J. Algebra Comb. Discrete Appl.
9(3) • 133–147

Received: 4 September 2020
Accepted: 7 May 2022

Journal of Algebra Combinatorics Discrete Structures and Applications

Computation of minimum d-hop connected dominating set of trees in $O(n)$ time

Research Article

Amita Samanta Adhya, Sukumar Mondal , Sambhu Charan Barman , Jonecis A. Dayap

Abstract: For a graph $G = (V, E)$ and a fixed constant $d \in \mathbb{N}$, a subset D_{hd} of the vertex set V is a *d-hop connected dominating set* of the graph G if each vertex $t \in V$ is situated at most d-steps from at least one vertex $z \in D_{hd}$, that is, $d(t, z) \leq d$, and the subgraph of G induced by D_{hd} is connected. If D_{hd} has minimum cardinality, then it is a minimum d-hop connected dominating set. In this paper, we present two $O(n)$-time algorithms for computing a minimum D_{hd} of trees with n vertices. We also design an algorithm to find the central vertices of a tree. Besides that, we also study some properties related to hop-domination on trees.

2010 MSC: 05C30, 05C12

Keywords: D-hop connected domination, Domination number, Trees

1. Introduction

A simple, connected, undirected graph without any cycles is called a *tree*. Trees are the very simplest category of graphs with enormous descriptive structure. Cayley [9] introduced the term "tree" first. Macky [25] made a tree database with at most 18 vertices. A rooted tree comes with a specified vertex, known as the root, which is distinct from the other nodes by being the ancestor node of every node

Amita Samanta Adhya; Research Centre in Natural and Applied Sciences (Department of Computer Science), Raja N. L. Khan Women's College (Autonomous), India (email: amitasamanta1@gmail.com).
Sukumar Mondal; Department of Mathematics, Raja N. L. Khan Women's College (Autonomous), India (email: smnlkhan@gmail.com).
Sambhu Charan Barman (Corresponding Author); Department of Mathematics, Shahid Matangini Hazra Government General Degree College for Women, India (email: barman.sambhu@gmail.com).
Jonecis A. Dayap; Department of Mathematics and Sciences, University of San Jose-Recoletos, Cebu, Philippines (email: jdayap@usjr.edu.ph).

ISSN 2148-838X

and every edge of the tree is generated either directly or indirectly from the root. The root is always drawn at the top of the tree as shown in Figure 1(b). The *eccentricity* $E(z)$ of a vertex z in tree T is the maximum distance from vertex z to any other vertex y of T that is, $E(z) = max\{d(z, y) : y \in V\}$. Maximum eccentricity is called the *diameter* of the tree T, i.e., $diameter(T) = max\{E(z) : z \in V\}$. A vertex with minimum eccentricity is the *central vertex*. Every tree has either one or two central vertices. A tree is called *central tree* if it has single central vertex and *bi-central tree* if it has two central vertices. Some applications of trees are counting of saturated-hydrocarbons, cryptographic application in modern computer science and formation of electrical circuits, etc. [20]. A tree is one of the popular and useful data structure in modern computer science. There are different types of trees for different purposes. For storing image data effectively, some hierarchical data structures such as quadtrees, octrees are utilized [35]. For fast key searching in database systems, we use self-balancing binary search trees named AVL trees.

By the graph structure $G = (V(G), E(G))$, we mean that V is the node set or vertex set of G and E is the edge set of G, wherein $|V(G)| = n$, $|E(G)| = m$. For an arbitrary graph G, if each node of V is dominated by at least one node of a node set $B \subseteq V$, then B is a *dominating-set (D-set)* of G. A D-set with minimum cardinality is called a *minimum D-set*. The cardinality of a minimum D-set is called the *domination number (D-number)* and it is denoted by $\gamma(G)$. A node set $D_{hd} \subseteq V$ is a *d-hop dominating-set (d-hop D-set)* of graph G if every node $t \in V$ is situated at most d-distances from at least one node $z \in D_{hd}$, that is, $d(t, z) \leq d$, where d is a fixed positive integer. Now, a node set $D_{hd} \subseteq V$ is a *d-hop connected dominating-set (d-HCDS)* of graph G if every node $t \in V$ is situated at most d-distances from at least one node $z \in D_{hd}$, that is, $d(t, z) \leq d$, and the subgraph of G induced by D_{hd} is connected. A d-HCDS with minimum cardinality is called a minimum d-hop connected dominating-set (MdHCDS). The d-hop domination number of a graph G is the cardinality of a minimum d-hop connected dominating-set of G, and it is denoted by $\gamma_{hk}(G)$.

1.1. Survey of the related works

Domination is always an interesting and vital topic to researchers. Claude Berge, in his book, [8] defined the basic idea of the D-number $\gamma(G)$ of a graph for the first time. The term D-set and D-number were first used by Ore [28]. About twenty years ago, Haynes et al. [21] wrote a revolutionary book on fundamentals of domination, and they recorded over twelve thousand research articles on domination and its different parameters in graphs. Sampathkumar et al. [31] introduced the term "Connected domination number" first. Later, Dorbec et al. [17] established independent-domination in cubic graphs. Variations of domination like connected domination [15], edge domination, k-tuple domination and k-hop domination [16, 24, 33], weighted domination, paired-domination, perfect domination, secured domination, independent-domination, roman domination have been briefly discussed in the literature [10, 21, 23]. Slater renames a k-hop Dominating set as a k-basis [33]. Besides these, many researchers studied various properties of graphs with respect to domination ([11, 13, 21, 22, 34, 36–38]). Like most of the general graph problems, MdHCDS problem is NP-Complete for random graphs ([39]), even in unit disc graphs (in short, UDG), this problem is NP-Complete ([27]). On planar graphs, Demaine et al. [16] proposed an algorithm that takes $O(n^4)$ time to determine k-hop dominating set with the fewest members. Natarajan et al.[26] have done some fundamental works on hop domination number on some special class of graphs. Also, on permutation-graphs Rana et al. [29] set up an effective algorithm to find a distance-k dominating set. Later, Ayyaswamy et al. [3] worked on the upper and lower limits of the hop D-number on trees. Besides these, on UDGs and random graphs many researchers set up some algorithms for solving k-hop Connected D-set problem [14, 19, 30, 41]. Also, Basuchowdhuri et al. [5] determined influential nodes for traditional communication networks using k-hop D-set. Kundu et al. [24] set up an optimal algorithm for determining a k-hop D-set on trees. It is a general work of the prior result to find an 1-hop D-set (with the fewest cardinality) of a tree [13]. Favaron et al. [18] showed that the diameter of the domination k-critical graphs is at most $2(k-1)$ when $k \geq 2$. Later, Ramy S. Shaheen [32] showed the bounds for the 2-domination number of toroidal grid graphs. Also, Barman et al. [4] designed an $O(n)$ time algorithm

to find minimum d-hop D-set on interval graphs. Recently Adhya et al. [1] presented an algorithm for computing minimum k-hop connected D-set in $O(n)$ time on permutation graphs.

1.2. Applications

To the best of our knowledge, domination is one of the important and fast-developing research problems in the core graph theory. Some useful applications [2, 6, 7] of domination are found in many vital areas, such as computer-based communication networks, wireless-radio stations, kernels of games, coding-theory, modelling of biological-networks, land-surveying and problems of finding radar-stations. Besides these, d-hop domination has lots of applications in facility location problems(FLP). Domination appears in FLP for fixed number of facilities(like relief centres, disaster control centres [2]) and if somebody efforts to shorten the distance to travel so that he can get the service from the closest facility. The concept of d-HCDS is used in ad-hoc networks [40, 42] to enhance the performance of efficiency in communication. Ad-hoc networks have no fixed infrastructure. These networks are used in applications like search and rescue, mobile commerce and military battlefields.

1.3. Main outcome

Here, we present two $O(n)$ time algorithms to determine an MdHCDS on trees, where $n = |V|$. The concept of the problem is the same with computing a connected D-set, only it differs concerning the number of steps/hops which is required for reaching all the members of V. Actually we generalize minimum connected hop domination problem as minimum d-hop connected D-set problem of trees.

1.4. Organization of our paper

Section 2 describes the formation of BFS-tree $T'(u)$ and $T(v)$ of tree T. Here we also present some notations which are used to solve our proposed problem. We state and prove some essential properties related with MdHCDS of trees in Section 3. Besides these, we also present two algorithms: one for co-ordinates generation and other for finding central vertex(s) of tree T. In section 4, we present two complete optimal algorithms for computing MdHCDS on trees. In this section, we also calculate time and space complexity of our main algorithms.

2. Construction of BFS-trees $T'(u)$ and $T(v)$

In graph theory, there are several graph traversal technique, one of them is BFS technique. It can form a BFS-tree. Many researchers used BFS to solve different problems. A polynomial time algorithm which runs in $O(n+m)$ time is available for construction of a BFS-tree on an arbitrary graph. On tree [12] Chen et al. presented an efficient algorithm for creating a BFS-tree. In this paper, first we make a BFS-tree $T'(u)$ taking u as root, where u is an arbitrary vertex of a tree T. After then, we make another BFS-tree $T(v)$ taking v as root, where v is any leaf node at last level(highest level) of $T'(u)$. These two BFS-trees can be made separately in $O(m+n)$ time $\approx O(n)$ time, because $m = n-1$. The BFS-tree $T(v)$, taking root at $v = 1$ of the tree of Figure 1(a) is shown in Figure 2(a), taking $u = 11$. We denote the level of a node x belongs to any BFS-tree rooted at y by the symbol $level(x)$ and defined by $level(x) = d(x, y)$, assuming that $level(v) = 0$.

Lemma 2.1. *If w be any vertex at the last level of $T(v)$ then diameter$(T) = d(v, w)$.*

Proof. Let $T(V, E)$ be an arbitrary tree. Now, we construct a BFS-tree $T'(u)$ taking u, an arbitrary vertex of the tree T as root. After then, we make another BFS-tree $T(v)$ taking v, any leaf node at last level(highest level) of $T'(u)$ as root. Since the tree $T'(u)$ is a BFS-tree and level of v is maximum on it, then v is a farthest vertex from u, i.e., $d(u, v) = max\{d(u, x) : x \in V(T)\}$, i.e., $E(u) = d(u, v)$. Again, $T(v)$ is also a BFS-tree, so, w, a vertex at highest level on $T(v)$ is a farthest vertex from v, i.e., $d(v, w) = max\{d(v, y) : y \in V(T)\}$, i.e., $E(v) = d(v, w)$. Now, there are three cases may arise.
Case 1: If $level(w) = level(u)$ on $T(v)$, then u, w are both farthest vertices from v, i.e., $d(v, u) = d(v, w) = max\{d(x, y) : x, y \in V(T)\} = diameter(T)$.
Case 2: If the shortest path between v and w passes through u, then $d(v, w) = d(v, u) + d(u, w) = max\{d(x, y) : x, y \in V(T)\} = diameter(T)$.
Case 3: If $level(w) > level(u)$ on $T(v)$ and the shortest path between v and w does not pass through u, then $d(v, w) = d(v, u) + d(u, w) - 2\times$length of the common part of the shortest paths from u to v and u to w. Also, $d(v, w) = max\{d(x, y) : x, y \in V(T)\} = diameter(T)$. Hence the result is proved. □

Corollary 2.2. *If H is the height of the BFS-tree $T(v)$ then $diameter(T) = diameter(T(v)) = diameter(T'(u)) = H$.*

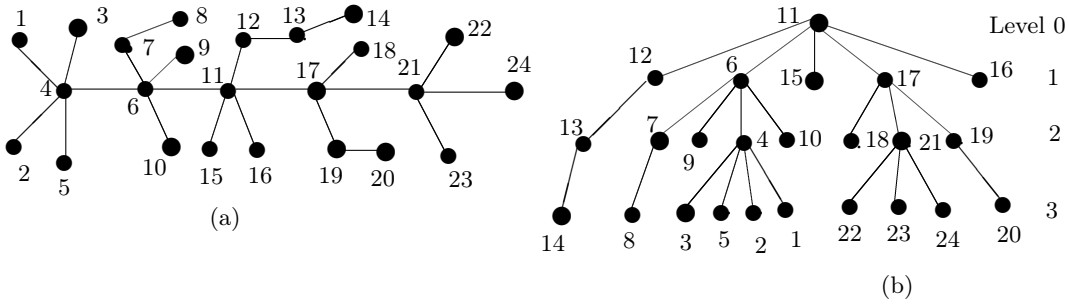

Figure 1. **(a) A tree T and (b) BFS-tree $T'(u)$ of tree T .**

2.1. Identification of central-path lying on BFS-tree $T(v)$

We consider H as the height of the formed BFS-tree $T(v)$. Also we presume that v_{last} is an arbitrary node at level H on $T(v)$. Now we consider the shortest path $v_{last} \to parent(v_{last}) \to parent(parent(v_{last})) \to \cdots \to v$ between v and v_{last} as the *central-path* between v and v_{last}.
We also consider that if a node lying on the central-path at level j, then we denote it by v_j^*.

2.2. Notations

In this subsection, we present some useful notations that are needed throughout this paper.

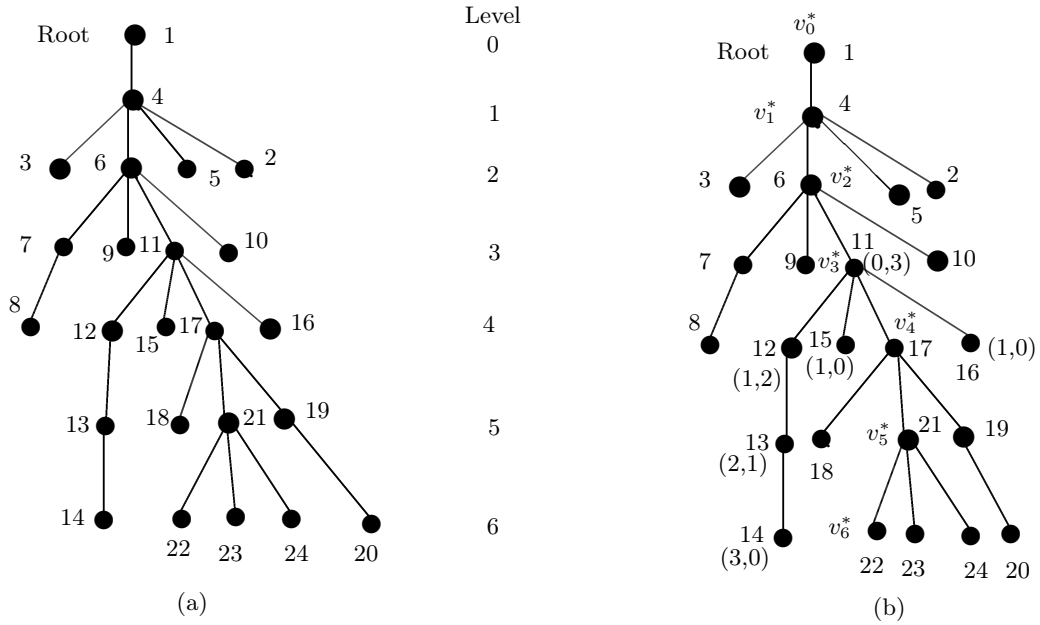

Figure 2. **(a) BFS-tree $T(v)$ and (b) BFS-tree $T(v)$ after 2-tuple weight assignment.**

v_r^*	: The node lying on the central-path of BFS-tree $T(v)$ at level r.
$T_s(v_r^*)$	: Rooted sub-tree (v_r^* as root) of $T(v)$ contains all the vertices imitate from : v_r^* (excluding the nodes lying on the central-path except v_r^*). Here v_r^* is the : source vertex of any other vertex of that sub-tree.
M_j	: M_j is the set of vertices at level j on the BFS-tree $T(v)$.
x_j	: x_j is any member of the set $M_j - \{v_j^*\}$.
H	: Height of the BFS-tree $T(v)$.
d	: d is a fixed positive integer.
D_{hd}	: D_{hd} is MdHCDS.
$u(r, j)$	: Vertex u with 2-tuple weight assignment, where r, j are respectively 1st and : 2nd weight components on $T(v)$.
$parent(z)$	: Parent node of the vertex z.
$x \hookrightarrow y$	: Shortest path between x and y.

2.3. Weight assignment on vertices

Here we describe the weight assignment process to some vertices, when $d < \lfloor (H/2) \rfloor$. For the BFS-tree $T(v)$, we assign a new kind of 2-tuple weights (of the form (x, y), where x, y are respectively the 1st and 2nd weight components) on the vertices of rooted sub-trees $T_s(v_{d+1}^*), T_s(v_{d+2}^*)$, $\ldots, T_s(v_{H-d-1}^*)$ rooted at respectively $v_{d+1}^*, v_{d+2}^*, \ldots, v_{H-d-1}^*$. First, select the vertex v_{d+1}^* and assign its 1st weight component as 0 and then move to child nodes and assign their first weight components as 1. After that for each child nodes of v_{d+1}^*, we move to its child nodes and place their first weight components as 2. This process will be continued until all the vertices of the rooted sub-trees rooted at v_{d+1}^* get their first weight components. Then we start back tracing from the last level (highest level) of the sub-tree $T_s(v_{d+1}^*)$, selecting (one by one) a vertex z having no 2nd weight component and assign its 2nd weight component as 0. After that we assign the 2nd weight component of the parent vertex of z as 1 and for the parent of the parent vertex of z as 2 and continue until v_{d+1}^* gets 2nd weight component. Again we start back tracing from the 2nd last level (2nd highest level) of the sub-tree $T_s(v_{d+1}^*)$, selecting

(one by one) a vertex y (if exits) having no 2nd weight component and assign its 2nd weight component as 0. After that we assign the 2nd weight component (if it's absent) of the parent vertex of y as 1 and for the parent of the parent vertex of y as 2 and so on. This process will be continued until all the members of the sub-rooted trees rooted at v^*_{d+1} get their second weight components. It is to be keep in mind that during this weight assignment process, if a vertex of $T_s(v^*_{d+1})$ has already got 2nd weight component then we skip it. We apply the same procedure to assign the 2-tuple weight assignment on the vertices of rooted sub-trees $T_s(v^*_{d+2}), T_s(v^*_{d+3}), \ldots, T_s(v^*_{H-d-1})$ separately. In this way, we assign the 2-tuple weight assignment (as shown in Figure 2(a)) on all the vertices of the rooted sub-trees, rooted at v^*_r, where $r \in \{d+1, d+2, \ldots, H-d-1\}$.

2.4. Algorithm for 2-tuple weight assignment

Algorithm 2TUP-WT
Input : Tree $T(v)$.
Output: 2-tuple weight assignment of $\cup_{r=d+1}^{H-d-1} V(T_s(v^*_r))$.
Initially $d(< \lfloor (H/2) \rfloor) \in \mathbb{N}$.
Step 1: Identify the nodes v^*_i, $i = 0, 1, ..., H$, lying on the central-path of $T(v)$.
Step 2: Set $r = d + 1$.
Step 3: Do {
Step 3.1: Select a vertex v^*_r and assign its 1st weight component as 0.
Step 3.2: Move to the child nodes of v^*_r on $T_s(v^*_r)$ and assign their 1st weight components as 1.
Step 3.3: For each child nodes of v^*_r, move to its child nodes and assign their 1st weight components as 2.
Step 3.4: Continue till all the members of the set $V(T_s(v^*_r))$ get their 1st weight components.
Step 3.5: Compute the level of any vertex placed at last level (highest level) of $T_s(v^*_{d+1})$ and assign it with L.
Step 3.6: Do{
Step 3.6.1: Assign the 2nd weight components of all the vertices of $T_s(v^*_{d+1})$ at L^{th} level as 0, if unavailable; else skip.
Step 3.6.2: For each vertex z at L^{th} level of $T_s(v^*_{d+1})$ move to its parent vertex and assign the 2nd weight component of the $parent(z)$ as 1, if unavailable; else skip.
Step 3.6.3: Similarly assign 2nd weight component of $parent(parent(z))$ as 2 and continue until all internal nodes (including v^*_{d+1}) of the path between z and v^*_{d+1} get 2nd weight components.
Step 3.6.4: Set $L = L - 1$.
} while $(L > r)$.
Step 3.7: $r = r + 1$.
} while $(r \leq (H - d - 1))$.
End 2TUP-WT.

2.5. Illustrative example of 2-tuple weight assignment

To describe the procedures of the above algorithm (Algorithm **2TUP-WT**), we first consider the tree $T(v)$ of Figure 2(a). So, $H = 6$. Initially, we assume that $d = 2 \leq \lfloor (H/2) \rfloor$. In Step 1, we identify the members on the central-path (considering $v_{last} = 22$) as $v^*_0 = 1, v^*_1 = 4, v^*_2 = 6, v^*_3 = 11, v^*_4 = 17, v^*_5 = 21, v^*_6 = 22$. In Step 2, we set $r = 2+1 = 3$. In Step 3, we enter in the do-while loop for $(r \leq (H-d-1))$ to assign 2-tuple weight on vertices. In step 3.1, we select the root $v^*_3 = 11$ of the rooted sub-tree $T_s(v^*_{d+1}) = T_s(v^*_3)$ and assign its 1st weight component as 0 and write like $11(0, -)$. After that, in Step

3.2, we move to its child nodes and assign their 1st weight components as 1, $12(1,-), 15(1,-), 16(1,-)$. In Step 3.3, for each child nodes of 11, we move to their child nodes and assign their 1st 1st weight components as 2, i.e, $13(2,-)$. Similarly vertex 14 get its 1st weight component as $14(3,-)$ in Step 3.4. Thus all vertices of the tree $T_s(v_3^*)$ are assigned with their 1st weight components. In Step 3.5, we compute $L = 6$. In Step 3.6, we enter into another do-while loop for $L > r$. In this loop we will the 2nd weight components of the vertices of $T_s(v_3^*)$. At first, in Step 3.6.1, we move to vertices placed at the last level (6th) of $T_s(v_3^*)$ and see that there is only one vertex which is 14 and it has no 2nd weight component. So we assign 2nd weight component of the vertex 14 as 0, i.e., $14(3,0)$. In Step 3.6.2, we select the parent node of 14, which is 13 and put its 2nd weigh component as $13(2,1)$. Similarly, in Step 3.6.3, the vertices 12 and 11 get their 2nd weight components as 2 and 3 respectively and write like $12(1,2)$ and $11(0,3)$. Next in Step 3.6.4, we reset $L = L - 1 = 5$ and move to the Step 3.6.1 as $L > r$. Since, all vertices of $T_s(v_3^*)$ at level 5 have 2nd weight components, so move to the Step 3.6.4, and reset Again, we move to the Step 3.6.1 (as $L > r$) and see that there are two vertices 15 and 16 which have no 2nd weight components. So we assign 2nd weight components of 15 and 16 as 0. Since $parent(15) = parent(16) = 11$ and vertex 11 has already got 2nd weight component, so, we move to the Step 3.6.4, and reset $L = L - 1 = 3$. Since $L < r$, so the present do-while loop for $L > r$ is finished and move to the Step 3.7. Here we reset $r = r + 1 = 4$. Since, $r > H - d - 1 = 3$, so the do-while loop for $r \leq H - d - 1$ is finished. Thus the execution process of Algorithm **2TUP-WT** is finished. The final 2-tuple weight assignment of the vertices of $T(v)$ is shown in Figure 2(b).

Theorem 2.3. *The run time of Algorithm* **2TUP-WT** *is* $O(n)$.

Proof. In Step 1, the time needed for identifying the vertices v_i^*, $i = 0, 1, 2, \ldots, H$, on the central path of the tree $T(v)$ is $O(n)$. Step 2 takes constant time. Since, $V(T_s(v_{d+1}^*)), V(T_s(v_{d+2}^*)), \ldots,$ $V(T_s(v_{H-d-1}^*))$ are mutually disjoint, so, in Step 3, the time for assigning 1st weight components of the vertices $\cup_{r=d+1}^{H-d-1} V(T_s(v_r^*))$ is $O(n+m) \approx O(n)$ time (in a worst case) and the time required for assigning 2nd weight components of the same vertices is also $O(n)$ time. So, Step 3 can be finished in $O(n)$ time. Hence, the overall time complexity of Algorithm **2TUP-WT** is $O(n)$. □

2.6. Algorithm for computation of central node(s) of T

Algorithm T-CENTER
Input : Tree T.
Output: Central node(s) of tree T.
Step 1: Make BFS-tree $T'(u)$ taking u as root, where u is an arbitrary vertex.
Step 2: Make BFS-tree $T(v)$ taking v as root, where v is an arbitrary pendant vertex at highest level of $T'(u)$.
Step 3: Identify the central-path and determine the height H.
Step 4: If H is even then central node of T is $v_{\lfloor H/2 \rfloor}^*$
Else central nodes of T are $v_{\lfloor H/2 \rfloor}^*$ and $v_{\lfloor H/2 \rfloor + 1}^*$.
End If
End T-CENTER.

The Algorithm T-CENTER provides that the central vertex of the tree T shown in the Figure 1(a) is $V_3^* = 11$.

Theorem 2.4. *The run time of Algorithm* **T-CENTER** *for determining the central node(s) of tree* T *is* $O(n)$*, where* $|V| = n$.

Proof. At Step 1 and Step 2, two BFS-trees $T'(u)$ and $T(v)$ can be made separately in $O(n)$-time. In Step 3, the members of the central-path and the height H can be determined in $O(n)$-time, because there

is only $(H+1)(\leq n)$ nodes located on the central-path. Again, we can finish Step 4 in constant time. So, overall time complexity of the algorithm **T-CENTER** is $O(n)$. □

3. Important properties related to MdHCDS D_{hd} of trees.

Here, we have proved some emergent results related to MdHCDS D_{hd} of trees.

Lemma 3.1. *If $r \leq \lfloor H/2 \rfloor$ and z_j is any leaf node at j^{th} level on $T_s(v_r^*)$ then $d(v_r^*, x_j) \leq r$.*

Proof. We know, by Corollary 2.2, that $H = diameter(T) = diameter(T(v))$, i.e., $v_0^* \to v_1^* \to \cdots \to v_r^* \to v_{r+1}^* \to \cdots \to v_H^*$ is the longest shortest path on both trees T and $T(v)$. Now, if $r \leq \lfloor H/2 \rfloor$ and z_j be any leaf node at j^{th} level on $T_s(v_r^*)$, then $r < j \leq H$. Again, $d(v_0^*, v_r^*) = r$ and $d(v_r^*, v_H^*) = (H-r)$. So, $d(v_r^*, v_H^*) \geq d(v_0^*, v_r^*)$ as $r \leq \lfloor H/2 \rfloor$. Let us assume that $d(v_r^*, z_j) > r$. So $(j-r) > r \Rightarrow j > 2r \Rightarrow (j-2r) > 0$. Therefore, $d(v_H^*, z_j) = d(v_H^*, v_r^*) + d(v_r^*, z_j) = (H-r) + (j-r) = H + (j-2r) > H$ as $(j-2r) > 0$. So, $v_H^* \hookrightarrow v_r^* \hookrightarrow z_j$ is the longest shortest path on $T(v)$ which is impossible as diameter is H. So, our assumption is wrong. Therefore, $d(v_r^*, z_j) \leq r$. □

Lemma 3.2. *If $\lfloor H/2 \rfloor < r$ and z_j is any leaf node at j^{th} level on $T_s(v_r^*)$ then $d(v_r^*, z_j) \leq H - r$.*

Proof. We know, by Corollary 2.2, that $H = diameter(T) = diameter(T(v))$, i.e., $v_0^* \to v_1^* \to \cdots \to v_r^* \to v_{r+1}^* \to \cdots \to v_H^*$ is the longest shortest path on both trees T and $T(v)$. Now, if $r > \lfloor H/2 \rfloor$ and z_j be any leaf node at j^{th} level on $T_s(v_r^*)$, then $\lfloor H/2 \rfloor < r < j$. Also, $d(v_0^*, v_r^*) = r$ and $d(v_r^*, v_H^*) = (H-r)$. Since, $r > \lfloor H/2 \rfloor$, $d(v_0^*, v_r^*) > d(v_r^*, v_H^*)$. Let us assume that $d(v_r^*, z_j) > H - r$. Now, $d(v_r^*, z_j) = (j-r)$. So, $(j-r) > (H-r)$ that implies $j > H$, i.e., $d(v_0^*, z_j) = j > H$ which is impossible, as H is the diameter of T. So, our assumption is wrong. Therefore, $d(v_r^*, z_j) \leq (H-r)$. □

Corollary 3.3. *If z_r be any member at even level $r > 1$ on $T(v)$, then its source vertex will be any one of $\{v_{r-1}^{*,}, v_{r-2}^*, \ldots, v_{r/2}^*\}$.*

Corollary 3.4. *If z_r be any member at odd level $r > 2$ on $T(v)$, then its source vertex will be any one of $\{v_{r-1}^{*,}, v_{r-2}^*, \ldots, v_{(r+1)/2}^*\}$.*

Lemma 3.5. *The tree $T(v)$ with at least three nodes has only one internal node at level 1.*

Proof. Let T be any tree having at least three nodes. Now, for constructing $T(v)$, at first we make a BFS-tree $T'(u)$ where u is any leaf node on T. Then we construct $T(v)$, where v is any leaf node at the highest level(last level) on $T'(u)$. This implies that the vertex v is incident only one edge i.e., v is the parent of only one internal vertex on $T(v)$. □

Corollary 3.6. *There is no leaf nodes at level 1 on $T(v)$.*

Lemma 3.7. *For each $z \in \cup_{r=0}^{2d} M_r$, $d(v_d^*, z) \leq d$.*

Proof. Let z_r be any member of $\{M_r - v_r^*\}$, for $r = 2, 3, \ldots, 2d$. So, for $r = 0, 1, \ldots, d$, $d(v_d^*, v_r^*) \leq (d-r)$. Now if r is even and $2 \leq r \leq d$ then, using Corollary 3.3, $d + 2 - r \leq d(z_r, v_d^*) \leq d$(as $z_r \to v_{r-1}^* \to v_r^* \to v_{r+1}^* \to \cdots \to v_d^*$ or $z_r \to z_{r-1} \to v_{r-2}^* \to v_{r-1}^* \to v_r^* \to v_{r+1}^* \to \cdots \to v_d^*$ or $\ldots z_r \to z_{r-1} \to \cdots \to z_{(r/2)+1} \to v_{r/2}^* \to v_{(r/2)+1}^* \to \cdots \to v_d^*$). Again if r is odd and $3 \leq r \leq d$ then, using Corollary 3.4, $d + 2 - r \leq d(z_r, v_d^*) \leq (d-1)$(as $z_r \to v_{r-1}^* \to v_r^* \to v_{r+1}^* \to \cdots \to v_d^*$ or $z_r \to z_{r-1} \to v_{r-2}^* \to v_{r-1}^* \to v_r^* \to v_{r+1}^* \to \cdots \to v_d^*$ or $\ldots z_r \to z_{r-1} \to \cdots \to z_{((r+1)/2)+1} \to v_{(r+1)/2}^* \to v_{((r+1)/2)+1}^* \to \cdots \to v_d^*$). Therefore, $d(v_d^*, z) \leq d$, $\forall\ z \in \cup_{r=0}^{d} M_r$. Again, $d(v_d^*, v_r^*) \leq (r-d)$, for $r = d+1, d+2, \ldots, 2d$. Also if r is even and $d+1 \leq r \leq 2d-2$, then $r - d \leq d(v_d^*, z_r) \leq d$(as $z_r \to v_{r-1}^* \to v_{r-2}^* \to \cdots \to v_d^*$ or $z_r \to z_{r-1} \to v_{r-2}^* \to v_{r-3}^* \to \cdots \to v_d^*$ or $\ldots$or $z_r \to z_{r-1} \to$

$z_{r-2} \to \cdots \to z_{d+1} \to v_d^*$ or $z_r \to z_{r-1} \to \cdots \to z_d \to v_{d-1}^* \to v_d^*$ or $z_r \to z_{r-1} \to \cdots \to z_d \to z_{d-1} \to v_{d-2}^* \to v_{d-1}^* \to v_d^*$ or $\ldots$ or $z_r \to z_{r-1} \to \cdots \to z_{(r/2)+1} \to v_{r/2}^* \to v_{(r/2)+1}^* \to \cdots \to v_d^*$). Again if r is odd and $d+1 \leq r < 2d-2$, then $r-d \leq d(v_d^*, z_r) \leq d-1$(as $z_r \to v_{r-1}^* \to v_{r-2}^* \to \cdots \to v_d^*$ or $z_r \to z_{r-1} \to v_{r-2}^* \to v_{r-3}^* \to \cdots \to v_d^*$ or $\ldots$or $z_r \to z_{r-1} \to z_{r-2} \to \cdots \to z_{d+1} \to v_d^*$ or $z_r \to z_{r-1} \to \cdots \to z_d \to v_{d-1}^* \to v_d^*$ or $z_r \to z_{r-1} \to \cdots \to z_d \to z_{d-1} \to v_{d-2}^* \to v_{d-1}^* \to v_d^*$ or $\ldots$or $z_r \to z_{r-1} \to \cdots \to z_{((r+1)/2)+1} \to v_{(r+1)/2}^* \to v_{((r+1)/2)+1}^* \to \cdots \to v_d^*$). Also for $r = 2d-1, 2d$ $d(v_d^*, z_r) = d$ (or $d-1$) if r is even (or odd) (as $z_r \to v_{r-1}^* \to v_{r-2}^* \to \cdots \to v_d^*$ or $z_r \to z_{r-1} \to v_{r-2}^* \to v_{r-3}^* \to \cdots \to v_d^*$ or $\ldots$or $z_r \to z_{r-1} \to z_{r-2} \to \cdots \to z_{d+1} \to v_d^*$). So, $d(v_d^*, z) \leq d$, $\forall\ z \in \cup_{r=d+1}^{2d} M_r$. Therefore $d(v_d^*, z) \leq d$, $\forall\ z \in \cup_{r=0}^{2d} M_r$. □

Corollary 3.8. *v_d^* is a possible member of D_{hd}.*

Corollary 3.9. *All the members of $\cup_{r=1}^{d} V(T_s(v_r^*))$ are within d distances from v_d^*.*

Lemma 3.10. *For each $z \in \cup_{r=1}^{d} V(T_s(v_{H-r}^*))$, $d(v_{H-d}^*, z) \leq d$.*

Proof. Let z_r be any node at level r on $T(v)$ and $(H-d) < r \leq H$. Also, let $z_r \in \cup_{r=1}^{d} V(T_s(v_{H-r}^*))$, i.e., source vertex of z_r be one of the set $\{v_{H-d}^*, v_{H-d+1}^*, ..., v_{H-1}^*\}$. Now, $d(v_{H-d}^*, v_H^*) = d$ and $d(v_{H-d}^*, z_r) = (r-H+d) \leq d$ (as $z_r \to v_{r-1}^* \to v_{r-2}^* \to \cdots \to v_{H-d}^*$ or $z_r \to z_{r-1} \to v_{r-2}^* \to v_{r-3}^* \to \cdots \to v_{H-d}^*$ or $\ldots$ or $z_r \to v_{H-d}^*$). So, the result is proved. □

Now, by observing the last result, we present the following result.

Corollary 3.11. *v_{H-d}^* is a possible member of D_{hd}.*

Lemma 3.12. *If $u(r,j) \in T_s(v_l^*(0, r+j))$, $j \geq d$ and $v_l^*(0, r+j) \in D_{hd}$, then $u(r+j-d, d), u(r+j-d-1, d+1), \ldots, u(1, r+j-1)$ are the possible members of D_{hd}.*

Proof. Let $u(r,j) \in T_s(v_l^*(0, r+j))$, $j \geq d$ and $v_l^*(0, r+j) \in D_{hd}$, that is there exists a path of length $(r+j)$ imitate from $v_l^*(0, r+j)$ and $u(r+j, 0)$ passing through $u(r,j)$ of the form $v_l^*(0, r+j) \to u(1, r+j-1) \to u(2, r+j-2) \to \cdots \to u(r,j) \to u(r+1, j-1) \to \cdots \to u(r+j-d-1, d+1) \to u(r+j-d, d) \to u(r+j-d+1, d-1) \to \cdots \to u(r+j, 0)$. So, $d(v_l^*(0, r+j), u(r+j, 0)) = r+j \geq r+k$ and $d(u(r+j-d, d), u(r+j, 0)) = d$. Therefore, $u(r+j-d, d)$ is a possible member of D_{hd}. Also, $v_l^*(0, r+j) \in D_{hd}$. So, for computing d-HCDS, we have to select $u(r+j-d, d), u(r+j-d-1, d+1), \ldots, u(1, r+j-1)$ as the possible member of D_{hd}. □

Lemma 3.13. *If $H \leq d$, then $D_{hd} = \{v_r^*\}$, where r refers any single member of {0, 1, 2,3,...,H}.*

Proof. Let z_r be a member of $M_r - \{v_r^*\}$, for $r = 2, 3, \ldots, H$. Now, $d(v_0^*, v_H^*) = H \leq d$. We know (Corollary 3.6) that there is no leaf nodes at level 1 on $T(v)$. So v_0^* be a probable member of D_{hd} because $d(v_0^*, z_r) = r \leq d$ (as $v_0^* \to v_1^* \to \ldots \to v_{r-1}^* \to z_r$ or $v_0^* \to v_1^* \to \ldots \to v_{r-2}^* \to z_{r-1} \to z_r$ or $v_0^* \to v_1^* \to \ldots \to v_{i-3}^* \to z_{r-2} \to z_{r-1} \to z_r$ or...or $v_0^* \to v_1^* \to z_2 \to z_3 \to \ldots \to z_{r-3} \to z_{r-2} \to z_{r-1} \to z_r$), for $r = 2, 3, \ldots, H$. Furthermore, if r is even and $2 \leq r \leq H$ then $d(z_r, v_H^*) \leq H \leq d$ (as $z_r \to v_{r-1}^* \to v_r^* \to \ldots \to v_H^*$ or $z_r \to z_{r-1} \to v_{r-2}^* \to v_{r-1}^* \to v_r^* \ldots \to v_H^*$ or $\ldots$ or $z_r \to z_{r-1} \to \ldots \to z_{(r-(r/2)+1)} \to v_{(r-(r/2)}^* \to v_{(r-(r/2)+1)}^* \to \ldots \to v_H^*$). Again, if r is odd and $3 \leq r \leq H$ then $d(z_r, v_H^*) \leq H \leq d-1 < d$ (as $z_r \to v_{r-1}^* \to v_r^* \to \ldots \to v_H^*$ or $z_r \to z_{r-1} \to v_{r-2}^* \to v_{r-1}^* \to v_r^* \ldots \to v_H^*$ or $\ldots$ or $z_r \to z_{r-1} \to \ldots \to z_{(r-((r+1)/2)+1)} \to v_{r-((r+1)/2)}^* \to v_{(r-((r+1)/2)+1)}^* \to \ldots \to v_H^*$). Hence, v_H^* is a possible member of D_{hd}. Similarly, other members of the central path on $T(v)$ may be possible members of D_{hd}. So, if $H \leq d$, then $D_{hd} = \{v_r^*\}$, where r refers any single member of {0, 1, 2,...,H}. □

Lemma 3.14. *d-HDS D_{hd} of a tree T is*

$$D_{hd} = \begin{cases} \{z : z \text{ is the central node(s) of } T\} & \text{if } \lfloor H/2 \rfloor = d, \\ \{z : z \text{ any single central vertices of } T\} & \text{if } \lfloor H/2 \rfloor < d. \end{cases}$$

Proof. Let $diameter(T) = diameter(T(v)) = H$. Now, two cases may appear —
Case 1: $d > \lfloor H/2 \rfloor$ and Case 2: $d = \lfloor H/2 \rfloor$.
Case 1: In this case, $d(v_0^*, v_H^*) = H$. Now two sub-cases may appear here. Sub-case 1.1: when H is even. In this sub-case, T is mono-centric and the central vertex is $v^*_{(H/2)}$. So, $d(v^*_{(H/2)}, x) \leq \frac{H}{2} < d, \forall\ z \in V(T)$. Therefore, $D_{hd} = \{v^*_{H/2}\}$. Sub-case 1.2: When H is odd. In this subcase T is bi-centric, and the central vertices are $v^*_{\lfloor H/2 \rfloor}$ and $v^*_{\lfloor H/2 \rfloor+1}$. So, $d(v^*_{\lfloor H/2 \rfloor}, z) \leq \lfloor r/2 \rfloor + 1 \leq d$ and $d(v^*_{\lfloor H/2 \rfloor+1}, z) \leq \lfloor H/2 \rfloor + 1 \leq d, \forall\ z \in V(T)$. Therefore, $D_{hd} = \{v^*_{\lfloor H/2 \rfloor}\}$ or $D_{hd} = \{v^*_{\lfloor H/2 \rfloor+1}\}$.
Case 2: In this case, two sub-cases may appear. Sub-case 2.1: when H is even. In this sub-case, T has single central vertex $v^*_{H/2}$. So, $d(v^*_{H/2}, x) \leq H/2 = d, \forall\ z \in V(T)$. Therefore, $D_{hd} = \{v^*_{H/2}\}$. Sub-case 2.2: When H is odd. In this sub-case T has two central vertices $v^*_{\lfloor H/2 \rfloor}$ and $v^*_{\lfloor H/2 \rfloor+1}$. So, $d(v^*_{\lfloor H/2 \rfloor}, z) \leq \lfloor r/2 \rfloor = d\ \forall\ z \in \cup_{r=0}^{H-1} M_r$ but $d(v^*_{\lfloor H/2 \rfloor}, z_H) = \lfloor r/2 \rfloor + 1 = d + 1 > d$, where $z_H \in M_H$. For instance $d(v^*_{\lfloor H/2 \rfloor}, v^*_H) = \lfloor r/2 \rfloor + 1 = d + 1$ (as $v^*_H \to v^*_{H-1} \to v^*_{H-2} \to \ldots \to v^*_{\lfloor H/2 \rfloor}$). Again, $d(v^*_{\lfloor H/2 \rfloor+1}, z) \leq \lfloor r/2 \rfloor = d\ \forall\ z \in \cup_{r=1}^{H} M_r$ but $d(v^*_{\lfloor H/2 \rfloor+1}, v_0^*) = d + 1 > d$ (as $v^*_H \to v^*_{H-1} \to v^*_{H-2} \to \ldots \to v_0^*$). So, all the vertices of T are within d distances from the members of the set $D_{hd} = \{v^*_{\lfloor H/2 \rfloor}, v^*_{\lfloor H/2 \rfloor+1}\}$. Hence the result. □

4. Complete algorithms and their complexities

Here, we have presented two optimal algorithms for determining minimum d-HCDS D_{hd} of tree T.

4.1. First complete algorithm

From the results discussed in Section 3, it is observed that when $d \leq \lfloor H/2 \rfloor$ and if H is even, then $v^*_{H/2}$ is a possible member of D_{hd}, and if H is odd, then $v^*_{\lfloor H/2 \rfloor}$ and $v^*_{\lfloor H/2 \rfloor+1}$ are two possible members of D_{hd}. Two possible cases for selecting the members of D_{hd} are discussed in Lemma 3.14. The first complete Algorithm **MDHCDST1** is designed for computing a MdHCDS D_{hd} of tree T, presented below.

Algorithm MDHCDST1
Input: A tree $T(V, E)$.
Output: MdHCDS D_{hd} of tree T.
Step 1**:** Make the BFS-tree $T'(u)$ taking u as root, where u is any arbitrary vertex.
Step 2**:** Make BFS-tree $T(v)$ taking v as root, where v is any leaf node at highest level of $T'(u)$.
Step 3**:** Identify central path and compute height H of T and set H= diameter(T).
Step 4**:** Find the center(s) of T.//by Algorithm T-CENTER//
Step 5**:** If $d \leq \lfloor H/2 \rfloor$, then
  Step 5.1**:** Construct a subtree $T_1(v)$ of $T(v)$, by removing the set V_1 of all pendant vertices from $T(v)$.
  Step 5.2**:** For $r = 2, 3, ..., d$, construct the subtree $T_r(v)$ of $T_{r-1}(v)$, by removing the set V_r of all pendant vertices from $T_{r-1}(v)$.
  Step 5.3**:** Assign $D_{hd} = \{V(T_d(v))\}$.
  Else D_{hd}= $\{z : z$ is any one central node of $T.\}\}$ //Lemma 3.14//
  End If
End MDHCDST1.

If we apply the **Algorithm MDHCDST1** for $d = 2$ on the tree T of Figure 1 (a), then we get a minimum 2-hop connected dominating set $D_{h2} = \{6, 11, 17, 12\}$.

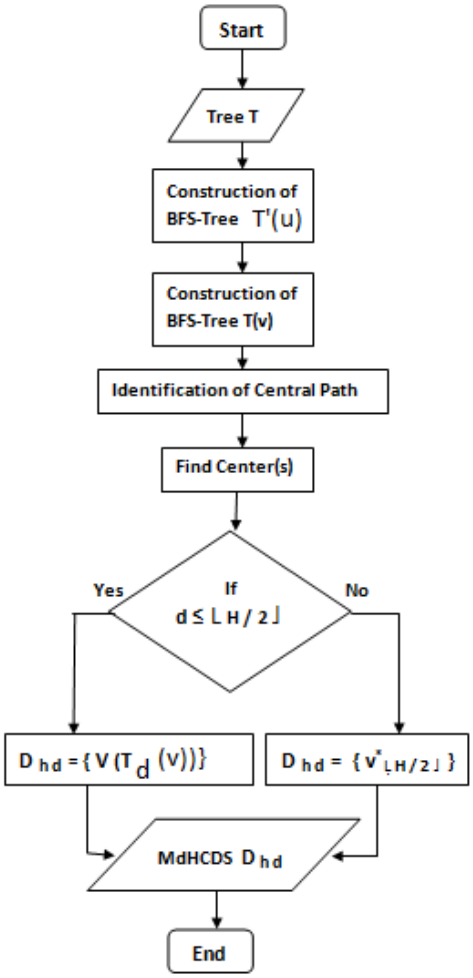

Figure 3. **Flowchart of Algorithm MDHCDST1.**

Lemma 4.1. *The Algorithm* **MDHCDST1** *gives a MdHCDS* D_{hd} *of trees.*

Proof. In Algorithm **MDHCDST1**, if $d \leq \lfloor H/2 \rfloor$ (Step 4), then we set $D_{hd} = V(T_{\lfloor H/2 \rfloor}(v))$. In this case, D_{hd} is a minimum because if we eliminate anyone pendent vertex from $T_{\lfloor H/2 \rfloor}(v)$, then $d(x, y) \geq d$, for all $x \in V(T)$ and $y \in V(T_{\lfloor H/2 \rfloor}(v))$. Again D_{hd} is connected as $T_{\lfloor H/2 \rfloor}(v)$ is a connected sub-graph of T. Moreover, by Lemma 3.13, when H is even and $d > \lfloor H/2 \rfloor$, then T is mono-centric and the central node is $v^*_{H/2}$. For this reason we set $D_{hd} = \{v^*_{H/2}\}$. Again, when H is odd and $d > \lfloor H/2 \rfloor$, then T is bi-centric and the central nodes are $v^*_{\lfloor H/2 \rfloor}$ and $v^*_{\lfloor H/2 \rfloor + 1}$. Also each of these vertices can dominate each vertex of T by d or less steps. Therefore, we set $D_{hd} = \{v^*_{\lfloor H/2 \rfloor}\}$ or $\{v^*_{\lfloor H/2 \rfloor + 1}\}$. Therefore, D_{hd} is a MdHCDS of trees. □

Theorem 4.2. *The run time of the Algorithm* **MDHCDST1** *is* $O(n)$*, where* $|V| = n$.

Proof. Step 1 and Step 2 need $O(n)$ time, respectively, for building the BFS-trees $T'(u)$ and $T(v)$. In Step 3, we can compute the members of the central path and the diameter in $O(n)$ time as $H=$ diameter(T) and there are only $H + 1 \leq n$ vertices in the central path on $T(v)$. In Step 4, central node(s)

of $T(v)$ can be identified in $O(n)$ time (Theorem 2.4). At first iteration of Step 5, we find, and then delete the set V_1 of pendent vertices of tree $T(v)$ and it can be done in $|V_1|$ time. Similarly, other iterations can be finished. Hence, Step 5 takes $O(\cup_{r=1}^{\lfloor H/2 \rfloor}|V_r|) \approx O(n)$ time as V_r, $r = 1, 2, 3, \ldots, \lfloor H/2 \rfloor$ are mutually disjoint. Therefore, the Algorithm **MDHCDST1** runs in $O(n)$-time. □

4.2. Second complete algorithm

In Section 3, we proved that if $H \leq d$, then v_l^* can be accepted as an element of D_{hd}, where l is only one arbitrary element of $\{0, 1, \ldots, H\}$ (Lemma 3.12). Also, if $H > d$, then by Corollary 3.6, Corollary 3.9 and Lemma 3.10, we can take the vertices of the set $\{v_d^*, v_{d+1}^*, \ldots, v_{H-d}^*\} \cup \{u(r, j) \in T^*(v) : j \geq d\}$ as members of D_{hd}. All probable cases for choosing the elements of D_{hd} are already discussed in section 3. The second complete Algorithm **MDHCDST2** for determining a MdHCDS D_{hd} of tree T is presented below.

Algorithm MDHCDST2
Input: A tree T.
Output: MdHCDS D_{hd} of tree T.
Initially $D_{hd} = \phi$.
Step 1: Make BFS-tree $T'(u)$ taking u as root, where u is any vertex of $V(T)$.
Step 2: Make BFS-tree $T(v)$ taking v as root, where v is any leaf node at highest level of $T'(u)$.
Step 3: Identify the central-path and determine the vertices lying on that central-path of $T(v)$ and denote them by $v_r^*, r = 0, 1, \ldots, H$.
Step 4: Do 2-tuple weight assignment of $V(T_s(v_r^*))$, $r = d+1, d+2, \ldots, H-d-1$.
Step 5: If $H \leq d$, then
$D_{hd} = D_{hd} \cup \{v_l^*\}$, where l is only one arbitrary element of $\{0, 1, \ldots, H\}$. //Lemma 3.12//
Else
$D_{hd} = D_{hd} \cup \{v_d^*, v_{d+1}^*, \ldots, v_{H-d}^*\} \cup \{u(r, j) \in T^*(v) : j \geq d \text{ and } r \neq 0\}$
// (Lemma 3.10, Corollary 3.6 and Corollary 3.9)//
End If
End MDHCDST2.

If we apply the **Algorithm MDHCDST2** for $d = 2$ on the tree T of Figure 1 (a), then we get a M2HCDS $D_{h2} = \{6, 11, 17, 12\}$.

Lemma 4.3. *The Algorithm* **MDHCDST2** *gives a MdHCDS D_{hd} of trees.*

Proof. We have noticed in Algorithm **MDHCDST2** that if $H \leq d$ then, using the result of Lemma 3.12, we find $D_{hd} = \{v_l^*\}$, where l is only one arbitrary element of $\{0, 1, 2, 3, \ldots, H\}$. So, in that case, D_{hd} is MdHDS. If $H > d$, we select first member of D_{hd} as v_d^*, by Corollary 3.6, which dominates $|\cup_{r=0}^{d} M_r|$ vertices & $d(v_{H-d}^*, z) = d$, and choose second element of D_{hd} as v_{H-d}^* (by Corollary 3.9) because $d(v_{H-d}^*, z) = d$, $\forall\ z \in M_H$. In Step 4, we do 2-tuple weight assignment of the vertices of the sub-trees $T_s(v_{d+1}^*), T_s(v_{d+2}^*), \ldots, T_s(v_{H-d-1}^*)$ rooted at, respectively, $v_{d+1}^*, v_{d+2}^*, \ldots, v_{H-d-1}^*$. These roots are consecutively adjacent and lie on the central path on $T(v)$. So these are essential for making connectedness of D_{hd}. Besides these, we select some vertices whose second weight components are greater or equal to d (Lemma 3.10). During this selection process, we always keep in mind that selected members cover the maximum number of nodes of V and every node of V is located within d distances from at least one element of D_{hd}. So, D_{hd} is MdHDS. Now, we have to prove that D_{hd} is connected. Since some members of D_{hd} stay consecutively on the central path of BFS-tree $T(v)$, so they form a sub-central path, so they are connected. The remaining members of D_{hd} which are imitated from some vertices lie on the central-path, they are connected with the previous sub-central path (Lemma 3.10). Hence, D_{hd} is connected. Therefore, D_{hd} is a MdHCDS of trees. □

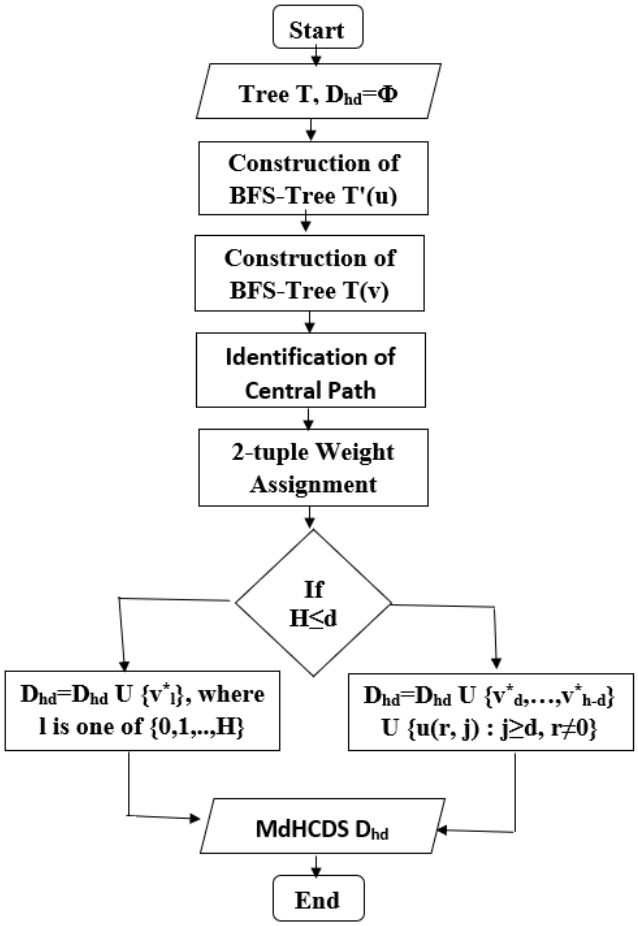

Figure 4. Flowchart of Algorithm MDHCDST2.

Theorem 4.4. *The Algorithm* **MDHCDST2** *runs in* $O(n)$ *time, where* $n = |V|$.

Proof. Step 1 takes $O(n)$ time for constructing the BFS-tree $T'(u)$. Similarly $O(n)$ time is required for building the BFS-tree $T(v)$ in Step 2. In Step 3, the vertices on the central path can be identified in $O(n)$ time, as there is only $H + 1 \leq n$ vertices in the central path. For 2-tuple weight assignment of the vertices of $V(T_s(v_r^*), r = d + 1, d + 2, \ldots, H - d - 1$, $O(n)$ time is needed, in Step 4, as $V(T_s(v_r^*)), r = d+1, d+2, \ldots, H-d-1$ are mutually disjoint. If $H \leq d$, then we can compute D_{hd} in constant time as $|D_{hd}| = 1$. On the other hand if $H > d$, the time needed to find the members of D_{hd} is $O(n)$. Therefore, overall time complexity of Algorithm **MDHCDST2** is $O(n)$ time. □

5. Conclusion

Domination is always an attractive and crucial problem to the researchers who work in graph theory. Within the various types of dominations, we can apply connected domination in many practical-life problems, and many researchers have done much research on this in the past. Nowadays, in advanced

graph theory, minimum d-hop connected domination problem is a momentous research topic because it can be applied in communication networks, such as wireless mobile networks.

In this paper, we propose two algorithms for the determination of an MdHCDS of trees. In future, we have a plan to design an $O(n)$ time algorithm to determine an MdHCDS of the social network graph, unit-disk graph, circular-arc graph or other intersection graphs.

Acknowledgment: We would like to thank the anonymous reviewers and editors for their careful reading of our manuscript and their valuable comments and suggestions to improve our paper. We would also like to express our deep and sincere gratitude to our colleagues and friends for encouraging us to complete this research work and giving continuous support throughout this research.

References

[1] A. S. Adhya, S. Mondal, S. C. Barman, An optimal algorithm to find minimum k-hop connected dominating set of permutation graphs, Asian-Eur. J. Math. 14(4) (2021) 2150049–2150061.
[2] B. Awerbuch, D. Peleg, Sparse partitions (extended abstract), In 31st Annual Symposium on Foundations of Computer Science, IEEE Comput. Soc. Press (1990) 503–513.
[3] S. K. Ayyaswamy, B. Krishnakumari, C. Natarajan, Y. B. Venkatakrishnan, Bounds on the hop domination number of a tree, Proc Math Sci. 125(4) (2015) 449–455.
[4] S. C. Barman, M. Pal, S. Mondal, An optimal algorithm to find minimum k-hop dominating set of interval graphs, Discrete Math Algorithms Appl. 11(2) (2019) 1950016–1950034.
[5] P. Basuchowdhuri, S. Majumder, Finding influential nodes in social networks using minimum k-hop connected dominating set, Proceedings of International Conference on Applied Algorithms, 8321 (2014) 137–151.
[6] D. A. Benson, M. S. Boguski, D. J. Lipman, J. Ostell, B. F. Ouellette, B. A. Rapp, D. L. Wheeler, GenBank, Nucleic Acids Res. 27(1) (1999) 12–17.
[7] S. Berchtold, D. Keim, H. P. Kriegel, The X-tree: an index structure for high-dimensional data, Proceedings of the 22nd Conference on Very Large Databases (1996) 28–39.
[8] C. Berge, The theory of graphs and its applications, Translated by Alison Doig, Methuen & Co., Ltd., London (1962).
[9] A. Cayley, On the theory of the analytical forms called trees, Lond. Edinb. Dublin philos. mag. j. sci. 13(85) (1857) 172–176.
[10] G. J. Chang, Algorithmic aspect of domination in graphs, Handbook of Combinatorial Optimization, Second Edition, Springer, New York (2013) 339–405.
[11] H. S. Chao, F. R. Hsu, R. C. T. Lee, An optimal algorithm for finding the minimum cardinality dominating set on permutation graphs, Discrete Appl. Math. 102(3) (2000) 159–173.
[12] C. C. Y. Chen, S. K. Das, Breadth-first traversal of trees and integer sorting in parallel, Inf. Process. Lett. 41(1) (1992) 39–49.
[13] E. J. Cockayne, R. M. Dawes, S. T. Hedetniemi, Total domination in graphs, Networks 10(3) (2006) 211–219.
[14] D. Cokuslu, K. Erciyes, A hierarchical connected dominating set based clustering algorithm for mobile ad hoc networks, Proceedings of 15th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (2007) 60–66.
[15] A. D'Atri, M. Moscarini, Distance-hereditary graphs, Steiner trees, and connected domination, SIAM J. Comput. 17(3) (1988) 521–538.
[16] E. D. Demaine, M. Hajjaghay, D. M. Thilikos, Fixed parameter algorithm for (k, r) center in planar graphs and map graphs, ACM Trans. Algorithms 1(1) (2003) 33–47.
[17] P. Dorbec, M. A. Henning, M. Montassier, J. Southey, Independent domination in cubic graphs, J

Graph Theory 80(4) (2015) 329–349.
[18] O. Favaron, M. A. Henning, C. M. Mynhart, J. Puech, Total domination in graphs with minimum degree three, J Graph Theory 34(1) (2000) 9–19.
[19] X. Gao, W. Wu, X. Zhang, X. Li, A constant-factor approximation for d-hop connected dominating sets in unit disk graph, Int. J. Sens. Netw. 12(3) (2012) 125–136.
[20] F. Harary, Graph theory, Addison-Wesley, Massachusetts (1969).
[21] T. W. Haynes, S. T. Hedetniemi, P. J. Slater, Fundamentals of domination in graphs: selected topics, Marcel Dekker, New York (1998).
[22] T. W. Haynes, S. T. Hedetniemi, P. J. Slater, Domination in graphs: advanced topics, Marcel Dekker, New York (1998).
[23] S. T. Hedetniemi, R. Laskar, Topics on domination, North-Holland, Amsterdam (1991).
[24] S. Kundu, S. Majumder, A linear time algorithm for optimal k-hop connected dominating set of a tree, Inf. Process. Lett. 116(2) (2016) 197–202.
[25] B. D. McKay, On the spectral characterisation of trees, Ars Combin. 3 (1977) 219–232.
[26] C. Natarajan, S. K. Ayyaswamy, Hop domination in graphs-II, An. Stt. Univ. Ovidius Constanta 23(2) (2015) 187–199.
[27] T. N. Nguyen, D. T. Huynh, Connected d-hop dominating sets in mobile ad hoc networks, 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (2006) 1–8.
[28] O. Ore, Theory of graphs, Amer. Math. Soc. Colloq. Pub., Providence, RI, 38 (1967).
[29] A. Rana, A. Pal, M. Pal, An efficient algorithm to solve the distance k-domination problem on permutation graphs, J. Discret. Math. Sci. Cryptogr. 19(2) (2016) 241–255.
[30] M. Q. Rieck, S. Pai, S. Dhar, Distributed routing algorithms for multi-hop ad hoc networks using d-hop connected d-dominating sets, Comput. Netw. 47(6) (2005) 785–799.
[31] E. Sampathkumar, H. B. Walikar, The connected domination number of a graph, J. Matm. Phys. Sci. 13(6) (1979) 607–613.
[32] R. S. Shaheen, Bounds for the 2-domination number of toroidal grid graphs, Int. J. Comput. Math. 86(4) (2009) 584–599.
[33] P. J. Slater, R-domination in graphs, J. ACM 23(3) (1976) 446–450.
[34] D. P. Sumner and P. Blitch, Domination critical graphs, J. Comb. Theory. Ser. B, 34(1) (1983) 65–76.
[35] L. K. Swift, T. Johnson, P. E. Livadas, Parallel creation of linear octrees from quadtree slices, Parallel Process. Lett. 4(4) (1994) 447–453.
[36] L. C. Van Der Merwe, C. M. Mynhardt, T. W. Haynes, Criticality index of total domination, Proceedings of the Twenty-ninth Southeastern International Conference on Combinatorics, Graph Theory and Computing, Congr. Numer. 131 (1998) 67–73.
[37] L. C. Van Der Merwe, C. M. Mynhardt, T. W. Haynes, Total domination edge critical graphs, Utilitas Math. 54 (1998) 229–240.
[38] L. C. Van Der Merwe, C. M. Mynhardt, T. W. Haynes, Total domination edge critical graphs with maximum diameter, Discuss. Math. Graph Theory 21 (2001) 187–205.
[39] T. H. P. Vuong, D.T. Huynh, Adapting d-hop dominating sets to topology changes in ad hoc networks, 9th International Conference on Computer Communications and Networks (2000) 348–353.
[40] A. K. Yadav, R. S. Yadav, R. Singh, A. K. Singh, Connected dominating set for wireless ad hok networks: a survey, Int. J. Eng. Syst. Model. Simul. 7(1) (2015) 22–34.
[41] S. Yau, W. Gao, Multi-hop clustering based on neighborhood benchmark in mobile ad-hoc networks, Mob. Netw. Appl. 12 (2007) 381–391.
[42] J. Yu, N. Wang, G. Wang, D. Yu, Connected dominating sets in wireless ad hok and sensor networks-a comprehensive survey, Comput. Commun. 36(2) (2013) 121–134.