Egerváry Research Group
on Combinatorial Optimization

Technical reportS

TR-2011-11. Published by the Egerváry Research Group, Pázmány P. sétány 1/C, H–1117, Budapest, Hungary. Web site: www.cs.elte.hu/egres. ISSN 1587–4451.

Recognizing graphic degree sequences and generating all realizations

Zoltán Király

April 23, 2012

Recognizing graphic degree sequences and generating all realizations

Zoltán Király*

Abstract

In this paper we give a very simple, linear-time algorithm for deciding whether a given sequence of integers is graphic. We also give an implementation of the famous algorithm of Havel and Hakimi, which runs in $O(n \log \log n)$.

In the second half of the paper we present a new algorithm, which, given a graphic degree sequence, generates all graph realizations of this degree sequence. As the output may be exponential, this algorithm is not necessarily polynomial, but it is the first such algorithm with a polynomial delay, actually it runs with quadratic delay.

Keywords: graphic degree sequence, Erdős-Gallai theorem, Havel-Hakimi algorithm, enumeration, polynomial delay

1 Introduction

Throughout the paper $[n]$ will denote the set $\{1, 2, \ldots, n\}$. Some linear-time decision algorithms were known in the literature [6, 4] for graphic degree sequences, our aim in this direction is two-fold: we give a simpler algorithm, without any multiplication, and using only constant additional memory if the input is already sorted; and we give an extended version calculating also some useful auxiliary values.

We could not find any reference about the best running time (best implementation) of the famous algorithm of Havel and Hakimi, we give one with $O(n \log \log n)$ steps.

The main purpose of this paper is to give an enumeration algorithm, which, given a graphic degree sequence, generates all graph realizations of this degree sequence. As the output may be exponential, this algorithm is not necessarily polynomial, but runs with quadratic delay. This means, that after $O(n^2)$ steps after beginning it gives the adjacency matrix of one realization, and then, always in $O(n^2)$ steps, generates the adjacency matrix of the next realization, taking care that no realization is reported twice, and that all possible realizations are reported. This worst case delay time is clearly optimal.

*Department of Computer Science and EGRES (MTA-ELTE), Eötvös University, Pázmány Péter sétány 1/C, Budapest, Hungary. Research was supported by grants (no. CNK 77780 and no. CK 80124) from the National Development Agency of Hungary, based on a source from the Research and Technology Innovation Fund, and also by T'AMOP grant 4.2.1./B-09/1/KMR-2010-0003.

2 Recognizing graphic degree sequences in linear time

Here we use the classical theorem of Erdős and Gallai. A sequence of integers is called *graphic*, if it is a degree sequence of some (simple undirected) graph. We call a sequence of integers $\mathbf{d} = d_1, \ldots, d_n$ *sorted*, if $d_1 \geq d_2 \geq \ldots \geq d_n$ (if not stated explicitly otherwise, the length of a sequence $\mathbf{d}$ is always assumed to be n).

Theorem 2.1 (Erdős-Gallai [1])**.** *A sorted sequence of natural numbers with even sum is graphic if and only if for every* $1 \leq k \leq n-1$

$$\Delta_k := k(k-1) + \sum_{j=k+1}^{n} \min(d_j, k) - \sum_{i=1}^{k} d_i \geq 0. \tag{1}$$

We are presenting a linear-time algorithm, which decides, given a sequence of n natural numbers, whether this sequence is graphic. We will not detail the initialization phase of the algorithm, it consists of three well-known linear-time phases, namely

a) Checking that the integers are bounded by 0 and $n-1$,

b) Checking the parity of the sum (e.g., by XOR-ing them),

c) Sorting them by counting (or bucket) sort.

We remark that actually Step a) is needed only for guaranteeing that Step c) can be done in linear time, and checking that the input really consists of non-negative numbers. (If the input is guaranteed to consist of natural numbers and is already sorted, Steps a) and c) can be omitted.) So we will detail the algorithm from the point, where we know that $n-1 \geq d_1 \geq d_2 \geq \ldots \geq d_n \geq 0$ and the sum is even; otherwise we return value FALSE in the initialization phase.

Algorithm LIN-RECOG-GRAPHIC

```
LRG(n, d_1, ..., d_n):
    w := n; b := 0; s := 0; c := 0
    for k = 1..n
        b := b + d_k
        c := c + w - 1
        while (w > k and d_w ≤ k)
            s := s + d_w
            c := c - k
            w--
        if b > c + s
            then return(FALSE)
            else if w = k then return(TRUE)
```

Theorem 2.2. *Algorithm* LIN-RECOG-GRAPHIC *correctly recognizes graphic sequences and runs in time* $O(n)$.

This theorem will be a consequence of the next one. We note, that this algorithm uses constant additional space (if the input is already sorted), and does not make any multiplication. For the generating algorithm of Section 4 we will need an extension, that stores some useful auxiliary values for all k.

Algorithm LIN-RECOG-GRAPHIC-EXT

```
LRGE(n, r, d_1, ..., d_n):
    w := n; b := 0; s := 0; c := 0; ℓ_1 := 0; ℓ_2 := 0
    for k = 1..n
        b := b + d_k
        c := c + w − 1
        while (w > k and d_w ≤ k)
            s := s + d_w
            c := c − k
            w−−
        w_k := w + 1
        Δ_k := c + s − b
        if k < r then
            if (Δ_k = 0 or [(Δ_k = 1 and d_r ≤ k)]) then ℓ_1 := k
            if (Δ_k = 0 and d_r ≤ k and ℓ_2 = 0) then ℓ_2 := w_{k−1} − 1
        if Δ_k < 0
            then return(FALSE)
            else if w = k then
                k* := k
                return(TRUE, max(ℓ_1, ℓ_2))
```

We will explain the purpose of ℓ_1 and ℓ_2 later. Let k^* be the smallest value k, for which $d_{k+1} \leq k$.

Theorem 2.3. *Algorithm* LIN-RECOG-GRAPHIC-EXT *correctly recognizes graphic sequences and runs in time* $O(n)$. *It correctly calculates* k^*, *and for each* $k \leq k^*$, *the calculated value* w_k *is the smallest index* i, *such that* $i > k$ *and* $d_i \leq k$; *and the calculated value* Δ_k *is the excess of the* k*th inequality defined in (1).*

Proof. The way we calculated w_k ensures the statement about it, consequently we calculated k^* also correctly. An equivalent form of the equation (1) is the following.

$$\Delta_k = k(w_k - 2) + \sum_{j=w_k}^{n} d_j - \sum_{i=1}^{k} d_i \geq 0. \tag{2}$$

To prove this, it is enough to observe, that $\min(d_j, k) = k$, if $j < w_k$ and $\min(d_j, k) = d_j$ if $j \geq w_k$, and $\sum_{j=k+1}^{w_k-1} k = k(w_k - k - 1)$ and $k(k-1) + k(w_k - k - 1) = k(w_k - 2)$.

Observe, that we stop after checking equation (2) for k^*. By induction one can show, that for each $k \leq k^*$ we calculate the following values:

- b is the sum of the big degrees, i.e., $b = \sum_{i=1}^{k} d_i$,
- s is the sum of the small degrees, i.e., $s = \sum_{j=w_k}^{n} d_j$,
- $c = k(w_k - 2)$.

For proving the correctness of the checking part we only have to show, that if we checked the inequality (1) for $k = 1, \ldots, k^*$ then it is automatically satisfied for all larger values of k. This is a simple calculation, a slightly weaker form was already proved by Tripathi and Vijay [7]. Suppose $k' > k^*$ and $\sum_{i=1}^{k^*} d_i \leq k^*(k^* - 1) + \sum_{j=k^*+1}^{n} \min(d_j, k^*) = k^*(k^* - 1) + \sum_{j=k^*+1}^{n} d_j$, and we have to prove $\sum_{i=1}^{k'} d_i \leq k'(k' - 1) + \sum_{j=k'+1}^{n} d_j$. The left side increases by $\sum_{i=k^*+1}^{k'} d_i$ and the right side increases by $(k' - k^*)(k' + k^* - 1) - \sum_{i=k^*+1}^{k'} d_i$. As the terms in the sums are at most k^* and $k' + k^* - 1 \geq 2k^*$, we are done. The running time is obviously $O(n)$. □

A subset $J \subseteq [n]$ will be identified by its characteristic vector $\boldsymbol{\chi}^J$, where $\boldsymbol{\chi}^J$ is a 0-1 vector of length n and $\chi_i^J = 1$ if and only if $i \in J$.

Let $\mathbf{d}$ be a graphic sequence. We call the (i, j)-sibling of $\mathbf{d}$, denoted by $\mathbf{d}^{i,j}$, the following sequence: we increase d_i by one and then decrease d_j by one.

Our goal is to determine for each pair (i, j), whether the (i, j)-sibling of $\mathbf{d}$ is graphic or not.

Lemma 2.4 (Havel, Hakimi [3, 2])**.** *If* $\mathbf{d}$ *is a graphic sequence, and* $d_i < d_j$ *then* $\mathbf{d}^{i,j}$ *is also graphic.*

Proof. Let G be a realization of $\mathbf{d}$ on vertex set $[n]$, where $d_G(k) = d_k$. As $d_i < d_j$, there is vertex v connected to j, but not connected to (and different from) i. After deleting edge vj and adding edge vi, the resulting graph is obviously a representation of $\mathbf{d}^{i,j}$. □

A simple corollary of this lemma is the following famous theorem.

Theorem 2.5 (Havel, Hakimi [3, 2])**.** *The sequence* $d_1 \geq d_2 \geq \ldots \geq d_n \geq d_{n+1}$ *is graphic if and only if the following sequence (of length* n*) is graphic: we delete* d_{n+1} *and decrease the* d_{n+1} *largest remaining values by one.*

A triplet (P, r, s) is called a *preset*, if $P \subseteq [r]$ and $r \leq s \leq n$. A preset is thought as a trace – of a set $S \subseteq [n]$ of size s – on set $[r]$. Therefore we call a set $S \subseteq [n]$ an *extension* of preset (P, r, s), if $|S| = s$ and $S \cap [r] = P$. The leftmost extension of preset (P, r, s) is the set $P \cup \{r + 1, \ldots, r + (s - |P|)\}$. The following theorem will play an essential role in Section 4. In a slightly different form it was also stated in [5].

Theorem 2.6. *Suppose* $\mathbf{d}$ *is a sorted sequence,* (P, r, s) *is a preset, and* S *is the leftmost extension of* (P, r, s)*. There exists an extension* H *of* (P, r, s) *such that* $\mathbf{d} - \boldsymbol{\chi}^H$ *is graphic if and only if the sequence* $\mathbf{d} - \boldsymbol{\chi}^S$ *is graphic.*

Proof. Suppose $\mathbf{d} - \boldsymbol{\chi}^H$ is graphic. We can change $\boldsymbol{\chi}^H$ to $\boldsymbol{\chi}^S$ by a sequence of the following swaps. Take a pair of indices (i, j), such that $r < i < j$ and $\chi_i = 0$ and

$\chi_j = 1$, and increase χ_i by one, and then decrease χ_j by one. We had originally $d_i \leq d_j$, consequently $d_i - \chi_i < d_j - \chi_j$, thus Lemma 2.4 applies. □

The last theorem in this section is needed for getting quadratic delay time instead of a cubic one in Section 4.

Theorem 2.7. *Suppose we are given a value of r with the property that either $d_{r+1} < d_r$ or $r = n$. After running Algorithm* Lin-recog-graphic-ext *on the sorted graphic sequence $\mathbf{d}$, let $\ell^* = \max(\ell_1, \ell_2)$ be the second return value of the algorithm. Take any $\ell < r$ with the property that either $d_{\ell-1} > d_l$ or $\ell = 1$. Then the (ℓ, r)-sibling of $\mathbf{d}$ is not graphic if and only if $\ell \leq \ell^*$.*

Proof. Observe that by the conditions on values of ℓ and r, if $\mathbf{d}'$ denotes the sequence $\mathbf{d}^{\ell,r}$ after resorting then we have $d'_i = d_i$ except that $d'_\ell = d_\ell + 1$ and $d'_r = d_r - 1$ (i.e., $\mathbf{d}^{\ell,r}$ is already sorted).

First we prove that the condition is necessary, such that for any value $1 \leq \ell \leq \ell^*$ and $\ell < r$, the sibling $\mathbf{d}^{\ell,r}$ is not graphic. The value of ℓ_1, if it is not zero, is the largest value $k < r$, where either $\Delta_k = 0$, or $\Delta_k = 1$ and $d_r \leq k$. The value of ℓ_2, if it is not zero, is the largest index i, such that there exists a $r > k \geq d_r$ with properties $\Delta_k = 0$ and $k \leq d_i$. We suppose that ℓ^* got its final value in the kth main cycle, clearly $k < r$. Let Δ'_k denote $k(k-1) + \sum_{j=k+1}^{n} \min(d'_j, k) - \sum_{i=1}^{k} d'_i$.

The first case is when $\ell^* = \ell_1 = k$ and $\Delta_k = 0$. Now we have $\Delta'_k = \Delta_k + \min(d_r - 1, k) - \min(d_r, k) - (d_\ell + 1) + d_\ell \leq -1$, proving that $\mathbf{d}'$ is not graphic. The second case is when $\ell^* = \ell_1 = k$ and $\Delta_k = 1$ and $d_r \leq k$. Now we have again $\Delta'_k = \Delta_k + \min(d_r - 1, k) - \min(d_r, k) - (d_\ell + 1) + d_\ell \leq 1 - 1 - 1 \leq -1$, proving that $\mathbf{d}'$ is not graphic. The remaining third case is when $\ell^* = \ell_2 = w_{k-1} - 1$, and we also have $\Delta_k = 0$ and $d_r \leq k$. If $\ell \leq k$ then we have $\Delta'_k = \Delta_k + \min(d_r - 1, k) - \min(d_r, k) - (d_\ell + 1) + d_\ell \leq 0 - 1 - 1 \leq -2$, proving that $\mathbf{d}'$ is not graphic, so we may suppose $\ell > k$. Here we know that $d_\ell \geq d_{\ell_2} \geq k$. As $\Delta'_k = \Delta_k + \min(d_r - 1, k) - \min(d_r, k) + \min(d_\ell + 1, k) - \min(d_\ell, k) = \Delta_k + \min(d_r - 1, k) - \min(d_r, k) = 0 - 1$, proving that $\mathbf{d}'$ is not graphic.

Now we prove the much more important other part of the statement. Suppose that ℓ^* is the second return value of Algorithm Lin-recog-graphic-ext, $\ell < r$ and either $d_{\ell-1} > d_l$ or $\ell = 1$, and moreover suppose that $\mathbf{d}'$ is not graphic. Let k be the largest value with property $\Delta'_k \leq -1$.

Case 1. $k < \ell$.

$\Delta'_k = \Delta_k + \min(d_\ell + 1, k) - \min(d_\ell, k) + \min(d_r - 1, k) - \min(d_r, k) \leq -1$. As $\mathbf{d}$ is graphic, this may happen only if $\Delta_k = 0$, $d_\ell \geq k$, and $d_r \leq k$. In this case $\ell^* \geq \ell_2 \geq \ell$, because ℓ_2 is not smaller than the largest index with the property $d_{\ell_2} \geq k$.

Case 2. $\ell \leq k < r$.

$\Delta'_k = \Delta_k + \min(d_r - 1, k) - \min(d_r, k) - (d_\ell + 1) + d_\ell \leq -1$. This may happen either if $\Delta_k = 0$, or if $\Delta_k = 1$ and $d_r \leq k$. In both cases $\ell^* \geq \ell_1 \geq k \geq \ell$.

Case 3. $r \leq k$.

$\Delta'_k = \Delta_k - (d_r - 1) + d_r - (d_\ell + 1) + d_\ell \leq -1$, a contradiction, as sequence $\mathbf{d}$ is graphic. □

Corollary 1. *Suppose* $\hat{\mathbf{d}}$ *is a sorted graphic sequence of length* $n+1$, $p = \hat{d}_{n+1} > 0$, *and* $H = [p]$. *Let* $\mathbf{d}^-$ *denote the sequence* $d_1, \ldots, d_n$, *and* $\mathbf{d}'$ *denote the sequence* $\mathbf{d}^- - \boldsymbol{\chi}^H$, *and* $\mathbf{d}$ *denote the sequence* $\mathbf{d}'$ *after resorting. Let* $\hat{r} = p+1$ *and* r *denote the largest index* j *with* $d_j = \hat{d}_{\hat{r}} = d'_{\hat{r}}$. *Moreover let* $1 \leq \hat{\ell} < \hat{r}$ *be a given value, and* ℓ *be the smallest index* i *with* $d_i = d'_{\hat{\ell}}$. *Suppose* ℓ^* *is the second return value of the Algorithm* Lin-recog-graphic-ext *running on* $\mathbf{d}$ *and value* r. *The* $(\hat{\ell}, \hat{r})$*-sibling of* $\mathbf{d}'$ *is not graphic, if and only if* $d'_{\hat{\ell}} \geq d'_{\hat{r}}$ *and* $\ell \leq \ell^*$.

Proof. Observe that $\mathbf{d}'$ is graphic by Theorem 2.5, and if $d'_{\hat{\ell}} < d'_{\hat{r}}$ then the $(\hat{\ell}, \hat{r})$-sibling of $\mathbf{d}'$ is also graphic by Lemma 2.4, moreover if $\hat{\ell} = \hat{r}$ then the $(\hat{\ell}, \hat{r})$-sibling of $\mathbf{d}'$ equals to $\mathbf{d}'$. Thus we may assume that ℓ and r satisfy the conditions of Theorem 2.7. It is easy to see that the $(\hat{\ell}, \hat{r})$-sibling of $\mathbf{d}'$ is the same multiset as the (ℓ, r)-sibling of $\mathbf{d}$, so by Theorem 2.7 we are done. □

3 Havel-Hakimi Algorithm in quasi-linear time

We will use the data structure of van Emde Boas [8] over the following universe: $U = \{0, 1, 2, \ldots, n, n+1\}$. In this data structure the following operation are defined:

- **newdict**(S, U, n) – creates an empty dictionary S that can contain at most n elements of U – running time (in the simplest form) is $O(|U|)$.
- **insert**(S, u) – inserts element $u \in U$ to the dictionary S, if it is not yet there – running time is $O(\log\log|U|)$.
- **delete**(S, u) – deletes element $u \in S$ from the dictionary S – running time is $O(\log\log|U|)$.
- **prev**(S, u) – gives back the largest $j \in S$, such that $j \leq u$ – running time is $O(\log\log|U|)$.
- **next**(S, u) – gives back the smallest $j \in S$, such that $j \geq u$ – running time is $O(\log\log|U|)$.

Now in the initialization phase we order $\mathbf{d}$ non-decreasingly and create an empty dictionary in time $O(n)$. Next we calculate the sequence of differences: $D_i := d_i - d_{i-1}$ for $i = 1 \ldots n$ using $d_0 = 0$. Later on we **do not update** the values d_i, only the values D_i. Our aim is that during the run of the algorithm S always contains all the indices i where $D_i > 0$. We also store $d_{\max}$, the actual highest degree, and we are always going to delete this one from the sequence.

Algorithm ALMOSTLIN-HAVEL-HAKIMI

```
ALHH(n, d_1, ..., d_n):
    d_0 := 0; d_max := d_n; newdict(S, U, n); insert(S, 0); insert(S, n + 1)
    for i = 1..n
        D_i := d_i − d_{i−1}
        if D_i > 0 then insert(S, i)
    for m = n..2 (step −1)
        if d_max = 0 then return(TRUE)
        c := m − d_max
        if c ≤ 0 then return(FALSE)
        if D_c > 0 then
            d_max := d_max − D_m − 1
            D_c−−
            if D_c = 0 then delete(S, c)
        else
            l :=prev(c)
            r :=next(c)
            if r > m then r := m
            if l = 0 then return(FALSE)
            D_{l+r−c} := 1
            insert(S, l + r − c)
            D_l−−
            if D_l = 0 then delete(S, l)
            d_max := d_max − D_m
            if r < m then d_max := d_max − 1
    return(TRUE)
```

4 Generating all realizations

The problem we deal with in this section is the following. Given a graphic degree sequence, we want to generate (and print out) all graphs (on labeled vertices) that realize the given degree sequence. As the total size of the output may be exponential, our goal is the following. After starting the algorithm, the first graph must be presented in time $O(n^2)$, and afterward each representation must be given exactly once, and within time $O(n^2)$ from the previous one. This is usually called an algorithm with quadratic delay.

Our algorithm can generate the output graphs either in adjacency matrix form or by edge list. We give the description when the output form is the adjacency matrix, the edge list type of output can be presented similarly: we insert edges at the beginning of the lists, so inserting, and deleting the last inserted edge can be done in $O(1)$.

High-level description of the algorithm. For notational simplicity we assume that the algorithm is called with a sequence of length $n + 1$. We give a recursive algorithm, that starts with the basic checkings and rearrangements as Algorithm LIN-

REC in Section 2. (Actually we will take care not calling recursively the algorithm, if the present degree sequence is not graphic.) Then we generate one-by-one all the possible edge-sets incident to the last vertex, delete the last vertex, decrease the degree of the neighbors by one, and if the resulting sequence is graphic, then we call recursively the algorithm with this sequence of length n. The difficulty lies in to enumerate these shorter graphic sequences themselves with polynomial delay. For this purpose we will use a binary tree of depth n.

Let $n \geq d_1 \geq d_2 \geq \ldots \geq d_n \geq d_{n+1} \geq 1$ be the sorted and parity-checked graphic degree sequence, and let v_i denote the label of the intended vertex having prescription d_i. Our goal is to determine the neighbor set of v_{n+1} in all admissible ways, where a set $S \subseteq [n]$ is called admissible, if $|S| = d_{n+1} = s$ and the following sequence of length n is graphic: $d'_i := d_i - 1$, if $i \in S$, and $d'_i := d_i$ otherwise ($1 \leq i \leq n,\ i \notin S$). We construct a depth n binary tree with root $\emptyset$ and edges labeled by 0 or 1: for every non-leaf vertex the edge going to its left child is labeled by 1 and the edge going to its right child is labeled by 0. A vertex of this tree at depth r is identified with the set $P \subseteq [r]$, where $i \in P$ if and only if the ith step on path from the root to this vertex goes to the left child.

Next we color the leaves: a leaf w is red, if the set S identified with w is admissible, otherwise it is black. Now, by a bottom-up approach we color all vertices of the tree. A vertex is red, if at least one of its children is red, otherwise it is black. A red vertex at depth r corresponding to a set P is labeled by the preset (P, r, s), note that a vertex identified with P cannot be red, if $|P| > s$. The heart of the algorithm is the following: we must visit all red leaves of the tree from left to right, stepping on the edges of the tree, and consuming time $O(n^2)$ between two visits. We are forbidden to visit a black vertex. The distance of any two red leaves is clearly at most $2n$, so we have to do one step (traversing one edge of the tree) in time $O(n)$. This can be done using Theorem 2.6, and it is not hard to see that this would give an algorithm with cubic delay. However our main goal is to give an algorithm with quadratic delay.

Detailed description of the algorithm. The algorithm is called by a sequence of length $n + 1$ consisting of natural numbers. We first check whether the input is graphic, and if not then we stop. We initialize an $(n+1) \times (n+1)$ all-0 matrix. Then we call the recursive main part of the algorithm.

The recursive algorithm starts by examining the value d_{n+1}, if it is zero then it calls itself with $d_1, \ldots, d_n$. Otherwise it sets $s = d_{n+1}$, and starts traversing the tree described above, avoiding to step onto a black vertex. Let d' denote the sequence $d_1 \geq \ldots \geq d_n$.

During traversing the tree, we keep track of the following information. If the current vertex is at depth r and is labeled by preset (P, r, s) then we have $\boldsymbol{\chi}^P$ in the last row and in the last column of the adjacency matrix in preparation, and we store the depth r, the actual size a (which is the cardinality of set P), and the current degree sequence $\boldsymbol{d' - \chi^P}$. At each visited tree-vertex we also store two bits of information indicating whether its left or right child is red. Determining these bits is the essence of the algorithm. We must note here, that if a vertex labeled by (P, r, s) is red and $a = |P| < s$ then by Theorem 2.6 its left child must be red; otherwise (if $a = |P| = s$) its right child must be red. Checking whether a vertex labeled by (P, r, s) is red can

be done easily in linear time (using Theorem 2.6), because it is red if and only if $r \leq s$ and the leftmost extension of (P, r, s) is graphic.

Going downwards in the tree: suppose we are at a tree vertex labeled by (P, r, s) and $r < n$. We check in linear time whether the degree sequence of the left child is red (we will omit this step when we are sure that it is), if not, then we know that the right child is red. If the left child is red we first go to there: we set χ^P_{r+1} to 1, increase r and a by one, change the two corresponding values of the adjacency matrix to 1, and proceed with that left child. After returning to this tree vertex, if the right child is also red then we go to the right child in an even simpler manner: we increase r by one and proceed with that right child. Clearly the time needed for one downward step is $O(n)$, and it is $O(1)$, if we can omit the graphicity-check.

Going upwards in the tree: if we are in a right child of a tree-vertex of depth $r-1$, we simply proceed with the parent after decreasing the value of r by one. If we are in a left child, then we set χ^P_r to 0, decrease a by one, and change the two corresponding values of the adjacency matrix to 0, and proceed with the parent after decreasing the value of r by one. The time for an upward step is $O(1)$.

When we reach a red leaf, the construction of the current edge list of v_{n+1} is finished, and we call recursively the same algorithm with the constructed length-n current degree sequence. The recursion stops if $n \leq 2$. In this case it is easy to determine the realizations out of the two possibilities, and when we find a good realization, we can print out the graph collected as the adjacency matrix (or the edge list) in time $O(n^2)$.

By these steps we never go to a black vertex, and, as the number of red vertices is less than twice the number of red leaves, it is not hard to see that this recursive algorithm can be done with polynomial delay. To prove our goal, that is a quadratic delay, we need one more trick: when we are going to the leftmost red leaf, we use Theorem 2.5, and we omit the graphicity-checks. Consequently we reach the leftmost red leaf in $O(n)$ time. However it is not obvious that this can be done with the required adjustments of the two bits at every intermediate vertex. This is the reason we stated Theorem 2.7 and Corollary 1. So when starting the recursive algorithm, we also run Lin-recog-graphic-ext with sequence $\boldsymbol{d'} - \boldsymbol{\chi}^{[s]}$ (and precalculating for each possible $\hat{\ell}$ the corresponding value ℓ), and using Corollary 1 we can determine in time $O(1)$ for all possible values of $\hat{\ell}$ whether the right child of any vertex on this leftmost admissible path is red or not.

The detailed running time analysis is as follows. Let $D(n)$ denote the maximum delay, i.e., the maximum number of steps between reporting two graphs, when the input degree sequence consists of $n + 1$ numbers. Let $WU(n)$ denote the warm-up delay, which is the number of steps needed from the beginning to start printing out the first realization, and $F(n)$ denote the finishing delay, that is the number of steps needed after printing out the last realization until observing that we are done. Moreover let $IWU(n)$ denote the number of inner steps of the current recursion level from the beginning to the first recursive call, and $ID(n)$ denote the maximum number of inner steps of the current recursion level between two calls, and $IF(n)$ denote the inner finishing delay, that is the number of steps needed after the last call until observing that we are done.

Theorem 4.1. *The algorithm above has the following delays.* $IWU(n) = O(n)$, $ID(n) = O(n^2)$, $IF(n) = O(n)$, *and, consequently,* $WU(n)$, $F(n)$, $D(n) = O(n^2)$.

Proof. When we start the algorithm, we do linear-time check and a counting sort. Then we traverse the tree to the leftmost red leaf as follows. We go to the left d_n times, and then we go to the right $n-1-d_n$ times. By Theorem 2.5 we arrive at the leftmost red leaf, so we do not have to check graphicity when going downwards, consequently each step can be done in $O(1)$ (by using the argument above), thus $IWU(n) = O(n)$. As going upwards can be done in $O(1)$, we have $IF(n) = O(n)$. Between two red leaves we traverse at most $2n$ edges of the tree in time $ID(n) = O(n^2)$. Consequently there is a constant c, such that $IWU(n) \leq c \cdot n$, $IF(n) \leq c \cdot n$ and $ID(n) \leq c \cdot n^2$.

To remaining part of the proof goes by induction. We are going to prove that $D(n)$, $WU(n)$, $F(N) \leq 3 \cdot c \cdot n^2$. We have that $WU(n) \leq c \cdot n + WU(n-1)$ and $F(n) \leq c \cdot n + F(n-1)$, this gives $WU(n)$, $F(n) \leq c \cdot n^2$. There are two possibilities of the underway delay. Either it is inside the recursive call, so it is not more than $3 \cdot c \cdot (n-1)^2 \leq 3 \cdot c \cdot n^2$, or it is not more than $F(n-1) + ID(n) + WU(n-1) \leq 2 \cdot c \cdot (n-1)^2 + c \cdot n^2 \leq 3 \cdot c \cdot n^2$. □

5 Acknowledgment

The author is very grateful to Antal Iványi and Péter L. Erdős for proposing nice problems, and also to Péter L. Erdős and István Miklós for helpful discussions.

References

[1] P. Erdős, T. Gallai Graphs with prescribed degrees (in Hungarian: Grfok elrt fok pontokkal), *Mat. Lapok* **11** (1960), pp. 264–274.

[2] S. L. Hakimi On the realizability of a set of integers as degrees of the vertices of a simple graph, *J. SIAM Appl. Math.* **10** (1962), pp. 496–506.

[3] V. Havel A remark on the existence of finite graphs (in Czech), *Časopis Pěst. Mat.* **80** (1955), pp. 477–480.

[4] P. Hell, D. Kirkpatrick Linear-time certifying algorithms for near-graphical sequences, *Discrete Mathematics* **309** (2009), pp. 5703–5713.

[5] H. Kim, Z. Toroczkai, P. L. Erdős, I. Miklós, L. A. Székely Degree-based graph construction, *J. Phys. A: Math. Theor.* **42** (2009), 392001

[6] M. Takahashi Optimization Methods for Graphical Degree Sequence Problems and their Extensions, *PhD. Thesis* Graduate School of Waseda University, (2007).

[7] A. Tripathi, S. Vijay A note on a theorem of Erdős & Gallai, *Discrete Mathematics* **265** (2003), pp. 417–420.

[8] P. van Emde Boas Preserving order in a forest in less than logarithmic time, *Proceedings of the 16th Annual Symposium on Foundations of Computer Science* **10** (1975), pp. 75–84.