アルゴリズム 42−12
(1994. 11. 18)

木の一般化ランク付け

周 暁　永井 伸明　西関 隆夫
東北大学大学院情報科学研究科

概要

本論文ではグラフの点ランク付けの一般化として，新たに c-ランク付けを定義する．即ち，グラフ G の c-ランク付けとは，G の点に整数のランクを付けて，しかも任意のランク i について，G からランクが i より大きい点を全て除去すると，どの連結成分にもランク i の点が高々 c 個しかないようにすることである．明らかに普通の点ランク付けは 1-ランク付けである．本文では与えられた木を，最小のランク数で c-ランク付けする線形時間のアルゴリズムを与える．

Generalized Rankings of Trees

Xiao Zhou, Nobuaki Nagai and Takao Nishizeki [1]
Department of System Information Sciences
Graduate School of Information Sciences
Tohoku University, Sendai 980-77, Japan

Abstract

In this paper we newly define a generalized vertex-ranking of a graph G as follows: for a positive integer c, a c-vertex-ranking of G is a labeling (ranking) of the vertices of G with integers such that, for any label i, every connected component of the graph obtained from G by deleting the vertices with label $> i$ has at most c vertices with label i. Clearly an ordinary vertex-ranking is a 1-vertex-ranking. We present a linear algorithm to find a c-vertex-ranking of a given tree using a minimum number of ranks for any bounded integer c.

---

[1] E-mail:(zhou|nishi)@ecei.tohoku.ac.jp, nagai@anakin.nishizeki.ecei.tohoku.ac.jp

1 Introduction

A *vertex-ranking* of a graph G is a labeling (ranking) of vertices of G with integers such that any path between two vertices with the same label i contains a vertex with label $j > i$. The *vertex-ranking problem* is to find a vertex-ranking of a given graph G using the minimum number of ranks (labels). The vertex-ranking problem is NP-complete in general [BDJ+94, Pot88]. On the other hand Schäffer has given a linear algorithm to solve the vertex-ranking problem for trees [Sch89]. Very recently Bodlaender *et al.* have given a polynomial-time algorithm to solve the vertex-ranking problem for graphs with bounded treewidth [BDJ+94]. The problem of finding an optimal vertex-ranking of G has applications in VLSI layout and in scheduling the manufacture of complex multi-part products [Sch89, IRV88]; it is equivalent to finding the minimum height vertex separator tree of G.

In this paper we newly define a generalization of an ordinary vertex-ranking. For a positive integer c, a *c-vertex-ranking* (or a *c-ranking* for short) of a graph G is a labeling of the vertices of G with integers such that, for any label i, every connected component of the graph obtained from G by deleting the vertices with label$> i$ has at most c vertices with label i. Clearly an ordinary vertex-ranking is a 1-vertex-ranking. The integer label of a vertex is called the *rank* of the vertex. The minimum number of ranks needed for a c-vertex-ranking of G is called the *c-vertex-ranking number* (or the *c-ranking number* for short) and denoted by $r_c(G)$. A c-ranking of G using $r_c(G)$ ranks is called an *optimal c-ranking* of G. The *c-ranking problem* is to find an optimal c-ranking of a given graph G. The problem is also NP-complete in general since the ordinary vertex-ranking problem is NP-complete [BDJ+94, Pot88]. Figure 1 depicts an optimal 3-ranking of a tree using three ranks, where vertex numbers are drawn in circles and ranks next to circles.

Consider the process of starting with a connected graph and partitioning it recursively by removing at most c vertices and incident edges from each of the remaining connected subgraphs until the graph becomes empty. The tree representing the recursive decomposition is called a c-vertex separator tree. Thus a c-vertex separator tree corresponds to a parallel computation scheme based on the process above. The c-vertex-ranking problem is equivalent to finding a c-vertex separator tree of the minimum height. Figure 2 illustrates a 3-vertex sparator tree of the tree depicted in Figure 1, where deleted vertex numbers are drawn in ovals.

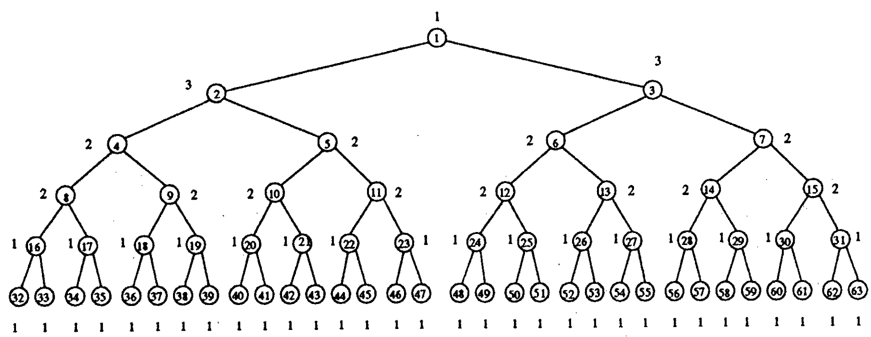

Figure 1: An optimal 3-vertex-ranking φ of a tree T.

In this paper we give a linear algorithm to solve the c-ranking problem on trees for any positive bounded integer c. Our algorithm uses techniques employed by Schäffer [Sch89] and Iyer *et al.* [IRV88] for the ordinary vertex-ranking problem as well as new techniques specific

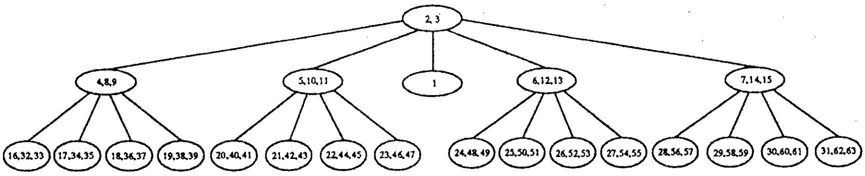

Figure 2: A 3-vertex sparator tree of the tree in Figure 1.

to the c-ranking problem.

2 Preliminaries

In this section we define some terms and present easy observations. Let $T = (V, E)$ denote a tree with vertex set V and edge set E. We often denote by $V(T)$ and $E(T)$ the vertex set and the edge set of T, respectively. We denote by n the number of vertices in T. T is a "free tree," but we regard T as a "rooted tree" for convenience sake: an arbitrary vertex of tree T is designated as the *root* of T. We will use notions as: root, internal vertex, child and leaf in their usual meaning. An edge joining vertices u and v is denoted by (u, v). The maximal subtree of T rooted at vertex v is denoted by $T(v)$.

The definition of a c-ranking immediately implies the following lemma.

Lemma 1. *Any c-ranking of a connected graph labels at most c vertices with the largest rank.*

For integers α and β, $\alpha \leq \beta$, we denote by $[\alpha, \beta]$ the set of integers between α and β, that is, $[\alpha, \beta] = \{\alpha, \alpha + 1, \cdots, \beta\}$. Let $[\alpha, \beta] = \phi$ if $\alpha > \beta$. Let φ be a c-ranking of a tree T. The number of ranks used by φ is denoted by $\#\varphi$. One may assume without loss of generality that φ uses the ranks in set $[1, \#\varphi]$. A vertex v of T and its rank $\varphi(v)$ are *visible* (from the root under φ) if all the vertices in the path from the root to v have ranks $\leq \varphi(v)$. Thus the root of T and $\#\varphi$ are visible. Denote by $L(\varphi)$ the list of ranks of all visible vertices, and call $L(\varphi)$ the *list of a c-ranking* φ of the rooted tree T. For an integer γ we denote by $count(L(\varphi), \gamma)$ the number of γ's contained in $L(\varphi)$, i.e., the number of visible edges with rank γ. The ranks in the list $L(\varphi)$ are sorted in non-increasing order. Thus the c-ranking φ in Figure 1 has the list $L(\varphi) = \{3, 3, 1\}$, $count(L(\varphi), 3) = 2$, $count(L(\varphi), 2) = 0$ and $count(L(\varphi), 1) = 1$. One can easily observe that $count(L(\varphi), \gamma) \leq c$ for each rank γ.

We define the *lexicographical order* $\prec$ on the set of non-increasing sequences (lists) of positive integers as follows: let $A = \{a_1, \cdots, a_p\}$ and $B = \{b_1, \cdots, b_q\}$ be two sets (lists) of positive integers such that $a_1 \geq \cdots \geq a_p$ and $b_1 \geq \cdots \geq b_q$, then $A \prec B$ if and only if there exists an integer i such that

(a) $a_j = b_j$ for all $1 \leq j < i$, and

(b) either $a_i < b_i$ or $p < i \leq q$.

We write $A \preceq B$ if $A = B$ or $A \prec B$. The non-increasing list obtained by merging two lists A and B is denoted by $A + B$. $A - [\alpha, \beta]$ denotes the list obtained from A by deleting all ranks $\gamma \in [\alpha, \beta]$. Obviously if $A \preceq B$ then $A - [1, \alpha] \preceq B - [1, \alpha]$ for any $\alpha \geq 1$.

A c-ranking φ of T is *critical* if $L(\varphi) \preceq L(\eta)$ for any c-ranking η of T. The optimal c-ranking depicted in Figure 1 is indeed critical. The list of a critical c-ranking of T is called the *critical list of tree* T and denoted by $L^*(T)$. Clearly any critical c-ranking is optimal, and $L^*(T)$ corresponds to an equivalent class of optimal c-rankings of T.

For a c-ranking φ of tree T and a subtree T' of T, we denote by $\varphi|T'$ a restriction of φ to $V(T')$: let $\varphi' = \varphi|T'$, then $\varphi'(v) = \varphi(v)$ for $v \in V(T')$.

3 Optimal c-Ranking

The main result of the paper is the following theorem.

Theorem 2. *An optimal c-ranking of a tree T can be found in linear time for any bounded integer c.*

In the remaining of this section we give a linear algorithm for finding a critical c-ranking of a tree T. Our algorithm uses the technique of "bottom-up tree computation." For each internal vertex u of a tree T, we construct a critical c-ranking of $T(u)$ from those of the subtrees rooted at u's children.

One can easily observe the following lemma.

Lemma 3. *Any tree T of n vertices has at most c vertices whose removal leaves subtrees each having no more than $2n/(c+3)$ vertices.*

By Lemma 3 we have the following lemma.

Lemma 4. *Every tree T of n vertices satisfies $r_c(T) \leq \lceil \log_{\frac{c+3}{2}} n \rceil + 1$.*

Proof. Recursively applying Lemma 3, one can construct a c-partition tree of height $h(n)$ satisfying the following recurrence relation

$$h(n) \leq 1 + h\left(\frac{2}{c+3}n\right).$$

Solving the recurrence, we have $h(n) \leq \lceil \log_{\frac{c+3}{2}} n \rceil$. Note that $h(1) = 0$. Hence $r_c(T) \leq h(n) + 1 \leq \lceil \log_{\frac{c+3}{2}} n \rceil + 1$. $\mathcal{Q.E.D.}$

Let $d(u)$ be the number of children of vertex u in T, and let $v_1, v_2, \cdots, v_{d(u)}$ be the children of u. Our idea is that a critical c-ranking of $T(u)$ can be constructed from any critical c-rankings φ_i of $T(v_i)$, $i = 1, 2, \cdots, d(u)$. One can easily observe that a vertex-labeling η of $T(u)$ is a c-ranking of $T(u)$ if and only if there are no more than c visible vertices of the same rank under η and $\eta|T(v_i)$ is a c-ranking of $T(v_i)$ for every i, $1 \leq i \leq d(u)$.

We first have the following lemma.

Lemma 5. *$T(u)$ has a critical c-ranking η such that $\eta|T(v_i) = \varphi_i$ for every i, $1 \leq i \leq d(u)$.*

Proof. Let η be an arbitrary critical c-ranking of $T(u)$. Clearly $L(\eta|T(v_i)) \succeq L(\varphi_i)$ for each i, $1 \leq i \leq d(u)$. If $L(\eta|T(v_i)) \succ L(\varphi_i)$, then let γ_i be an integer such that

(a) $L(\eta|T(v_i)) - [1, \gamma_i] = L(\varphi_i) - [1, \gamma_i]$, and

(b) $count(L(\eta|T(v_i)), \gamma_i) > count(L(\varphi_i), \gamma_i)$.

Otherwise let $\gamma_i = 0$. Let $\gamma_{\max} = \max\{\gamma_i \mid 1 \leq i \leq d(u)\}$. Construct a vertex-labeling η' of $T(u)$ from η and φ_i as follows:

$$\eta'(v) = \begin{cases} \max\{\eta(u), \gamma_{\max}\} & \text{if } v = u; \text{ and} \\ \varphi_i(v) & \text{if } v \in V(T(v_i)) \text{ and } i \in [1, d(u)]. \end{cases}$$

Since there is no visible rank $< \gamma_{\max}$ under η', η' is a c-ranking of $T(u)$. Since $L(\eta') \preceq L(\eta)$, η' is a critical c-ranking of $T(u)$ and $\eta'|T(v_i) = \varphi_i$ for all i, $1 \leq i \leq d(u)$. $\mathcal{Q.E.D.}$

Let $m = \max\{\#\varphi_i \mid 1 \le i \le d(u)\}$, then we have the following lemma.

Lemma 6. *$r_c(T(u)) = m$ or $m+1$.*

Proof. Clearly $m \le r_c(T(u))$. Therefore it suffies to prove that $r_c(T(u)) \le m+1$. One can extend φ_i, $1 \le i \le d(u)$, to a c-ranking η of $T(u)$ as follows:

$$\eta(v) = \begin{cases} m+1 & \text{if } v = u; \text{ and} \\ \varphi_i(v) & \text{if } v \in V(T(v_i)) \text{ and } i \in [1, d(u)]. \end{cases}$$

Thus $r_c(T(u)) \le \#\eta = m+1$. $\mathcal{Q.E.D.}$

The following Lemma 7 gives a necessary and sufficient condition for $r_c(T(u)) = m$.

Lemma 7. *$r_c(T(u)) = m$ if and only if there is a rank $\alpha \in [1, m]$ such that*
(a) $\sum_{i=1}^{d(u)} count(L(\varphi_i), \alpha) \le c-1$
and
(b) $\sum_{i=1}^{d(u)} count(L(\varphi_i), \gamma) \le c$ *for all ranks* $\gamma \in [\alpha+1, m]$.

Proof. $\Longleftarrow$: One can easily extend the cirtical c-rankings φ_i to a c-ranking η of $T(u)$ with $\#\eta = m$ as follows:

$$\eta(v) = \begin{cases} \alpha & \text{if } v = u; \text{ and} \\ \varphi_i(v) & \text{if } v \in V(T(v_i)) \text{ and } i \in [1, d(u)]. \end{cases}$$

Therefore $r_c(T(u)) \le \#\eta = m$, and hence by Lemma 6 $r_c(T(u)) = m$.

$\Longrightarrow$: Suppose that $r_c(T(u)) = m$. By Lemma 5 there is a c-ranking η of $T(u)$ such that $\eta|T(v_i) = \varphi_i$ for each i, $1 \le i \le d(u)$. Let $\alpha = \eta(u)$, then (a) and (b) above hold since η is a c-ranking of $T(u)$. $\mathcal{Q.E.D.}$

In order to find a critical c-ranking η of $T(u)$ from φ_i, $i = 1, 2, \cdots, d(u)$, we need the following two lemmas.

Lemma 8. *If $r_c(T(u)) = m+1$, then*

$$\eta(v) = \begin{cases} m+1 & \textit{if } v = u; \textit{ and} \\ \varphi_i(v) & \textit{if } v \in V(T(v_i)) \textit{ and } i \in [1, d(u)] \end{cases}$$

is a critical c-ranking of $T(u)$ and $L(\eta) = \{m+1\}$.

Proof. immediate. $\mathcal{Q.E.D.}$

Lemma 9. *If $r_c(T(u)) = m$, then*

$$\eta(v) = \begin{cases} \alpha & \textit{if } v = u; \textit{ and} \\ \varphi_i(v) & \textit{if } v \in V(T(v_i)) \textit{ and } i \in [1, d(u)] \end{cases}$$

is a critical c-ranking of $T(u)$, where $\alpha \in [1, m]$ is the minimum integer such that
(a) $\sum_{i=1}^{d(u)} count(L(\varphi_i), \alpha) \le c-1$
and
(b) $\sum_{i=1}^{d(u)} count(L(\varphi_i), \gamma) \le c$ *for every rank* $\gamma \in [\alpha+1, m]$.
Furthermore $L(\eta) = \sum_{i=1}^{d(u)} L(\varphi_i) - [1, \alpha-1] + \{\alpha\}$.

Proof. By Lemma 5 there is a critical c-ranking η' of $T(u)$ such that $L(\eta'|T(v_i)) = L(\varphi_i)$ for every i, $1 \le i \le d(u)$. Since $\alpha = \eta(u)$ is the minimum integer satisfying (a) and (b) above, $L(\eta) \preceq L(\eta')$ and hence η is a critical c-ranking of $T(u)$. Clearly

$$L(\eta) = \sum_{i=1}^{d(u)} L(\varphi_i) - [1, \alpha - 1] + \{\alpha\}.$$

Q.E.D.

By Lemmas 7, 8 and 9 above we have the following recursive algorithm to find a critical c-ranking of $T(u)$.

```
  Procedure Ranking(T(u));
  begin
1   if u is a leaf
        then return a trivial c-ranking: u → 1
2   else
3     begin
4       let v_1, v_2, ···, v_d(u) be the children of u;
5       for i := 1 to d(u) do Ranking(T(v_i));
6       find a critical c-ranking of T(v_i) from critical c-rankings
            of T(v_i), i = 1, 2, ···, d(u), by Lemmas 8 and 9;
7       return a critical c-ranking of T(u)
8     end
  end.
```

Clearly one can correctly find a critical c-ranking of a tree T by calling **Procedure** Ranking $(T(r))$ for the root r of T. Therefore it suffices to verify the time-complexity of the algorithm. Let φ_i, $i = 1, 2, \cdots, d(u)$, be a critical c-ranking of $T(v_i)$. Assume without loss of generality that $\#\varphi_1$ and $\#\varphi_2$ are the two largest, possibly equal, numbers among $\#\varphi_i$, $i = 1, 2, \cdots, d(u)$, and that $\#\varphi_1 \ge \#\varphi_2$. Let $\#\varphi_2 = 0$ if $d(u) = 1$. Let η be a critical c-ranking of $T(u)$ obtained from φ_i, $i = 1, 2, \cdots, d(u)$, at line 6. Then the following lemma holds, which will be proved later.

Lemma 10. *One execution of line 6 can be done in time $O(x_u + d(u) + c \cdot \#\varphi_2)$ where x_u is the number of vertices which were visible in $T(v_i)$ under φ_i, $i \in [1, d(u)]$, but are not visible in $T(u)$ under η.*

Once a vertex becomes non-visible, it will never become visible again. Furthermore $\sum d(u) \le n$ where the summation is taken over all internal vertices. Therefore the total time counted by the first term x_u and the second term $d(u)$ is $O(n)$ when **Procedure** Ranking is recursively called for all vertices. Let n_{u_2} be the number of vertices in the second largest tree among $T(v_i)$, $i = 1, 2, \cdots, d(u)$, if $d(u) \ge 2$. Then by Lemma 4 we have $\#\varphi_2 \le \lceil \log_{\frac{c+3}{2}} n_{u_2} \rceil + 1$. Let $V_2 = \{u \in V \mid d(u) \ge 2\}$. The following lemma implies that the total time counted by the third term $c \cdot \#\varphi_2$ is also $O(n)$. Thus the total running time of Ranking is $O(n)$. This completes the proof of Theorem 2.

Lemma 11. $\sum_{u \in V_2} \left(\lceil \log_{\frac{c+3}{2}} n_{u_2} \rceil + 1 \right) = O(n).$

Proof. For a tree T, let $S(T) = \sum_{u \in V_2} \left(\lceil \log_{\frac{c+3}{2}} n_{u_2} \rceil + 1 \right)$. We now prove by induction on n that

$$S(T) \leq 2n - \left(\lceil \log_{\frac{c+3}{2}} n \rceil + 1 \right). \tag{1}$$

Trivially Eq. (1) holds when $n = 1$. Now assume that Eq. (1) holds for any tree having at most $n-1$ vertices.

Let T be a tree with n vertices rooted at vertex u. One may assume that $d(u) \geq 2$. Let $v_1, v_2, \cdots, v_{d(u)}$ be the children of u, and let n_i, $i = 1, 2, \cdots, d(u)$, be the number of vertices of $T(v_i)$, respectively. Assume without loss of generality that $n_1 \geq n_2 \geq \cdots \geq n_{d(u)}$. Then $n_{u_2} = n_2$, and we have

$$\begin{aligned} S(T(u)) &= \sum_{i=1}^{d(u)} S(T(v_i)) + \lceil \log_{\frac{c+3}{2}} n_2 \rceil + 1 \\ &\leq \sum_{i=1}^{d(u)} \{2n_i - (\lceil \log_{\frac{c+3}{2}} n_i \rceil + 1)\} + \lceil \log_{\frac{c+3}{2}} n_2 \rceil + 1 \\ &\leq 2n - \{d(u) + 1 + \log_{\frac{c+3}{2}} n_1 + \sum_{i=3}^{d(u)} \log_{\frac{c+3}{2}} n_i\}. \end{aligned}$$

Since

$$\left(\frac{c+3}{2}\right)^{d(u)-1} n_1 n_3 n_4 \cdots n_{d(u)} \geq 2^{d(u)-1} n_1 \geq d(u) n_1 \geq n,$$

we have

$$d(u) + 1 + \log_{\frac{c+3}{2}} n_1 + \sum_{i=3}^{d(u)} \log_{\frac{c+3}{2}} n_i \geq \log_{\frac{c+3}{2}} n + 2 \geq \lceil \log_{\frac{c+3}{2}} n \rceil + 1.$$

Therefore $S(T(u)) \leq 2n - (\lceil \log_{\frac{c+3}{2}} n \rceil + 1)$. $\mathcal{Q.E.D.}$

We finally give an implementation of line 6 of **Procedure** Ranking, which finds a critical c-ranking η of $T(u)$ from the critical c-ranking φ_i, $i = 1, 2, \cdots, d(u)$.

Procedure Line-6($\varphi_1, \cdots, \varphi_{d(u)}, \eta$);
begin
  $\eta | T(v_i) := \varphi_i$ for each i, $i := 1, 2, \cdots, d(u)$;
                { determine the rank of u as follows. }
  if $d(u) = 1$ **then**
    begin
      find a smallest rank $\alpha \geq 1$ such that $count(L(\varphi_1), \alpha) \leq c - 1$;
      $\eta(u) := \alpha$;
      $L(\eta) := (L(\varphi_1) - [1, \alpha - 1]) + \{\alpha\}$
    end
  else { $d(u) \geq 2$ }
    begin
      find the two largest, possibly equal, numbers among $\#\varphi_i$, $i := 1, 2, \cdots, d(u)$;
    { assume w.l.o.g. that $\#\varphi_1$ and $\#\varphi_2$ are these largest numbers and $\#\varphi_1 \geq \#\varphi_2$. }

11 let $L_s := (L(\varphi_1) - [\#\varphi_2 + 1, \#\varphi_1]) + \sum_{i=2}^{d(u)} L(\varphi_i)$;
12 find a smallest rank $\alpha \in [1, \#\varphi_2]$ such that $count(L_s, \alpha) \leq c - 1$ and $count(L_s, \gamma) \leq c$ for all ranks $\gamma \in [\alpha + 1, \#\varphi_2]$;
13 **if** such a rank α exists **then**
14 **begin**
15 $\eta(u) := \alpha$;
16 $L(\eta) := (L(\varphi_1) - [1, \#\varphi_2]) + (L_s - [1, \alpha - 1]) + \{\alpha\}$
17 **end**
18 **else**
19 **begin**
20 $L_s := L_s + (L(\varphi_1) - [1, \#\varphi_2])$; $\{ L_s = \sum_{i=1}^{d(u)} L(\varphi_i) \}$
21 find a smallest rank $\alpha \in [\#\varphi_2 + 1, \#\varphi_1 + 1]$ such that $count(L_s, \alpha) \leq c - 1$;
22 $\eta(u) := \alpha$;
23 $L(\eta) := (L_s - [1, \alpha - 1]) + \{\alpha\}$;
24 **end**
25 **end**
end;

We are now ready to prove Lemma 10.

Proof of Lemma 10. As a data-structure to represent a list $L(\xi)$ of a c-ranking ξ, we use a linked list L_ξ consisting of records. Each record contains two items of data: rank $\gamma \in [1, \#\xi]$ and $count(L(\xi), \gamma)$ such that $count(L(\xi), \gamma) \geq 1$.

If $d(u) = 1$, then using linked list L_{φ_1} one can easily find α at line 4 in $O(x_u)$ time where $x_u = |L(\varphi_1) \cap [1, \alpha - 1]|$. It should be noted that all the x_u edges of ranks in $L(\varphi_1) \cap [1, \alpha - 1]$ were visible but they become non-visible after lines 5 and 6 are executed. Thus lines 3–7 can be done total in time $O(x_u)$. Similarly, if lines 20–23 are executed, then at line 21 one can easily find α in $O(x_u)$ time, and hence lines 20–23 can be done in time $O(x_u)$ time.

We now claim that if $d(u) \geq 2$ then lines 10–12 and 15–16 can be done total in time $O(d(u) + c \cdot \#\varphi_2)$. We construct a linked list L_s as follows. First set L_s as an empty list. For each $i \in [1, d(u)]$, add to L_s all ranks γ $(\leq \#\varphi_2)$ in L_{φ_i} in the decreasing order of γ until either $count(L_s, \gamma) > c$ or all such ranks γ have been added. Thus line 11 can be done in time $O(c \cdot \#\varphi_2)$. Clearly line 10 can be done in time $O(d(u))$ and lines 12, 15 and 16 in time $O(\#\varphi_2)$. Therefore lines 10–12 and 15–16 can be done total in time $O(d(u) + c \cdot \#\varphi_2)$. Thus **Procedure** Line-6 can bo done total in time $O(x_u + d(u) + c \cdot \#\varphi_2)$. $\mathcal{Q.E.D.}$

Remark

If c is not bounded, our algorithm takes time $O(cn/\log(c+3))$. Note that Lemma 11 holds for any integer c.

References

[BDJ+94] H. Bodlaender, J.S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Zs. Tuza. Ranking of graphs. In *Proc. of International Workshop on Graph-Theoretic Concepts in Computer Science*, Herrsching, Bavaria, Germany, 1994.

[IRV88] A. V. Iyer, H. D. Ratliff, and G. Vijayan. Optimal node ranking of trees. *Information Processing Letters*, 28, pp.225–229, 1988.

[Pot88] A. Pothen. The complexity of optimal elimination trees. Technical Report CS-88-13, Pennsylvania State University, U.S.A., 1988.

[Sch89] A. A. Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33(2), pp.91–99, 1989.