Vol.2019-AL-174 No.2
2019/9/17

On Hardness of Sublinear-Space Lex-DFS for Graphs of Maximum Degree Three*

Taisuke Izumi[1,a)]

Abstract: The lexicographic depth-first search (Lex-DFS) is a popular variant of the standard depth-first search, which is known as a P-complete problem under logspace reduction, i.e., unlikely to have an algorithm using only $O(\log n)$-bit working memory. From the upper-bound side, several algorithms attaining a space complexity below the trivial $O(n \log n)$-bit bound are recently proposed (where n is the number of vertices), but none of them achieves $o(n)$-bit space complexity. In this research direction, it is a common view that obtaining a Lex-DFS algorithm using $o(n)$-bit working memory is a challenging open problem. The main contribution of this paper is to provide a new insight into that challenge. Specifically, we show that Lex-DFS using $o(n)$-bit working memory for graphs of *maximum degree 3* is as hard as that for sparse graphs (i.e. graphs with $O(n)$ edges).

Keywords: space-efficient algorithm, space-complexity, Lexicographic order DFS

1. Introduction

1.1 Background and Motivation

Depth-First Search (DFS) is one of the most fundamental and elementary graph search algorithms with a huge number of applications. Lexicographic DFS (Lex-DFS) is a popular variant of DFS, which requires the search head always moves to the *first* undiscovered neighbor in the adjacency list of the current vertex (as long as it exists). Recently, it receives much attention to attempt lowering the space complexity of fundamental graph problems, including DFS, below their trivial upper bounds [1], [2], [3], [4], [5], [6], [7]. These researches are roughly motivated by the two aspects as follows: First, the space matter is serious in the big-data (i.e., too large inputs) and/or IoT (i.e., too small computational devices) era. Second, the challenge of revealing the space complexity of problems within class P still lies at the core of the computational complexity theory. One of the ultimate goals along this line is to prove or disprove the seminal P $\neq$ L conjecture [1]. Our study focuses on the space complexity of Lex-DFS, which can be motivated from both sides but much leans against the second one. We define the Lex-DFS problem as the one of outputting the Lex-DFS ordering of all vertices in streaming way, and measure its space complexity by the required working-memory size in the classical read-only model [8] where the system is equipped with three memory areas, read-only input memory, write-only output memory, and working memory. An input graph is initially written in the input memory, and the output must be written in the output memory.

To argue the complexity of sublinear-space algorithms, the notion of P-completeness under logspace reduction plays an important role, which is analogous to NP-completeness in P $\neq$ NP conjecture. A proof by Reif [9] shows that the Lex-DFS problem is P-complete under logspace reduction. It implies that no Lex-DFS algorithm only using $O(\log n)$-bit working memory exists unless P=L holds. Its counterpart is the recent progress on (Lex-)DFS algorithms achieving both polynomial time and $o(n \log n)$-bit space complexity, where n is the number of vertices in the input graph (note that $\Theta(n \log n)$ is the trivial upper bound for Lex-DFS). Initiated by Asano et al. [1] and Elmasry et al. [2], a series of papers by several authors explore the good time-space tradeoffs in implementing (Lex-)DFS using $o(n \log n)$-bit working memory. The state-of-the-art best achievable bounds are $O(m \log^* n)$ running time and $O(n)$-bit working memory[7], $O(m + n)$ running time and $O(n \log \log n)$-bit working memory[2], and $(m + n)$ running time and $O(n \log(m/n))$-bit working memory[3]. Looking at hidden coefficients, the smallest-space algorithm is the one by Asano et al. [1], which attains a polynomial time (with a high exponent) and $(n + o(n))$-bit usage of working memory. There is no algorithm achieving $o(n)$-bit working memory (or even weaker cn-bit memory for $c < 1$) so far. Obtaining such an algorithm is commonly recognized as a very challenging problem.

1.2 Our Result

The main result of this paper is the following theorem.

Theorem 1 Assume that there exists a polynomial-time

[1] Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya, Aichi, 466-8555, Japan.
a) t-izumi@nitech.ac.jp

[1] L is the class of problems decidable in poly(n) time using $O(\log n)$-bit working space.

Lex-DFS algorithm for any graphs of *maximum degree 3* which uses $O(n/f(n))$-bit working memory. Then there exists a polynomial-time Lex-DFS algorithm for any graphs of m edges using the working memory of $O(m/f(m))$ bits.

While these corollaries do not state any conclusive answer for the question of sublinear-space solvability of Lex-DFS, the author believes that they provide an interesting insight into it: If the barrier of $\Omega(n)$-bit space complexity for Lex-DFS is not so strong, the theorem above will be a powerful tool for breaking it because the constraint of maximum degree 3 substantially simplifies the problem.

1.3 Related Work

As stated above, the space complexity of Lex-DFS is one of the classical problems in the context of logspace computability. As well as P-completeness result by Reif [9], Anderson and Mayr also shows a restricted case of Lex-DFS (lexicographically first maximal path) is also P-complete [10]. The main interest in this early stage is s-t connectivity of directed graphs, in relation to the theory of NL-completeness. It is known that the s-t directed connectivity problem allows an algorithm with a space complexity sublinear of n [11], as well as its optimality within a (naturally) restricted class of algorithms [12]. It is also known that the space complexity of undirected s-t connectivity drops into $O(\log n)$ bits, which is proved in Reingold's celebrating paper [13]. Despite the relatively rich literature on s-t connectivity, the space complexity of lex-DFS receives less attention until recently, in particular, from the aspect of upper bounds. After the two concurrent results by Asano et al. [1] and Elmasry et al. [2], a few follow-up results has been proposed, which consider the $o(n \log n)$-bit solvability for a variety of fundamental graph problems: Lex-BFS [2], [3], single-source shortest path [2], biconnected component decomposition [4], [5], s-t numbering [4], maximum cardinality search [6], and so on. It is also becoming active to consider sublinear-space algorithms for fundamental non-graph-theoretic problems [14], [15], [16], [17], [18].

On the side of computational models, the read-only model is one of the classical models to think about the complexity of working memory. The first main topic in this model is the time-space tradeoffs for sorting and/or selection [8], [19], [20], [21], [22], [23]. Recently, more unconventional models are also considered; Stream model [24], restore model (algorithms can manipulate input memory but after the computation the initial input data must be recovered) [25], [26], and catalytic model (the algorithm can use a large memory which are already used for other purpose, and after the computation the memory state must be recovered to the initial one)[27]. Some of the results in those models allows a lex-DFS algorithm using only a small working memory, but they are incomparable to the ones for the standard read-only model. It is also worth touching the result by Barba et al.[28], which presents a general scheme of realizing stack machines using only a small memory space. While the dominant part of the memory usage in lex-DFS algorithms is to store the contents of the stack, the technique by Barba et al. only applies to the algorithms whose access pattern to stacks are non-adaptive. Thus that scheme does not work for realizing small-space Lex-DFS algorithms.

1.4 Organization of Paper

In Section 2, we introduce the model, notations, and several auxiliary matter for the Lex-DFS problem. Section 3 shows the proof of Theorem 1. Finally the paper is concluded in Section 4 as well as a short discussion on related open problems.

2. Preliminaries

2.1 Model and Notation

As stated in the introduction, throughout this paper (and also in past literature), we measure the space complexity of algorithms by the number of bits used for the working space excluding the space for inputs and outputs. More specifically, we adopt the *read-only model* [8], which is equipped with three memory areas, called input, working, and output memories respectively. The input memory is read-only, and the output memory is write-only. The memory-access model follows the standard RAM of $(\log n)$-bit words. This paper considers only undirected graphs as inputs. Let $G = (V_G, E_G)$ be any input graph of n vertices and m edges, which is stored in the input memory in the form of adjacency list A_G. We assume $V_G = [0, n-1]$, that is, each vertex in V_G is identified by an integer in $[0, n-1]$. Letting $N_G(v) \subseteq V_G$ be the set of v's neighbors in G, we refer to the neighbor list of $v \in V_G$ as $A_{G,v}$ and denote the i-th vertex in $A_{G,v}$ by $A_{G,v}[i]$ (index i starts from value one). We use notation $u <_{(G,v)} w$ for $u, w \in N_G(v)$ if u precedes w in $A_{G,v}$. We define the inverse mapping of $A_{G,v}$ as $A^{-1}_{G,v}$, that is, for any neighbor u of v, $A^{-1}_{G,v}[u]$ returns the order of u in $A_{G,v}$. For any graph H, we denote its vertex set and edge set by V_H and E_H respectively if they are not explicitly defined.

2.2 Lex-DFS

To explain the behavior of Lex-DFS, we utilize a coloring scheme used in [1]: Initially, all the vertices are white, and the search head h is placed at the starting vertex s. When vertex v is visited from vertex u, the color of v changes from white to gray and the search head moves from u to v. If there is no white neighbor of v, the search at v finishes. Then the color of v changes from gray to black, and the search head returns back to u. We also introduce the notion of time in the DFS search. The algorithm starts at time $t = 0$. At $t = 1$, the starting vertex s becomes gray. At each time $t \geq 1$, exactly one vertex changes its color, either from white to gray, or, from gray to black. At time t $= 2n$, the color of s becomes black and the search finishes. For vertex $v \in V$, the *discovery time* of v is defined as the time when the color of v is changed from white to gray. Similarly, we define the *leaving time* of v as the time when v changes

Algorithm 1 Lex-DFS Algorithm for graph G starting from s

```
1: $v_{cur} \leftarrow s$                                  ▷ $v_{cur}$ is the search head
2: (color $v_{cur}$ gray)
3: $t = 1$
4: while true do
5:     $v \leftarrow$ Pivot($t$)        ▷ Find the white neighbor of $v_{cur}$ with the smallest order in $A_{G,s}$.
6:     if $v = -1$ then                ▷ All neighbors have been already visited.
7:         (color $v_{cur}$ black)
8:         $v \leftarrow$ Parent($t$)                      ▷ Find the end of the gray path
9:         if $v = -1$ then                                ▷ All vertices are visited
10:            halt
11:        $v_{cur} \leftarrow v$                          ▷ Backtrack
12:    else
13:        $v_{cur} \leftarrow v$                          ▷ Forward
14:        (color $v_{cur}$ gray)
15:    $t \leftarrow t + 1$
```

its color from gray to black. Note that gray vertices always form a simple path from the starting vertex s to the one where the current search head lies.

In what follows, we fix the starting vertex s. While we introduce several notations and definitions which could depend on the starting vertex, the information of s is not explicitly stated in those notations and definitions. We refer to the task of the Lex-DFS search in graph G (starting from s) as $LDFS_G$. We define $h_{G,t}$ as the vertex pointed by the search head at time t in $LDFS_G$. The subgraph of G induced by the gray vertices at time t is denoted by $S_{G,t}$. With a small abuse of notations, we often treat $S_{G,t}$ as a sequence of vertices in V_{S_t} following the order of the gray path (starting from s). For any vertex $u \in S_{G,t}$, we also denote by $p_{G,t}(u)$ and $s_{G,t}(u)$ the (immediate) predecessor and successor of u in $S_{G,t}$, and denote by $S_{G,t}(u)$ the prefix of $S_(G,t)$ up to u (or the subgraph corresponding to it). The discovery time and leaving time of $u \in V$ in the task of $LDFS_G$ is respectively denoted by $d_G(u)$ and $l_G(u)$. Note that all of these notations are defined for the Lex-DFS task itself, and does not depend on algorithms implementing Lex-DFS. In the following argument, for all the notations introduced in this paper (including $N_G(v)$, $A_{G,v}$, $<_{G,v}$ and $A^{-1}_{G,v}$ defined in Section 2), the subscript G is omitted if G is the input graph.

The Lex-DFS algorithm based on the coloring scheme stated above is presented in Algorithm 1. The pseudocode does not prepare the memory space for storing the color of each vertex explicitly. Instead, all the tasks dependent on vertex coloring is encapsulated by two abstract procedures called Pivot(t) and Parent(t). The procedure Pivot(t) tries to find the white neighbor of h_t whose order with respect to $\leq_{h_t}$ is the smallest. If there is no white neighbor, it returns -1. The procedure Parent(t) returns the predecessor $p_t(h_t)$ in the current gray path. It returns -1 if $h_t = s$ holds.

3. Proof of Theorem 1

3.1 Reduction Gadget

This section provides the proof of the second reduction, which reduces the task of Lex-DFS for graph H of n vertices and m edges to that for a max-degree-3 graph H^3 of $O(m)$ vertices. The reduction uses only $O(\log n)$-bit working memory, i.e., it is a logspace reduction. We present the detailed construction of H^3 below:

Vertex set

Let $\delta(u)$ be the degree of u in H. The graph H^3 consists of $4m + 2n$ vertices. We associate a set $\hat{V}_u$ of $4\delta(u) + 2$ vertices in H^3, which forms a gadget Γ_u corresponding to u in H^3. The vertices in $\hat{V}_u$ are referred as $\hat{V}_h = \{x^u_0, x^u_1, \ldots, x^u_{2\delta(u)-1}, y^u_0, y^u_1, \ldots, y^u_{\delta(u)}, z^u_0, z^u_1, z^u_{\delta(u)}\}$. We define $\hat{X}^u = \{x^u_i \mid 0 \leq i \leq 2\delta(u) - 1\}$, $\hat{Y}^u = \{y^u_i \mid 0 \leq i \leq \delta(u)\}$, and $\hat{Z}^u = \{z^u_i \mid 0 \leq i \leq \delta(u)\}$.

Edge set

Precisely, we do not only specify the edge set E_{H^3}, but also its adjacency list A_{H^3}. First, we specify the edge in each gadget Γ_u. Let $a(v, i) = A_{H^3,v}[i]$ for short. We call edge $(v, a(v, i))$ the *i-th edge* of v.

- The edges inside Γ_u (see Figure 1): The first and second edges of the vertices in $\hat{X}^u$, $\hat{Y}^u$, and $\hat{Z}^u$ form three cycles respectively. Formally, $a(x^u_i, 1) = x^u_{(i+2\delta(u)-1 \bmod 2\delta(u))}$ and $a(x^u_i, 2) = x^u_{(i+1 \bmod 2\delta(u))}$ hold for any $0 \leq i \leq 2\delta(u) - 1$. Similarly, $a(y^u_i, 1) = y^u_{(i+\delta(u)-1 \bmod (\delta(u)+1))}$, $a(y^u_i, 2) = y^u_{(i+1 \bmod (\delta(u)+1))}$, $a(z^u_i, 1) = z^u_{(i+\delta(u)-1 \bmod (\delta(u)+1))}$, and $a(z^u_i, 2) = z^u_{(i+1 \bmod (\delta(u)+1))}$ hold for any $0 \leq i \leq \delta(u)$. For any $0 \leq i \leq \delta(u) - 1$, connect x^u_{2i+1} and y^u_i by the third edge of them, i.e., $a(x^u_{2i+1}, 3) = y^u_i$ and $a(y^u_i, 3) = x^u_{2i+1}$. In addition, connect $y^u_{\delta(u)}$ and $z^u_{\delta(u)}$ be their third edge, i.e., $a(y^u_{\delta(u)}, 3) = z^u_{\delta(u)}$ and $a(z^u_{\delta(u)}, 3) = y^u_{\delta(u)}$.

- The edges between two gadgets: Connect z^u_i and x^v_{2j-1} by their third edge if and only if $A_{H,u}[i+1] = v$ and $A_{H,v}[j+1] = u$ (recall that the indices of adjacency lists start from one). Note that there exists an edge crossing between $\hat{X}^u$ and $\hat{Z}^v$ if and only if there exists an edge crossing between $\hat{Z}^u$ and $\hat{X}^v$.

3.2 Simulation of $\boldsymbol{LDFS_H}$

The output sequence of Lex-DFS for H is obtained by running any Lex-DFS algorithm for H^3 with starting vertex x^s_0. In the run of the task $LDFS_{H^3}$, u is outputted when x^u_0 is discovered.

3.3 Correctness

Let $\hat{d}(u)$ be the earliest time when a vertex in $\hat{V}_u$ is discovered in $LDFS_{H^3}$. To prove the correctness of this reduction, we first presents an auxiliary lemma below:

Lemma 1 For any two vertices u and v such that $\hat{d}(u) < \hat{d}(v)$ holds, all the vertices in $\hat{V}_u$ is discovered by $\hat{d}(u)$.

Proof We first prove that if $h_{H^3,\hat{d}(u)} \in X^u$ holds then all the vertices in $\hat{V}_u$ are discovered by time $\hat{d}(v)$. Figure 2 almost proves this fact. Starting from x^u_{2i} for any $0 \leq i\delta(u) - 1$, the search head discovers all the vertices in $\hat{X}^u$ and then moves from x^u_{2i+1} to y^u_i. The search head discovers all the vertices in $\hat{Y}^u$ with termination at y^u_{i+1}. Then it starts the backtrack to $y^u_{\delta(u)}$. After moving from $y^u_{\delta(u)}$ to $z^u_{\delta(u)}$, it discovers all the vertices in $\hat{Z}^u$ without going out of Γ_u. The remaining issue is to show that $h_{H^3,\hat{d}(u)} \in \hat{X}^u$ holds for any $u \in V_H$. Suppose for contradiction that $h_{H^3,\hat{d}(u)} \notin \hat{X}^u$ (i.e., $h_{H^3,\hat{d}(u)} \in \hat{Z}^u$) holds for some $u \in V_H$. Then we have $h_{H^3,\hat{d}(u)-1} \in \hat{X}^w$ for some w. Letting $x^w_j = h_{H^3,\hat{d}(u)-1}$, the search head finishes the backtrack at all the vertices in Z^w. However, $\hat{X}^w$ has an outgoing incident edge to a vertex in $\hat{V}_u$. Let $z^w_{j'} \in Z^w$ be the vertex having an outgoing edge to a vertex in $\hat{V}_u$. Then the search head finishes the backtrack at $z^w_{j'}$ despite leaving an undiscovered neighbor of $z^w_{j'}$. It is a contradiction. □

Lemma 2 For any $u, v \in H$, $d_H(u) < d_H(v)$ implies $d_{H^3}(x^u_0) < d_{H^3}(x^v_0)$.

Proof Let $u_0, u_1, u_2, \ldots, u_{n-1}$ ($u_0 = s$) be the output sequence of $LDFS_H$, and $x_0, x_1, \cdots, x_{n-1}$ ($x_0 = s$) be the sequence of vertices in $\{x^v_0 \mid v \in V_H\}$ sorted by the order of their discovery time in $LDFS_{H^3}$. It suffices to show that $x_i = x^{u_i}_0$ holds for any $0 \leq i \leq n-1$. The proof is by the induction on i. (Basis) For $i = 0$, $u_0 = s$ and $x_0 = x^s_0$ holds. (Inductive Step) Suppose as the induction hypothesis that $x_{i'} = x^{u_{i'}}_0$ holds for any $i' \leq i$ and consider the case of $i+1$. Let $w_1, w_2, \ldots w_k$ be the sequence of the vertices whose leaving time is in $[d_H(u_i), d_H(u_{i+1})]$. Assume that this sequence is sorted in the order of their leaving times. Since the backtrack is performed on those vertices in $LDFS_H$, any vertex in $\sum_{0 \leq j \leq k} N_H(w_j)$ has been discovered at $d_H(u_i)$. By the induction hypothesis and Lemma 1, for any $1 \leq j \leq k$, all the destinations of the edges outgoing from Γ_{w_j} are discovered at $\hat{d}(u_i)$. Thus the run of $LDFS_{H^3}$ performs backtrack after discovering all the vertices in Γ_{u_i} until the search head goes back to a vertex $z^{w_k}_j$ in Z^{w_k}. Since $LDFS_H$ discovers u_{i+1} at the run of $\mathsf{Pivot}(d_H(u_{i+1}) - 1)$ executed when $h_{H,t} = w_k$, there exists j' such that $A_{H,w_k}[j'] = u_{i+1}$ holds and all the vertices in $A_{H,w_k}[j'']$ for $j < j'' < j'$ are already discovered. By Lemma 1 and the induction hypothesis, all the vertices in $\cup_{j \leq j'' \leq j'} \hat{V}_{A_{H,w_k}[j'']}$ are discovered when the search head goes back to $z^{w_k}_j$. Thus the destinations of the outgoing edges of Γ_{u_i} from $z^{u_i}_{j-1}, z^{u_i}_j, \ldots, z^{u_i}_{j'-2}$ are all discovered, and thus $z^{u_i}_{j'-1}$ has the first undiscovered neighbor in the backtrack process starting from $z^{u_i}_{j-1}$. That is, the search head discovers a vertex in $\Gamma_{u_{i+1}}$ next. By Lemma 1, it implies $x_{i+1} = x^{u_{i+1}}_0$. The lemma is proved. □

4. Concluding Remarks

In this paper, we proved that undirected Lex-DFS for maximum-degree-3 graphs is as hard as that for any sparse graphs with respect to sublinear-space solvability. The author believes that this result will give a new insight into the big question if there exist a lex-DFS algorithm using $o(n)$-bit working space or not. Our result yields several interesting open problems, which are listed below.

- Is it possible to obtain a reduction from the sublinear-space solvability for general graphs (not necessarily sparse) to maximum-degree-3 cases? The author conjectures it is true, but the proof is still missing.
- A more weaker form of Lex-DFS problem is the *lexicographically minimum maximal path* (Lex-min-max path) problem, which must output the prefix of the Lex-DFS ordering before the first backtrack point. Is it possible to obtain the reduction from Lex-DFS to Lex-min-max path preserving sublinear space complexity?
- Can we find any reason why sublinear-space Lex-DFS is difficult? In the case of directed s-t connectivity, it is shown that the currently-best algorithm by Barnes et al. [11] is space-optimal under the NNJAG computational model[12]. For example, can we prove that no sublinear-space algorithm for Lex-DFS in the NNJAG model?
- What other problems are as hard as sublinear-space Lex-DFS? Will our result be a powerful tool for identifying such a problem?

Acknowledgments This work was supported by JST SICORP Grant Number JPMJSC1606 and JSPS KAKENHI Grant Number JP19K11824, Japan.

References

[1] Asano, T., Izumi, T., Kiyomi, M., Konagaya, M., Ono, H., Otachi, Y., Schweitzer, P., Tarui, J. and Uehara, R.: Depth-First Search Using $$O(n)$$Bits, *International Symposium on Algorithms and Computation (ISAAC)*, pp. 553–564 (2014).

[2] Elmasry, A., Hagerup, T. and Kammer, F.: Space-efficient Basic Graph Algorithms, *International Symposium on Theo-*

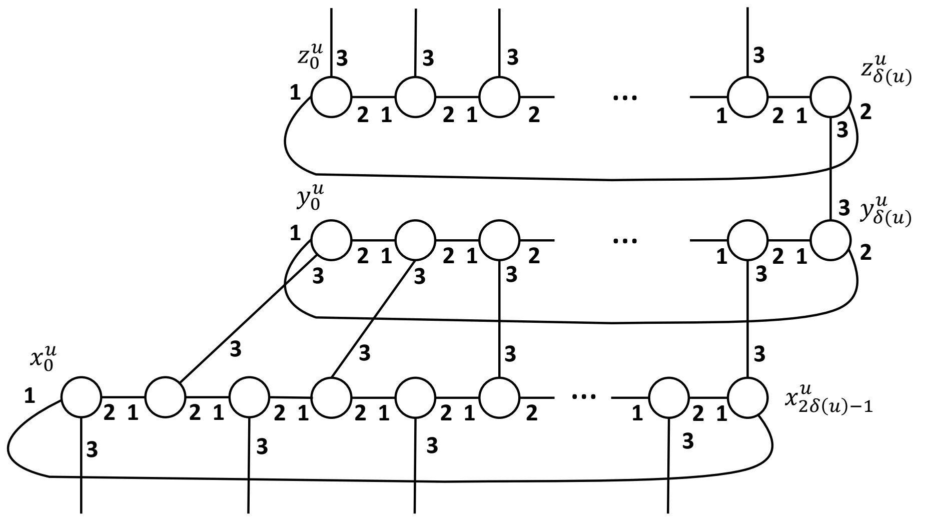

Fig. 1: Gadget Γ_u. The bold numbers written around each vertex are the order of the incident edges in its adjacency list.

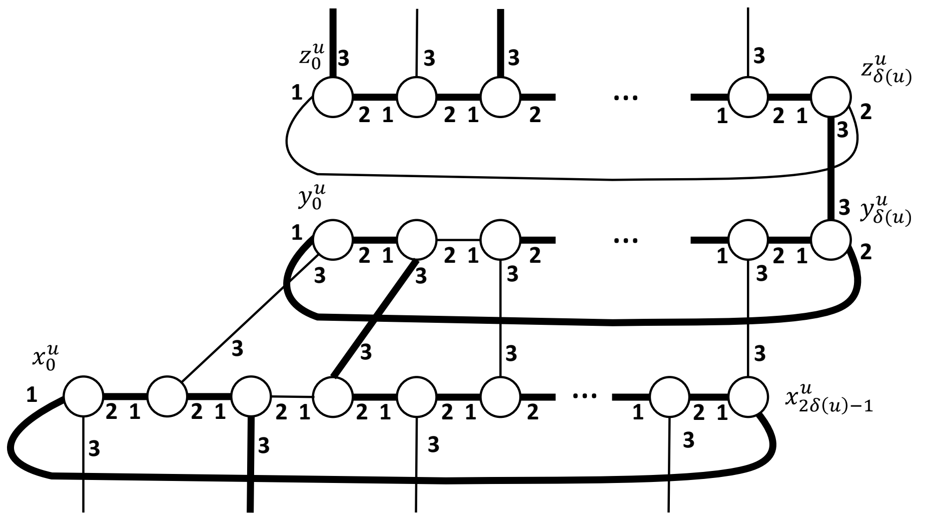

Fig. 2: The Lex-DFS tree in Γ_u. Assume that x_2^u is discovered first.

retical Aspects of Computer Science (STACS 2015), pp. 288–301 (online), DOI: 10.4230/LIPIcs.STACS.2015.288 (2015).

[3] Banerjee, N., Chakraborty, S. and Raman, V.: Improved Space Efficient Algorithms for BFS, DFS and Applications, *International Computing and Combinatorics Conference (COCOON)*, pp. 119–130 (2016).

[4] Chakraborty, S., Raman, V. and Satti, S. R.: Biconnectivity, Chain Decomposition and st-Numbering Using O(n) Bits, *International Symposium on Algorithms and Computation (ISAAC)*, Vol. 64, pp. 22:1–22:13 (online), DOI: 10.4230/LIPIcs.ISAAC.2016.22 (2016).

[5] Kammer, F., Kratsch, D. and Laudahn, M.: Space-Efficient Biconnected Components and Recognition of Outerplanar Graphs, *Algorithmica*, Vol. 81, No. 3, pp. 1180–1204 (online), DOI: 10.1007/s00453-018-0464-z (2019).

[6] Chakraborty, S. and Satti, S. R.: Space-efficient Algorithms for Maximum Cardinality Search, Its Applications, and Variants of BFS, *Jouanal of Combinatorial Optimization*, Vol. 37, No. 2, pp. 465–481 (online), DOI: 10.1007/s10878-018-0270-1 (2019).

[7] Hagerup, T.: Space-Efficient DFS and Applications: Simpler, Leaner, Faster, *CoRR*, (online), available from ⟨http://arxiv.org/abs/1805.11864⟩ (2018).

[8] Frederickson, G. N.: Upper bounds for time-space trade-offs in sorting and selection, *Journal of Computer and System Sciences*, Vol. 34, No. 1, pp. 19 – 26 (online), DOI: https://doi.org/10.1016/0022-0000(87)90002-X (1987).

[9] Reif, J. H.: Depth-first search is inherently sequential, *Information Processing Letters*, Vol. 20, No. 5, pp. 229 – 234 (online), DOI: https://doi.org/10.1016/0020-0190(85)90024-9 (1985).

[10] Anderson, R. and Mayr, E. W.: Parallelism and the maximal path problem, *Information Processing Letters*, Vol. 24, No. 2, pp. 121 – 126 (online), DOI: https://doi.org/10.1016/0020-0190(87)90105-0 (1987).

[11] Barnes, G., Buss, J. F., Ruzzo, W. L. and Schieber, B.: A Sublinear Space, Polynomial Time Algorithm for Directed S-t Connectivity, *SIAM Journal on Computing*, Vol. 27, No. 5, pp. 1273–1282 (online), DOI: 10.1137/S0097539793283151 (1998).

[12] Edmonds, J., Poon, C. K. and Achlioptas, D.: Tight Lower Bounds for st-Connectivity on the NNJAG Model, *SIAM Journal on Computing*, Vol. 28, No. 6, pp. 2257–2284 (online), DOI: 10.1137/S0097539795295948 (1999).

[13] Reingold, O.: Undirected Connectivity in Log-space, *Journal of the ACM*, Vol. 55, No. 4 (online), DOI: 10.1145/1391289.1391291 (2008).

[14] Kiyomi, M., Ono, H., Otachi, Y., Schweitzer, P. and Tarui, J.: Space-Efficient Algorithms for Longest Increasing Subsequence, *Theory of Computing Systems*, (online), DOI: 10.1007/s00224-018-09908-6 (2019).

[15] Wang, J. R.: Space-Efficient Randomized Algorithms for K-SUM, *Algorithms - ESA 2014*, pp. 810–829 (2014).

[16] Lincoln, A., Williams, V. V., Wang, J. R. and Williams, R. R.: Deterministic Time-Space Trade-Offs for k-SUM, *43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 55, pp. 58:1–58:14 (online), DOI: 10.4230/LIPIcs.ICALP.2016.58 (2016).

[17] Darwish, O. and Elmasry, A.: Optimal Time-Space Tradeoff

for the 2D Convex-Hull Problem, *Algorithms - ESA 2014*, pp. 284–295 (2014).

[18] Banyassady, B., Korman, M., Mulzer, W., van Renssen, A., Roeloffzen, M., Seiferth, P. and Stein, Y.: Improved Time-Space Trade-Offs for Computing Voronoi Diagrams, *34th Symposium on Theoretical Aspects of Computer Science (STACS)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 66, pp. 9:1–9:14 (online), DOI: 10.4230/LIPIcs.STACS.2017.9 (2017).

[19] Pagter, J. and Rauhe, T.: Optimal time-space trade-offs for sorting, *39th Annual Symposium on Foundations of Computer Science (STOC)*, pp. 264–268 (online), DOI: 10.1109/SFCS.1998.743455 (1998).

[20] Borodin, A. and Cook, S.: A Time-Space Tradeoff for Sorting on a General Sequential Model of Computation, *SIAM Journal on Computing*, Vol. 11, No. 2, pp. 287–297 (online), DOI: 10.1137/0211022 (1982).

[21] Chan, T. M.: Comparison-based Time-space Lower Bounds for Selection, *ACM Transactions on Algorithms*, Vol. 6, No. 2, pp. 26:1–26:16 (online), DOI: 10.1145/1721837.1721842 (2010).

[22] Munro, J. and Paterson, M.: Selection and sorting with limited storage, *Theoretical Computer Science*, Vol. 12, No. 3, pp. 315 – 323 (online), DOI: https://doi.org/10.1016/0304-3975(80)90061-4 (1980).

[23] Chan, T. M., Munro, J. I. and Raman, V.: Faster, Space-Efficient Selection Algorithms in Read-Only Memory for Integers, *Algorithms and Computation*, pp. 405–412 (2013).

[24] Khan, S. and Mehta, S. K.: Depth First Search in the Semi-streaming Model, *International Symposium on Theoretical Aspects of Computer Science (STACS)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 126, pp. 42:1–42:16 (online), DOI: 10.4230/LIPIcs.STACS.2019.42 (2019).

[25] Chakraborty, S., Mukherjee, A., Raman, V. and Satti, S. R.: A Framework for In-place Graph Algorithms, *26th Annual European Symposium on Algorithms (ESA 2018)* (Azar, Y., Bast, H. and Herman, G., eds.), Leibniz International Proceedings in Informatics (LIPIcs), Vol. 112, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 13:1–13:16 (online), DOI: 10.4230/LIPIcs.ESA.2018.13 (2018).

[26] Kammer, F. and Sajenko, A.: Linear-Time In-Place DFS and BFS on the Word RAM, *Algorithms and Complexity*, pp. 286–298 (2019).

[27] Buhrman, H., Cleve, R., Koucký, M., Loff, B. and Speelman, F.: Computing with a Full Memory: Catalytic Space, *Forty-sixth Annual ACM Symposium on Theory of Computing (STOC)*, pp. 857–866 (online), DOI: 10.1145/2591796.2591874 (2014).

[28] Barba, L., Korman, M., Langerman, S., Sadakane, K. and Silveira, R. I.: Space–Time Trade-offs for Stack-Based Algorithms, *Algorithmica*, Vol. 72, No. 4, pp. 1097–1129 (online), DOI: 10.1007/s00453-014-9893-5 (2015).