and \( \sigma = 0 \), then \( fa = W \) and \( sn = H \), hence \( d(H)(fs, 0) = 28 \). Now, when \( \sigma = 1 \) and \( \rho = 1 \), we have to return to Formula (11) for computing the value of the \( \text{msdsn}(fs, 1) \) = 5, and this result is obtained by the edge \( c_1 = (H, h_1) \), and then \( p_1 = (H, h_1) \). Hence, \( d(sn)(fs, 1) = 28 - 5 = 23 \), similarly, when \( \sigma = 2 \) and \( \rho = 2 \), therefore, \( d(sn)(fs, 2) = 28 - 6 = 22 \), this result has been obtained through the path \( p_2 = p_1 \cup (h_2, -) \), finally when \( \sigma = 3 \) and \( \rho = 3 \), therefore, \( d(sn)(fs, 3) = 28 - (-\infty) = \infty \), there is no edge that gives this result. It means the calculation on the subtree \( Sn_v \) has finished.

The rest of the results are presented in Table 3. Column 2 of Table 3 denoted by \( d(sn) \) contains a sequence of the sum of the distances from the vertex "sn" to all vertices in the subtree \( Sn \), which are computed by Formula (7) using Algorithm 2. Columns 4 – 7 of Table 3 show the values of the \( d(sn) \) that we have obtained from Formula (20).

**Definition 11.** The best path in the subtree \( Fa_{fa} \) can be defined as follows.

In Definition 9, we have used two cases as follows.

According to Formula (14) in Case 1, when \( fa = r \), the best edge of the adjacent edges, which are connected to the root \( r \), is the edge \( E_k \). In this case, the best path in the subtree.

\[
p_{\rho} = p_{\rho - 1} \cup E_{k^*}, \rho = 1, 2, \ldots, \ell - 1, \quad fs \neq E_{k^*}.
\]  

(21)

Now, when \( \rho = 1 \) and \( \sigma = 1 \), the targeted path is \( p_1 = p_0 \cup E_{k^*} \), hence \( p_1 = E_{k^*} \).

For example, in Figure 1, when we want to compute the value of the \( \text{msdfa}(fs, \rho) \) for the edge \( fs = (s, y) \), we delete the edge \( fs = (s, y) \) of the tree \( T_v \), and then we obtain the subtree \( Fa_{fa} = T_v \setminus Sn_v \). At \( \sigma = 1 \) and \( \rho = 1 \), the value of the \( \text{msdfa}(s, y, 1) \) is calculated by Formula (14), then \( \text{msdfa}(s, y, 1) = 24 \), is obtained by the edge \( E_1 = (s, a) \), hence \( p_1 = (s, a) \).

In Case 2 of Definition 9, when \( fa \neq r \), the values of the \( \text{msdfa}(fs, \rho) \) are computed by Formula (16), therefore, the best edge is an edge of the predecessor or descendant that is
associated with the vertex “fa” without the edge fs = (fa, sn) that we have deleted from
the tree T_a.

Note 1: If the condition in the first branch of Formula (16) satisfies and \( \pi(fa) + \text{msdfa}(g, \rho - 1) < \text{msdsn}_r \), the value of the \( \text{msdfa}(fs, \rho) \) equals to \( \text{msdsn}_r \).

Note 2: If the condition in the first branch of Formula (16) satisfies and \( \pi(fa) + \text{msdfa}(g, \rho - 1) > \text{msdsn}_r \), this means that the value of the \( \text{msdfa}(fs, \rho - 1) \) equals to \( \pi(fa) + \text{msdfa}(g, \rho) \). Accordingly, in Case 2, the best path in the subtree \( \text{Fa}_{fa} \) is defined by the following.

\[
p_{\rho} = \begin{cases} 
  p_{\rho - 1} \cup c_j & \text{if } fs = c_i \text{ and } \pi(fa) + \text{msdfa}(g, \rho - 1) < \text{msdsn}_r, \\
  p_{\rho - 1} \cup g & \text{otherwise, for } \rho = 1, \cdots, \ell - 1,
\end{cases}
\]

where the edge \( g = (gf, fa) \) is connected directly to the deleted edge \( fs = (fa, sn) \) and the \( c_j \) and \( c_i \) are the first and second best edges connected to the deleted edge \( fs = (fa, sn) \) according to (11) and (12), respectively.

For example, in Figure 1, when we compute the value of the \( \text{msdsn}(fs, \rho) \) for the edge \( fs = (h, d) \), we delete the edge \( fs = (h, d) \) of the tree \( T_a \), and then we obtain two subtrees \( S_{fa} \) and \( F_{fa} \).

We return to Formula (11) to compute \( \text{msdsn}(fs, \rho) \), at \( \rho = 1 \), the \( \text{msdsn}(D, h, d) = 11 \) is obtained by the edge \( c_1 = (d, v) \) and \( \text{msdsn}_1 = 1 \) is obtained by the edge \( c_2 = (d, \tau) \).

Now, when we want to calculate the value of the \( \text{msdfa}(fs, \rho) \) for the edge \( fs = (d, v) \). Firstly, we delete edge \( fs = (d, v) \) of the tree \( T_a \), and then we obtain two subtrees, which are \( S_{fa} = T_a \) and \( F_{fa} = T_a \setminus T_a \). According to Formula (16), \( g = (fg, fa) = (h, d) \), we note that the \( fs = c_1 = (d, v) \), then we use the first branch of Formula (16). At \( \rho = 1 \), \( \text{msdfa}(d, v, 1) = \max \{ \pi(fa) + \text{msdfa}(g, 0), \text{msdsn}_1 \} \), we compute the value of the \( \pi(fa) = 34 - 13 = 21 \), by Formula (17), \( \text{msdfa}(d, v, 1) = \max \{ 21 + 0, 1 \} \), i.e., \( \pi(fa) + \text{msdfa}(d, v, 1) = 21 > \text{msdsn}_1 = 1 \), therefore, we apply the second branch of Formula (22) to specify the best path in the subtree \( F_{fa}' \), and the best path is \( p_1 = (h, d) \).

Therefore, the sum of the distance form the path \( p_{\rho} \) in all vertices in the subtree \( F_{fa} \) is defined in the following equation.

\[
d(fa)(fs, \sigma) = d(fa)(fs, 0) - \text{msdfa}(fs, \sigma),
\]

where \( \sigma = 0, 1, \cdots, \ell - 1 \), for example, when \( \sigma = 0 \), \( \text{msdfa}(fs, 0) = 0 \), then \( d(fa)(fs, 0) = d(fa)(fs, 0) - \text{msdfa}(fs, 0) \). The term \( d(fa)(fs, \sigma) \) is the sum of the distances from the root “fa” of the subtree \( F_{fa} \) to all vertices in the subtree \( F_{fa} \) according to (8).

This means that the beginning of the targeted path is sitting at the vertex “fa” of the edge \( fs = (fa, sn) \), i.e., the best edge is either the predecessor of the vertex “fa” or the descendant of the vertex “fa”.

Hence, Formula (16) will drive the vertex “fa” to the best edge of the subtree \( F_{fa} \).

Therefore, the sum of the distance form the path \( p_1 \) in all vertices in the subtree \( F_{fa} \) equals to \( d(fa)(fs, 1) = d(fa)(fs, 0) - \text{msdfa}(fs, 1) \), where the value of the \( d(fa)(fs, 0) \) is constant for the edge “fa” at \( \sigma = 0, 1, \cdots, \ell - 1 \), i.e., it represents the initial value of Formula (23) and the value of the \( \text{msdfa}(fs, 1) \) is calculated by Formula (16). For more illustration, we use the example that we present in Figure 4 by the following.

Assume we pick the edge \( fs = (W, H) \) and \( \sigma = 0 \), then \( fa = W \), hence \( d = 73 \), and this value has been computed by Algorithm 3. Now, when \( \sigma = 1 \) and \( \rho = 1 \), \( d(W)(fs, 1) = (W)(fs, 0) - \text{msdfa}(fs, 1) = 73 - 21 = 52 \), where the value of the \( \text{msdfa}(fs, 1) \) is calculated by Formula (16), hence this result has been obtained through the edge \( g = (D, W) \), and \( p_1 = (D, W) \). Similarly, at \( \sigma = 2 \) and \( \rho = 2 \), \( d(W)(fs, 2) = 73 - 39 = 34 \), this result has been obtained through the edge \( g = (b, D) \), and \( p_2 = p_1 \cup (b, D) \), at \( \sigma = 3 \) and \( \rho = 3 \), \( d(W)(fs, 3) = 73 - 50 = 23 \), this result has been obtained through the edge \( g = (Q, Q) \), and \( p_3 = p_2 \cup (Q, Q) \), at \( \sigma = 4 \) and \( \rho = 4 \), \( d(W)(fs, 2) = 73 - 51 = 22 \), and this result has been obtained through the edge \( g = (Q, Q) \), and \( p_4 = p_3 \cup (Q, Q) \), where \( 1 \leq i \leq 10 \).
Finally, when 𝜎 = 5 and 𝜌 = 5, therefore, \(d(fa)(fs, 3) = 73 - (-\infty) = \infty\), there is no edge that gives this result. It means the calculation on the subtree \(Fa_n\) has finished.

The rest of the results are presented in Table 4. Column 1 of Table 4 is labeled by \(d(fa)\), which contains a sequence of the sum of the distances from the vertex \(fa\) to all vertices in the subtree \(Fa_n\), which are computed by Formula (8) using Algorithm 3, Columns 4 – 8 in Table 4 show the values of the \(d(fa)\) that we have obtained from Formula (23).

**Definition 12.** Let \(k\left(d(sn)(fs, \rho)\right)\) be used to compare the “distancesum” of all paths with \(d(sn)(fs, \rho)\) in the subtrees \(Sn_m\) for the the edge “fs” of length \(\rho = 1, 2, \cdots, \ell - 1\), i.e., the value of the \(k\left(d(sn)(fs, \rho)\right)\) represents the “distancesum” of a best path of length \(\rho = 1, 2, \cdots, \ell - 1\) in the subtrees \(Sn_m\), that starts from the vertex “sn” to the farthest leaf on the subtree \(Sn_m\). See [21,25].

The \(k\left(d(sn)(fs, \rho)\right)\) is defined by the following.

\[
k(d(sn)(fs, \rho)) = \begin{cases} 
  d(sn)(fs, \rho) & \text{if } \rho = 0, \\
  \min\{d(sn)(fs, \rho), k(d(sn)(fs, \rho - 1))\} & \text{otherwise.}
\end{cases}
\]  

(24)

**Definition 13.** Let \(k\left(d(fa)(fs, \rho)\right)\) be used to keep the “distancesum” at a minimum value of the \(d(fa)(fs, \rho)\) in the subtree \(Fa_n\) for the the edge “fs” at length \(\rho = 1, \cdots, \ell - 1\).

In other words, the value of the \(k\left(d(fa)(fs, \rho)\right)\) is the “distancesum” of a best path of length \(\rho = 1, \cdots, \ell - 1\) that begins from the vertex “fa” to the farthest leaf on the subtree \(Fa_n\).

The \(k\left(d(fa)(fs, \rho)\right)\) is defined by

\[
k(d(fa)(fs, \rho)) = \begin{cases} 
  d(fa)(fs, \rho) & \text{if } \rho = 0, \\
  \min\{d(fa)(fs, \rho), k(d(fa)(fs, \rho - 1))\} & \text{otherwise.}
\end{cases}
\]  

(25)

**Note 1:** In the following, we show how Formulas (24) and (25) are used. Given, \(fs = (W, H)\) on the tree network of Figure 4, where \(sn = H, fa = W\), because the value of the \(d(H)(fs, \sigma)\) equals to 28 at \(\rho = 0\), therefore, \(k(d(H)(fs, 0)) = 28\), by using the first branch of Formula (24).

Now, when \(\rho = 1\), at the same edge \(fs = (W, H)\), \(d(H)(fs, 1) = 23\), hence \(k(d(H)(fs, 1)) = \min\{28, 23\} = 23\), similarly, \(d(H)(fs, 2) = 22\), then \(k(d(H)(fs, 1)) = \min\{23, 22\} = 22\).

Finally, at \(\rho = 3, d(H)(fs, 3) = \infty\), therefore, \(k(d(H)(fs, 1)) = \min\{22, \infty\} = 22\).

Similarly, for the same edge \(fs = (W, H)\) regorging to the subtree \(Fa_n\), we use Formula (25), then \(k(d(W)(fs, 0)) = d(W)(fs, 0) = 73, k((W)(fs, 1)) = \min\{d(W)(fs, 1); k(d(W)(fs, 0))\} = \min\{52, 73\} = 52, k((W)(fs, 2)) = \min\{34, 52\} = 34, k((W)(fs, 3)) = \min\{23, 34\} = 23\), finally, at \(\rho = 4, k((W)(fs, 4)) = \min\{22, 23\} = 22\).

The rest of the results of the \(k(d(sn)(fs, \rho))\) and \(k(d(fa)(fs, \rho))\) are presented in Tables 5 and 6, respectively. Column 1 in both Tables 5 and 6 is labeled by \(k(d(sn)_{\rho})\) and \(k(d(fa)_{\rho})\) contains the sum of the distance form both vertices “sn” and “fa” to all vertices in the subtrees \(Sn_m\) and \(Fa_n\) that have been obtained by Formulas (24) and (25). The other columns in both tables give the values of the \(k(d(sn)_{\rho})\) and \(k(d(fa)_{\rho})\), which represent the “distancesum” from a path \(p_{\rho} = p_{\rho-1} \cup fs\) to all vertices in the two subtrees \(Sn_m\) and \(Fa_n\), with the length \(\rho \geq 1\), respectively.

In the following we present a mechanism to specify the optimal path on the whole tree.

**Definition 14.** Let \(\psi \) and \(\mu\) be two functions by which the best path in the two subtrees \(Sn_m\) and \(Fa_n\) is combined by picking the best edge of both (\(\psi, \mu\)) to obtain a composite path that represents the median path (\(\ell\)-core) for a tree network with the length \(\ell\).
These two functions are defined by

\[ \psi(fs, \rho) = k(d(sn)(fs, \rho - 1)) + d(fa)(fs, 0), \quad (26) \]

and

\[ \mu(fs, \rho) = k(d(\rho)(fs, \rho - 1)) + d(sn)(fs, 0), \quad (27) \]

where \( \rho = 1, \ldots, \ell - 1 \), the \( \psi(fs, \rho) \) denotes the sum of the distance from a candidate path \( p_{\rho+1} = p_\rho \cup fs \) to all vertices in the subtree \( Sn_\rho \) and \( \mu(fs, \rho + 1) \) denotes the sum of the distance from a candidate path \( p_{\rho+1} = p_\rho \cup fs \) to all vertices in the subtree \( Fa_\rho \).

For example, at the tree network depicted in Figure 4, if \( \rho = 1 \) and the edged is \( fs = (b, A) \), then \( \psi(fs, 1) = k(d[A](fs, \rho) + d(b)(fs, 0)) = 107 + 5 = 112 \), for \( fs = (b, Q) \), \( \psi(fs, 1) = 97 + 10 = 109 \), for \( fs = (b, D) \), \( \psi(fs, 1) = 97 \), for \( fs = (D, W) \), \( \psi(fs, 1) = 97 \) and for \( fs = (W, h) \), \( \psi(fs, 1) = 101 \).

The five values of the \( \psi(fs, 1) \) = \{112, 107, 97, 97, 101\}, denote the sum of the distances from each edge of \( \{b, A\}, \{b, Q\}, \{b, D\}, \{(D, W)\}, \{(W, h)\} \) to all vertices in the tree \( T_s \), e.g., the \( \psi(fs, 1) \) = 112, denotes the sum of the distances from the edge \( fs = (b, A) \) to all vertices in the tree \( T_s \).

Similarly, there are the five values of the \( \mu(fs, 1) \).

According to Problem (1) that describes the optimal path, we find the minimum distance among the set values of the \( \psi \) and \( \mu \).

We note that the minimum value is 97, that corresponds to either the edge \( (b, D) \) or \( (D, W) \), so the best path with length \( \ell = 1 \) is either \( p_1 = (D, W) \) or \( p_1 = (b, D) \). Now, we need to create a path from the both functions \( \psi \) and \( \mu \) that will represent the optimal path in the tree \( T_s \).

In general, the optimal path \( p_\rho \) is selected according to the following formulas:

\[ \alpha(fs, \rho) = \min\{\psi(fs, \rho), \mu(fs, \rho)\}, \quad (28) \]

\[ p_\rho = p_{\rho-1} \cup fs, \quad (29) \]

where \( \rho = 1, \ldots, \ell - 1 \), \( \forall \ fs \in \text{fath}, \ i.e., \ \forall \ fs \in E \).

Formula (28) means that the fs edge, which gives the minimum value of the function \( \alpha(fs, \rho) \), will be added to the the optimal path defined by Formula (29).

For example, at \( \rho = 1 \), for each fs on the tree \( T_s \), we obtain \( \alpha(fs, 1) = \min\{\psi(fs, 1), \mu(fs, 1)\} \).

Hence, the value of the \( \alpha(fs, 1) \), is the minimum sum of the distance from the edge “fs” to all vertices on the tree \( T_s \).

Therefore, the optimal path is \( p_1 = p_0 \cup fs \), where the \( p_0 = \emptyset \) is the initial (empty) path.

Similarly, when \( \rho = 1 \), for each fs on the tree \( T_s \), we obtain \( \alpha(fs, 1) = \min\{\psi(fs, 2), \mu(fs, 2)\} \).

Hence, the value of the \( \alpha(fs, 1) \) represents the minimum sum of the distance form the path \( p_1 \) to all vertices on the tree \( T_s \). Therefore, the optimal path is \( p_2 = p_1 \cup fs \), and so on up to the required length, i.e., \( |P| = |p_1| + |p_2| + \ldots = \ell \).

In general, \( \forall \ \rho \geq 0 \), the value of the \( \alpha(fs, \rho) \) represents the minimum sum of the distances from the path \( p_{\rho+1} \) to all vertices on a tree \( T_s \) network. For further illustration, see Tables 7–9, regorging the tree \( T_s \) in Figure 4.
### Table 9. The one-core path with length $\ell = 5$ for the tree $T_r$ in Figure 4.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\delta$</th>
<th>$\ell$</th>
<th>$P^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>21</td>
<td>1</td>
<td>$p_1 = (b, D)$</td>
</tr>
<tr>
<td>79</td>
<td>39</td>
<td>2</td>
<td>$p_1 = p_1 \cup (D, W)$</td>
</tr>
<tr>
<td>62</td>
<td>56</td>
<td>3</td>
<td>$p_1 = p_2 \cup (W, H)$</td>
</tr>
<tr>
<td>51</td>
<td>67</td>
<td>4</td>
<td>$p_1 = p_1 \cup (b, Q)$</td>
</tr>
<tr>
<td>46</td>
<td>72</td>
<td>5</td>
<td>$p_5 = p_4 \cup (H, h_3)$</td>
</tr>
</tbody>
</table>

### 4.2. Recursive Modified and Improved Algorithm

The modifications and suggestions that are presented in the above sections lead us to design the following recursive modified and improved algorithm as a developed algorithm finds a core for a tree network rooted at a vertex $r$, with a length equals to $\ell$.

In the following, we present the main algorithm that achieves this purpose. Therefore, we have depicted Algorithm 7, in Figure 5.

**Algorithm 7** For finding $\ell$-core of a tree

**Output:** $P$ is the optimal path with length $\ell$.

**Input:** fath and dim $\leq \ell < 2 \text{dim}$ given by user.

1. for $\rho \leftarrow 1$ to $\ell$ do
2. for each $fs \in \text{fath}$ do
3. compute the $\psi$ and the $\mu$ by using Formulas (26) and (27).
4. compute $\alpha = \min\{\psi, \mu\}$.
5. set $p_{\rho+1} = p_{\rho} \cup fs$, where “fs” corresponding the value of $\alpha$ at $\rho$.
6. end for
7. end for
8. return $P = p_\ell$ is the optimal path with length $\ell$.

---

**Figure 5.** Flowchart of the developed and efficient median path algorithm.

### 5. Numerical Results

Some test examples of tree networks are run on a PC with Intel(R) Core(TM) i5-3230M CPU@2.60GHz 2.60 GHz with RAM 4.00GB of memory on Windows 10 operating system, by utilizing MATLAB version 8.5.0.197613 (R2015a; MathWorks, Inc., Natick, MA, USA).

As we have mentioned, our aim is to find the optimal path on a tree network.
5.1. Time Complexity Analysis

The time complexity of the main algorithm (Algorithm 7) can be computed by the following.

We compute the time complexity of all six algorithms, and then the worst time complexity of these six algorithms represents the time complexity of Algorithm 7.

Since Algorithm 1 visits each edge of the matrix fath only once, i.e., for each iteration $i$ of $N$, Algorithm 1 visits an edge of the matrix fath $1 \leq k \leq |L_{j-1}|$ time, where $j = 1, 2, \ldots, d$ denotes the index of the level, which belongs to the matrix fath and $\eta = 1, 2, \ldots, j$ denotes the index of the level, which belongs to the matrix fath.

Accordingly, Algorithm 1, has two loops are the outer loop that denoted by $i = 1 : N$ and the inner loop that denoted by $k = 1 : N_t$.

Before we compute the time complexity of Algorithm 1, which has two loops (outer and inner), we present a general example for an algorithm that has two loops (outer and inner loop) (Algorithm 8).

\begin{algorithm}[H]
\caption{General algorithm contains two loops (outer and inner)}
\begin{algorithmic}[1]
\State for $i = 1 : n$
\For{$k = 1 : m$}
\State Statement 1
\State Statement 2
\EndFor
\State Statement 3
\EndFor
\end{algorithmic}
\end{algorithm}

We analyze the time complexity of Algorithm 8, as follows: for each $i$ ($n$ times), we iterate over all $k$ ($m$ times), which leads $n \times m = nm$ iterations, hence the time complexity of this example is $O(nm)$ if $m = n$, then the time complexity of Algorithm 8, is $O(n^2)$.

Since Algorithm 1 is similar to Algorithm 8, both algorithms have nested loops (outer and inner loop). But, Algorithm 1 runs in a different way.

We analyze the time complexity of Algorithm 1 as follows: the outer loop of Algorithm 1 is $i = 1 : N$, and the inner loop is $k = 1 : N_t$, and if Algorithm 1 runs like Algorithm 8, then the time complexity of Algorithm 1 is $O(N_t \times N) = O(n^2)$, but Algorithm 1 does not work like Algorithm 8.

Assume Algorithm 1 at the first level $L_1 \subset \text{fath}$, then Algorithm 1 finds for the predecessor of the vertex $f_{a_1}$ in the level $L_{1-1} = L_0$, of the matrix fath. Hence, the worst case of $i = 1 : |L_1|$ is rebated $|L_0|$ iterations. Also, the worst case of $i = 1 : |L_2|$ is rebated $|L_0| + |L_1|$ iterations in the worst case of $i = 1 : |L_3|$ is rebated $|L_0| + |L_1| + |L_2|$ iterations in the worst case of $i = 1 : |L_4|$ is rebated $|L_0| + |L_1| + |L_2| + |L_3|$ iterations, and so on in the worst case of $i = 1 : |L_d|$ is rebated $|L_0| + |L_1| + |L_2| + |L_3| + \ldots + |L_{d-1}|$ iterations.

In general, the worst case of $i = 1 : N$ is rebated $\sum_{j=0}^{d-1} |L_j| = N - (1 + |L_d|)$ iterations.

Therefore, for each $i = 1 : N$ cannot be iterated over all $k$ ($N_t$ times).

Hence, the running mechanism of Algorithm 1 is less costly compared to the running mechanism of Algorithm 8. Therefore, the time complexity of Algorithm 8 is greater than the time complexity of Algorithm 1.

Therefore, it can be deduced that the time complexity of Algorithm 1 is less than $O(n^2)$, and greater than $O(n \log(n))$, it can be formed as $O(n \log(n)) < O(nx) < O(n^2)$.

Now, we analyze the time complexity of Algorithms 2 and 3 as follows:

The time complexity of Lines 1:3 of Algorithm 2, is $O(n)$ and the time complexity of Lines 4:8 of Algorithm 2, is $O(n)$. The time complexity of Lines 16:22 of Algorithm 2 is $O(1)O(n) = O(n)$, hence the time complexity of Lines 12:26 of Algorithm 2 is $O(n)$. Therefore, the time complexity of Algorithm 2 is $O(n) + O(n) + O(n) = 3O(n) = O(n)$.

Similarly, the time complexity of Algorithm 3 is $O(n)$. 

Now, we analyze the time complexity of Algorithms 4 and 5 as follows:

The time complexity of the outer loop, which begins at Line 1 of Algorithm 4, is \( n-1 \) times and the time complexity of the inner loop, which is listed in Lines 8:10 of Algorithm 4, is \( \varphi, (n-1) \) time.

Therefore, the time complexity of Algorithm 4 is \( O(n) \) time.

While the time complexity of Algorithm 5 is \( O(N) \) where \( N << n \).

The time complexity of Algorithm 6 is \( O(n-1) + O(N) = O(n) \).

Finally, the time complexity of Algorithm 7 is computed as follows:

Algorithm 7 has two loops (outer and inner loop), where in the outer loop the length \( \ell \) (constant) is given an advance that represents the length of the median path that we intend to find, hence the cost of the outer loop is \( \ell \) time. The cost of the inner loop in Algorithm 7 is computed as follows:

The worst of the time complexity of Algorithms 1–6 is \( O(xx) \) time, where \( O(n\log(n) < O(xx) < O(n^2) \).

Therefore, the time complexity of Algorithm 7 is \( O(xx) \).

According to the above discussion, we present the following.

**Theorem 1.** Algorithm 7 finds the median path of a tree network with length \( \ell \) in \( O(xx) \) time, where \( O(n\log(n) < O(xx) < O(n^2) \).

**Proof.** Since the distancesum \( D(v) \) for each \( v \in V \) can be implemented in \( O(n) \) time and by using the above results. Then, Algorithm 7 runs in \( O(xx) \) time where \( O(n\log(n) < O(xx) < O(n^2) \).\)

5.2. Examples of Practical Networks

The mathematical modeling process is an important tool used to formulate many applied life problems to be forms of mathematical problems. At the core of every policy or planning decision, there are intended options to obtain one or more outcomes. The field of optimization is concerned with how this process can be quantitatively modeled, and, within the restrictions of these quantitative models, how the best decisions can be made. In other words, it is a science that researches the best solution among many solutions when there exists no exact (known) solution to the problem.

In recent years, the problem of finding the optimal locations of servers on an operating network as a given tree network. See for example [20,29,30].

Hence, in the following, we present two practical examples on a tree network.

5.2.1. Example 1

This problem represents a small city, and in this town there is a medical center facility that provides the town with several medical services.

The director of the medical center intends to implement a children’s vaccination campaign against polio, and then he formed two teams to achieve this aim. The resident blocks of the city are associated with the medical center by three roads: the first one is on the right of the medical center, the second is on the left of the medical center and the third is in front of the medical center.

However, both teams have to move in the two paths only and they cannot visit all resident blocks on the two roads. The financial capabilities to visit all facilities are not available.

So, we present a solution to this problem as follows.

We design the map of the city according description above as a tree network, we consider the medical center as a root of a tree and the three roads are edges associated with the root. See Figure 6.
We use Algorithm 7 to find the optimal path that will provide effort and money for the medical team.

Assume $r$ is a root of the tree $T_1$ in Figure 6. From this figure, we show how Algorithm 7 runs. The optimal path that we advise the team to go by is the red path, which is shown in Figure 6 and in Table 10.

**Table 10.** The one-core path with length $\ell = 6$ for the tree $T_1$ in Figure 6.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\delta$</th>
<th>$\ell$</th>
<th>$P^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>178</td>
<td>40</td>
<td>1</td>
<td>$p_1 = (r, d)$.</td>
</tr>
<tr>
<td>147</td>
<td>71</td>
<td>2</td>
<td>$p_2 = p_1, (d, m)$.</td>
</tr>
<tr>
<td>124</td>
<td>94</td>
<td>3</td>
<td>$p_3 = p_2, (r, h)$.</td>
</tr>
<tr>
<td>111</td>
<td>107</td>
<td>4</td>
<td>$p_4 = p_3, (h, g)$.</td>
</tr>
<tr>
<td>110</td>
<td>108</td>
<td>5</td>
<td>$p_5 = p_4, (g, g)$, or $p_4, (m, m)$.</td>
</tr>
<tr>
<td>109</td>
<td>109</td>
<td>6</td>
<td>$p_6 = p_5, (g, g)$, $(m, m)$.</td>
</tr>
</tbody>
</table>

5.2.2. Example 2

The objective is to locate a path of the length in Figure 7. We are asked to select the best path on the tree network that will represent one path for a tram on a city.
Our Algorithm 7 finds this path, which is denoted by the $P^*$ and listed in Column 4 of Table 11, also denoted by the colored line on Figure 7.

Note 1: the minimum sum of the distance equals $\alpha = 234$.

Note 2: the maximum possible number obtained from this path compare to the other paths equals to $\delta = 134$.

Table 11. The one-core path with length $\ell = 16$ for the tree in Figure 7.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\delta$</th>
<th>$\ell$</th>
<th>$P^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>348</td>
<td>20</td>
<td>1</td>
<td>$p_1 = (1, 21)$</td>
</tr>
<tr>
<td>328</td>
<td>40</td>
<td>2</td>
<td>$p_2 = (1, 21), (1, 41)$.</td>
</tr>
<tr>
<td>309</td>
<td>59</td>
<td>3</td>
<td>$p_3 = (1, 21), (1, 41), (21, 22)$.</td>
</tr>
<tr>
<td>293</td>
<td>75</td>
<td>4</td>
<td>$p_4 = (1, 21), (1, 41), (21, 22), (41, 42)$.</td>
</tr>
<tr>
<td>280</td>
<td>88</td>
<td>5</td>
<td>$p_5 = p_4, (2, 23)$.</td>
</tr>
<tr>
<td>270</td>
<td>98</td>
<td>6</td>
<td>$p_6 = p_5, (42, 48)$.</td>
</tr>
<tr>
<td>261</td>
<td>107</td>
<td>7</td>
<td>$p_7 = p_6, (48, 49)$.</td>
</tr>
<tr>
<td>254</td>
<td>114</td>
<td>8</td>
<td>$p_8 = p_7, (23, 24)$.</td>
</tr>
<tr>
<td>250</td>
<td>118</td>
<td>9</td>
<td>$p_9 = p_8, (24, 27)$ or $p_9, (49, 50) \backslash (49, 54)$.</td>
</tr>
<tr>
<td>246</td>
<td>122</td>
<td>10</td>
<td>$p_{10} = p_9, (24, 27), (49, 50) \backslash (49, 54)$.</td>
</tr>
<tr>
<td>243</td>
<td>125</td>
<td>11</td>
<td>$p_{11} = p_{10}, (27, 28)$ or $p_{10}, (50, 51) \backslash (54, 55)$.</td>
</tr>
<tr>
<td>240</td>
<td>128</td>
<td>12</td>
<td>$p_{12} = p_{11}, (27, 28), (50, 51) \backslash (54, 55)$.</td>
</tr>
<tr>
<td>238</td>
<td>130</td>
<td>13</td>
<td>$p_{13} = p_{12}, (28, 29)$ or $p_{12}, (51, 52) \backslash (55, 56)$.</td>
</tr>
<tr>
<td>236</td>
<td>132</td>
<td>14</td>
<td>$p_{14} = p_{13}, (28, 29), (51, 52) \backslash (55, 56)$.</td>
</tr>
<tr>
<td>235</td>
<td>133</td>
<td>15</td>
<td>$p_{15} = p_{14}, (29, 30)$ or $p_{14}, (52, 53) \backslash (56, 57)$.</td>
</tr>
<tr>
<td>234</td>
<td>134</td>
<td>16</td>
<td>$p_{16} = p_{15}, (29, 30), (52, 53) \backslash (56, 57)$.</td>
</tr>
</tbody>
</table>

6. Concluding Remarks

We have modified and improved several algorithms to obtain a developed and an efficient algorithm that finds an $\ell$–core of an unweighted tree network. The suggested algorithm is applied on a tree network that represents facilities in a large city. The aim of that is to specify the best path for public transportation to provide a service for a maximum number possible of people who are located at the facilities and neighborhoods of these facilities.

As a future investigation, one may consider a two-core problem of a weighted tree network.

A meta-heuristic strategy can be considered to develop algorithms solved with a problem tree network, for instance median path problems, central problems, and eccentricity problems. Hybridized algorithms for finding one-core or two-core of an unweighted and weighted tree will be considered.


Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare that they have no conflicts of interest.
References


3. Beineke, L.W.; Wilson, R.J.; Cameron, P.J. Topics in Algebraic Graph Theory; Cambridge University Press: Cambridge, UK, 2004; Volume 102.


5. Deo, N. Graph Theory with Applications to Engineering and Computer Science; Courier Dover Publications: Mineola, NY, USA, 2017.

6. Van Steen, M. An Introduction to Graph Theory and Complex Networks; Computer Science Department, VU Amsterdam: Amsterdam, The Netherlands, 2010.


9. Cui, S.; Chen, Y.; Li, X. A Robust and Efficient UAV Path Planning Approach for Tracking Agile Targets in Complex Environments. Machines 2022, 10, 931. [CrossRef]


29. Batta, R.; Berman, O. A location model for a facility operating as an M/G/k queue. Networks 1999, 39, 717–728. [CrossRef]


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.