Fully leafed induced subtrees

Alexandre Blondin Massé1 Julien de Carufel2 Alain Goupil2
Mélodie Lapointe1 Émile Nadeau1 Élise Vandomme1

1 Laboratoire de Combinatoire et d’Informatique Mathématique, Université du Québec à Montréal, Canada
2 Laboratoire Interdisciplinaire de Recherche en Imagerie et en Combinatoire, Université du Québec à Trois-Rivières, Canada

received 2017-09-XX, revised XXXX-XX-XX, accepted XXXX-XX-XX.

Let $G$ be a simple graph of $n$ vertices. We consider the problem $IS_{i,\ell}$ of deciding whether there exists an induced subtree with exactly $i \leq n$ vertices and $\ell$ leaves in $G$. We also study the associated optimization problem, that consists in computing the maximal number of leaves, denoted by $L_G(i)$, realized by an induced subtree with $i$ vertices, for $2 \leq i \leq n$. We compute the values of the map $L_G$ for some classical families of graphs and in particular for the $d$-dimensional hypercubic graphs $Q_d$, $d \leq 6$. Then we prove that the $IS_{i,\ell}$ problem is in general NP-complete. We also describe a nontrivial branch and bound algorithm that computes the function $L_G$ for any simple graph $G$. In the special case where $G$ is a tree, we provide a $O(n^3\delta)$ time and $O(n^2)$ space algorithm, where $\delta$ is the maximum degree of $G$. Finally, we exhibit a bijection between the set of discrete derivative of the sequences $L_G(i)_{3 \leq i \leq |G|}$, where $G$ is a caterpillar, and the set of prefix normal words.

Keywords: induced subtrees, leaf, optimization problem, prefix normal words

1 Introduction

Several decision and optimization problems concerning subgraphs, and in particular induced subgraphs, of a given simple graph have been studied in the past decades [NOdM12]. For instance, special cases according to whether the induced subgraphs are complete graphs [Rob01, TT77], chain graphs (see [AP16] and references therein) have been considered.

In 1984, Payan et al. [PTX84] discussed the maximum number of leaves, called the leaf number, that can be realized by a spanning tree of a given graph. This problem, called the maximum leaf spanning tree problem (MLST), is known to be NP-complete even in the case of regular graphs of degree 4 [GJ79], but some exact values were obtained for particular classes of graphs, as well as lower and upper bounds in other cases (see [KW91] and references therein). Spanning trees with many leaves are of interest in telecommunication networks, where they naturally optimize the energy transfer [BCL05].

An interesting variation of the MLST problem consists in replacing spanning trees by induced subtrees. The maximal number of leaves in induced subtrees has been recently investigated [BMdCGS17]. More
precisely, the authors give explicit formulas for the maximal number of leaves in any tree-like polyomino and polycube of given size. Since the underlying graph of a polyomino (resp. polycube) is an induced subgraph of the infinite square lattice (resp. cubic lattice), this result can be reformulated as the determination of the maximal number of leaves in an induced subtree of \((\mathbb{Z}^d, A_d)\) with \(i\) vertices for \(d = 2, 3\), where \(A_d = \{(p, p') \in \mathbb{Z}^d \times \mathbb{Z}^d \mid \operatorname{dist}(p, p') = 1\}\) and \(\operatorname{dist}\) is the Euclidean distance.

Induced subtrees have also been considered by Erdős et al., who showed in 1986 that the problem \(\text{IS}^\geq i\) of finding an induced subtree of a given graph \(G\) with more than \(i\) vertices is NP-complete \cite{ESS86}. In the data mining community, the detection of subgraph patterns, in particular of induced subtrees are used in information retrieval \cite{Zak02}. This requires efficient algorithms for the enumeration of induced subtrees. For instance, Wasa et al. \cite{WUA14} proposed an efficient parametrized algorithm for the generation of induced subtrees in a graph.

In this paper, we consider the following decision problem and its associated optimization problem.

**Problem 1.1** \((\text{IS}^\geq i)_1\). Given a simple graph \(G\) and two positive integers \(i\) and \(\ell\), does there exist an induced subtree of \(G\) with \(i\) vertices and \(\ell\) leaves?

**Problem 1.2** \((\text{MLIS})\). Given a simple graph \(G\), what is the maximum number \(L_G(i)\) of leaves that can be realized by an induced subtree of \(G\) with \(i\) vertices, for \(i \in \{2, \ldots, |G|\}\)?

Given a graph \(G\) and an induced subtree \(T\) of \(G\) having \(i\), we say that \(T\) is fully leafed if its number of leaves is exactly \(L_G(i)\). Examples of fully leafed induced subtrees are given in Figure 1.

The manuscript is organized as follows. Basic notions are recalled in Section 2 and a proof of the NP-completeness of the decision problem \((\text{IS}^\geq i)_1\) is given. We also study the function \(L_G\) in classical families of graphs. We describe a general branch and bound algorithm to compute \(\text{MLIS}\) in Section 3. In Section 4 we extend a polynomial algorithm to compute the function \(L_G\) when \(G\) is a tree so that the problem \(\text{MLIS}\) is in the class \(P\) for the particular case of trees. Section 5 is devoted to the relationship between the family of caterpillar graphs and prefix normal words. More precisely, we investigate the discrete derivative sequence \(\Delta L_T\) of the leaf function \(L_T\) of a tree \(T\) which is a binary word over the alphabet \(\{0, 1\}\). We prove that for caterpillar graphs the set \(\{\Delta L_C : C\text{ is a caterpillar}\}\) is precisely the set of prefix normal words. We then conclude with some perspectives on future work in Section 6.

## 2 Fully leafed induced subtrees

We refer the reader to \cite{Die10} for a basic introduction to graph theory, but for the sake of setting the notation clearly, we recall several notions. All graphs considered in this text are simple and undirected unless stated otherwise. Let \(G = (V, E)\) be a graph with vertex set \(V\) and edge set \(E\). Given two vertices \(u\) and \(v\) of \(G\), we denote by \(\operatorname{dist}(u, v)\) the distance between \(u\) and \(v\), that is the number of edges in a shortest chain between \(u\) and \(v\). The degree of a vertex \(u\) is the number of vertices that are at distance 1 from \(u\) and is denoted by \(\operatorname{deg}(u)\). We denote by \(n_i(G)\) the number of vertices of degree \(i\) in \(G\) and by \(n(G) = |V|\) the total number of vertices of \(G\) which is called the size of \(G\).

Let \(T = (V, E)\) be a tree, that is a connected and acyclic graph. A vertex \(u \in V\) is called a leaf of \(T\) when \(\operatorname{deg}(u) = 1\). Let \(V_1\) be the set of all leaves of \(T\). We say that \(T\) is a caterpillar if the induced subgraph \(T[V \setminus V_1]\), called its spine, is a chain, i.e. all leaves of \(T\) are adjacent to a single central chain of \(T\). Caterpillars are the focus of Section 5.

For \(U \subseteq V\), the subgraph of \(G\) induced by \(U\), denoted by \(G[U]\), is the graph \(G[U] = (U, E \cap P_2(U))\). An induced subtree of \(G\) by \(U\) is an induced subgraph that is a tree.
Fig. 1: Fully leafed induced subtrees in different graphs. (a) In a finite graph (the subtree of $i = 11$ vertices appears in black). (b) In the cubic grid, the leaves are omitted for sake of visibility. (c) In the square grid. (d) In the hexagonal grid. (e) In the triangular grid. The color of each cell indicates its degree: blue for degree 4, red for degree 3, yellow for degree 2 and green for degree 1.
The next definitions and notation are useful in the study of $\text{IS}_i^\ell$ and MLIS problems.

**Definition 2.1 (Leaf function).** Given a finite or infinite graph $G = (V, E)$, let $T_G(i)$ be the family of all induced subtrees of $G$ with exactly $i$ vertices. The leaf function of $G$, denoted by $L_G$, is the function with domain $\{0, 1, 2, \ldots, n(G)\}$ defined by $L_G(0) = 0, L_G(1) = 1$ and, for $i \geq 2$,

$$L_G(i) = \max\{n_1(T) \mid T \in T_G(i)\}.$$

As is customary, we set $\max\emptyset = -\infty$. An induced subtree $T$ of $G$ with $i$ vertices is called fully leafed when $n_1(T) = L_G(i)$.

**Remark 2.1.** Since a single vertex is not a leaf, in principle, we should have $L_G(1) = 0$. However, for conveniency, we set $L_G(1) = 1$. Also, it is worth mentioning that we always have $L_G(2) = 2$ in any graph $G$ with at least one edge.

The following observations are immediate.

**Proposition 2.1.** Let $G$ be a graph with $n \geq 3$ vertices. If $G$ is non-isomorphic to $K_n$, the complete graph on $n$ vertices, then $L_G(3) = 2$.

**Proposition 2.2.** For any simple graph $G$ with at least 3 vertices, the sequence $(L_G(i))_{i=3, \ldots, n(G)}$ is non-decreasing if and only if $G$ is a tree.

**Proof:** If $G$ is a tree, then $L_G(i)$ cannot be decreasing because if a subtree $T_1$ of $G$ contains a subtree $T_2$ then $n_1(T_1) \geq n_1(T_2)$. If $G$ is not a tree, then either it contains a cycle or it is not connected. In both cases, $G$ has no subtree with $n(G)$ vertices. Therefore $L_G(n(G)) = -\infty$ and $L_G(2) = 2$ which implies that there exists a decreasing step in the sequence $L_G(i)$.

We now describe the complexity of solving the problem $\text{IS}_i^\ell$.

**Theorem 2.1.** The problem $\text{IS}_i^\ell$ of determining whether there exists an induced subtree with $i$ vertices and $\ell$ leaves in a given graph is NP-complete.

**Proof:** It is clear that $\text{IS}_i^\ell$ is in the class NP. To show that it is NP-complete, we reduce it to the problem $\text{INDUCED SUBTREE (IS)}^{>\ell}$: Given a graph $G$ and a positive integer $i$, does there exist an induced subtree of $G$ with strictly more than $i$ vertices? The problem $\text{IS}^{>\ell}$ was shown to be NP-complete by Erdős et al. in 1986 [ESS86].

Consider the map $f$ that associates to an instance $(G, i)$ of $\text{IS}^{>\ell}$ with $G = (V, E)$ the instance $(H, 2(i + 1), i + 1)$ of $\text{IS}_i^\ell$ such that the graph $H$ is obtained as a copy of $G$ and an additional copy $V'$ of $V$ with an edge between $v \in V$ and its corresponding vertex $v' \in V'$ for each $v \in V$. Clearly, the map $f$ is computable in polynomial time as the graph obtained has $2|V|$ vertices and $|E| + |V|$ edges. Figure 2 illustrates the construction on a particular graph.

If $(G, i)$ is a positive instance of $\text{IS}^{>\ell}$, i.e. an instance for which the answer is yes, then $f(G, i) = (H, 2(i + 1), i + 1)$ is a positive instance of $\text{IS}_i^\ell$. Indeed, assume that the graph $G$ has an induced subtree with strictly more than $i$ vertices. Then $G$ has in particular an induced subtree $T$ with $i + 1$ vertices. So, in $H$, the set containing the vertices of $T$ and the $i + 1$ corresponding vertices of $V'$ induces a subtree with $2(i + 1)$ vertices and exactly $i + 1$ leaves. Conversely, if the instance $f(G, i) = (H, 2(i + 1), i + 1)$ is a positive instance of $\text{IS}_i^\ell$, then $(G, i)$ is a positive instance of $\text{IS}^{>\ell}$. Indeed, assume that $H$ contains an induced subtree $T$ with $2(i + 1)$ vertices and $i + 1$ leaves. Observe that, in $H$, any set of $k$ vertices
of \( V \) and \( j \) vertices of \( V' \) induces a non-connected subgraph if \( k < j \). Hence, the induced subtree \( T \) has at most \( i + 1 \) vertices in \( V' \). In other words, \( T \) has at least \( i + 1 \) vertices in \( V \). So the vertices of \( T \) that belong to \( V \) induce a subtree in \( G \) that has at least \( i + 1 \) vertices.

Therefore, \( IS^{>1} \leq IS_i^j \) and \( IS_i^j \) is NP-complete.

Fig. 2: Illustration of the polynomial transformation \( f(G, i) = (H, 2(i + 1), i + 1) \)

We end this section by computing the function \( L_{G}(i) \) for well known families of graphs. First, we consider classical families of finite graphs. Proofs are omitted as they are straightforward.

Complete graphs \( K_n \). For the complete graph with \( n \) vertices,

\[
L_{K_n}(i) = \begin{cases} i, & \text{if } 0 \leq i \leq 2; \\ -\infty, & \text{if } 3 \leq i \leq n. \end{cases}
\]

since any induced subgraph of \( K_n \) with more than two vertices contains a cycle.

Cycles \( C_n \). For the cyclic graph \( C_n \) with \( n \) vertices, we have

\[
L_{C_n}(i) = \begin{cases} i, & \text{if } 0 \leq i \leq 2; \\ 2, & \text{if } 3 \leq i < n; \\ -\infty, & \text{if } i = n. \end{cases}
\]

Wheels \( W_n \). For the wheel \( W_n \) with \( n + 1 \) vertices,

\[
L_{W_n}(i) = \begin{cases} i, & \text{if } 0 \leq i \leq 2; \\ i - 1, & \text{if } 3 \leq i \leq \lfloor \frac{n}{2} \rfloor + 1; \\ 2, & \text{if } \lfloor \frac{n}{2} \rfloor + 2 \leq i \leq n - 1; \\ -\infty, & \text{if } n \leq i \leq n + 1. \end{cases}
\]

Complete bipartite graphs \( K_{p,q} \). For the complete bipartite graph \( K_{p,q} \) with \( p + q \) vertices,

\[
L_{K_{p,q}}(i) = \begin{cases} i, & \text{if } 0 \leq i \leq 2; \\ i - 1, & \text{if } 3 \leq i \leq \max(p, q) + 1; \\ -\infty, & \text{if } \max(p, q) + 1 < i \leq p + q. \end{cases}
\]
Hypercubes $Q_d$. For the hypercube graph $Q_d$ with $2^d$ vertices, the computation of $L_{Q_d}$ seems more involved. Using the branch and bound algorithm described in Section 3, we were only able to compute the values up to $d = 6$ (see Table 1).

\[
\begin{array}{cccccccccccccccccc}
\text{n} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 \\
L_{Q_2}(n) & 0 & 1 & 2 & 2 & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\
L_{Q_3}(n) & 0 & 1 & 2 & 3 & 3 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\
L_{Q_4}(n) & 0 & 1 & 2 & 3 & 3 & 4 & 5 & 4 & 5 & 6 & 6 & 6 & 7 & 7 & 7 & 7 & 8 & 8 \\
L_{Q_5}(n) & 0 & 1 & 2 & 2 & 3 & 4 & 5 & 6 & 5 & 6 & 7 & 8 & 8 & 9 & 9 & 10 & 11 & 11 \\
L_{Q_6}(n) & 0 & 1 & 2 & 2 & 2 & 3 & 4 & 5 & 6 & 5 & 6 & 7 & 8 & 8 & 9 & 9 & 10 & 11 \\
\end{array}
\]

Tab. 1: The leaf function $L_{Q_d}(i)$ for $2 \leq d \leq 6$. The symbol $*$ is used instead of $-\infty$ to gain some space.

Infinite planar lattices. Blondin Massé et al. have computed the map $L_{\text{Squ}}(i)$ on the regular square grid with respect to the 4-adjacency relation [BMdCGS17]:

\[
L_{\text{Squ}}(i) = \begin{cases}
2, & \text{if } i = 2, \\
1 + i, & \text{if } i = 3, 4, 5, \\
L_{\text{Squ}}(i - 4) + 2, & \text{if } i \geq 6.
\end{cases}
\]

A similar argument leads to the computation of $L_{\text{Hex}}(i)$ and $L_{\text{Tri}}(i)$ on the hexagonal and the triangular lattices.

\[
L_{\text{Hex}}(i) = \begin{cases}
2, & \text{if } i = 2, 3, 4, 5, 6, \\
L_{\text{Hex}}(i - 2) + 1, & \text{if } i \geq 4.
\end{cases}
\]

\[
L_{\text{Tri}}(i) = \begin{cases}
2, & \text{if } i = 2, 3, 4, 5, 6, \\
L_{\text{Tri}}(i - 2) + 1, & \text{if } i \geq 4.
\end{cases}
\]

In all three cases, the maps verify linear recurrences. Therefore, it is easy to see that their asymptotic growth is $i/2$. Notice that the functions $L_{\text{Hex}}$ and $L_{\text{Tri}}$ are identical.

The infinite cubic lattice. The authors of [BMdCGS17] also gave the maximal number of leaves $L_{Z^3}(i)$ in induced subgraphs with $i$ vertices of the discrete space $Z^3$. This also satisfies a linear recurrence

\[
L_{\text{Cub}}(i) = \begin{cases}
f(i) + 1, & \text{if } i = 6, 7, 13, 19, 25, \\
f(i), & \text{if } 2 \leq i \leq 40 \text{ and } i \neq 6, 7, 13, 19, 25, \\
f(i - 41) + 28, & \text{if } 41 \leq i \leq 84, \\
L_{\text{Cub}}(i - 41) + 28, & \text{if } i \geq 85.
\end{cases}
\]

where $f$ is the function defined by

\[
f(i) = \begin{cases}
\lfloor (2i + 2)/3 \rfloor, & \text{if } 0 \leq i \leq 11, \\
\lfloor (2i + 3)/3 \rfloor, & \text{if } 12 \leq i \leq 27, \\
\lfloor (2i + 4)/3 \rfloor, & \text{if } 28 \leq i \leq 40.
\end{cases}
\]

whose asymptotic growth of $28i/41$ is slightly greater than in the 2D cases.
3 Computing the leaf function of a graph

We now describe a branch and bound algorithm that computes the leaf function \( L_G(i) \) for an arbitrary simple graph \( G \). We propose an algorithm based on a data structure that we call an induced subtree configuration.

**Definition 3.1.** Let \( G = (V, E) \) be a simple graph and \( \Gamma = \{ \text{green}, \text{yellow}, \text{red}, \text{blue} \} \) be a set of colors. An induced subtree configuration of \( G \) is an ordered pair \( C = (c, H) \), where \( H \) is a stack of other induced subtree configurations of \( G \), called the history of \( C \), and \( c : V \to \{ \text{green}, \text{yellow}, \text{red}, \text{blue} \} \) is a map satisfying the following conditions:

(i) The subgraph induced by \( \{ v \in V \mid c(v) = \text{green} \} \) is a tree;

(ii) If \( c(u) = \text{green} \) and \( \{ u, v \} \in E \), then \( c(v) \in \{ \text{green}, \text{yellow}, \text{red} \} \).

(iii) If \( c(u) = \text{yellow} \) and \( U = \{ v \in V \mid c(v) = \text{green} \} \), then \( \text{Card}(U \cap N(u)) = 1 \), where \( N(u) \) denotes the set of neighbors of \( u \).

When the context is clear, \( C \) is simply called a configuration.

Roughly speaking, a configuration is an induced subtree enriched with information that allows one to generate other induced subtrees either by extending, by excluding or by backtracking. The colors assigned to the vertices can be interpreted as follow. The **green** vertices are the confirmed vertices to be included in a subtree. Since each **yellow** vertex is connected to exactly one **green** vertex, any of them can be safely added to the green subtree to create a new induced subtree. The **red** vertices are those that are excluded from any possible tree extension. A **red** vertex is excluded either because it is adjacent to more than one **green** vertex and its addition would create a cycle or because it is explicitly excluded for generation purposes, by calling the operation **EXCLUDE_VERTEX** which is defined below. At last, the **blue** vertices are available vertices that have not been “seen” yet and which could be considered later. For reasons that are explained in the next paragraphs, it is convenient to save the configurations from which \( C \) was obtained in the stack \( H \).

Figure 3(a) illustrates an induced subtree configuration. The vertices and edges colored in **green** outline the induced subtree. The **yellow** vertices and edges show the possible extension of this tree. The vertices 14 and 15 are colored in **red** since they are both connected to two **green** vertices. Although the vertex 9 is colored in **red**, it would also have been possible to color it in **yellow** as well since it is connected to exactly one **green** vertex. Similarly, vertices 12, 13 and 16 could be colored either in **blue** or **red** since they are not adjacent to the tree.

Let \( C = (c, H) \) be a configuration of some simple graph \( G = (V, E) \), where \( c \) is a coloring of \( V \), and \( H \) is a stack of configurations. We considere the following operations acting on \( C \):

- **C.VERTEX_TO_ADD()** returns any non **green** vertex in \( G \) that can be safely colored in **green**. If no such vertex exists, it returns **none**. Note that the color of the returned vertex is always **yellow**, except when \( C = \emptyset \), where the color is **blue**.

- **C.ADD_TO_SUBTREE(v)** first pushes a copy of \( C \) on the top of \( H \), sets the color of \( v \) to **green** and updates the colors of the neighborhood of \( v \) accordingly.

- **C.EXCLUDE_VERTEX(v)** first pushes a copy of \( C \) on the top of \( H \) and then sets the color of \( v \) to **red**.
Fig. 3: Induced subtree configurations. The green edges outline the green subtree and the yellow ones outline the possible extensions, (a) The configuration $C$. (b) The configuration $C.\text{addToSubtree}(11)$.

- $C.\text{undoLastOperation}()$ changes $C$ back to the configuration on top of $H$ and then pop the top of $H$. In other words, it cancels the last operation applied on $C$, which is either an inclusion or an exclusion.

To illustrate these operations, let $C$ be the configuration of Figure 3(a). Then $C.\text{vertexToAdd}()$ could return one of the yellow vertices 7, 8, 10 or 11. Let $C'$ be the configuration obtained from $C$ after calling $C.\text{addToSubtree}(11)$. Then we have to update the colors of vertices 10, 11 and 12 by setting $c(11) \leftarrow \text{green}, c(10) \leftarrow \text{red}$ and $c(12) \leftarrow \text{yellow}$, as illustrated in Figure 3(b).

For optimization purposes, it is worth mentioning that it is not necessary to keep a complete copy of the configurations when saving them in the history $H$. It suffices to keep track of the vertex $v$ on which the operation was performed, the type of the operation (either an inclusion or an exclusion), together with the local color changes performed on the neighborhood of the vertex $v$. Keeping this optimization in mind, it is easy to show that the operations $\text{addToSubtree}(v)$ and $\text{undoLastOperation}()$, in the case where the last operation is an inclusion of a vertex $v$, are done in $O(\deg(v))$ time. Also, the operations $\text{excludeVertex}(v)$ and $\text{undoLastOperation}()$, in the case where the last operation is an exclusion of a vertex $v$, are done in $O(1)$. To achieve this complexity, it is sufficient to store the degree of green vertices, the vertex which caused a vertex to become red, and the number of green and red vertices, as well as the current number of leaves.

It is quite straightforward to use configurations for the generation of all induced subtrees of a graph $G$. Starting with the empty configuration $C = (c, \emptyset)$, where $c$ is the function that maps each vertex to the color blue, it is sufficient to recursively build configurations by branching is included or excluded from the current green tree.

While iterating over all possible configurations, if we want to compute the leaf function $L_G$, it is obvious that some configurations should be discarded whenever they cannot extend to interesting configurations. Given an induced subtree configuration of $n$ green vertices, we define for $n' \geq n$ the function
Fully leafed induced subtrees

$C$.LEAFPOTENTIAL($n'$), which computes an upper bound on the number of leaves that can be reached by adding $n' - n$ vertices to the current configuration $C$. To compute this upper bound we imagine an optimistic scenario in which all available and close enough yellow and blue vertices can safely be colored in green without creating a cycle, whatever the order. Keeping this idea in mind, we start by partitioning the yellow and blue vertices according to their distance from the green vertices. While computing this distance, red vertices are treated as if they had been removed from the graph.

Algorithm 1 Computation of the leaf potential for $n'$

1: function LEAFPOTENTIAL($C$ : configuration, $n'$ : natural)
2:  $n$ ← number of green vertices
3:  $\ell$ ← number of leaves in the green subtree
4:  $k$ ← number of yellow vertices adjacent to an inner green subtree vertex
5:  if $n + k \geq n'$ then
6:      return $\ell + (n' - n)$
7:  else
8:      $(n, \ell) \leftarrow (n + k, \ell + k)$
9:  end if
10:  $d \leftarrow 1$
11:  while $n < n'$ and there are available vertices at distance at most $d$ from the green vertices do
12:     Let $v$ be a highest degree available vertex $\triangleright$ The degree does not count red vertices
13:     if $n + \deg(v) - 1 \leq n'$ then
14:        $(n, \ell) \leftarrow (n + \deg(v) - 1, \ell + \deg(v) - 2)$
15:     else
16:        return $\ell + (n' - n) - 1$
17:     end if
18:     Remove $v$ from accessible vertices
19:     $d \leftarrow d + 1$
20:  end while
21:  return $\ell$
22: end function

These ideas are summarized in Algorithm 1 which computes an upper bound on the number of leaves that can be realized from a configuration of $n$ green vertices extended to a configuration of $n'$ green vertices. The first part of Algorithm 1 consists in completing the green subtree. More precisely, a configuration $C$ is called complete if each yellow vertex is adjacent to a leaf of the green tree. We first check if $C$ is complete and, if it is not the case, we increase $n$ and $\ell$ as if we were completing the green subtree (Lines 5–9). Next, we choose a vertex $v$ among all available vertices at distance $d$, and we update $n$ and $\ell$ as if we were adding $v$ to $C$ as well as all of its neighbors, which would only be leaves (Lines 13–17). This process is repeated until the size of the “optimistic subtree” reaches $n'$. We now prove that Algorithm 1 yields an upper bound on the maximum number of leaves that can be realized. It is worth mentioning that, in order to obtain a nontrivial bound, we restrict the available vertices to those that are not too far from the current green subtree, by increasing $d$ at each iteration.

Proposition 3.1. Algorithm 1 returns an upper bound on the number of leaves that can be realized starting
with a given configuration $C$. More precisely, any extension of $C$ to a configuration of $n'$ vertices has at most $C.\text{LEAF}\text{POTENTIAL}(n')$ leaves.

**Proof:** Let $C$ be a configuration whose *green* subtree has size $n$ and

$$Y_1 = \{\text{yellow vertices of } C \text{ at distance 1 of the inner vertices of the *green* tree}\}.$$

Let $p' = C.\text{LEAF}\text{POTENTIAL}(n')$ and suppose that $C$ can be extended to a configuration $C'$ with $n'$ *green* vertices and $\ell'$ leaves, with $\ell' > p'$. Let $v_1, v_2, ..., v_k$ be, in order, the vertices that became inner vertices in $C$ to reach $C'$ and $v'_1, v'_2, ..., v'_{k'}$ be the vertices chosen by the procedure $\text{LEAF}\text{POTENTIAL}(n')$. Clearly, $k \leq k'$, otherwise $\ell'$ cannot be greater than $p'$. Without loss of generality, we can assume that if $v_i$ and $v_j$ are at the same distance from the *green* subtree and $\deg(v_j) \leq \deg(v_i)$ then $i \leq j$ (otherwise, we simply swap any pair of vertex $v_i$ and $v_j$ not satisfying this condition). Moreover, we know that $v_i$ is at most at distance $i$. This implies

$$\deg(v_i) \leq \deg(v'_1), \deg(v_1) \leq \deg(v'_i), ..., \deg(v_k) \leq \deg(v'_{k'}).$$

At most all vertices of $Y_1$ can be included so that the configuration has more leaves than the previous one without adding inner vertices. Then, for each inner vertex included, at most all its neighbors can be included without adding an inner vertices. Similarly, including $v_i$ as an inner vertex implies that at most $\deg(v_i) - 1$ leaves are gained. Therefore,

$$\ell \leq |Y_1| + \sum_{i=1}^{k} (\deg(v_i) - 1) - k < |Y_1| + \sum_{i=1}^{k'} (\deg(v'_i) - 2) = \ell'$$

which is a contradiction, showing that such a configuration $C'$ cannot exist. \hfill \Box

It follows from Proposition [3.1] that a configuration $C$ of $n$ *green* vertices and $n_r$ *red* vertices cannot be extended to a subtree with more leaves than prescribed by the current leaf function $L$ if

$$C.\text{LEAF}\text{POTENTIAL}(n') < L(n')$$

for all $n \leq n' \leq n + |G| - n_r$. \hfill (1)

We conclude this section by presenting Algorithm 2 which computes the function $L$ for an arbitrary simple graph $G$. The idea consists simply in enumerating all possible configurations, discarding those that cannot be extended to fully leaved induced subtrees.

Based on Proposition [3.1] and the previous discussion, the following result is immediate.

**Theorem 3.1.** Algorithm 2 returns the leaf function $L_G$ of $G$.

## 4 Fully leaved induced subtrees of trees

It turns out that the MLIS problem can be solved in polynomial time when restricted to the class of trees. Hereafter, we describe in details an algorithm with polynomial time complexity based on the dynamic programming paradigm.

Before going further, we need additional definitions. A *rooted tree* is a couple $\hat{T} = (T, u)$ where $T = (V, E)$ is a tree and $u \in V$ is a distinguished vertex called the *root* of $\hat{T}$. Rooted trees have a natural
Fully leafed induced subtrees

Algorithm 2 Leaf function computation

1: function LEAFFUNCTION(G: graph): function
2:     function EXPLORECONFIGURATION() 
3:         u ← C.VERTEXTOADD() 
4:         if u = none then 
5:             i ← the number of green vertices in C 
6:             ℓ ← the number of leaves in C 
7:             L[i] ← max(L[i], ℓ) 
8:         else if Condition (1) is not satisfied then 
9:             C.ADDTOSUBTREE(u) 
10:             EXPLORECONFIGURATION() 
11:         end if 
12:     end function 
13:     C ← ∅ (the empty configuration) 
14:     L[i] ← 0 for i = 0, 1, ..., n(G) 
15:     EXPLORECONFIGURATION() 
16:     return L 
17: end function 

orientation with arcs pointing away from the root. A leaf of a rooted tree is therefore a vertex v with outdegree \( \text{deg}^+(v) = 0 \). In particular, if a rooted tree consists of a single vertex, then this vertex is a leaf. The functions \( n(\hat{T}) \) and \( n_1(\hat{T}) \) are defined accordingly by

\[
n(\hat{T}) = n(T) \quad \text{and} \quad n_1(\hat{T}) = \text{Card}\left( \{ v \in T \mid \hat{T} = (T, u) \text{ and } \text{deg}^+(v) = 0 \} \right).
\]

Similarly, a rooted forest \( \hat{F} \) is a collection of rooted trees. It follows naturally that

\[
n_1(\hat{F}) = \sum_{\hat{T} \in \hat{F}} n_1(\hat{T}).
\]

Let \( \hat{T} \) be any rooted tree with \( n \) vertices and \( L_{\hat{T}} : \{0, 1, \ldots, n\} \to \mathbb{N} \) be defined by

\[
L_{\hat{T}}(i) = \max\{n_1(\hat{T}') : \hat{T}' \preceq \hat{T} \text{ and } n(\hat{T}') = i\},
\]

where \( \preceq \) denotes the relation “being a rooted subtree with the same root”. Roughly speaking, \( L_{\hat{T}}(i) \) denotes the maximum number of leaves that can be realized by some rooted subtree of size \( i \) of \( \hat{T} \). This map is naturally extended to rooted forests. Let \( \hat{F} = \{\hat{T}_1, \ldots, \hat{T}_k\} \) be a rooted forest and set

\[
L_{\hat{F}}(i) = \max\left\{ \sum_{j=1}^k n_1(\hat{T}_j') : \hat{T}_j' \preceq \hat{T}_j \text{ and } \sum_{j=1}^k n(\hat{T}_j') = i \right\}.
\]
Let $C(i, k)$ be the set of all weak compositions $\lambda = (\lambda_1, \ldots, \lambda_k)$ of $i$ in exactly $k$ non-negative parts. Then Equation (3) is equivalent to

$$L_\widehat{F}(i) = \max \left\{ \sum_{j=1}^{k} L_\widehat{T}_j(\lambda_j) \mid \lambda \in C(i, k) \right\}.$$  

(4)

Assuming that $L_\widehat{T}_j$ is known for $j = 1, 2, \ldots, k$, a naive computation of $L_\widehat{F}$ using Equation (4) is not done in polynomial time, since

$$|C(i, k)| = \binom{i + k - 1}{i}.$$  

The next lemma ensures that $L_\widehat{F}$ can be computed in polynomial time.

**Lemma 4.1.** Let $k \geq 1$ be an integer and $\widehat{F} = \{\widehat{T}_1, \ldots, \widehat{T}_k\}$ be a rooted forest with $n$ vertices. Then, for $i \in \{0, \ldots, n\}$,

$$L_\widehat{F}(i) = \begin{cases} L_{\widehat{T}_1}(i) & \text{if } k = 1, \\ \max\{L_{\widehat{T}_1}(j) + L_\widehat{F}(i-j) \mid \max\{0, i-n(\widehat{F}')\} \leq j \leq \min\{i, n(\widehat{T}_1)\}\} & \text{if } k \geq 2, \end{cases}$$

(5)

where $\widehat{F}' = \{\widehat{T}_2, \ldots, \widehat{T}_k\}$. Therefore, if $L_\widehat{T}_j$ is known for $j = 1, 2, \ldots, k$, then $L_\widehat{F}$ can be computed in $O(kn^2)$ time.

**Proof:** The first part follows from Equation (4) and the fact that, for $k \geq 2$, we have

$$C(i, k) = \{(j, \lambda_1, \lambda_2, \ldots, \lambda_{k-1}) \mid 0 \leq j \leq i, \lambda \in C(i-j, k-1)\}.$$

For the time complexity, one notices that for a given $i$, the recursive step of Equation (5) is applied $k - 1$ times, where each step is done in $O(n)$. Since $L_\widehat{F}(i)$ is computed for $i = 1, 2, \ldots, n$, the total time complexity is $O(kn^2)$.

Finally, we describe how one computes $L_\widehat{T}_j$ from its children.

**Lemma 4.2.** Let $\widehat{T}$ be some rooted tree with root $u$. Let $\widehat{F}$ be the rooted forest induced by the children of $u$. Then

$$L_\widehat{T}(i) = \begin{cases} i & \text{if } i = 0, 1, \\ L_\widehat{F}(i-1) & \text{if } 2 \leq i \leq n(\widehat{T}). \end{cases}$$

(6)

**Proof:** The cases $i = 0, 1$ are immediate. Assume that $i \geq 2$. Since any rooted subtree of $\widehat{T}$ must in particular include the root $u$ and since $u$ is not a leaf, all the leaves are in $\widehat{F}$ and the result follows.

Combining Lemmas 4.1 and 4.2, we obtain the following result.

**Theorem 4.1.** Let $T = (V, E)$ be a undirected tree with $n \geq 2$ vertices. Then $L_T$ can be computed in $O(n^3\delta)$ time and $O(n^2)$ space where $\delta$ denotes the maximal degree of a vertex in $T$. 

Fully leafed induced subtrees

Proof: For any edge \{u, v\} ∈ E of T, if \{u, v\} is removed from T, then the remaining graph is a set of two rooted trees: a tree \(\hat{T}(v \rightarrow u)\) rooted in u and a tree \(\hat{T}(u \rightarrow v)\) rooted in v. Using Lemmas 4.1 and 4.2, we compute the values of \(L_{\hat{T}(u \rightarrow v)}\) and \(L_{\hat{T}(v \rightarrow u)}\) for each edge \{u, v\}, making sure to store the results obtained recursively to avoid computing them twice. The overall time complexity is

\[
\sum_{\{u,v\}\in E} \left( \mathcal{O}(\deg(u)n^2) + \mathcal{O}(\deg(v)n^2) \right) = \mathcal{O}(n^3\delta),
\]

by Lemma 4.1 and the fact that |E| = n - 1.

Next, let the function \(L_{\{u,v\}}: \{0, 1, 2, \ldots, n\} \rightarrow \mathbb{N}\) be defined by

\[
L_{\{u,v\}}(i) = \max \left\{ L_{\hat{T}(u \rightarrow v)}(j) + L_{\hat{T}(v \rightarrow u)}(i - j) \mid 1, i - n(\hat{T}(v \rightarrow u)) \leq j \leq i - 1, n(\hat{T}(u \rightarrow v)) \right\},
\]

i.e. \(L_{\{u,v\}}(i)\) is the maximum number of leaves that can be realized by all subtrees of T containing the edge \{u, v\} and having i vertices. Clearly, \(L_{\{u,v\}}\) is computed in time \(\Theta(n)\) when the functions \(L_{\hat{T}(u \rightarrow v)}\) and \(L_{\hat{T}(v \rightarrow u)}\) have been computed. Hence, since any optimal subtree with \(i \geq 2\) vertices has at least one edge, the optimal value \(L_T(i)\) must be stored in at least one edge, so that

\[
L_T(i) = \begin{cases} 
\max \left\{ L_{\{u,v\}}(i) \mid \{u, v\} \in E \right\} & \text{if } i < n(T) \\
n(T) & \text{if } i = n(T) 
\end{cases}
\]

which is computed in \(\Theta(n)\) time as well. The global time complexity is therefore \(\mathcal{O}(n^3\delta)\), as claimed. Finally, the space complexity of \(\mathcal{O}(n^2)\) follows from the fact that each of the \(n-1\) edges stores information of size \(\mathcal{O}(n)\).

---

**Example 4.1.** Consider the tree depicted in Figure 4 without the orientation and with only one edge between u and v. By Theorem 4.1, the computation of \(L_T(i)\) with \(i \in \{2, \ldots, 14\}\) requires to first compute the function \(L_{\{x,y\}}\) for each edge \(\{x, y\} \in E\). Figure 4 illustrates the computation of \(L_{\{u,v\}}\) for the specific edge \{u, v\}. Indeed, the blue arc \(\{v, u\}\) stores the value of \(L_{\hat{T}(u \rightarrow v)}\). As \(L_{\hat{T}(u \rightarrow v)}\) is computed recursively on the subtrees rooted in the children, the other blue arcs hold intermediate values necessary for \(L_{\hat{T}(v \rightarrow u)}\). Similarly, the red edges hold the intermediate values of the recursive computation of \(L_{\hat{T}(u \rightarrow v)}\).
5 Caterpillars and prefix normal words

The leaf functions obtained in Section 2 suggest that they have a very specific structure. A natural question would be to find a way to describe or decide if, given some function $L$, there exists a graph $G$ whose leaf function is exactly $L$. Although we do not answer this general question, in this section, we provide interesting observations and, in particular, we exhibit a connection with a family of words named prefix normal.

For this purpose, we first start by recalling basic terminology about words. The reader is referred to Lothaire for a complete introduction [Loo97]. A word $w$ is a finite concatenation of letters, $w = w_1 w_2 \cdots w_n$, where the $w_i$’s are letters. The length of a word $w$, denoted by $|w|$, is the number $n$ of its letters. The alphabet $\text{Alph}(w)$ of a word $w$ is the set of letters occurring in $w$. The number of occurrences of the letter $a$ in $w$ is denoted by $|w|_a$. Two words $u$ and $v$ are called abelian equivalent if $|u|_a = |v|_a$ for all letters $a$. A word $u$ is a factor of $w$ if $w = p \cdot u \cdots s$ for some words $p, s$. If $p$ (respectively $s$) is the empty word, then $u$ is called a prefix (resp. suffix) of $w$. We denote by $\text{Pref}_i(w)$ (resp. $\text{Suff}_i(w)$) the unique prefix (resp. suffix) of $w$ of length $i$, and by $\text{Fac}(w)$ (respectively $\text{Pref}(w)$, $\text{Suff}(w)$) the set of all its factors (resp. prefixes, suffixes). A language is any set of words, either finite or infinite.

A first useful tool is the word of first differences.

**Definition 5.1** (Leaf word). Let $G$ be a simple graph of $n$ vertices and $L_G$ be its associated leaf function. The leaf word of $G$, denoted by $\Delta L_G$, is a word on the alphabet $\mathbb{N} \cup \{\omega\}$ defined by

$$\Delta L_G = (L_G(i + 1) - L_G(i))_{3 \leq i \leq n - 1},$$

where we set $L_G(i + 1) - L_G(i) = \omega$, whenever $L_G(i) = -\infty$ or $L_G(i + 1) = -\infty$.

**Remark 5.1.** Since $L_G(0) = 0$, $L_G(1) = 1$ and $L_G(2) = 2$ for any graph having at least one edge, we restrict the domain of $\Delta L_G$ to $i \geq 3$.

It follows directly from Definition 5.1 that

$$L_G(j) - L_G(i) = |\text{Suff}_{j-i}(\text{Pref}_{j-5}(\Delta L_G))|_1$$

for $2 \leq i \leq j \leq n$.

We are interested in sequences of integers corresponding to leaves in induced subtrees of graph.

**Definition 5.2** (Leaf sequence). Let $\ell$ be a finite sequence of positive integers. We say that $\ell$ is a leaf sequence if there exists some simple graph $G$ such that $\ell = (L_G(n))_{n=3,\ldots,n(G)}$.

A natural problem follows directly from Definition 5.2.

**Problem 5.1.** Given a sequence of positive integers $\ell$, is $\Delta \ell$ a leaf function?

As a first step, we describe the possible alphabets of a leaf word.

**Lemma 5.1.** Let $G$ be a simple graph of more than 3 vertices and $w$ its associated leaf word. Then $\text{Alph}(w) \subseteq \{1, 0, -1, -2, \ldots, \omega\}$.

**Proof:** Let $L_G$ be the leaf function of $G$. Obviously, $L_G(i + 1) - L_G(i)$ could be equal to 1, 0, $-1$ or $\omega$, as seen for instance in the leaf function $L_{Q_3}$ of the cube graph, computed in Section 2. Also, $L_G(i + 1) - L_G(i)$ may take any negative integer value: Figure 5 shows a family of graphs for which $-n$ appears in $\Delta L_G$ for any $n \in \mathbb{N}$. 
It remains to show that for any graph $G$, positive integers $k > 1$ cannot occur in $\Delta L_G$. Arguing by contradiction, assume that $G$ is a graph with $n$ vertices such that $\Delta L_G$ contains some letter $k \in \mathbb{N} - \{0, 1\}$. By Definition 5.1, we have that $L_G(i) - L_G(i + 1) = k$ for some $i$ smaller than $n$. This means that there exists an induced subtree $T$ of $G$ with $i + 1$ vertices and $L_G(i) + k$ leaves. Let $T'$ be the subgraph obtained by removing any leaf of $T$. Then $T'$ is an induced subtree with $i$ vertices and $L_G(i) + k - 1 > L_G(i)$ leaves, contradicting the definition of $L_G(i)$.

Fig. 5: A graph with leaf word $\Delta L = 1^n, -n$

Another simple yet useful observation is that the alphabet of the leaf word indicates whether the associated graph is a tree.

Lemma 5.2. Let $G$ be a simple graph with at least 3 vertices. Then $\text{Alph}(\Delta L_G) \subseteq \{0, 1\}$ if and only if $G$ is a tree.

Proof: Follows from Proposition 2.2 and Definition 5.2.

We were not able to provide a complete answer to Problem 5.1 for general graphs. However, when restricting our attention to caterpillar graphs, there is an interesting connection with the so-called prefix normal words. A binary word $u$ on the alphabet $\{0, 1\}$ is called prefix normal if for any prefix $p \in \text{Pref}(u)$ and any factor $f \in \text{Fac}(u)$, the condition $|p| = |f|$ implies $|p|_1 \geq |f|_1$. Prefix normal words were recently considered in [BFL+14, BFL+17].

We prove the following theorem.

Theorem 5.1. Let $L : \{0, 1, 2, \ldots, n\} \rightarrow \mathbb{N}$ be a function. Then $L$ is the leaf function of some caterpillar $C$ if and only if $\Delta L$ is a prefix normal binary word.

To prove Theorem 5.1, we need additional definitions and lemmas. As a first step, it is convenient to consider directed versions of caterpillars.

Definition 5.3. A caterpillar sequence $D = (d_1, \ldots, d_k)$ is a sequence of nonnegative integers such that $d_1, d_k \geq 1$ and $\sum_{i=1}^{k} d_i \geq 2$, where the number $k$ is called its length. Given another caterpillar sequence $D' = (d'_1, \ldots, d'_k)$ of length $k'$, we say that $D'$ is a caterpillar subsequence of $D$, and we write $D' \preceq D$, if there exists a positive integer $i$ such that $d'_j \leq d_{i+j}$ for $j = 1, 2, \ldots, k'$. The size of $D$ is given by

$$n(D) = k + \sum_{i=1}^{k} d_i$$
and its number of leaves is

\[ n_1(D) = \sum_{i=1}^{k} d_i. \]

The name “caterpillar sequence” comes from the fact that we can associate to each caterpillar sequence \((d_1, \ldots, d_k)\) a graph that is a caterpillar with spine \((v_1, \ldots, v_k)\) with \(d_i\) leaves attached to the vertex \(v_i\) for all \(i \in \{1, \ldots, k\}\). Therefore, the size of a caterpillar sequence is exactly the number of vertices of the associated graph. In particular, to each caterpillar sequence corresponds a unique caterpillar graph up to isomorphism. Conversely, to each caterpillar graph, we can associate at least one caterpillar sequence \(D\) by choosing arbitrarily an orientation of its spine, say \((v_1, \cdots, v_k)\), and by setting

\[ d_1 = \text{deg}(v_1) - 1, d_k = \text{deg}(v_k) - 1 \text{ and } d_i = \text{deg}(v_i) - 2 \forall i \in \{2, \ldots, k-1\}. \]

Caterpillar subsequences corresponds exactly with induced subtrees. Finally, given any caterpillar sequence \(D\), we denote by \(L_D\) the leaf function of any of its associated caterpillar graph.

Some caterpillar subsequences are of particular interest. Let \(C = (d_1, \ldots, d_k)\) be a caterpillar sequence of size \(n\). Given an integer \(i \in \{3, 4, \ldots, n\}\), the left caterpillar subsequence of size \(i\) of \(C\) is defined by

\[ \text{Left}_i(C) = \begin{cases} (2), & \text{if } i = 3; \\ (d_1, \ldots, d_a - 1, 1), & \text{if } \text{Left}_{i-1}(C) = (d_1, \ldots, d_a), \text{ with } a \leq k; \\ (d_1, \ldots, d_a' + 1), & \text{if } \text{Left}_{i-1}(C) = (d_1, \ldots, d_a'), \text{ with } a \leq k \text{ and } d_a' \leq d_a. \end{cases} \]  

Similarly, the right caterpillar subsequence of size \(i\) of \(C\) is given by

\[ \text{Right}_i(C) = \begin{cases} (2), & \text{if } i = 3; \\ (1, d_b - 1, \ldots, d_k), & \text{if } \text{Right}_{i-1}(C) = (d_b, \ldots, d_k), \text{ with } b \geq 1; \\ (d_b' + 1, \ldots, d_k), & \text{if } \text{Right}_{i-1}(C) = (d_b', \ldots, d_k), \text{ with } b \geq 1 \text{ and } d_b' \leq d_b. \end{cases} \]

It is easy to show by induction on \(i\) that \(\text{Left}_i(C) = (d_1, \ldots, d_a, \alpha)\), where \(\alpha\) and \(a\) are the unique integers satisfying \(\sum_{m=1}^{a}(d_m + 1) + \alpha = i\), and that \(\text{Right}_i(C) = (\beta, d_b, \ldots, d_k)\), where \(\beta\) and \(b\) are the unique integers satisfying \(\sum_{m=1}^{b}(d_m + 1) + \beta = i\).

Caterpillars can be built naturally by reading binary words.

**Definition 5.4.** We define the reading caterpillar \(C(w)\) of a binary word \(w\) recursively on \(|w|\) as follows. The reading caterpillar \(C(\epsilon)\) of the empty word is the caterpillar sequence given by the tuple \((2)\), i.e. it corresponds to a simple chain on three vertices. Let \(w = ua\) be a binary word with \(u \in \{0, 1\}^*\), \(a \in \{0, 1\}\) and \(C(u) = (d_1, \ldots, d_k)\) be the reading caterpillar of \(u\). Then the reading caterpillar of \(w\) is

\[ C(w) = \begin{cases} (d_1, \ldots, d_k - 1, 1), & \text{if } a = 0; \\ (d_1, \ldots, d_k + 1), & \text{if } a = 1. \end{cases} \]

The following observations are deduced directly from Definition 5.4.

**Lemma 5.3.** Let \(w\) be a binary word on \(\{0, 1\}\), \(a \in \{0, 1\}\) and \(3 \leq i \leq |w| + 3\). Then

(i) \(\text{Left}_i(C(w)) = C(\text{Pref}_{i-3}(w))\).
Lemma 5.4. Fully leafed induced subtrees

Proof: Let \( C(w) = (d_1, \ldots, d_k) \).

(i) Follows by induction on \( i \) and by Equations (8) and (10).

(ii) By Definition 5.4, on one hand, if \( a = 0 \), then

\[
    n_1(C(wa)) = \left( \sum_{j=1}^{k} d_i \right) - 1 + 1 + 2 = n_1(C(w)) = n_1(C(w)) + a
\]

On the other hand, if \( a = 1 \), then

\[
    n_1(C(wa)) = \left( \sum_{j=1}^{k} d_i \right) + 1 + 2 = n_1(C(w)) + 1 = n_1(C(w)) + a.
\]

(iii) Follows from (i).

(iv) Follows from (i) by replacing \( \text{Left}_i \) by \( \text{Right}_i \) and \( \text{Pref}_{i-3} \) by \( \text{Suff}_{i-3} \).

(v) This is a special case of (iv) with \( i = |w| + 3 \).

An interesting property of reading caterpillars is that their leaf function can be computed incrementally.

**Lemma 5.4.** Let \( w \) be a prefix normal word and \( 3 \leq i \leq |w| + 3 \). Then

\[
    L_{C(w)}(i) = n_1(\text{Left}_i(C(w))).
\]

**Proof:** Assume first that \( i = |w| + 3 \). Then \( L_{C(w)}(i) = n_1(\text{Left}_i(C(w))) \), since \( \text{Left}_i(C(w)) \) is the only caterpillar subsequence of \( C(w) \). Therefore, we can assume that \( i \leq |w| + 2 \). Moreover, it is immediate that \( L_{C(w)}(i) \geq n_1(\text{Left}_i(C(w))) \), since \( \text{Left}_i(C(w)) \) is a caterpillar subsequence of \( C(w) \), so that we only have to prove that \( L_{C(w)}(i) \leq n_1(\text{Left}_i(C(w))) \). We proceed by induction on \( |w| \).

**Basis.** If \( |w| = 0 \), then \( w = \varepsilon \) and \( i = 3 \), so that

\[
    L_{C(w)}(i) = L_{C(\varepsilon)}(3) = 2 = n_1(\text{Left}_3(C(\varepsilon))) = n_1(\text{Left}_i(C(w))) \leq n_1(\text{Left}_i(C(w))).
\]

**Induction.** Since \( |w| > 0 \), there exists a word \( w' \) and a letter \( a \) such that \( w = w'a \). Now, arguing by contradiction, assume that there exists a caterpillar subsequence \( D \) of \( C(w) \) of size \( i \) such that \( n_1(D) > n_1(\text{Left}_i(C(w))) \). If \( D \preceq C(w') \), then

\[
    n_1(D) > n_1(\text{Left}_i(C(w))) = n_1(\text{Left}_i(C(w'))) (\text{since } i < |w| + 3) = L_{C(w')}(i) \quad \text{(by the induction hypothesis),}
\]

contradicting the maximality of $L C_{C(w')}(i)$. Hence, $D \not\subseteq C(w')$.

Next, assume that $D = Right_i(C(w))$. Then we would have

$$|\text{Pref}_{i-3}(w)| + 2 = n_1(\text{Left}_{i}(C(w))) < n_1(D) = n_1(\text{Right}_i(C(w))) = |\text{Suff}_{i-3}(w)| + 2,$$

i.e. $|\text{Pref}_{i-3}(w)| < |\text{Suff}_{i-3}(w)|$, contradicting the assumption that $w$ is prefix normal.

It remains to consider the case where $D$ is neither a left or right subcaterpillar of $C(w)$. Let $C(w) = (c_1, c_2, \ldots, c_k)$ and $D = (d_1, d_2, \ldots, d_{k'})$. Since $D \not\subseteq C(w)$ but $D \not\supseteq C(w')$, we have $k' \leq k$ and $d_j \leq c_{j+k-k'}$ for $j = 1, 2, \ldots, k'$. Let $j$ be the largest index such that $d_j < c_{j+k-k'}$ (such an index $j$ exists since $D \not= \text{Right}_i(C(w))$) and let

$$D' = (d_1, d_2, \ldots, d_{j-1}, d_j + 1, d_{j+1}, \ldots, d_{k'} - 1).$$

Clearly, $D' \subseteq C(w')$, but $n_1(D') = n_1(D)$, so that a similar sequence of relations such as (12)$\rightarrow$(14) by substituting $D$ by $D'$ leads to another contradiction, concluding the proof. \hfill $\square$

As a consequence of Lemma 5.3, we have the first part of Theorem 5.1.

**Corollary 5.1.** Let $w$ be a prefix normal word. Then $\Delta L_{C(w)} = w$.

**Proof:** For $3 \leq i \leq |w| + 3$, we have

$\Delta L_{C(w)}(i) = L_{C(w)}(i + 1) - L_{C(w)}(i)
= n_1(\text{Left}_{i+1}(C(w))) - n_1(\text{Left}_{i}(C(w)))
= (|\text{Pref}_{i-2}(w)| + 2) - (|\text{Pref}_{i-3}(w)| + 2)
= w_{i-3},$

as claimed. \hfill $\square$

Clearly, not every caterpillar sequence can be obtained as a reading caterpillar. However, we now prove the last part of Theorem 5.1 which implies that reading caterpillars are good representatives when studying leaf sequences.

The following simple observation about non prefix normal words is key in proving the other implication.

**Proposition 5.1.** Let $w$ be a binary word on $\{0, 1\}$ that is not prefix normal. Then there exist two abelian equivalent words $u$ and $u'$, such that $u0 \in \text{Pref}(w)$ and $1u' \in \text{Fac}(w)$.

**Proof:** Let $w$ be a non prefix normal word. Since $w$ is non prefix normal, there exists at least a prefix $p$ and a factor $f$ having the same length such that $|p|_1 < |f|_1$. Without loss of generality, we can assume that $|p|$ and $|f|$ are as small as possible. Let $p = ua$ and $f = bu'$ for some letters $a, b$. Since $|p|$ and $|f|$ are minimal, we have $|u|_1 \geq |w|_1$. Therefore,

$$|u|_1 + a = |ua|_1 = |p|_1 < |f|_1 = |bu'|_1 = |u'|_1 + b \leq |u|_1 + b,$$

which can only be verified if $a = 0$ and $b = 1$. \hfill $\square$

In order to complete the proof, we introduce an operation called $\text{graft}$ acting on caterpillar sequences.
**Definition 5.5.** Let $C = (d_1, d_2, \ldots, d_k)$ and $C' = (d'_1, d'_2, \ldots, d'_l)$ be two caterpillar sequences. The graft of $C$ and $C'$ is the caterpillar sequence

$$\circled{C} = (d_1, d_2, \ldots, d_{k-1}, d_k + d'_1 - 2, d'_2, \ldots, d'_l).$$

The graft of the caterpillars $(4, 1)$ and $(3, 0, 1)$ is depicted in Figure 6.

The maps $n$ and $n_1$ interact well with the graft operation. Given two caterpillar sequences $C$ and $C'$ be two caterpillars, we have

$$n_1(\circled{C}) = n_1(C) + n_1(C') - 2 \quad (15)$$

$$n(\circled{C}) = n(C) + n(C') - 3. \quad (16)$$

The graft operation is useful for decomposing a caterpillar sequence into smaller ones.

**Lemma 5.5.** Let $C$ be a caterpillar sequence of size $n$. Then, for any integer $i \in \{3, 4, \ldots, n-3\}$,

$$C = \text{Left}_i(C) \circ \text{Right}_{n+3-i}(C).$$

**Proof:** Write $C = (d_1, \ldots, d_k)$. Then

$$\text{Left}_i(C) \circ \text{Right}_{n+3-i}(C) = (d_1, \ldots, d_a, \alpha) \circ (\beta, d_b, \ldots, d_k) = (d_1, \ldots, d_a, \alpha + \beta - 2, d_b, \ldots, d_k),$$

where $\alpha$, $\beta$, $a$ and $b$ satisfy $\sum_{m=1}^{a} (d_m + 1) + \alpha = i$ and $\sum_{m=b}^{k} (d_m + 1) + \beta = n + 3 - i$. The only possibility is $\alpha + \beta = d_{a+1} + 2 = d_{b-1} + 2$. \(\square\)

We are now ready to describe the shape of the leaf sequences of caterpillars.

**Lemma 5.6.** Let $C$ be a caterpillar. Then $\Delta L_C$ is prefix normal.

**Proof:** Let $w = \Delta L_C$. We proceed by contradiction, i.e. we assume that $w$ is not prefix normal. By Proposition 5.1 there exist two words $p$ and $f$, with $|p|_1 = |f|_1$ such that $pf \in \text{Pref}(w)$ and $1f \in \text{Fac}(w)$. Let $n - 3$ be the rightmost index of an occurrence of the factor $1f$ in $w$, i.e. such that $1f = \text{Suff}_w(|\text{Pref}_{n-3}(w)|)$, and $C'$ be a caterpillar subsequence of $C$ of size $n$ such that $n_1(C') = L_C(n(C'))$. Also, let $A = \text{Left}_n(1f)(C')$ and $B = \text{Right}_1|1f|+3(C')$ so that, by Lemma 5.5 we have $C' = A \circ B$. 

---

**Fig. 6:** The graft of the caterpillar $(4, 1)$ and $(3, 0, 1)$. 

Fully leafed induced subtrees
Then

\[ n_1(B) = n_1(C') - n_1(A) + 2 = L_C(n(C')) - n_1(A) + 2, \]

\[ \geq L_C(n) - L_C(n - |1f|_1) + 2, \quad \text{(by definition of } L) \]

\[ = |1f|_1 + 2, \quad \text{(by definition of } w) \]

\[ = |f|_1 + 3 \]

\[ = |p|_1 + 3 \]

\[ > |p|_1 + 2 \]

\[ = |p0|_1 + 2 \]

\[ = L_C(|p0| + 3), \]

which is absurd since \( n(B) = |p0| + 3 \).

\[ \Box \]

**Proof of Theorem 5.1** Follows from Corollary 5.1 and Lemma 5.6

To conclude this section, we summarize the results obtained so far as follows (see Figure 7). Given a family \( A \) of graphs, let \( \mathcal{L}(A) \) be the language over all possible leaf words \( \Delta L_G \) for \( G \in A \). Let \( \mathcal{T} \) be the family of all trees and \( \mathcal{C} \) the family of all caterpillars.

Lemma 5.2 implies \( \mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{G} - \mathcal{T}) = \emptyset \). Also, by Theorem 5.1, we have shown that

\[ \mathcal{L}(\mathcal{C}) = \{ w \mid w \text{ is prefix normal on } \{0, 1\} \}. \]

As caterpillars are particular trees, one might wonder whether their language is the same. However, it turns out that \( \mathcal{L}(\mathcal{C}) \nsubseteq \mathcal{L}(\mathcal{T}) \). A smallest counter-example is depicted in Figure 8, whose corresponding leaf word is 1101011011, which is not a prefix normal word, by taking the pair of words \( p = 11010 \) and \( f = 11011 \).

6 Concluding Remarks

In this paper, we have investigated the map \( L_G \) giving the maximal number of leaves in induced subtrees of \( G \) of given size. We have developed basic algorithms for the computation of the function \( L_G \) on general
Fully leafed induced subtrees

graphs and on trees. We proved that the the language of prefix normal words is essentially the language
of caterpillars.

Several questions remain open. In particular the identification of the language \( \mathcal{L}(T) \) is eluding us. We
have identified the reading caterpillar \( C(w) \) as a representative of the set of caterpillars having a given
prefix normal word \( w \), but the size of these sets and the combinatorial relationship between their elements
remain to be investigated. The enumeration of fully leafed induced subtrees of particular graphs have not
started yet. Improving and specializing the algorithms presented in this paper would be of interest to us.
Finally, it is not clear whether the algorithm described in Theorem 4.1 is optimal or its time complexity
analysis could be refined.

As a last perspective, it would be interesting to study the following natural problems (finding, counting
and generating) derived from the definition of fully leafed induced subtrees.

**Problem 6.1** (FLIS\(^i\)). Given a simple graph \( G \), find a fully leafed induced subtree of \( G \) having \( i \) vertices.

**Problem 6.2** (CFLIS\(^i\)). Given a simple graph \( G \), count the number of fully leafed induced subtrees of \( G \)
having \( i \) vertices.

**Problem 6.3** (AFLIS\(^i\)). Given a simple graph \( G \), find all fully leafed induced subtrees of \( G \) having \( i \)
vertices.

Since \( \text{IS}_i \) is NP-complete, it follows that the above problems are NP-hard and CFLIS\(^i\) is \#P-hard.
However, since \( \text{IS}_i \) is polynomial for trees, one might wonder if the problem CFLIS\(^i\) is also polynomial
and if AFLIS\(^i\) can be solved with an enumeration algorithm with polynomial time delay and polynomial
space.

**References**

[AP16] D. Allison and D. Paulusma. New Bounds for the Snake-in-the-Box Problem. *ArXiv e-prints*, March 2016.

[BCL05] A. Boukerche, X. Cheng, and J. Linus. A performance evaluation of a novel energy-aware
data-centric routing algorithm in wireless sensor networks. *Wireless Networks*, 11(5):619–
635, 2005.

[BFL+14] Péter Burcsi, Gabriele Fici, Zsuzsanna Lipták, Frank Ruskey, and Joe Sawada. On combinatorial
generations on prefix normal words. In *Proceedings of the 28th Ann. Symp. on Comb. Pattern Matching (CPM 2014)*, volume 8486 of *Lect. Notes in Comput. Sci.*, pages 60–69, 2014.

[BFL+17] Péter Burcsi, Gabriele Fici, Zsuzsanna Lipták, Frank Ruskey, and Joe Sawada. On prefix
normal words and prefix normal forms. *Theoretical Computer Science*, 659:1 – 13, 2017.
Alexandre Blondin Massé, Julien de Carufel, Alain Goupil, and Maxime Samson. Fully leafed tree-like polyominoes and polycubes. In Combinatorial Algorithms. 28th International Workshop, IWOCA 2017, New-Castle, Australia, Springer, 2017. To appear.

Reinhard Diestel. Graph theory, volume 173 of Graduate Texts in Mathematics. Springer, Heidelberg, fourth edition, 2010.

Paul Erdős, Michael Saks, and Vera T. Sós. Maximum induced trees in graphs. J. Combin. Theory Ser. B, 41(1):61–79, 1986.

Michael R. Garey and David S. Johnson. Computers and intractability. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

Daniel J. Kleitman and Douglas B. West. Spanning trees with many leaves. SIAM J. Discrete Math., 4(1):99–106, 1991.

M. Lothaire. Combinatorics on words. Cambridge Mathematical Library. Cambridge University Press, Cambridge, 1997. With a foreword by Roger Lyndon and a preface by Dominique Perrin, Corrected reprint of the 1983 original, with a new preface by Perrin.

Jaroslav Nešetřil and Patrice Ossona de Mendez. Sparsity, volume 28. Springer-Verlag, 2012. Algorithms and Combinatorics.

C. Payan, M. Tchuente, and N. H. Xuong. Arbres avec un nombre maximum de sommets pendants. Discrete Math., 49(3):267–273, 1984.

J.M. Robson. Finding a maximum independent set in time O(2n/4). Labri, January 2001.

J.M. Tarjan and A.E. Trojanowsky. Finding a maximum independent set. SIAM Journal on Computing, 6(3):537–546, 1977.

Kunihiro Wasa, Hiroki Arimura, and Takeaki Uno. Efficient enumeration of induced subtrees in a K-degenerate graph. In Algorithms and computation, volume 8889 of Lecture Notes in Comput. Sci., pages 94–102. Springer, Cham, 2014.

Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’02, pages 71–80, New York, NY, USA, 2002. ACM.