Basic Research in Computer Science

BRICS RS-96-9

Lower Bounds for Dynamic Transitive Closure, Planar Point Location, and Parentheses Matching

Thore Husfeldt
Theis Rauhe
Søren Skyum

BRICS Report Series **RS-96-9**

ISSN 0909-0878 **April 1996**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark**

**Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.dk/`
`ftp ftp.brics.dk (cd pub/BRICS)`

LOWER BOUNDS FOR DYNAMIC TRANSITIVE CLOSURE, PLANAR POINT LOCATION, AND PARENTHESES MATCHING

THORE HUSFELDT, THEIS RAUHE, AND SØREN SKYUM

BRICS*
Department of Computer Science, University of Aarhus
Ny Munkegade, DK–8000 Århus C, Denmark

Abstract. We give a number of new lower bounds in the cell probe model with logarithmic cell size, which entails the same bounds on the random access computer with logarithmic word size and unit cost operations.

We study the signed prefix sum problem: given a string of length n of zeroes and signed ones, compute the sum of its ith prefix during updates. We show a lower bound of $\Omega(\log n/\log\log n)$ time per operations, even if the prefix sums are bounded by $\log n/\log\log n$ during all updates. We also show that if the update time is bounded by the product of the worst-case update time and the answer to the query, then the update time must be $\Omega\big(\sqrt{(\log n/\log\log n)}\big)$.

These results allow us to prove lower bounds for a variety of seemingly unrelated dynamic problems. We give a lower bound for the dynamic planar point location in monotone subdivisions of $\Omega(\log n/\log\log n)$ per operation. We give a lower bound for the dynamic transitive closure problem on upward planar graphs with one source and one sink of $\Omega(\log n/(\log\log n)^2)$ per operation. We give a lower bound of $\Omega\big(\sqrt{(\log n/\log\log n)}\big)$ for the dynamic membership problem of any Dyck language with two or more letters. This implies the same lower bound for the dynamic word problem for the free group with k generators. We also give lower bounds for the dynamic prefix majority and prefix equality problems.

1. Introduction

We introduce a new technique for proving lower bounds for dynamic algorithms in the cell probe model. With this, we easily derive lower bounds for half a dozen natural problems, including the following:

Computational Geometry: Dynamic planar point location in monotone subdivisions cannot be solved faster than $\Omega(\log n/\log\log n)$. The important algorithm of Preparata and Tamassia [16] achieves upper bounds of $O(\log^2 n)$ per update and $O(\log n)$ per query.

Graph algorithms: Dynamic transitive closure in planar acyclic digraphs with one source and one sink that are on the same face cannot be solved faster than $\Omega(\log n/(\log\log n)^2)$. Tamassia and Preparata [17] achieve a logarithmic upper bound for this problem.

Parentheses matching: The dynamic membership problem for the Dyck languages (alternatively, the dynamic word problem for the free group) cannot be solved faster than $\Omega\big(\sqrt{\log n/\log\log n}\big)$ per operation. Frandsen et al. [7] show polylogarithmic upper bounds for this problem.

*Basic Research in Computer Science, Centre of the Danish National Research Foundation.

No lower bounds (in the cell probe model) for any of these problems have been published to the knowledge of the authors, even though an exponentially worse bound of $\Omega(\log\log n/\log\log\log n)$ can be seen to hold for most of them, using an unpublished result by Beame and Fich [3], that improves work of Miltersen [13]. Lower bounds for harder variants of dynamic transitive closure and dynamic planar point location follow from [9, 12].

In the rest of the paper, our nomenclature for dynamic problems omits the prefix 'dynamic' for brevity, since all our problems are of this type.

1.1. **Roadmap.** The Introduction contains a short presentation of our model of computation and recalls a seminal result for this, the *Fredman–Saks bound* [10], that will be important to us.

The main part of the paper proves two theorems that can be viewed as generalisations of the Fredman–Saks bound. The lower bounds for planar point location and transitive closure follow from the first theorem, while the bound for parentheses matching follows from the second.

Much of the present paper can be enjoyed without knowledge of the proof of the Fredman–Saks bound; hence we present our results as a reduction to that result rather than modifying the original proof [10], trading elegance for (what we hope is) readability.

1.2. **The Cell Probe Model.** Our lower bounds work in the cell probe model [19], where the only resource is memory access—all computation is for free; we consider word size $\mathrm{O}(\log n)$ for concreteness. Hence our bounds hold on all natural (and some unnatural) models of random access machines with logarithmic word size. For example, the cost or availability of operations like multiplication does not enter into the argument. Also, there are no assumptions on the the size of the memory or its organisation relative to the structure of the instance.

The cell probe model is so strong that no lower bound better than $\Omega(\log n/\log\log n)$ is known for *any* problem in Polynomial Time. This is discouraging in light of the fact that the best known algorithm for many dynamic problems is 'recompute from scratch.' However, recent breakthroughs in algorithms for random access machines [1, 2, 18] show that widely-held beliefs about the relevance *for real computers* of lower bounds in weaker models may be mistaken.

Unfortunately, cell probe lower bounds are often hard to come by and the range of general techniques is limited. We believe that the techniques of the present paper are widely applicable; we substantiate this claim by proving new lower bounds for several well-studied problems from various fields.

1.3. **Prefix Parity.** We use a result of Fredman and Saks [10] that gives a lower bound on the complexity of the *prefix parity* problem: given a vector $x_1, \ldots, x_n$ of bits, maintain a data structure that is able to react to the following operations for all $i = 1, \ldots, n$:

change(i): negate the value of x_i,

parity(i): return $\bigoplus_{j=1}^{i} x_j$, the parity of the first i elements.

The Fredman–Saks bound. *Let t_{u} denote the worst-case update time and let t_{q} denote the worst-case query time for any solution of the prefix parity problem. Then*

$$t_{\mathrm{q}} \in \Omega\Big(\frac{\log n}{\log(t_{\mathrm{u}}\log n)}\Big). \tag{1}$$

Especially, no algorithm can run faster than $\Omega(\log n/\log\log n)$ time per operation. Dietz [6] shows this bound to be tight.

We mention at this point that the Fredman–Saks bound holds for *amortised* complexity also. So do all our results; we claim this without proof and will not mention it again to keep the presentation simple.

2. Prefix Balancing

2.1. **Signed Prefix Sum.** To prove our lower bounds, we introduce the *signed prefix sum* problem. Given a vector $y_1, \ldots, y_n \in \{0, \pm 1\}^n$, maintain a data structure that is able to react to the following operations for all $i = 1, \ldots, n$:

change(i, a)**:** let $y_i = a \in \{0, \pm 1\}$,

sum(i)**:** return $\sum_{j=1}^{i} y_j$.

Obviously, the Fredman–Saks bound holds for signed prefix sum, since it is a generalisation of the prefix parity problem. The data structure of Dietz [6] can be used for an optimal $\Theta(\log n/\log\log n)$ implementation.

2.2. **Range Reduction.** We first show that signed prefix sum remains difficult even when the range of $\sum_{j\le i} y_j$ is reduced.

Theorem 1. *Let t_{u} denote the worst-case update time and let t_{q} denote the worst-case query time for any solution of the signed prefix sum problem with the restriction that at all times during the updates,*

$$\Big|\sum_{j=1}^{i} y_j\Big| = \mathrm{O}\Big(\frac{\log n}{\log\log n}\Big) \quad \textit{for all } 1 \le i \le n. \tag{2}$$

Then (1) *holds.*

Proof. Let $x \in \{0,1\}^n$ be an instance of the prefix parity problem and assume that we have a solution to the signed prefix sum problem that works under the restriction (2). We construct an instance $y \in \{0, \pm 1\}^n$ to the latter such that $y_i = 0$ if and only if $x_i = 0$. Note that since $1 = -1 \bmod 2$ we can answer the query **parity**(i) by returning the parity of **sum**(i). To prove the desired lower bound we only have to keep the value of $\sum_{j\le i} y_j$ small. The rest of the proof explains how to construct y with this property—namely, how to choose values from $\{-1, +1\}$ for the nonzero elements.

Write

$$m = \left\lceil \frac{\log n}{\log\log n} \right\rceil$$

in the rest of the paper. We introduce sequences $w_1, \ldots, w_m$ such that w_1 is a subsequence of y and w_{k+1} is a subsequence of w_k. The sequences are defined as follows: w_1 contains exactly the nonzero entries of y. The elements of each sequence are either *single* or *coupled* with a neighbouring element. The singles in the kth sequence constitute the elements of w_{k+1}.

Note that every nonzero element of y is coupled at most once among all sequences. The values of these elements will be chosen from $\{+1, -1\}$ such that the sum of each couple is zero.

We will maintain a distance invariant: that there are at least $\log n$ elements between any two singles. This ensures that $|w_{k+1}| \le \lceil |w_k|/\log n \rceil$, so w_m contains

at most a (single) element, which we will pair up with a dummy element y_{n+1} for simplicity.

Let us see how to maintain the invariant during updates. Whenever a bit in x is flipped, a nonzero element in y becomes zero or vice versa, which means that an element is inserted into or deleted from w_1.

insertions: Consider the case where a new element y_i is inserted into list w_k. If there are no singles among the nearest $2\log n$ neighbours of y_i then we can insert y_i as a single in w_k and as a new element in w_{k+1}.

Otherwise, if there is a single y_j close to i, we have to change the coupling of elements in w_k. According to the invariant, all elements in the $\log n$ neighbourhood of y_j are coupled. We make new couples of these elements and y_j and y_i and remember to delete y_j from w_{k+1}.

deletions: Consider now the case where we want to delete y_i from w_k. If y_i is a single then we remove it and delete it from w_{k+1}. The case where y_i is coupled with some y'_i is handled as above, since deleting y_i corresponds to removing the couple and inserting y'_i as a single.

In both cases, the distance invariant is maintained, at most $\mathrm{O}(\log n)$ are recoupled and at most one element inserted into or deleted from w_{k+1}. Hence the update time for each **change** operation in the worst case is

$$t_{\mathrm{u}} = \mathrm{O}\Big(\frac{\log^2 n}{\log\log n} \cdot t\Big), \tag{3}$$

where t is the update time of the data structure for signed prefix sum.

Now for the query operation. We first explain how the values from $\{-1, +1\}$ are given to the nonzero element of y. The rule is straightforward and arbitrary: when a new couple is created, the leftmost element is assigned the value -1 and the rightmost $+1$.

To find $\sum_{j\le i} y_j$, first note that we can ignore couples (y_i, y'_i) with $i, i' \le j$, since their sum is zero. So we restrict our attention to elements y_i in couples (y_i, y'_i) with $i \le j < i'$. In every sequence w_k there can be at most one such couple (because elements are coupled with a neighbour in the same sequence), so the number of contributing y_i is at most m. This ensures that the range condition (2) holds and proves correctness. The bound on the query time follows from the Fredman–Saks bound. □

2.3. **Planar Point Location.** A classical problem in Computational Geometry is *planar point location*: given a subdivision of the plane, i.e. a partition into polygonal regions induced by the straight-line embedding of a planar graph, determine the region of query point $q \in \mathbb{R}^2$.

In the dynamic version, updates consist of insertion and deletion of vertices or (chains of) edges. An important restriction of the problem, for which our bound will apply, considers only *monotone* subdivisions, where the subdivision consists of polygons that are monotone (so no straight line crosses any polygon more than twice). Preparata and Tamassia [16] give an algorithm that runs in time $\mathrm{O}(\log^2 n)$ per operation. Several other dynamic algorithms for this and other types of subdivisions have been found since, see [4] for a survey.

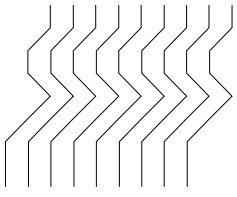


Fig. 1

To prove a lower bound for this problem we construct a monotone subdivision from the signed prefix sum instance $y \in \{0, \pm 1\}^n$. This is easier drawn than explained formally; Fig. 1 shows the subdivision corresponding to $y = (0,\ 0,\ +1,\ +1,\ -1,\ 0,\ +1\ ,0)$. There are 2 unbounded polygons at the sides and $2m$ strip-like ones with common sides and common top and bottom corners at infinity. Each of the strip-like polygons mimics the path described by y with $+1$ meaning 'go right' and -1 meaning 'go left.'

Now comes the only, but crucial, use of our range reduction (2): when y is changed, the subdivision can be updated in polylogarithmic time (given a fast data structure for planar point location) because is is so narrow.

To answer a sum query for the ith prefix, we query the names of the polygons that contain the points $(0, 0)$ and $(0, i)$ (assuming some appropriate placement of the origin). The distance between these polygons is precisely the answer, so we can indeed implement a data structure for signed prefix sum. We conclude that (1) is a lower bound on the time per operation for planar point location in monotone subdivisions.

3. Binary Search

Using a simple binary search strategy, we can prove lower bounds for a variety of other problems. The cleanest application is for the majority function. This tells us a bit about the complexity of range searching as well. Following the same melody, we give lower bounds for a dynamic graph problem.

3.1. **Prefix Majority.** The *prefix majority* problem is defined in analogy with the prefix parity problem from Sect. 1.3; the query operation is

majority(i)**:** return 'true' iff there are more ones than zeroes in the ith prefix.

We will show how to use Thm. 1 to prove a lower bound of

$$t_{\mathrm{q}} = \Omega\Big(\frac{\log n}{\log\log^2 n}\Big), \qquad \text{if } t_{\mathrm{u}} = \log^{\mathrm{O}(1)} n \tag{4}$$

for any implementation, where t_{u} and t_{q} denote the update and query time, respectively. We know no better upper bound than $\mathrm{O}(\log n/\log\log n)$ per operation (again, Dietz' data structure [6]), so the result leaves a double-logarithmic gap.

To see that (4) holds, let $y \in \{0, \pm 1\}^n$ be an instance of signed prefix sum. We first construct an instance $x \in \{0, 1\}^{2n}$ to the prefix majority problem in the obvious way:

$$-1 \mapsto 00, \qquad 0 \mapsto 01, \qquad +1 \mapsto 11. \tag{5}$$

The majority of a prefix of this instance is one if and only if the signed sum of the corresponding prefix in y is positive. This is the main idea.

To learn the *exact* values of y's prefixes we maintain $2m+1$ bitstrings x_i, where $x_i = (11)^i x$ for positive i and $x_i = (00)^i x$ for negative i. We encourage the reader to check that this facilitates a binary search for the exact number of ones in x's prefixes. The query time for this (and hence for the prefix sum of y) is $t_{\mathrm{q}} \log m$, and since the update time is polylogarithmic if t_{u} is, the bound (4) follows from Thm. 1.

Note that while we still use the range reduction (2) to maintain our construction in polylogarithmic time during *updates* (as in the previous application), we now also use it to reduce the *query* time.

3.2. **Range Searching.** A fundamental algorithmic problem is *range searching*; we can put the above result in that framework. The problem is to maintain a set $S \subseteq \mathbb{R}^d$ (for our lower bound, $d = 1$ is hard enough) under the following operations:

insert(x)**:** insert a point at coordinate $x \in \mathbb{R}^d$ into S,
delete(x)**:** remove the point at $x \in \mathbb{R}^d$ from S,
report(R)**?:** how many points are the in $R \cap S$, where R is a rectangle in $\mathbb{R}^d$.

The problem has been studied for many other query operations and our understanding of its complexity varies with the type of query. For *counting* (as above), the Fredman–Saks bound applies even in one dimension. On the other hand, the problem of *existential range queries* (return 'yes' iff $R \cap S$ is nonempty) is among the most interesting problems at the time of writing, see [14] for some results.

Our lower bound applies to versions of the problem where the query operation involves the majority function in some disguise. Here is one:

insert(x, c)**:** insert $x \in R^d$ of colour $c \in \{\text{blue}, \text{red}\}$ into S,
delete(x)**:** remove the point at x if it exists,
blue(R)**:** are there more blue than red points in $R \cap S$?

This corresponds to asking questions like 'among the students aged 20 to 25, are there more males than females?'. Alternatively, in the monochromatic setting, we can ask: 'Are there more students aged 20 to 25 than 23 to 30?', reflected in the following query:

more(R_1, R_2)**:** is $|R_1 \cap S| > |R_2 \cap S|$?

We leave it to the reader formalise this and show that (4) is a lower bound.

3.3. **Upward planar graphs.** A graph is planar if it can be embedded in the plane without crossing edges. A digraph is *upward planar* if it admits a planar embedding where all edges are directed upward, i.e. their projection on the y-axis is positive; such a graph is clearly acyclic.

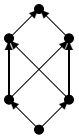


Fig. 2

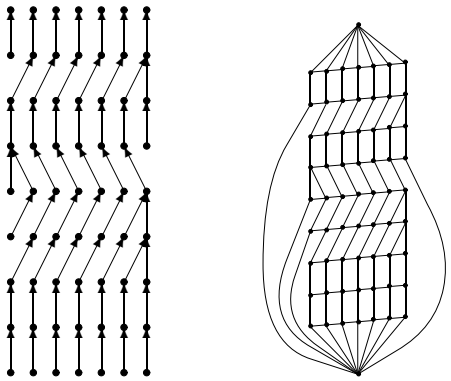
There are planar dags that are not upward planar, like shown in Fig. 2. A digraph is a *source–sink* graph, or *st*-graph for short, if it has only one vertex with no incoming edges (the source s) and only one vertex with no outgoing edges (the sink t). It is well known that a graph is upward planar iff it is the subgraph of an acyclic planar *st*-graph that has s and t on the same face. Fig. 2 shows the last condition to be necessary. The survey [5] contains a recent list of references to other characterisations of these classes and many applications in graph drawing; see [17] for more applications.

We give a lower bound of (4) for the *transitive closure*, i.e. for data structures that handle the following operations:

insert(u, v)**:** insert an edge from vertex u to vertex v,
delete(u, v)**:** delete the edge from u and v,
path(u, v)**:** 'Is there a path from u to v?'

Our bound holds even under the severe restriction that at all times, the graph remains an upward planar *st*-graph (with the same embedding). The prize for this generality, compared to a related result [9, 12], is a double-logarithmic factor in the lower bound.

The construction is very similar to that for planar point location in Sect. 2.3. From an instance $y \in \{0, \pm 1\}^n$ of signed prefix sum, we construct a digraph $G = (V, E)$. The vertex set consists of the source s, the sink t, and $2m + 3$ vertices for



(a) (b)

Fig. 3

each letter y_i:

$$V = \{\, v_{ij} \mid 1 \le i \le n+1, -m-1 \le j \le m+1 \,\}.$$

The ith row is connected to its upper neighbour according to the value of y_i:

$$\{\, (v_{ij}, v_{(i+1)j'}) \mid 1 \le i \le n, -m-1 \le j \le m+1 \,\},$$

$$\text{where } j' = \begin{cases} j + y_i, & \text{if } |j + y_i| \le m+1, \\ m+1, & \text{if } j + y_i = m+2, \\ -m-1, & \text{if } j + y_i = -m-2. \end{cases}$$

For example, if y is $(0, 0, +1, +1, -1, 0, +1, 0)$ the edges look like Fig. 3(a). Note how the path starting in $(1, 0)$ (the middle vertex in the bottom row) mimics $s_i = \sum_{j \le i} x_j$. Indeed, there is a path from $(1, 0)$ to $(i + 1, u)$ for $1 \le i \le n$ and $-m \le u \le m$ if and only if $s_i = u$. We are going to use the transitive closure data structure to detect this.

First, we finish the construction by adding some more edges that have only technical significance and make sure that G is an st-graph. At the ends of the graph, $2m + 3$ edges connect s to the bottom row and $2m + 1$ edges connect the topmost row to t,

$$\{\, (s, v_{1j}), (v_{j(n+1)}, t) \mid -m-1 \le j \le m+1 \,\}.$$

At the top- and bottommost rows, edges connect s to all vertices that would otherwise be sources:

$$\{\, (s, v_{i(m+1)}) \mid y_{i-1} = -1 \,\} \cup \{\, (s, v_{i(-m-1)}) \mid y_{i-1} = 1 \,\}.$$

From this we construct two graphs G_+ and G_-. In G_+, edges connect every vertex (u, v) with $1 \le u \le n$, $-m - 1 \le v \le m$ to $(u, v + 1)$. The other graph G_- is constructed symmetrically, with all (u, v) connected to $(u, v - 1)$.

Figure 3(b) depicts it for our example. We have displaced the vertex rows slightly to make clear that all edges are directed upward. Arrows are removed for readability. The vertices at the bottom and top are s and t, respectively.

The desired property of G_+ is this: If $j = s_i$ then there is a path from v_{10} not only to v_{ij} but also to $v_{ij'}$ for any $j' \ge j$ but still none for $j' < j$. Likewise, in G_-, there is a path to $v_{ij'}$ for any $j' \le j$ and none for $j' > j$. But now we can do a binary search; the rest of the proof is similar to Sect. 3.1.

4. Randomised Prefix Balancing

We return to the signed prefix sum problem and state and prove our second theorem. In this version, the time for a query is expressed in terms of the size of its answer.

For a quick motivation, assume that we have a data structure that can only check for zero, i.e. answer 'zero' iff $\sum_{j\le i} y_j = 0$. Then we can use an exhaustive search strategy to solve signed prefix sum: (Assume for simplicity that the sum is positive.)

1. pad y with a number of zeroes to the left,
2. change a padded zero to -1 and query again,
3. repeat from 2 until we get the answer 'zero.'

Clearly, the number of iterations is exactly the number of -1s we used to balance the prefix sum down to zero, which in turn is the answer to the query. Theorem 2 allows us to give lower bounds for such a check-for-zero data structure.

Theorem 2. *Consider any solution to the signed prefix sum problem. For $1 \le i \le n$, let t_{q}^i denote the time for **sum**(i) and let t_{u} denote the worst-case time for any update. Then*

$$t_{\mathrm{u}} = \Omega\Bigl(\sqrt{\frac{\log n}{\log\log n}}\,\Bigr), \qquad \text{if } t_{\mathrm{q}}^i = \mathrm{O}\Bigl(t_{\mathrm{u}} \cdot \Bigl|\sum_{j=1}^{i} y_j\Bigr|\Bigr). \tag{6}$$

This bound holds even with the restriction (2)*.*

The proof uses the well-known fact that in a series of independent and fair coin flips, even though the difference between the number of heads and tails after the nth trail may be as big as n, the *expected* value is much smaller:

$$E\bigl(|\#\text{heads} - \#\text{tails}|\bigr) = \Theta(\sqrt{n}\,).$$

For a proof, solve Problem 1.6 of [15].

For this idea to work, we first have to observe that the Fredman–Saks bound works for *expected* query time as well.

Lemma 1. *Let t_{u} denote the worst-case update time and let t_{q} denote the expected query time for any solution of the prefix parity problem. Then*

$$t_{\mathrm{q}} = \Omega\Bigl(\frac{\log n}{\log(t_{\mathrm{u}} \log n)}\Bigr).$$

Sketch of proof. Equation (2) in Theorem 3 of [10] states the bound for q being the *worst-case* query time. This can be extended to expected time using Yao's Minimax principle. □

Proof of Theorem. We return to the proof of Theorem 1 and modify our scheme for giving values $\{-1, +1\}$ to the nonzero elements of y. The rule is quite straightforward: when a new couple is created, either the leftmost element is assigned the value -1 and the rightmost $+1$ or vice versa, depending on a fair coin toss.

But then, by Probability Theory, we have

$$E\Bigl(\Bigl|\sum_{j=1}^{i} y_j\Bigr|\Bigr) = \Theta(\sqrt{m}\,), \tag{7}$$

where the expectation is over the coin tosses used to determine the values of each couple.

Hence the expected time for a prefix parity query is

$$\mathrm{O}\left(t_{\mathrm{u}} \cdot \sqrt{\frac{\log n}{\log\log n}}\right).$$

The theorem now follows from the above lemma. □

4.1. **Prefix Equality.** Consider yet another relative to the prefix parity and majority problems, the prefix *equality* problem. The query operation is:

equal(i)**:** return 'true' iff the number of ones equals the number of zeroes in the ith prefix.

We will show that

$$t = \Omega\left(\sqrt{\frac{\log n}{\log\log n}}\right) \tag{8}$$

is a lower bound on the worst-case time per operation for any implementation.

To see this, consider any algorithm for prefix equality with worst-case time t per operation. From an instance $y \in \{0, \pm 1\}^n$ of signed prefix sum we construct two strings $x_+, x_- = (01)^m x$, where $x \in \{0,1\}^{2n}$ is constructed as in (5). For every signed prefix sum query we perform an exhaustive search by repeating the following until we get the answer 'true':

1. use **equal** to see if x_+ or x_- balance,
2. flip a zero among the first $2m$ letters of x_+ to one,
3. flip a one among the first $2m$ letters of x_- to zero.

The query time is $\mathrm{O}(t \cdot \sum_{j \le i} y_j)$ (we remember to change both strings back) and the update time is $2t$. The last theorem provides the stated bound on t.

The reader should now be able to show lower bounds of the same size for upward planarity testing or for planar point location where the query returns 'yes' iff two points are in the same polygon.

5. Parentheses Matching

The motivation for our last problem comes from modern editors. In many of them, a rudimentary syntax check is performed during editing, we focus on the feature of matching parentheses. Frandsen et al. [7] give polylogarithmic upper bounds for this problem; their lower bounds leave an exponential gap. Using the last theorem, we improve these bounds.

5.1. **Dyck Languages.** The language of *properly balanced parentheses* contains strings like ( ) and ( )(( )) but not)). The notion of balancedness also makes sense if we add more types of parentheses: ([ ])( ) balances but [ ) does not.

More formally, let $A = \{a_1, \ldots, a_k\}$ and $\bar{A} = \{\bar{a}_1, \ldots, \bar{a}_k\}$ be two disjoint sets of opening and closing symbols, respectively. For example, the pair $A = \{(, [, \mathbf{do}, \mathbf{if}\}$ and $\bar{A} = \{),], \mathbf{od}, \mathbf{fi}\}$ captures the nested structure of programming languages. The *one-sided Dyck language* D_k over $A \cup \bar{A}$ is the context-free language generated by the following grammar:

$$S \to SS \mid a_1 S \bar{a}_1 \mid \cdots \mid a_k S \bar{a}_k \mid \epsilon.$$

Closely related is the *two-sided Dyck language* D'_k over $A \cup \bar{A}$ defined by

$$S \to SS \mid a_1 S \bar{a}_1 \mid \bar{a}_1 S a_1 \mid \cdots \mid a_k S \bar{a}_k \mid \bar{a}_k S a_k \mid \epsilon.$$

This corresponds to two-sided cancellation, so now also)(and (][) balance, while [] still does not.

The two-sided Dyck language has an algebraic interpretation. If we identify $\bar{a}_i$ with a_i^{-1} and view concatenation as the product operator then $x \in D'_k$ if and only if x equals the identity in the free group generated by A. For example, $\bar{a}_1 a_2 \bar{a}_2 a_1 \in D'_2$ because $a_1^{-1} a_2 a_2^{-1} a_1$ evaluates to unity.

The Dyck languages bear the name of the German mathematician Walther von Dyck (1856–1934). They are covered in detail in Harrison's classical treatment [11].

5.2. **The Membership problem.** We consider the problem of maintaining membership in D_k or D'_k of a string from $(A \cup \bar{A})^n$ dynamically. Given a vector $x \in (A \cup \bar{A})^n$ of even length, initially a_1^n, maintain x under the following operations for any Dyck language D:

change(i, a)**:** change x_i to $a \in A \cup \bar{A}$,
member: return 'yes' if and only if $x \in D$.

Alternatively, we can use this set of updates for analysing the *word* problem for the free group. Here, the **member** query returns 'yes' if and only if $\prod_i x_i = 1$. (In this context, **product** or **identity** may be better names for the query.) However, we will refrain from distinguishing between the word problem for the free group and the membership problem for two-sided Dyck languages. Frandsen, Miltersen, and Skyum [8] study dynamic word problems for other monoids.

5.3. **Interval queries.** We begin by showing that (8) is a lower bound for the two *single-letter* languages with a more powerful query:

interval(i, j)**:** return 'yes' if $x_i \dots x_j \in D$.

Let $y \in \{0, \pm 1\}^n$ be an instance of signed prefix sum. Construct an instance of the Dyck problem,

$$h(y_n) h(y_{n-1}) \dots h(y_1)\, \bar{a}^{4n}$$

where

$$h(y_i) = \begin{cases} aaaa, & \text{if } y_i = +1, \\ aaa\bar{a}, & \text{if } y_i = 0, \\ a\bar{a}a\bar{a}, & \text{if } y_i = -1. \end{cases}$$

Note that for any i, the string $h(y_i) \dots h(y_1) \bar{a}^{2(i+s)}$ balances if and only if $s = \sum_{j \le i} y_j$. Therefore to answer a **sum** query, we check all intervals for $x = 0$, -1, $+1$, -2, $+2$, ..., until an interval balances. This takes time $O(t \cdot |s_i|)$ in the worst case, where t is the time for an **interval**-query. The bound follows from Theorem 2.

5.4. **Lower Bound for Language Membership.** We now show that for Dyck languages with *two or more* letters, (8) is a lower bound even for language membership, i.e. using the original **member**-query.

We first prove the claim for D'_2. We will use the **member** query for D'_2 to solve an instance of the problem from the last section.

Let $x \in \{a, \bar{a}\}^n$ be an instance of the interval problem for D'_1. Let

$$y = a\bar{a}x_1 a\bar{a}x_2 a\bar{a} \dots a\bar{a}x_n a\bar{a}x^{\mathrm{R}}$$

be an instance of the membership problem and note $y \in D_2'$. To answer a query **interval**(i, j) we merely insert a matching pair of other parentheses in y at the corresponding place:

$$y' = a\bar{a}x_1a\bar{a}\ldots x_{i-1}bbx_i\ldots x_j\overline{b}\overline{b}x_{j+1}\ldots x^{\mathrm{R}},$$

where x^{R} denotes x reversed. It is easy to see that $y' \in D_2'$ iff $x_i \ldots x_j \in D_1'$. After the query, y' is changed back to y.

In the one-sided case, we have to extend both ends of the instance with parentheses to

$$y = a^{2n}a\bar{a}x_1a\bar{a}x_2a\bar{a}\ldots a\bar{a}x_na\bar{a}x^{\mathrm{R}}\bar{a}^2n,$$

just to make sure $y \in D_2$. The rest of the proof is the same.

Acknowledgements. The authors thank Gudmund Skovbjerg Frandsen and Peter Bro Miltersen for their co-operation.

References

[1] Arne Andersson. Sublogarithmic searching without multiplications. In *Proc. 36thFOCS*, pages 655–663. IEEE Computer Society, 1995.

[2] Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. Sorting in linear time? In *Proc 27thSTOC*, pages 427–436, 1995.

[3] Paul Beame and Faith Fich, 1994. Personal communication, reported by Peter Bro Miltersen.

[4] Yi-Jen Chiang and Roberto Tamassia. Dynamic algorithms in computational geometry. Technical Report CS-91-24, Dept. of Comp. Sc., Brown University, 1991.

[5] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Algorithms for drawing graphs: an annotated bibliography. Available via anonymous ftp from wilma.cs.brown.edu in /pub/papers/compgeo/gdbiblio.ps.Z, 1994.

[6] Paul F. Dietz. Optimal algorithms for list indexing and subset rank. In *Proc. First Workshop on Algorithms and Data Structures (WADS)*, volume 382 of *Lecture Notes in Computer Science*, pages 39–46. Springer Verlag, Berlin, 1989.

[7] Gudmund Skovbjerg Frandsen, Thore Husfeldt, Peter Bro Miltersen, Theis Rauhe, and Søren Skyum. Dynamic algorithms for the Dyck languages. In *Proc. 4th WADS*, volume 955 of *Lecture Notes in Computer Science*, pages 98–108. Springer, 1995.

[8] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. In *Proc 34th FOCS*, pages 470–479, 1993.

[9] Michael L. Fredman and Monika Rauch Henzinger. Lower bounds for fully dynamic connectivity problems in graphs. Manuscript, preliminary version in STOC 94.

[10] Michael L. Fredman and Michael E. Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st STOC*, pages 345–354, 1989.

[11] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[12] P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130:203–236, 1994.

[13] Peter Bro Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Proc. 26th STOC*, pages 625–634. ACM, 1994.

[14] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. In *Proc. 27th STOC*, pages 103–111. ACM, 1995.

[15] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[16] Franco P. Preparata and Roberto Tamassia. Fully dynamic point location in a monotone subdivision. *SIAM Journal of Computing*, 18(4):811–830, 1989.

[17] Roberto Tamassia and Franco P. Preparata. Dynamic maintenance of planar digraphs, with applications. *Algorithmica*, 5:509–527, 1990.

[18] Mikkel Thorup. On ram priority queue. In *Proc 7thAnn. Symp. on Discrete Algorithms (SODA)*, pages 59–67, 1996.

[19] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, July 1981.

Recent Publications in the BRICS Report Series

RS-96-9 **Thore Husfeldt, Theis Rauhe, and S ren Skyum.** ***Lower Bounds for Dynamic Transitive Closure, Planar Point Location, and Parentheses Matching*****. April 1996. 11 pp. To appear in** ***Algorithm Theory: 5th Scandinavian Workshop,*** **SWAT '96 Proceedings, LNCS, 1996.**

RS-96-8 **Martin Hansen, Hans Hüttel, and Josva Kleist.** ***Bisimulations for Asynchronous Mobile Processes*****. April 1996. 18 pp. Appears in** ***Tbilisi Symposium on Language, Logic, and Computation*****, 1995.**

RS-96-7 **Ivan Damgård and Ronald Cramer.** ***Linear Zero-Knowledegde - A Note on Efficient Zero-Knowledge Proofs and Arguments*****. April 1996. 17 pp.**

RS-96-6 **Mayer Goldberg.** ***An Adequate Left-Associated Binary Numeral System in the λ-Calculus (Revised Version)*****. March 1996. 19 pp. Accepted for** ***Information Processing Letters*****. This report is a revision of the BRICS Report RS-95-38.**

RS-96-5 **Mayer Goldberg.** ***Gödelisation in the λ-Calculus (Extended Version)*****. March 1996. 10 pp.**

RS-96-4 **Jørgen H. Andersen, Ed Harcourt, and K. V. S. Prasad.** ***A Machine Verified Distributed Sorting Algorithm*****. February 1996. 21 pp. Abstract appeared in** ***7th Nordic Workshop on Programming Theory,*** **NWPT '7 Proceedings, 1995.**

RS-96-3 **Jaap van Oosten.** ***The Modified Realizability Topos*****. February 1996. 17 pp.**

RS-96-2 **Allan Cheng and Mogens Nielsen.** ***Open Maps, Behavioural Equivalences, and Congruences*****. January 1996. 25 pp. A short version of this paper is to appear in the proceedings of CAAP '96.**

RS-96-1 **Gerth Stølting Brodal and Thore Husfeldt.** ***A Communication Complexity Proof that Symmetric Functions have Logarithmic Depth*****. January 1996. 3 pp.**