# A Linear Time Algorithm for Computing Longest Paths in Cactus Graphs

Minko Markov, Mugurel Ionut Andreica, Krassimir Manev, Nicolae Tapus

Serdica J. Computing **6** (2012), 287–298

**Serdica**
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

# A LINEAR TIME ALGORITHM FOR COMPUTING LONGEST PATHS IN CACTUS GRAPHS

Minko Markov, Mugurel Ionuţ Andreica*, Krassimir Manev, Nicolae Ţăpuş

Abstract. We propose an algorithm that computes the length of a longest path in a cactus graph. Our algorithm can easily be modified to output a longest path as well or to solve the problem on cacti with edge or vertex weights. The algorithm works on rooted cacti and assigns to each vertex a two-number label, the first number being the desired parameter of the subcactus rooted at that vertex. The algorithm applies the divide-and-conquer approach and computes the label of each vertex from the labels of its children. The time complexity of our algorithm is linear in the number of vertices, thus improving the previously best quadratic time algorithm.

**Introduction.** Computing the length of a path of maximum length in an undirected graph is a problem that arises naturally. The graph can have positive edge weights, in which case the length of any path is the sum of its weights, or no edge weights, in which case the length of a path is the number of its edges.

*ACM Computing Classification System* (1998): G.2.2.

*Key words:* algorithmic graph theory, longest path, cactus graphs.

*The work performed by this author was partially funded by the Romanian National Council for Scientific Research (CNCS)-UEFISCDI under research grant PD_240/2010 (AATOMMS – contract no. 33/28.07.2010), from the PN II – RU program, and by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the financial agreement POSDRU/89/1.5/S/62557.

Both versions are known to be $\mathcal{NP}$-complete [10]. The two major ways to tackle with $\mathcal{NP}$-completeness are the parameterized approach and the approximation approach.

It is known that LONGEST PATH is fixed parameter tractable [13]. Recent research [6] shows that if certain restrictions are imposed on the graph there is an algorithm that is subexponential in the parameter. For a detailed introduction to the Parameterized Complexity Theory, see [7]. From the approximation perspective, the problem is not approximable in polynomial time within a multiplicative constant unless $\mathcal{P} = \mathcal{NP}$ [11]. The approximation algorithm with best approximation ratio so far has approximation ratio that is, asymptotically, close to linear [3]. Other relevant results are [4], [1], [16], [9], [8], [12], and [15].

Yet another way to tackle with $\mathcal{NP}$-completeness on graphs is to construct fast, that is polynomial time, algorithms on restricted graph classes. A linear time algorithm for LONGEST PATH on edge weighted trees was constructed by Dijkstra around 1960 (see [5] for description and formal verification). For several decades the trees were the only natural graph class for which a polynomial time algorithm for LONGEST PATH was known. Then Uehara and Uno [14] proposed polynomial time algorithms for that problem on cacti and block graphs, and showed it can be solved efficiently on graphs with interval representation. Their algorithm on cacti has $O(|V(G)|^2)$ time complexity, the bound being tight, and is based on a generalization of Dijkstra's algorithm.

In 2009 Andreica and Ţăpuş proposed [2] a linear time algorithm for longest paths in cactus graphs. Independently, at precisely the same time Manev and Markov constructed a linear time algorithm for the same problem on the same graph class. Those algorithms were remarkably similar and this article contains the combined effort of both teams. Our algorithm is based on an idea that is radically different from the idea of Uehara and Uno. First we turn the cactus into a rooted cactus, which can easily be done in linear time, and then work from the leaves upwards to the root. Each vertex $u$ of the rooted cactus is associated with two numbers: the length of a longest path in the subcactus rooted at $u$ and the length of a longest path in that subcactus having $u$ as one endpoint. We call that ordered pair, *the label* of the vertex. Clearly, the desired answer is the first number in the label of the root. The label of each nonleaf vertex is computed only from the labels of its children.

**1. Background.** We consider undirected graphs without multiple edges or self loops. Let $G = (V, E)$ be a graph. If the vertex set of a graph $G$ is not named explicitly we denote it by $V(G)$. Likewise, $E(G)$ is the set of the edges.

*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* $p$. If $p'$ and $p''$ are vertex disjoint paths in $G$, we denote that by $p' \perp p''$. If $u$ is an endpoint of $p'$, we denote that by writing $p'$ is a $u$-path . If $n \geq 2$, by $p - u_1$ we denote the path $u_2, e_2, \ldots, e_{n-1}$, $u_n$. *A cycle* $s$ in $G$ is a similar sequence of alternating vertices and edges with the only difference that two vertices, namely $u_1$ and $u_n$, coincide, and $n \geq 4$. *The length* of a path or cycle $z$ is the number of edges in it and is denoted by $|z|$, the set of vertices is denoted by $V(z)$, and the set of edges by $E(z)$. For any cycle $s$ and any vertex $u \in s$, we use the notation $s - u$ with the obvious meaning. Since we do not consider multigraphs, any path or cycle $z$ can be described uniquely by listing its distinct vertices in the order they occur in $z$. Any cycle $s$ has $2|s|$ equivalent descriptions. Any of them is *the order of the vertices in* $s$.

Let $G$ be a graph and $p = u_1, u_2, \ldots, u_n$ be a path in $G$. *A subpath* of a path $p$ is a contiguous subsequence of $p$ that is a path. To *break* $p$ *into subpaths* is to define subpaths of $p$, say $p_1$, $p_2$, ..., $p_t$, such that the vertices of $p$ are partitioned among them. There are $t - 1$ edges in $p$ that are not in any $p_i$, therefore $|p| = t - 1 + \sum_{i=1}^{t} |p_i|$. To *cover* $p$ *with subpaths* is to define subpaths of $p$, say $p^1$, $p^2$, ..., $p^s$, such that one endpoint of $p^1$ coincides with one endpoint of $p^2$, the other endpoint of $p^2$ coincides with one endpoint of $p^3$, etc., the other endpoint of $p^{s-1}$ coincides with one endpoint of $p^s$. Clearly, $|p| = \sum_{i=1}^{s} |p^i|$.

*The concatenation* of paths is the opposite of breaking. Let $q_1$, $q_2$, ..., $q_k$ be pairwise vertex disjoint paths in $G$, such that $q_i = u_i, \ldots, v_i$, for $1 \leq i \leq k$. Let $(v_i, u_{i+1}) \in E(G)$ for $1 \leq i \leq k-1$. *The concatenation of* $q_1$, $q_2$, ..., $q_k$ is the path $q = u_1, \ldots, v_1, u_2, \ldots, v_2, \ldots, u_k, \ldots, v_k$. Clearly, $|q| = k-1+\sum_{i=1}^{k} |q_i|$. *The chain* of paths is the opposite of covering. Let $q^1$, $q^2$, ..., $q^l$, where $q^i = x^i, y^i, \ldots, z^i$ for $1 \leq i \leq l$, be paths in $G$ that are vertex disjoint except that $x^{i+1} = z^i$ for $1 \leq i \leq l - 1$. *The chain of* $q^1$, $q^2$, ..., $q^l$ is the path $q = x^1, y^1, \ldots, z^1$, $y^2, \ldots, z^2, \ldots, z^{l-1}, y^l, \ldots, z^l$. Clearly, $|q| = \sum_{i=1}^{l} |q^i|$.

Let $G$ be a graph and $s = v_1, v_2, \ldots, v_n$ be a cycle in it. *A path in* $s$ is a path in $G$ such that its vertices form a contiguous subsequence of $s$. To *break* $s$ *into paths* is to define a set of paths in $s$, say $p_1$, $p_2$, ..., $p_t$, $1 \leq t \leq n$, such that the vertices of $s$ are partitioned among them. Clearly, $|s| = t + \sum_{i=1}^{t} |p_i|$. With respect to a fixed description of $s$, the paths can be defined uniquely only by their lengths. Any sequence of nonnegative integers $c = c_1, c_2, \ldots, c_t$, such that $1 \leq t \leq n$ and $t+\sum_{i=1}^{t} c_i = n$ is said to *define a breaking of* $s$ *into paths*, the paths being $p_1 = v_1, v_2, \ldots, v_{c_1+1}$, etc., $p_t = v_{c_1+c_2+\ldots+c_{t-1}+t}, v_{c_1+c_2+\ldots+c_{t-1}+t+1}, \ldots, v_n$.

*A circular array* is any array $A[1 \dots n]$ such that $n \geq 3$ and $A[1]$ and $A[n]$ are neighbours in the array. In the context of circular arrays, the distance between any two positions is defined as follows. For any $i, j \in \{1, 2, \dots, n\}$, $\text{dist}(i, j) = \max\{|i - j|, n - |i - j|\}$.

*A cactus graph*, or cactus for short, is a connected graph in which every edge is in at most one cycle. A cactus $G$ in which one vertex is distinguished as *the root* is called *a rooted cactus.* The root is denoted by $\text{root}(G)$. Assume $u = \text{root}(G)$ and $u$ is a vertex in $r$ cycles $s_1, s_2, \dots, s_r$ and there are $d + r$ connected components after deleting $u$ from $G$. Those components are called *the minions of* $u$. The $d$ minions that are connected to $u$ by single edges are *the tree minions* and the other are *the cycle minions*. For any minion $G'$ we write $G' + u$ to denote the subgraph of $G$ induced by $V(G') \cup \{u\}$. Every tree minion is a rooted cactus, its root being the vertex adjacent to $u$. The cycle minions are not rooted. In every cycle minion $H^i$, the path $s_i - u$ is *the basis*. The connected components after removing all the edges of the basis from any cycle minion are rooted cacti, the roots being the corresponding vertices from that basis. See Figure  on the facing page for illustration. The basis of $H^1$, for instance, is the path $w_1^1, w_2^1, \dots, w_{t_1}^1$. For every vertex $x \in V(G)$ there is a subcactus rooted at $x$ that we denote by $G[x]$. For instance, on Figure , $G[w_2^1] = H_2^1$. The roots of the tree minions are *the tree children of* $u$. The tree children and the vertices in the bases of the cycle minions are *the children of* $u$. On Figure  the tree children of $u$ are $v_1, \dots, v_d$, and the children of $u$ are $v_1, \dots, v_d, w_1^1, \dots, w_{t_1}^1, \dots, w_1^r, \dots, w_{t_r}^r$. A vertex with no children is *a leaf*.

For any cactus $G$ and any $u \in V(G)$, by $\text{longest}(G)$ we denote the length of any longest path in $G$ and by $\text{longest}(G, u)$, the length of any longest $u$-path in $G$. A *labeled rooted cactus* is a rooted cactus $G'$ such that every vertex has a *label* associated with it. The label of any $u \in V(G')$ is the ordered pair of nonnegative integers $(l_1(u), l_2(u))$ such that $l_1(u) = \text{longest}(G'[u])$ and $l_2(u) = \text{longest}(G'[u], u)$.

## 2. The algorithm, its verification, and complexity analysis.

Our main procedure is called Longest Cactus.

```
Longest Cactus(G: labeled rooted cactus, u: the root of G)
1  (* Computes the length of a longest path in G *)
2  choose any vertex u ∈ V(G)
3  turn G into a rooted cactus with root u
4  Compute Label(G, u)
5  return l_1(u)
```

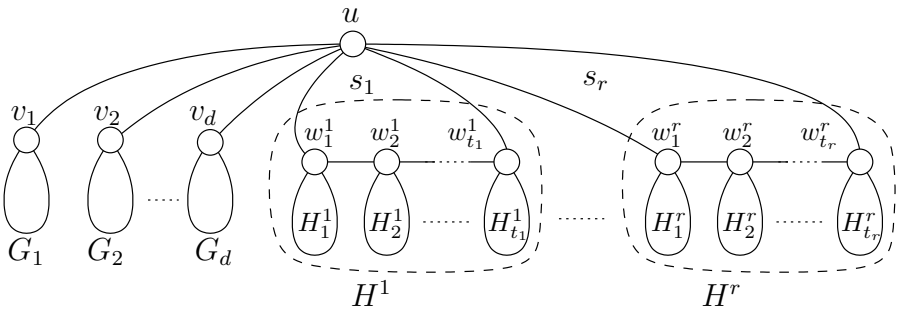


Fig. 1. The rooted cactus from COMPUTE LABEL

The procedure COMPUTE LABEL is the gist of the algorithm. The rooted cactus it refers to is shown on Figure 1. We use the notation "$\{\}_{\mathrm{M}}$" for multisets. The procedure calls another procedure AUX defined below.

```
COMPUTE LABEL(G: labeled rooted cactus, u: the root of G)
 1  (* Computes the label of the root *)
 2  if u a leaf
 3      (l_1(u), l_2(u)) ← (0, 0)
 4  else
 5      let the tree children of u be v_1, v_2, ..., v_d
 6      let the cycle minions of u be H^1, ..., H^r
 7      let the basis of H^i be w_1^i, w_2^i, ..., w_{t_i}^i for 1 ≤ i ≤ r
 8      for every child a of u do
 9          COMPUTE LABEL(G[a], a)
10      P ← {l_2(v_i) + 1 | 1 ≤ i ≤ d}_M
11      for i ← 1 to r do
12          q_i ← max{l_2(w_k^i) + max{k, t_i − k + 1} | 1 ≤ k ≤ t_i}
13      Q ← {q_i | 1 ≤ i ≤ r}_M
14      for i ← 1 to r do
15          z_i ← AUX([0, l_2(w_1^i) .. , l_2(w_{t_i}^i)])
16      x ← max{P ∪ Q}_M
17      y ← second-max{P ∪ Q}_M
18      z ← max{z_i | 1 ≤ i ≤ r}
19      m ← max{l_1(a) | a is a child of u}
20      l_1(u) ← max{x + y, z, m}
21      l_2(u) ← x
```

```
AUX(A[1...n]: circular array of nonnegative integers)
 1  (* Returns the maximum number t such that for some *)
 2  (*  i,j ∈ {1,2,...,n}, i ≠ j, t = A[i] + A[j] + dist(i,j). *)
 3  let B[0...n] and C[1...n] be linear arrays of nonnegative integers
 4  B[0] ← 0
 5  for i ← 1 to n do
 6      B[i] ← max{B[i−1], A[i] − (i−1)}
 7      C[i] ← B[i−1] + A[i] + (i−1)
 8  x ← max{C[i] | 1 ≤ i ≤ n}
 9  for i ← 1 to n do
10      B[i] ← max{B[i−1], A[i] + (i−1)}
11  C[n] ← A[n] + 1
12  for i ← n−1 downto 2 do
13      C[i] ← max{C[i+1], A[i] + n − (i−1)}
14  y ← max{B[i] + C[i+1] | 1 ≤ i ≤ n−1}
15  return max{x, y}
```

**Lemma 1.** *Whenever the execution of the first* ***for*** *loop (lines 5–7) of* AUX *is at line 7 and* $i \geq 2$, $C[i]$ *is assigned* $\max\{A[k] + A[i] + i - k \mid 1 \leq k < i\}$.

Proof. It is fairly obvious that at line 7 the value $B[i-1]$ is such that

$$B[i-1] = \begin{cases} 0, \text{ if } A[k] - (k-1) \leq 0 \ \ \forall k \text{ such that } 1 \leq k < i \\ \max\{A[k] - (k-1) \mid 1 \leq k < i\}, \text{ else} \end{cases}$$

However, $A[1] - (1-1)$ cannot be negative, therefore there is at least one nonnegative value in the sequence $A[k] - (k-1), 1 \leq k < i$, so we can say simply that $B[i-1]$ at line 7 is $B[i-1] = \max\{A[k] - k + 1 \mid 1 \leq k < i\}$. It follows that indeed $C[i]$ is assigned the value $\max\{A[k] - k + 1 \mid 1 \leq k < i\} + A[i] + i - 1 = \max\{A[k] + A[i] + i - k \mid 1 \leq k < i\}$. □

**Corollary 1.** $x$ *is assigned the value*

$$\max\{A[i] + A[j] + j - i \mid 1 \leq i < j \leq n\}$$

*at line* 8 *of* AUX.

**Lemma 2.** $y$ *is assigned the value*

$$\max\{A[i] + A[j] + n - (j-i) \mid 1 \leq i < j \leq n\}$$

*at line* 14 *of* AUX.

P r o o f. Consider the **for** loop at lines 9–10. Since $A[1] + (1 - 1) \geq 0$, it is the case that $\forall i, 1 \leq i \leq n, B[i] = \max\{A[k] + (k-1) \mid 1 \leq k \leq i\}$, whenever that **for** loop terminates.

Now consider the **for** loop at lines 12–13. Think of the assignment at line 11 as $C[n] = A[n] + n - (n-1)$. Having that in mind, it is fairly obvious that after that **for** loop terminates, it is the case that

$$C[i] = \max\{A[k] + n - (k-1) \mid i \leq k \leq n\}, \forall i, 2 \leq i \leq n.$$

From these two considerations it follows immediately that at line 14, $y$ is assigned the value $\max\{A[i] + (i-1) + A[j] + n - (j-1) \mid 1 \leq i < j \leq n\} = \max\{A[i] + A[j] + n - (j-i) \mid 1 \leq i < j \leq n\}$. □

**Lemma 3.** *Procedure* Aux *is correct.*

P r o o f. It follows immediately from Corollary 1 and Lemma 2 that Aux indeed returns the maximum number $t$ such that for some $i, j \in \{1, 2, \ldots, n\}, i \neq j$, $t = A[i] + A[j] + \text{dist}(i, j)$. □

**Lemma 4.** *Using the naming convention suggested by Figure , for any $i$ such that $1 \leq i \leq r$, for any longest $u$-path $p$ in $H^i + u$,*

$$|p| = \max\{\mathit{longest}(H^i_k, w^i_k) + \max\{k, t_i + 1 - k\} \mid 1 \leq k \leq t_i\}$$

P r o o f. First note that because the graph is a cactus, any $u$-path $p$ of positive length in $H^i + u$ is the chain of precisely two subpaths, one of them—call it $p'_k$—a path in $s$ with endpoints $u$ and $w^i_k$ for some index $k$ such that $1 \leq k \leq t_i$, and the other one—call it $p''_k$—a $w^i_k$-path in $H^i_k$. Clearly, $|p| = |p'_k| + |p''_k|$. For any $k$, $1 \leq k \leq t_i$, there are two choices for $p'_k$: one with length $k$ and the other one with length $t_i + 1 - k$. Furthermore, the choice of $p''_k$ does not depend on the choice of $p'_k$. Obviously, $|p|$ is maximised when $p'_k$ and $p''_k$ are maximised independently. □

**Lemma 5.** Compute Label *computes correctly the label of $G[u]$.*

P r o o f. By structural induction on $G$. The basis is when the root is a leaf and it obviously holds. Assume Compute Label computes correctly the label of every child of $u$. We first prove that $l_2(u)$ is computed correctly. Let $p$ be any longest $u$-path in $G$ and $p' = p - u$. $p'$ is entirely in some minion of $u$. First assume $p'$ is in some tree minion $G_i$. In this case $p'$ is a longest path in $G_i$

with one endpoint $v_i$. By the induction hypothesis, COMBINE LABEL correctly computes the label of $G_i$, therefore $l_2(v_i) = |p'|$. Note that $|p| = |p'| + 1$, therefore $|p| = l_2(v_i) + 1$, and the maximum element of the multiset $P$ (line 10) is precisely $l_2(v_i) + 1$. Via the assigments at lines 16 and 21, $l_2(u)$ is set to $l_2(v_i) + 1$ and that equals $|p|$.

Now assume $p'$ is in some cycle minion $H^i$. But then $p$ is a longest $u$-path in $H^i + u$, so by Lemma 4, $|p| = \max_{1 \le k \le t_i} \{\max\{k, t_i + 1 - k\} + \text{longest}(H^i_k, w^i_k)\}$. By the induction hypothesis $\text{longest}(H^i_k, w^i_k) = l_2(w^i_k)$, therefore the maximum element of the multiset $Q$ (lines 12 and 13) is precisely $|p|$. Via the assignments at lines 16 and 21, $l_2(u)$ is set to $|p|$.

Next we prove that $l_1(u)$ is computed correctly. Let $q$ be any longest path in $G$. The following five possibilities are exhaustive.

1. $u$ is an internal vertex in $q$ and the endpoints of $q$ are in two different minions $J_1$ and $J_2$ of $u$.
2. $u$ is an internal vertex in $q$ and both endpoints of $q$ are in the same minion of $u$. This minion must be a cycle minion, say $H^j$.
3. $u$ is an endpoint of $q$. In this case $u$ must have a single minion.
4. $u \notin q$ and for a single child $u'$ of $u$, $q$ is in $G[u']$.
5. $u \notin q$ and there are at least two children of $u$ such that $q$ contains vertices from the subcacti rooted at them. Clearly, all such children are in the same cycle minion $H^j$ and they form a subpath in the basis of $H^j$.

First assume 1 is the case. If both $J_1$ and $J_2$ are tree minions of $u$, then $q$ is a concatenation of three paths: $q_1$, $q'$, and $q_2$, such that $q_i$ is a longest $\text{root}(J_i)$-path in $J_i$ for $i = 1, 2$, and $q'$ is the single vertex $u$. In this subcase $|q| = |q_1| + |q_2| + 2$. By the induction hypothesis, $|q_1| + 1$ and $|q_2| + 1$ are two maximum elements in the multiset $P$ (line 10). Those two values are assigned to $x$ and $y$ (lines 16 and 17) and then their sum is assigned to $l_1(u)$ at line 20.

If $J_1$ is a tree minion and $J_2$ is a cycle minion of $u$, then $q$ is a concatenation of two paths $q_1$ and $q_2$, such that $q_1$ is a longest $\text{root}(J_1)$-path in $J_1$ and $q_2$ is a longest $u$-path in $J_2 + u$. In this subcase $|q| = |q_1| + 1 + |q_2|$. By the induction hypothesis, $|q_1| + 1$ is a maximum element in $P$. By Lemma 4 and the induction hypothesis, $|q_2|$ is a maximum element in $Q$. These two values are assigned to $x$ and $y$ (lines 16 and 17) and their sum is assigned to $l_1(u)$ at line 20.

If $J_1$ and $J_2$ are cycle minions of $u$, then $q$ is the chain of two paths $q_1$ and $q_2$, such that $q_i$ is a longest $u$-path in $J_i + u$, for $i = 1, 2$. In this subcase

$|q| = |q_1| + |q_2|$. By Lemma 4 and the induction hypothesis, $|q_1|$ and $|q_2|$ are two maximum elements in $Q$. Via lines 16, 17, and 20, the sum of these two values is assigned to $l_1(u)$. Note that $x$ and $y$ cannot possibly be the lengths of paths that are in the same (cycle) minion.

Now assume 2 is the case. Clearly, $q$ is a longest path in $H^j + u$ that contains at least one edge of $s_j$. Consider the **for** loop at lines 14–15 of COMPUTE LABEL. Think of the array $[0, l_2(w_1^i), \ldots, l_2(w_{t_i}^i)]$ as a circular array and apply Lemma 3. Conclude that after each call to AUX (line 15), $z_i$ is a maximum sum of any two distinct elements of the circular array plus the distance between them. Clearly, that equals the length of a longest path in $H^i + u$ that contains an edge of $s_i$. It follows that at line 18, the value assigned to $z$ is precisely $|q|$. Therefore, the value assigned to $l_1(u)$ at line 20 is $|q|$.

Now assume 3 is the case. First assume the only minion of $u$ is a tree one, namely $G_1$. Let $q' = q - u$. Clearly, $q'$ is a longest $v_1$-path in $G_1$. By the induction hypothesis, $P$ is $\{|q'| + 1\}_{\mathrm{M}}$ (line 10), so at line 16, $x = |q'| + 1$, that is $|q|$, and at line 17, $y = 0$. Then at line 20 the value assigned to $l_1(u)$ is precisely $|q|$. Now assume the only minion of $u$ is a cycle one, namely $H^1$. It is obvious $q$ contains an edge from $s_1$. Using considerations similar to case 2 above, we conclude that $|q|$ is assigned to $z$ at line 18, and the value assigned to $l_1(u)$ at line 20 is $|q|$.

Now assume 4 is the case. By the induction hypothesis, $l_1(u') = |q|$. So at line 19, $|q|$ is assigned to $m$ and the value assigned to $l_1(u)$ at line 20 is $|q|$.

Finally, assume 5 is the case. Since $q$ contains a cycle edge $s_j$, we use considerations similar to case 2 above to conclude that $|q|$ is assigned to $z$ at line 18, and the value assigned to $l_1(u)$ at line 20 is $|q|$. □

It is trivial to conclude that since COMPUTE LABEL computes correctly the label of the whole rooted cactus (Lemma 5), LONGEST CACTUS computes the length of a longest path in it.

**Lemma 6.** *The recurrence relation*

$$T(1) = \Theta(1)$$
$$T(n) = \sum_{i=1}^{m} T(n_i) + \Theta(m) \tag{1}$$

*for any numerical partition $n_1, n_2, \ldots, n_m$ of $n - 1$ has solution $T(n) = \Theta(n)$.*

Proof. We prove that $T(n) = O(n)$ by induction on $n$. Assume there are positive constants $b$ and $c$ such that $T(n) \leq c.n - b$. Assume the bigger of the

two hidden constants in the "$\Theta(m)$" expression is $k$. By the induction hypothesis,

$$\begin{aligned} T(n) &\leq \sum_{i=1}^{m}(c.n_i - b) + k.m \\ &= c\sum_{i=1}^{m}(n_i) - b.m + k.m \\ &= c(n-1) + m(k-b) \\ &= c.n - c + m(k-b) \\ &\leq c.n - b, \quad \text{if } k - b < 0 \text{ and } c > b \ . \end{aligned}$$

The fact that $T(n) = \Omega(n)$ is obvious. □

**Lemma 7.** Compute Label *runs in time* $\Theta(|V(G)|)$.

P r o o f. Assume $T(n)$ denotes the number of elementary operations performed by Compute Label on a cactus with $n$ vertices. Assume $m$ is the number of the chidren of $u$ and there are $n_i$ vertices in the subcactus rooted at child number $i$, for $1 \leq i \leq m$. Then $\sum_{i=1}^{m} n_i = n - 1$ since the vertices of $G$ are partitioned among $u$ and the subcacti rooted at the children of $u$. Furthermore, $\sum_{i=1}^{m} T(n_i)$ is the number of elementary operations performed by the recursive calls of Compute Label (line 9).

It is obvious the execution of lines 10–13 and 16–21 takes time $\Theta(m)$. Note that Aux runs in linear time and conclude the overall running time of the **for** loop at lines 14–15 is

$$\Theta(t_1 + 1) + \Theta(t_2 + 1) + \ldots + \Theta(t_r + 1) = \Theta\left(\sum_{i=1}^{r} t_i\right) = \Theta(m) \ .$$

Therefore, once the recursive calls at line 9 are done, Compute Label works in $\Theta(m)$ time. Apply Lemma 6 to conclude the running time of Compute Label is $\Theta(|V(G)|)$. □

**3. Longest paths in weighted cacti.** It is easy to see our algorithm can be modified to compute a longest path as well without violating the linear running time. Furthermore, it can be modified to compute in linear time the length of a longest path or the path itself of weighted cacti. In that case, every vertex or edge $x$ has a nonnegative real *weight* called $w(x)$, and the length of

a path is the sum of the weights of its edges and vertices. [2] describes how to accomplish the linear time computation on weighted cacti.

**4. Conclusions and future work.** It is natural to try to extend the idea of the algorithm that uses labels to more general classes of graphs with tree-like structure, for instance outerplanar graphs, which means cycles with chords. The chordless cycles of the cacti allow us to compute in constant time the minimum or maximum distance between any two vertices in the cycle, provided the vertices are indexed in accordance with their order in the cycle. If chords are present we need additional computation to find these distances, possibly increasing the asymptotic running time.

## REFERENCES

[1] Alon N., R. Yuster, U. Zwick. Color-coding. *J. ACM*, **42** (1995), No 4, 844–856.

[2] Andreica M. I., N. Ţăpuş. Central placement of storage servers in tree-like content delivery networks. In: Proc. of the IEEE Eurocon 2009, ISBN 978-1-4244-3860-0, 1901–1908. `https://mail.cs.pub.ro/~mugurel.andreica/publications/[eurocon2009]central_placement_storage_servers_tree-like_CDNs/[eurocon2009]Andreica_Tapus-StorageServers_CDN_TreeLike.pdf`.

[3] Björklund A., T. Husfeldt. Finding a path of superlogarithmic length. *SIAM J. Comput.*, **32** (2003), No 6, 1395–1402.

[4] Björklund A., T. Husfeldt, P. Kaski, M. Koivisto. Narrow sieves for parameterized paths and packings. Computing Research Repository, abs/1007.1161, 2010. `http://arxiv.org/abs/1007.1161`.

[5] Bulterman R. W., F. W. van der Sommen, G. Zwaan, T. Verhoeff, A. J. M. van Gasteren, W. H. J. Feijen. On computing a longest path in a tree. *Information Processing Letters*, **81** (2002), No 2, 93–96.

[6] Dorn F., F. V. Fomin, D. M. Thilikos. *Catalan structures and dynamic programming in h-minor-free graphs.* In: SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 2008, Society for Industrial and Applied Mathematics, 631–640.

[7] Downey R. G., M. R. Fellows. Parameterized Complexity. Springer, 1999.

[8] Gabow H. N. Finding paths and cycles of superpolylogarithmic length. *SIAM Journal of Comput.*, **36** (2007), No 6, 1648–1671. www.cs.colorado.edu/~hal/u.pdf

[9] Gabow H. N., S. Nie. Finding long paths, cycles and circuits. In: ISAAC '08: Proceedings of the 19th International Symposium on Algorithms and Computation, Springer-Verlag, Berlin, Heidelberg, 2008, 752–763.

[10] Garey M., D. Johnson. Computers and Intractability. W. H. Freeman and Co., New York, USA, 1979.

[11] Karger D. R., R. Motwani, G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, **18** (1997), No 1, 82–98.

[12] Koutis I. Faster algebraic algorithms for path and packing problems. In: Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I, ICALP '08, Berlin, Heidelberg, 2008, Springer-Verlag, 575–586. http://www.cs.cmu.edu/~jkoutis/papers/MultilinearDetection.pdf

[13] Monien B. How to find long paths efficiently. Annals of Discrete Mathematics, **25** (1985), 239–254.

[14] Uehara R., Y. Uno. On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, **18** (2007), No 5, 911–930.

[15] Williams R. Finding paths of length $k$ in $O^*(2^k)$ time. *Information Processing Letters*, **109** (2009), No 6, 315–318.

[16] Zhang Z., H. Li. Algorithms for long paths in graphs. *Theoretical Computer Science*, **377** (2007), No 1–3, 25–34.

*Minko Markov*
*Krassimir Manev*
*Department of Computing Systems*
*Faculty of Mathematics and Informatics*
*"St. K. Ohridski" University of Sofia*
*5, J. Bourchier Blvd, P.O. Box 48*
*1164 Sofia, Bulgaria*
*e-mail:* minkom@fmi.uni-sofia.bg
*e-mail:* manev@fmi.uni-sofia.bg

*Mugurel Ionuţ Andreica*
*Nicolae Ţăpuş*
*Computer Science Department*
*Politehnica University of Bucharest*
*Splaiul Independenţei 313, sector 6*
*Bucharest, Romania*
*e-mail:* mugurel.andreica@cs.pub.ro

*Received March 21, 2012*
*Final Accepted June 7, 2012*