A Linear Time Algorithm for Computing Longest Paths in 2-Trees

Minko Markov∗ Tzvetalin S. Vassilev†
Krassimir Manev‡

Abstract
We propose a practical linear time algorithm for the Longest Path problem on 2-trees.

Keywords: algorithmic graph theory, longest path, treewidth, 2-tree

1 Introduction
2 Background

We consider undirected graphs without multiple edges or self loops. Let $G = (V, E)$ be a graph. To delete a vertex $u$ from $G$ means to transform $G$ into $G' = (V \setminus u, E \setminus \{e \in E | u \in e\})$. We denote the vertex deletion by $G' = G - u$. To remove an edge $e$ from $G$ means to transform $G$ into $G'' = (V, E \setminus \{e\})$. We use the minus sign to denote the edge removal as well: $G'' = G - e$. If the vertex set of a graph $G$ is not named explicitly we denote it by $V(G)$. Likewise, $E(G)$ is the edge set. $K_n$ is the complete graph with $n$ vertices.

A path in $G$ is a sequence $p = u_1, e_1, u_2, e_2, \ldots, e_n-1, u_n$, for some $n \geq 1$, of alternating distinct vertices $u_1, u_2, \ldots, u_n$ and edges $e_1, e_2, \ldots, e_n-1$ such that for $1 \leq i < n$, $e_i = (u_i, u_{i+1})$. $u_1$ and $u_n$ are called the endpoints of $p$, and the remaining vertices are the internal vertices of $p$. We abuse the set-theoretical notation “$\in$” by writing “$u_1 \in p$” and “$e_1 \in p$” to denote the

∗Department of Computing Systems, Faculty of Mathematics and Informatics, “St. Kliment Ohridski” University of Sofia, 5 J. Bourchier Blvd, P.O. Box 48, BG-1164 Sofia, Bulgaria. Email: minkom@fmi.uni-sofia.bg
†Department of Computer Science and Mathematics, Nipissing University, 100 College Drive, Box 5002 North Bay, Ontario P1B 8L7, Canada. Email: tzvet-talv@nipissingu.ca
‡Department of Computer Science, New Bulgarian University, 21 Montevideo St., 1618 Sofia, Bulgaria. Email: kmanev@nbu.bg
facts that \( u_1 \) and \( e_1 \), respectively, are in \( p \), etc. If \( n \geq 2 \), by \( p - u_1 \) we denote the path \( u_2, e_2, \ldots, e_{n-1}, u_n \). The length of a path \( p \) is the number of edges in it and is denoted by \( |p| \). A subpath of \( p \) is a contiguous subsequence of \( p \) that is a path. Suppose \( q_1, q_2, \ldots, q_s \) are paths in \( G \). We say that they cover \( p \) in that order if:

- \( q_i \) is a subpath of \( p \) for \( 1 \leq i \leq s \), and
- one endpoint of \( q_1 \) coincides with one endpoint of \( p \), the other endpoint of \( q_1 \) coincides with one endpoint of \( q_2 \), the other endpoint of \( q_2 \) coincides with one endpoint of \( q_3 \), etc., the other endpoint of \( q_{s-1} \) coincides with one endpoint of \( q_s \), the other endpoint of \( q_s \) coincides with the other endpoint of \( p \).

We denote the fact that \( q_1, q_2, \ldots, q_s \) cover \( p \) in that order by writing \( p = q_1 \oplus q_2 \oplus \ldots \oplus q_s \). Clearly, \( |p| = \sum_{i=1}^{s} |q_i| \). When the covering of \( p \) is understood, the paths that take part in it—in the current naming scheme these are \( q_1, q_2, \ldots, q_s \)—are called the constituents of \( p \).

**Notation 1.** Let \( G \) be a graph. Let \( u \) and \( v \) be any distinct vertices in \( G \).

- “\( u \)-path” means path with one endpoint \( u \).
- “\( u \)-to-\( v \)-path” means path with endpoints \( u \) and \( v \).
- “\( u \)-in-\( v \)-path” means path with one endpoint \( u \) in which \( v \) is an internal vertex.
- “in-\( u \)-in-\( v \)-path” means path in which \( u \) and \( v \) are internal vertices.
- “\( u \)-not-\( v \)-path” means path with one endpoint \( u \) and not containing \( v \).
- For any two paths \( p \) and \( q \) in \( G \), “\( p \perp q \)” is an abbreviation for “\( p \) and \( q \) are vertex disjoint”.
- “max path” is an abbreviation for “path of maximum length”, “max \( u \)-path” is an abbreviation for “\( u \)-path of maximum length”, etc.
- “\( \langle p, q \rangle \) are max-sum \( (u, v) \)-paths” means \( p \) and \( q \) are paths in \( G \) such that:
  - \( p \) is a \( u \)-path, \( q \) is a \( v \)-path, \( p \perp q \), and
  - \( |p| + |q| \) is maximum over all such pairs of paths.

The angle brackets in the latter notation denote ordered pairs.
Figure 1: An example of a 2-tree.

Suppose $G$ is a connected graph. A separator of $G$ is any nonempty vertex set $U \subset V$, such that $V \setminus U$ can be partitioned into two nonempty sets $X$ and $Y$, such that any path in $G$ with one endpoint from $X$ and the other one, from $Y$, contains a vertex from $U$. To split a separator $U$ means the following. Let $G'$ be the subgraph of $G$ induced by $U$. We first delete $U$ from $G$, which clearly leads to the appearance of some connected components $H_1, H_2, \ldots, H_k$ for some $k \geq 2$, and transform $H_i$ into $(V(H_i) \cup U, E(H_i) \cup E(G') \cup E''_i)$, where $E''_i = \{(u, v) \in E(G) \mid u \in V(H_i) \text{ and } v \in U\}$. We denote the splitting operation by $G \ominus U$. In case $U = \{u\}$ we may write $G \ominus u$.

**Definition 1** (2-tree). A graph is a 2-tree if and only if it can be constructed according to the following rules.

1. $K_2$ is a 2-tree.

2. If $G'$ is a 2-tree, $e = (v_i, v_j)$ is any edge in $E(G')$, and $u$ is a vertex not in $V(G')$, then $G = (V(G') \cup \{u\}, E(G') \cup \{(u, v_i), (u, v_j)\})$ is a 2-tree.

Figure 1 shows a 2-tree. Clearly, it can be constructed according to Definition 1: start with the edge $(a, b)$ and then add the remaining vertices in the alphabetical order. Let $G$ be any 2-tree. If $G$ has precisely two vertices we say it is trivial, otherwise it is non-trivial. The edges of $G$ are classified into peripheral edges and interior edges as follows. If $G$ is $K_2$ or $K_3$ then all its edges are peripheral edges. Otherwise, $G$ is obtained from some smaller
2-tree $G'$ as in Definition 1. Using the naming convention of the Definition, the edge $e = (v_i, v_j)$ becomes interior regardless of its status before, and the newly added edges $(u, v_i)$ and $(u, v_j)$ become peripheral edges. For example, in the 2-tree on Figure 1, the edges $(a, b), (c, j)$, and $(w, y)$ are interior, while $(a, d), (b, n)$, and $(y, z)$ are peripheral.

Let $G$ be a non-trivial 2-tree. We call the induced $K_3$’s of $G$, the faces of $G$. Every face is identified by its three vertices, e.g. $F = (o, v, w)$ on Figure 1. Clearly, every peripheral edge in $G$ belongs to precisely one face, and for every interior edge $e = (v_i, v_j)$, $\{v_i, v_j\}$ is a separator of $G$ whose splitting results in the appearance of at least two 2-trees. We say shortly we split $e$, rather than its vertices, and write $G \ominus e$. We emphasise $G \ominus e$ is a set of non-trivial 2-trees.

A rooted 2-tree is 2-tree in which one edge is chosen to be special and is called the root. We denote the fact that $e$ is the root of $G$ by writing $e = \text{root}(G)$. If the root is a peripheral edge we say $G$ is simple, and if $G$ is non-trivial as well we define its root face and that is the face that contains the root. If the root is an interior edge we do not define any root face and we say the rooted tree is complex. In the latter case, $G \ominus e$ is a set of rooted simple non-trivial 2-trees, each one with root $e$. We call those, the folios of $G$. Clearly, the following inductive definition of rooted two tree is equivalent to the one we just mentioned.

Let $G$ be a rooted simple non-trivial 2-tree with root $(u, v)$ and root face $F = (u, v, w)$. Clearly, $(G - e) \ominus w$ consists of two connected components which we call, the branches of $G$. Each of them is considered to be a rooted 2-tree with root the edge from $F$ that is in it. Figure 2 shows a simple rooted 2-tree and a complex one schematically; the former one (Figure 2(a)) in terms of its branches and the latter one (Figure 2(b)), in terms of its folios.

It follows in every rooted 2-tree, every interior edge has one or more simple 2-trees rooted at it, and every edge is the root of some rooted 2-tree. The leaves of any rooted non-trivial 2-tree are the peripheral edges that are distinct from the root. The leaf of any rooted trivial 2-tree is its only edge, i.e. the root.

\textbf{Definition 2.} Let $G$ be any rooted 2-tree and $(u, v) = \text{root}(G)$. Then,

\begin{align*}
L_G &= \{p \mid p \text{ is a max path in } G\} \\
L_G(u \cdot v) &= \{p \mid p \text{ is a max } u\text{-to-}v\text{-path in } G\} \\
L_G(u \cdot \cdots \cdot v) &= \{p \mid p \text{ is a max } u\text{-in-}v\text{-path in } G\} \\
L_G(v, \neg v) &= \{p \mid p \text{ is a max } u\text{-not-}v\text{-path in } G\} \\
L_G(u \perp v) &= \{(p, q) \mid (p, q) \text{ are max-sum } (u, v)\text{-paths}\}.
\end{align*}
(a) A simple rooted 2-tree with root \((u, v)\) and root face \(F\). The branches are \(G_1\) and \(G_2\). Their roots are \((u, w)\) and \((w, v)\), respectively.

(b) A complex rooted 2-tree with root \((u, v)\). The folios are \(G_1, G_2, \ldots, G_k\). Each folio is a simple rooted 2-tree with root \((u, v)\).

Figure 2: A simple and a complex 2-tree.
Furthermore,

\[ \ell_G = |p|, \text{ for any } p \in L_G \]
\[ \ell_G(u \cdots v) = |p|, \text{ for any } p \in L_G(u \cdots v) \]
\[ \ell_G(u \cdots v \cdots) = |p|, \text{ for any } p \in L_G(u \cdots v \cdots) \]
\[ \ell_G(u, \neg v) = |p|, \text{ for any } p \in L_G(u, \neg v) \]
\[ \ell_G(u \perp v) = |p| + |q|, \text{ for any } (p, q) \in L_G(u \perp v). \]

\[\square\]

**Definition 3** (label). Let \( G \) be a rooted 2-tree with root \( e = (u,v) \). The label of \( e \) is the ordered septuple \( \tilde{\lambda}(e) = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7) \), where:

\[
\begin{align*}
\lambda_1 & = \ell_G, \\
\lambda_2 & = \ell_G(u \cdots v), \\
\lambda_3 & = \ell_G(u \cdots v \cdots), \\
\lambda_4 & = \ell_G(u, \neg v), \\
\lambda_5 & = \ell_G(v \cdots u \cdots), \\
\lambda_6 & = \ell_G(v, \neg u), \\
\lambda_7 & = \ell_G(u \perp v)
\end{align*}
\]

\[\square\]

The graphs we consider are not oriented, therefore both \((u,v)\) and \((v,u)\) are equivalent descriptions of the same edge. However, some of the elements of \(\tilde{\lambda}(e)\), namely \(\lambda_3, \lambda_4, \lambda_5, \) and \(\lambda_6\), are not invariant under interchanging \(u\) and \(v\). For instance, it may be the case that \(\ell_G(u \cdots v \cdots) \neq \ell_G(v \cdots u \cdots)\), etc. We resolve that issue by postulating the following. If the root is described as \((u, v)\) then the elements of the label are constructed according to that description and the mentioned definition. It follows that if the root of the same 2-tree was described as \((v, u)\) the label would be \((\lambda_1, \lambda_2, \lambda_5, \lambda_6, \lambda_3, \lambda_4, \lambda_7)\). In other words, we impose an order on the two vertices in the root edge when defining the label.

Every edge in a rooted 2-tree can be assigned a label because, as we observed, every edge is the root of some rooted 2-tree. Obviously, the label of any leaf is \((1,1,0,0,0,0,0)\). We place a tilde sign above the names of ordered tuples, e.g. \(\tilde{\lambda}(e)\) or \(\tilde{\lambda}\). The same name but with an index \(i\), \(1 \leq i \leq 7\), and without the tilde above it, refers to the \(i\)-th elements of the tuple, e.g. \(\lambda_2(e)\) is the second element of \(\tilde{\lambda}(e)\) and \(\lambda_2\) is the second element of \(\tilde{\lambda}\), etc.
3 Our algorithm, its verification and time complexity analysis

3.1 The algorithm

LONGEST Path\((G: \text{rooted 2-tree}, e: \text{the root of } G)\)
1 \hspace{1em} \triangleright \text{Computes the length of a longest path}
2 \hspace{1em} \bar{\lambda} \leftarrow \text{COMPUTE LABEL}(G, e)
3 \hspace{1em} \text{return } \bar{\lambda}_1

COMPUTE LABEL\((G: \text{rooted 2-tree}, e: \text{the root})\)
1 \hspace{1em} \triangleright \text{Computes the label of the root}
2 \hspace{1em} \text{if } G \text{ is trivial}
3 \hspace{1em} \bar{\lambda} \leftarrow (1, 1, 0, 0, 0, 0, 0)
4 \hspace{1em} \text{else}
5 \hspace{1em} \text{let } e = (u, v)
6 \hspace{1em} \text{let } G \ominus e \text{ be } \{G^1, G^2, \ldots, G^k\}
7 \hspace{1em} \text{for } i \leftarrow 1 \text{ to } k \text{ do}
8 \hspace{1em} \text{let the root face of } G^i \text{ be } (u, v, w_i)
9 \hspace{1em} \mu^i \leftarrow \text{COMPUTE LABEL}(H^1, (u, w_i))
10 \hspace{1em} \bar{\mu}^i \leftarrow \text{COMPUTE LABEL}(H^2, (w_i, v))
11 \hspace{1em} \bar{\nu}^i \leftarrow \text{COMBINE ON FACE}(\mu^1, \mu^2)
12 \hspace{1em} \bar{\lambda} \leftarrow \text{COMBINE ON EDGE}(\bar{\nu}^1, \bar{\nu}^2, \ldots, \bar{\nu}^k)
13 \hspace{1em} \text{return } \bar{\lambda}

COMBINE ON EDGE\((\bar{\lambda}^1, \bar{\lambda}^2, \ldots, \bar{\lambda}^k: \text{label})\)
1 \hspace{1em} \triangleright \text{Computes } \bar{\lambda} = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7), \text{ the label of rooted non-trivial 2-tree whose folios have labels } \bar{\lambda}^1, \bar{\lambda}^2, \ldots, \bar{\lambda}^k.
2 \hspace{1em} \triangleright \text{non-trivial 2-tree whose folios have labels } \bar{\lambda}^1, \bar{\lambda}^2, \ldots, \bar{\lambda}^k.
3 \hspace{1em} \text{if } k = 1
4 \hspace{1em} \bar{\lambda} \leftarrow \bar{\lambda}^1
5 \hspace{1em} \text{else}
6 \hspace{1em} x \leftarrow \max \{\lambda_i^1 \mid 1 \leq i \leq k\}
7 \hspace{1em} y \leftarrow \max \{\lambda_i^2 + \lambda_j^0 \mid 1 \leq i, j \leq k, i \neq j\}
8 \hspace{1em} \lambda_7 \leftarrow \max \{x, y\}
9 \hspace{1em} \lambda_2 \leftarrow \max \{\lambda_i^2 \mid 1 \leq i \leq k\}
10 \hspace{1em} \lambda_4 \leftarrow \max \{\lambda_i^4 \mid 1 \leq i \leq k\}
11 \hspace{1em} \lambda_6 \leftarrow \max \{\lambda_i^6 \mid 1 \leq i \leq k\}
12 \hspace{1em} x \leftarrow \max \{\lambda_i^3 \mid 1 \leq i \leq k\}
function positive(ψ, x)
let λ_k = max{x, y}
x ← max{λ_1^i | 1 ≤ i ≤ k}
y ← max{λ_2^j + λ_4^j | 1 ≤ i, j ≤ k, i ≠ j}
λ_5 ← max{x, y}
x ← max{λ_1 | 1 ≤ i ≤ k}
x_1 ← max{λ_2^i + λ_4^i | 1 ≤ i, j ≤ k, i ≠ j}
x_2 ← max{λ_2^j + λ_4^j | 1 ≤ i, j ≤ k, i ≠ j}
x_3 ← max{λ_2^j + λ_4^j | 1 ≤ i, j ≤ k, i ≠ j}
y_1 ← max{λ_2^i, λ_4^i | 1 ≤ i, j ≤ k, i ≠ j}
y_2 ← max{λ_2^i, λ_4^i | 1 ≤ i, j ≤ k, i ≠ j}
z ← 0
if k ≥ 3
z ← max{λ_2^i + λ_4^i + λ_6^i | 1 ≤ i, j ≤ k, i ≠ j, t ≠ i}
λ_1 ← max{x, λ_2, λ_3, λ_4, λ_5, λ_6, x_1, x_2, x_3}
return λ

Let ψ be a function of several nonnegative variables, one of which is x. The function positive(ψ, x) is defined as follows:

\[
\text{positive}(\psi, x) = \begin{cases} 0, & \text{if } x = 0 \\ \psi, & \text{else} \end{cases}
\]

**COMBINE ON FACE(\(\bar{\lambda}_1\), \(\bar{\lambda}_2\): label)**

1. Computes \(\bar{\lambda} = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7)\), the label of a rooted simple non-trivial 2-tree with root \((u, v)\) whose branches have roots \((u, w)\) and \((w, v)\) and labels \(\bar{\lambda}^1\) and \(\bar{\lambda}^2\), respectively.
2. λ_2 ← \(\lambda_1^2 + \lambda_4^2\)
3. \(x_1 ← \text{positive}(1 + \lambda_2^2, \lambda_4^2)\)
4. \(x_2 ← \text{positive}(\lambda_2^2 + \lambda_4^2, \lambda_6^2)\)
5. \(x_3 ← \text{positive}(1 + \lambda_2^3, \lambda_6^3)\)
6. \(x_4 ← \text{positive}(1 + \lambda_2^4 + \lambda_4^4, \lambda_6^4)\)
7. \(λ_3 ← \max {x_1, x_2, 1 + \lambda_2^2, x_3, x_4}\)
8. \(λ_4 ← \max {\lambda_2^3, \lambda_1^4, \lambda_1^2 + \lambda_4^4}\)
9. \(x_1 ← \text{positive}(1 + \lambda_2^5, \lambda_4^5)\)
10. \(x_2 ← \text{positive}(\lambda_2^3 + \lambda_5^3, \lambda_4^5)\)
11. \(x_3 ← \text{positive}(1 + \lambda_2^4, \lambda_4^5)\)
12. \(x_4 ← \text{positive}(1 + \lambda_2^3 + \lambda_4^5, \lambda_4^5)\)
13. \(λ_5 ← \max {x_1, x_2, 1 + \lambda_2^4, x_3, x_4}\)
14. \(λ_6 ← \max {\lambda_2^5, \lambda_5^3, \lambda_1^2 + \lambda_4^4}\)
Algorithm 3.2 Verification

**Lemma 1.** Algorithm \textsc{Combine on Face} is correct under the assumption that $\lambda^1$ and $\lambda^2$ are the correct labels of the branches.

**Proof:** Assume $G$ is a simple nontrivial rooted 2-tree as shown on Figure 2(a) on page 5, the edge $(u, v)$ is called $e$, and $\tilde{\lambda}^1$ and $\tilde{\lambda}^2$ are the labels of $G_1$ and $G_2$, respectively. We prove the correctness of the assignments to $\lambda_i$ in the order they appear in the algorithm. Let $q$ be the path $q = u, v$.

Consider any $p \in L_G(u \cdots v)$. $p$ is covered by two paths such that one is in $L_{G_1}(u \cdots w)$ and the other one, in $L_{G_2}(w \cdots v)$. To see why, assume $w \not\in p$. Then $p$ must coincide with $q$, so $|p| = 1$. But there is a length 2, $u$-to-$v$-path in $G$, namely $u, w, v$. It follows $w \not\in p$. Further, $w$ is an internal vertex in $p$, therefore $e \not\in p$. Now it is obvious there exist paths $p'$ and $q'$, such that $p = p' \oplus q'$ and $p'$ is a $u$-to-$v$-path in $G_1$ and $q'$ is a $w$-to-$v$-path in $G_2$.

Furthermore, $p' \in L_{G_1}(u \cdots w)$ and $q' \in L_{G_2}(w \cdots v)$ because $|p| = |p'| + |q'|$ and $|p'|$ and $|q'|$ are maximised independently. By the inductive assumption, $\lambda^1_2 = |p'|$ and $\lambda^2_2 = |q'|$. So the assignment at line 4 is correct.

Now we argue about the assignment at line 9. Consider any $p \in L_G(u \cdots v \cdots)$. The following five subcases are exhaustive:

1. $w \not\in p$. Then $p = q \oplus p'$ for some path $p'$ in $G_2$. Clearly, $p' \in L_{G_2}(v \cdots w)$, so $|p'| = \ell_{G_2}(v \cdots w)$. Recall that $\lambda^2_2 = \ell_{G_2}(v \cdots w)$. By the inductive assumptions, $\lambda^2_6 = \ell_{G_2}(v \cdots w)$, therefore $|p| = 1 + \lambda^2_6$.

2. $w \in p$ and $(u, v) \not\in p$. It must be the case that $p = p' \oplus q'$, such that $p' \in L_{G_1}(u \cdots w)$ and $q' \in L_{G_2}(w \cdots v \cdots)$. So, $|p| = \lambda^1_2 + \lambda^2_3$.

3. $w \in p$, $(u, v) \in p$, and $w$ is an endpoint of $p$. Then $|p| = 1 + \lambda^2_2$.

4. $w \in p$, $(u, v) \in p$, $w$ is an internal vertex in $p$, and the endpoint of $p$ is not $u$ in $G_2$. Then $|p| = 1 + \lambda^2_3$.

5. $w \in p$, $(u, v) \in p$, $w$ is an internal vertex in $p$, and the endpoint of $p$ is not $u$ in $G_1$. Then $|p| = 1 + \lambda^2_2 + \lambda^1_6$. 

\begin{verbatim}
17 $\lambda_7 \leftarrow \max \{\lambda^2_1 + \lambda^2_6, \lambda^2_1 + \lambda^2_5, \lambda^2_1 + \lambda^2_3, \lambda^2_1 + \lambda^2_2, \lambda^2_1 + \lambda^2_1, \lambda^2_1 + \lambda^2_2, \lambda^2_1 + \lambda^2_3, \lambda^2_1 + \lambda^2_5, \lambda^2_1 + \lambda^2_6, \lambda^2_1 + \lambda^2_7\}$
18 $x_1 \leftarrow \lambda^2_1 + \lambda^2_6$
19 $x_2 \leftarrow \lambda^2_1 + \lambda^2_5$
20 $x_3 \leftarrow \lambda^2_1 + \lambda^2_3$
21 $x_4 \leftarrow \lambda^2_1 + \lambda^2_2$
22 $\lambda_1 \leftarrow \max \{\lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_1, x_1, x_2, x_3, x_4, \lambda_7 + 1\}$
23 $\lambda \leftarrow (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7)$
24 return $\lambda$
\end{verbatim}
At line 9 we assign to $\lambda_3$ the maximum of those five quantities. Figure 3 illustrates the said five subcases in order and for each subcase, how $|p|$ is computed from the applicable $\lambda^i_j$’s.

Figure 3: The five subcases when $p \in L_{G}(u \cdots v \cdots)$ (line 9 of Combine on Face).

Note that the right-hand sides of lines 5–8 are wrapped in the function positive(). The reason is that we want to discard one or more subcases among 1, 2, 4, and 5, if one constituent of $p$ is a zero length path. For instance, consider subcase 2, illustrated on Figure 3(b). Assume $G_2$ is trivial. We must discard subcase 2 because $v$ cannot be an internal vertex in $p$ when $G_2$ is trivial and $w$ is between $u$ and $v$, which is implied by subcase 2. Discarding subcase 2 is equivalent to assigning $x_2 \leftarrow 0$ at line 6. Suppose we do not use the function positive() and perform the direct assignment $x_2 \leftarrow \lambda_1^2 + \lambda_2^3$ at line 6. What will happen is $x_2 \leftarrow \lambda_1^2$ since $\lambda_2^3 = 0$. However, $\lambda_2^3$ can be arbitrarily large so we may assign incorrectly a nonzero value to $x_2$. Likewise, if $G_2$ is trivial then we have to discard subcase 1 (see Figure 3(a)) by assigning $x_1 \leftarrow 0$ rather than $x_1 \leftarrow 1 + 0$, which is what will happen unless positive() is used at line 5. And so on.

Now we argue about the assignment at line 10. Consider any $p \in L_{G}(v \cdots u \cdots)$. The argument is identical to the argument about the correctness of the assignment at line 9 after the following change of names: swap $G_1$ with $G_2$, swap $u$ with $v$, substitute $\lambda_1^2$ with $\lambda_2^3$, and consequently it suffices to compute the maximum of three, not four, quantities. Figure 4 illustrates the mentioned four cases for $p$ with respect to line 10 and for each possibility, how $|p|$ is computed from the applicable $\lambda^i_j$’s.
\[ \lambda_1^3, \text{ substitute } \lambda_2^6 \text{ with } \lambda_1^4, \text{ and substitute } \lambda_2^1 \text{ with } \lambda_4^1. \] Figure 5 illustrates the five distinct possibilities for \( p \) with respect to line 15. Subfigures 5(a)–5(e) correspond to subfigures 3(a)–3(e) under the said name swaps and substitutions.

\[ \begin{align*}
(a) \quad & 1 + \lambda_4^1 \\
(b) \quad & \lambda_1^3 + \lambda_2^6 \\
(c) \quad & 1 + \lambda_2^6 \\
(d) \quad & 1 + \lambda_3^2 \\
(e) \quad & 1 + \lambda_2^6 + \lambda_3^2
\end{align*} \]

Figure 5: The five subcases when \( p \in L_G(v \cdots u \cdots) \) (line 15 of \texttt{Combine on Face}).

Now we argue about the assignment at line 16. Consider any \( p \in L_G(v, \neg u) \). The argument is identical to the argument about the correctness of the assignment at line 10 after the following change of names: swap \( G_1 \) with \( G_2 \), swap \( u \) with \( v \), substitute \( \lambda_4^1 \) with \( \lambda_2^6 \), substitute \( \lambda_1^3 \) with \( \lambda_2^1 \), substitute \( \lambda_2^6 \) with \( \lambda_1^2 \), and substitute \( \lambda_3^2 \) with \( \lambda_1^4 \). Figure 6 illustrates the four cases for \( p \) with respect to line 16 and for each possibility, how \( |p| \) is computed from the applicable \( \lambda_i^j \)'s.

Now we argue about the assignment at line 17. Consider any \( \langle p_1, p_2 \rangle \in L_G(u \perp v) \). Note that \( e \not\in p_1 \) and \( e \not\in p_2 \). Furthermore, it cannot be the case that simultaneously \( w \in p_1 \) and \( w \in p_2 \). Furthermore, \( w \not\in p_1 \) implies \( p_1 \in L_{G_1}(u, \neg w) \) and \( w \not\in p_2 \) implies \( p_2 \in L_{G_2}(w, \neg v) \). So, if \( w \not\in p_1 \) and \( w \not\in p_2 \) then \( |p_1| + |p_2| = \lambda_4^1 + \lambda_2^6 \). Suppose \( w \in p_1 \). If \( w \) is an endpoint of \( p_1 \) then \( |p_1| + |p_2| = \lambda_4^1 + \lambda_2^6 \). Suppose \( w \) is an internal vertex in \( p_1 \). Let the endpoint of \( p_1 \) that is not \( w \) be \( z \). If \( z \in G_1 \) then \( |p_1| + |p_2| = \lambda_4^1 + \lambda_2^6 \). If \( z \in G_2 \) then \( p_1 \) is covered by two paths \( p_1' \) and \( p_1'' \), such that \( p_1 \) is in \( G_1 \) and with endpoints \( u \) and \( w \), and \( p_1'' \) is in \( G_2 \) with one endpoint \( w \) and is vertex
Figure 6: The four cases for $p \in L_G(v, -u)$ (line 16 of Combine on Face).
To compute $|p|$ we consider only 6(a), 6(c), and 6(d).

disjoint with $p_2$. Since $|p_1| + |p_2| = |p'_1| + |p''_2| + |p_2|$ and because of the way these three paths are situated in $G_1$ and $G_2$, it is clear that the sum is maximised when $|p'_1|$, on one hand, and $|p''_1| + |p_2|$, on the other hand, are maximised independently. In other words, when $p'_1 \in L_{G_1}(u \cdots w)$ and $\langle p''_1, p_2 \rangle \in L_{G_2}(w \perp v)$. Therefore, $|p_1| + |p_2| = \lambda^1_2 + \lambda^2_7$.

If $w \in p_2$ we use completely analogous arguments to prove that $|p_1| + |p_2| = \lambda^1_4 + \lambda^2_5$. Figure 7 illustrates the seven possibilities for $p_1$ and $p_2$, and how $|p_1| + |p_2|$ is computed for each possibility.

Figure 7: The seven possibilities for $\langle p_1, p_2 \rangle \in L_G(u \perp v)$ (line 17 of Combine on Face).

Now we argue about the assignment at line 22. Consider any longest path $p$ in $G$. Each of $u$ and $v$ can be, independently from the other one, an endpoint, an internal vertex, or not be at all, in $p$. There are nine possibilities altogether and we have already covered five of them, in that order: $p$ being $u$-to-$v$-path, $u$-in-$v$-path, $u$-not-$v$-path, $v$-in-$u$-path, $v$-not-$u$-path. The results are stored in $\lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6$, respectively. The other four possibilities do not have corresponding elements in $\lambda$ and they are covered in the remainder of the proof.

First suppose $u, v \not\in p$. If $w \not\in p$ either, then $p$ lies either entirely in $G_1$, or in $G_2$. Assume $p$ is in $G_1$. Obviously, $p$ is a longest path in $G_1$, so by the inductive hypothesis $|p| = \lambda^1_1$. Likewise, if $p$ is in $G_2$ then $|p| = \lambda^2_1$. If $w \in p$ it must be the case that $|p| = \lambda^1_6 + \lambda^2_1$. 

12
Now suppose $p$ contains $u$ as an internal vertex and does not contain $v$. Clearly, it is not possible both endpoints of $p$ to be in $G_2$. The following four possibilities are exhaustive for this subcase. They are illustrated on Figure 8

- Both endpoints of $p$ are in $G_1$ and $w \notin p$. Then $|p| = \lambda_1^1$. See Figure 8(a).

- Both endpoints of $p$ are in $G_1$ and one of them is $w$. Then $|p| = \lambda_1^1$. See Figure 8(b).

- Both endpoints of $p$ are in $G_1$, none of them is $w$, but $w \in p$. Then $|p| = \lambda_1^1$. See Figure 8(c).

- One endpoint of $p$ is in $G_2$. Then $p$ is covered by two paths $p_1$ and $p_2$ such that $p_1 \in L_{G_1}(w \cdots u \cdots)$ and $p_2 \in L_{G_2}(w, \neg v)$. Consequently, $|p| = \lambda_1^3 + \lambda_2^2$. See Figure 8(d).

![Figure 8](image)

Figure 8: The four cases when $p$ contains $u$ as internal vertex and does not contain $v$.

Now suppose $p$ contains $v$ as an internal vertex and does not contain $u$. Clearly, this is a mirror case of the former one. Figure 9 depicts the analogous four possible subcases and in each one says how $|p|$ is computed.

![Figure 9](image)

Figure 9: The four cases when $p$ contains $v$ as internal vertex and does not contain $u$. 

13
The only remaining case to consider is that $p$ contains $u$ and $v$ as internal vertices. If $e \not\in p$ then $p$ has $w$ as an internal vertex between $u$ and $v$ and consequently $p$ is covered by two paths $p_1$ and $p_2$, such that $p_1$ is in $G_1$ and has $u$ as internal vertex and $w$ as an endpoint, and $p_2$ is in $G_2$ and has $v$ as internal vertex and $w$ as an endpoint. Clearly, $p_1 \in L_{G_1}(w \cdots u \cdots)$ and $p_2 \in L_{G_2}(w \cdots v \cdots)$, therefore $|p| = \lambda_1^2 + \lambda_2^2$. Suppose $e \in p$. Suppose we remove $e$ from $p$, without removing $u$ or $v$. Clearly, this edge removal leads to the appearance of two paths $p_1$ and $p_2$, such that $p_1$ is a $u$-path, $p_2$ is a $v$-path, $p_1 \perp p_2$, and $|p_1| + |p_2|$ is maximum over all such paths. Then $|p| = |p_1| + |p_2| + 1$. Note that $\langle p_1, p_2 \rangle \in L_G(u \perp v)$, therefore $|p| = 1 + \lambda_7$. □

**Lemma 2.** Algorithm Combine on Edge is correct under the assumption that $\lambda^1, \lambda^2, \ldots, \lambda^k$ are the correct labels of the folios.

**Proof:** Assume $G$ is a rooted 2-tree with folios $G_1, G_2, \ldots, G_k$ as shown on Figure 2(b) on page 5. Let $\lambda^i$ be the label of $G_i$, for $1 \leq i \leq k$. If $k = 1$ then $G$ is the same folio, or not. The former means

\[
\exists \text{ such that } 1 \leq i \leq k \text{ and } 1 \leq j \leq k \text{ and } i \neq j, \text{ p is mostly in } G_i \text{ and } G_j \text{ if } V'(V(G_i) \cap V(G_j)) \neq \emptyset \text{ and } V'(V(G_i) \cup V(G_j)) \neq \emptyset. \]

For any $i$ and $j$ such that $1 \leq i \leq k$ and $1 \leq j \leq k$ and $i \neq j$, $p$ is mostly in $G_i$ and $G_j$ if $V'(V(G_i) \cap V(G_j)) \neq \emptyset$ and $V'(V(G_i) \cup V(G_j)) \neq \emptyset$ and $V' \subseteq V(G_i) \cup V(G_j)$.

Consider any $\langle p_1, p_2 \rangle \in L_G(u \perp v)$. Either both of them are mostly in the same folio, or not. The former means $\langle p_1, p_2 \rangle \in L_G(u \perp v)$ for some $i \in \{1, 2, \ldots, k\}$. The latter means $p_1 \in L_{G_i}(u \cdots v)$ and $p_2 \in L_{G_j}(v \cdots u)$ for some $i, j \in \{1, 2, \ldots, k\}$ such that $i \neq j$. Clearly, the assignment at line 6 is correct under the assumption and the one at line 7, to the latter one. It follows the assignment at line 8 is correct. The two said possibilities are illustrated schematically on Figure 10.

The verification of lines 9, 10, and 11 is straightforward. Any path from $L_G(u \cdots v)$ has to be mostly in a single folio, therefore

\[\ell_G(u \cdots v) = \max \{\ell_{G_i}(u \cdots v) \mid 1 \leq i \leq k\}\]

Likewise,

\[\ell_G(u, \neg v) = \max \{\ell_{G_i}(u, \neg v) \mid 1 \leq i \leq k\}\]

and

\[\ell_G(v, \neg u) = \max \{\ell_{G_i}(v, \neg u) \mid 1 \leq i \leq k\}\]

14
Consider any \( p \in \mathcal{L}_G(u \ldots v) \). Clearly, \( p \) is covered by two paths \( p_1 \) and \( p_2 \) such that \( p_1 \) has endpoints \( u \) and \( v \), and \( p_2 \) has one endpoint \( v \). Either both \( p_1 \) and \( p_2 \) are mostly in the same folio, or not. The former means there is one folio such that \( p \) is mostly in it, i.e. \( p \in \mathcal{L}_G(u \ldots v) \) for some \( i \in \{1, 2, \ldots, k\} \). The latter means \( p \) is mostly in precisely two folios, that is, \( p_1 \in \mathcal{L}_G(u \ldots v) \) and \( p_2 \in \mathcal{L}_G(v, -u) \) for some \( i, j \in \{1, 2, \ldots, k\} \) such that \( i \neq j \). The assignment at line 12 is with respect to the former possibility, the one at line 13, with respect to the latter one. It follows the assignment at line 14 is correct. The two said possibilities are illustrated schematically on Figure 11.

The argument about the correctness of the assignment at line 17 is completely analogous to the argument about line 14.

Now we argue about the assignment at line 27. Consider any longest path \( p \) in \( G \). Each of \( u \) and \( v \) can be, independently from the other one, an endpoint, an internal vertex, or not be at all, in \( p \). There are nine possibilities altogether and we have already covered five of them, in that order: \( p \) being \( u \)-to-\( v \)-path, \( u \)-not-\( v \)-path, \( v \)-in-\( u \)-path, \( u \)-in-\( v \)-path, \( v \)-not-\( u \)-path. The results are stored in \( \lambda_2, \lambda_4, \lambda_6, \lambda_3, \lambda_5 \), respectively. The other four possibilities do not have corresponding elements in \( \bar{\lambda} \) and they are covered in the remainder of the proof.
First suppose \( u, v \not\in p \). Then \(|p| = \max \{\lambda_i^1 \mid 1 \leq i \leq k\} \). Clearly, this case is taken care of by the assignment at line 18. Now suppose \( p \) contains \( u \) as an internal vertex and does not contain \( v \). We consider the following two subcases. Figure 12 illustrates them.

- \( p \) is mostly in a single folio. Then \(|p| = \max \{\lambda_i^1 \mid 1 \leq i \leq k\} \) where \( 1 \leq i \leq k \). See Figure 12(a). This subcase is also taken care of by the assignment at line 18.

- \( p \) is covered by two paths \( p_1 \) and \( p_2 \) such that \( p_1 \) is mostly in some folio \( G_i \) and \( p_2 \) is mostly in some folio \( G_j \) where \( i \neq j \). In this subcase, \(|p| = \max \{\lambda_i^1 + \lambda_j^1 \mid 1 \leq i \leq k, 1 \leq j \leq k, i \neq j\} \), which we compute at line 22. See Figure 12(b).

![Figure 12: The two cases when \( p \) contains \( u \) as internal vertex and does not contain \( v \). In each case, \( p \) is drawn with thick line.](image)

Now suppose \( p \) contains \( v \) as an internal vertex and does not contain \( u \). Clearly, this is a mirror case of the former one and is divided into analogous subcases. In the first one, \(|p| = \max \{\lambda_i^1 \mid 1 \leq i \leq k\} \) as before. In the second one, \(|p| = \max \{\lambda_i^6 + \lambda_j^6 \mid 1 \leq i \leq k, 1 \leq j \leq k, i \neq j\} \), which we compute at line 23.

The only remaining case to consider is the one in which \( p \) contains both \( u \) and \( v \) as internal vertices. Clearly, \( p \) is covered by three paths \( p_1 \), \( p_2 \), and \( p_3 \), such that \( p_2 \) is a \( u \)-to-\( v \)-path, \( p_1 \) is a \( u \)-path and \( p_3 \) is a \( v \)-path.

First suppose that \( p \) is mostly in a single folio. Then \(|p| = \max \{\lambda_i^1 \mid 1 \leq i \leq k\} \), which possibility is covered by the assignment at line 18. Now suppose that \( p \) is mostly in exactly two folios \( G_i \) and \( G_j \), which includes the possibility that the edge \((u,v)\) is in \( p \). The following subcases, illustrated on Figure 13, are exhaustive.

- For some folio \( G_i \), either \( p_1 \) and \( p_2 \) are mostly in \( G_i \) (Figure 13(a)) or \( p_2 \) and \( p_3 \) are mostly in \( G_i \) (Figure 13(b)). In the former subcase, \( p_1 \oplus p_2 \in L_{G_i}((v \cdots u) \cdots) \) and \( p_3 \in L_{G_j}(v, \neg u) \), so \(|p| = \max \{\lambda_i^5 + \lambda_j^0 \mid 1 \leq i \leq k, 1 \leq j \leq k, i \neq j\} \). We compute that value at line 20. In the
latter subcase, \( p_2 \oplus p_3 \in L_{G_i}(u \ldots v \ldots) \) and \( p_1 \in L_{G_j}(u, \neg v) \), therefore \( |p| = \max \{ \lambda_i^1 + \lambda_j^2 \} \) where \( 1 \leq i \leq k, 1 \leq j \leq k, i \neq j \), which value we compute at line 19.

- For some folio \( G_i \), \( p_1 \) and \( p_3 \) are mostly in \( G_i \) (Figure 13(c)). Clearly, \( p_2 \in L_{G_j}(u \cdots v) \) and \( \langle p_1, p_3 \rangle \in L_{G_i}(u \perp v), \) so \( |p| = \max \{ \lambda_i^3 + \lambda_j^4 \} \) where \( 1 \leq i \leq k, 1 \leq j \leq k, i \neq j \), which value we compute at line 21.

Finally, suppose \( p \) is mostly in exactly three folios. That is possible only when \( k \geq 3 \), that is why \( z \) is initialized with zero at line 24 and the assignment at line 26 is executed only if \( k \geq 3 \). Under the current assumptions, \( p \) is covered by three paths \( p_1, p_2, \) and \( p_3, \) such that \( p_2 \) has endpoints \( u \) and \( v, p_1 \) is a \( u \)-path and \( p_3 \) is a \( v \)-path. Furthermore, every \( p_i \) is mostly in a distinct folio. Figure 14 illustrates the three subpaths in \( G \).

Finally, suppose \( p \) is mostly in exactly three folios. That is possible only when \( k \geq 3 \), that is why \( z \) is initialized with zero at line 24 and the assignment at line 26 is executed only if \( k \geq 3 \). Under the current assumptions, \( p \) is covered by three paths \( p_1, p_2, \) and \( p_3, \) such that \( p_2 \) has endpoints \( u \) and \( v, p_1 \) is a \( u \)-path and \( p_3 \) is a \( v \)-path. Furthermore, every \( p_i \) is mostly in a distinct folio. Figure 14 illustrates the three subpaths in \( G \).

![Figure 13: The subcases when \( p \) is mostly in exactly two folios.](image)

Finally, suppose \( p \) is mostly in exactly three folios. That is possible only when \( k \geq 3 \), that is why \( z \) is initialized with zero at line 24 and the assignment at line 26 is executed only if \( k \geq 3 \). Under the current assumptions, \( p \) is covered by three paths \( p_1, p_2, \) and \( p_3, \) such that \( p_2 \) has endpoints \( u \) and \( v, p_1 \) is a \( u \)-path and \( p_3 \) is a \( v \)-path. Furthermore, every \( p_i \) is mostly in a distinct folio. Figure 14 illustrates the three subpaths in \( G \).

![Figure 14: \( p \) is mostly in three folios: \( |p| = \max \{ \lambda_i^3 + \lambda_j^4 + \lambda_t^6 | i \neq j \neq t \neq j \} \).](image)

Note that \( |p| = |p_1| + |p_2| + |p_3| \) and \( p_1 \in L_{G_j}(u, \neg v), p_2 \in L_{G_i}(u \cdots v), \) and \( p_3 \in L_{G_t}(v, \neg u) \) for some pairwise distinct \( i, j, \) and \( t \) from \( \{ 1, 2, \ldots, k \} \).

The correctness of the assignment at line 26 is obvious.

The remainder of the correctness proof of algorithm LONGEST PATH is straightforward. Having in mind the inductive Definition 1, and Lemma 1 and Lemma 2, a proof by structural induction of the correctness of Algorithm COMPUTE LABEL follows immediately. And based on that, the correctness of LONGEST PATH is obvious.
3.3 Time complexity analysis

The complexity of Longest Path obviously equals the complexity of Compute Label. In general, for every folio, Compute Label (CL) makes two calls to itself and a call to Combine on Face (CoF). After that, there is a single call to Combine on Edge (CoE).

The complexity of Combine on Face is obviously $\Theta(1)$. Now we argue the complexity of Combine on Edge is $\Theta(k)$ where $k$ is the number of folios. That may not be immediately obvious: for instance, computing the maximum at line 7 requires $\Theta(k^2)$ time if done in a naive straightforward way, and likewise line 26 takes $\Theta(k^3)$ time is implemented naively. We explain how to compute line 7 in $\Theta(k)$:

- Compute in $\Theta(k)$ a maximum $m_4$ and a second maximum $s_4$ of $\lambda_i^4$, for $1 \leq i \leq k$, and store the index of $m_4$.
- Compute in $\Theta(k)$ a maximum $m_6$ and a second maximum $s_6$ of $\lambda_i^6$, for $1 \leq i \leq k$, and store the index of $m_6$.
- If $m_4$ and $m_6$ have different indices then $y \leftarrow m_4 + m_6$.
- Otherwise, $y \leftarrow \max \{m_4 + s_6, m_6 + s_4\}$.

Obviously, lines 13, 16, 19, 20, 21, 22, and 23 can be computed in $\Theta(k)$ likewise. Finally, note that line 26 can also be computed in $\Theta(k)$ in a similar fashion by computing a maximum, second maximum, and third maximum for each list, recording the indices the first and second maxima, and then dealing with a constant number of possible situations, in each one computing the maximum of a constant number of summands.

Therefore, the complexity of Compute Label on an arbitrary 2-tree of $n$ vertices is captured by the following recurrence relation:

$$T(n) = \sum_{i=1}^{k} \left( T(n_i^1) + T(n_i^2) + \Theta(1) \right) + \Theta(k)$$

$$= \sum_{i=1}^{k} \left( T(n_i^1) + T(n_i^2) \right) + \Theta(k)$$

where $k$ is the number of folios and for folio number $i$, $n_i^1$ and $n_i^2$ are the numbers of vertices of its branches. Let $n_i$ be the number of vertices in folio number $i$. Clearly,

$$\sum_{i=1}^{k} n_i = n + 2(k - 1)$$

$$n_i^1 + n_i^2 = n_i + 1$$
therefore
\[
\sum_{i=1}^{k}(n_i^1 + n_i^2) = \sum_{i=1}^{k}(n_i + 1) = n + 2(k - 1) + k = n + 3k - 2
\]

Apply Lemma 3 and conclude that the solution to recurrence (1) is \( T(n) = O(n) \). It follows our algorithm is a linear time one.

**Lemma 3.** Let the recurrence relation \( T(m) \) be defined as follows for \( m > 0 \):

\[
T(1) = \Theta(1) \\
T(m) = \sum_{i=1}^{q} T(m_i) + \Theta(q)
\]

where
- \( \forall i_1 \leq i \leq q (m_i \in \mathbb{N}^+ \text{ and } 1 \leq m_i < m) \), and
- \( \sum_{i=1}^{q} m_i = m + \Theta(q) \).

Then \( T(m) = O(m) \).

**Proof:**

By induction on \( m \). Assume there are positive constants \( b \) and \( c \) such that \( T(m) \leq cm - b \). By the inductive hypothesis,

\[
T(m) \leq \sum_{i=1}^{q} (c.m_i - b) + \Theta(q)
\]

\[
= c \sum_{i=1}^{q} (m_i) - b.m + \Theta(q)
\]

\[
= c(m + \Theta(q)) - b.m + \Theta(q)
\]

\[
= cm - bm + \Theta(q)
\]

\[
\leq cm - bm + pm,
\]

assuming the larger of the two hidden constants in the “\( \Theta(\)” expression in (2) is \( p \). Clearly, \( cm + (p - b)m < cm - b \) for all sufficiently large \( m \) if \( p - b < 0 \).

\[\square\]
3.4 Longest Path on weighted and partial 2-trees

4 Conclusions

5 Acknowledgments

References


