Some Results for Elementary Operations

Athanasios K. Tsakalidis

Department of Computer Engineering and Informatics,
University of Patras, 26501 Patras, Greece
tsak@cti.gr
www.tsakalidis.gr

Abstract. We present a number of results for elementary operations concerning the areas of data structures, computational geometry, graph algorithms and string algorithms. Especially, we focus on elementary operations like the dictionary operations, list manipulation, priority queues, temporal precedence, finger search, nearest common ancestors, negative cycle, 3-sided queries, rectangle enclosure, dominance searching, intersection queries, hidden line elimination and string manipulation.

1 Introduction

We consider a number of results derived in the last 25 years in the theory of efficient algorithms. For each of them, we present the main ideas, the main theorems, and we follow the path of their impact in the research community. We intentionally avoid technical descriptions and strictly mathematical definitions and we appoint them in a plausible manner. For each kind of the operations we offer a separate section with the respective references.

The model of computation we consider is the *Pointer Machine* (*PM-machine*), the *Pure Pointer Machine (PPM-machine)*, and the *Random-Access Machine* (*RAM-machine*). In a pointer machine, memory consists of a collection of *records*. Each record consists of a fixed number of *cells*. The cells have associated *types*, such as *pointer*, *integer*, *real*, and access to memory is only possible by "pointers". In other words, the memory is structured as directed graph with bounded out-degree. The edges of this graph can be changed during execution. Pointer machines correspond roughly to high-level programming languages without arrays. This PM-machine allows arithmetic capabilities on the content of data fields. In case that no arithmetic capabilities are allowed, we get the Pure Pointer Machine. The instruction set of the PPM-Machine consists of two types of instructions-pointer manipulation and control management. In contrast, the memory of a RAM consists of an array of cells. A cell is accessed through its address and hence address arithmetic is available. We assume in all the models that storage cells can hold arbitrary numbers and that the basic arithmetic and pointer operations take constant time. This is called the uniform cost assumption. All our machines are assumed to be deterministic.

Three types of complexity analysis are customary in the algorithms area: worst-case analysis, average-case analysis and amortized analysis. In a *worst-case analysis* worst-case bounds are derived for each single operation. This is

© Springer-Verlag Berlin Heidelberg 2009

the most frequent type of analysis. In an *average-case analysis*, we postulate a probability distribution on the operations of the abstract data type and computes the expected cost of the operations under this probability assumption. In an *amortized analysis*, we study the worst-case cost of a sequence of operations.

The externalization of a known data structure is the transformation of this data structure in such a way, that it can handle the same operations in an efficient way in case that the file handled cannot be stored entirely in the main memory. In this case, we have a main memory and a disk partitioned in blocks of size B. In this model of computation, the efficiency of the data structure used is measured by the number of *I/O-operations*. An *I/O-operation* (or simply *I/O*) is the operation of reading (or writing) a block from (or into) disk.

2 Dictionary Operations

Data structuring is the study of *concrete implementation* of frequently occurring *abstract data types*. An abstract data type is a set together with a collection of operations on the elements of the set. In the data type *dictionary*, the set is the powerset S of a universe U and the operations are *insertions* and *deletions* of the elements and the *test of membership* (*access*).

Note that the operations *insertion* and *deletion* are destructive, the old version of the set S is destroyed by the operations, and, in this case, the data structure used is called *ephemeral*. The nonedestructive version of the problem motivated the development *persistent data structures* [3]. We distinguish *partial* and $full$ persistence. A data structure is partially persistent, if all versions can be accessed but only the newest version can be modified and fully persistent if every version can be both accessed and modified.

A new data structure called *Interpolation Search Tree (IST)* is presented in [8], which supports interpolation search and insertions and deletions. The amortized insertion and deletion cost is $O(\log n)$. The expected search time in a random file is $O(\log \log n)$. This is not only true for the uniform distribution but for a wide class of probability distributions. Informally, a distribution defined over an interval I is *smooth*, if the probability density over any subinterval of I does not exceed a specific bound, however small this subinterval is (i.e., the distribution does not contain sharp peaks).

Given two functions f_1 and f_2, a density function $\mu = \mu[a,b](x)$ is (f_1, f_2)-*smooth* ([2], [8]), if there exists a constant β, such that for all c_1, c_2, c_3, $a \leq c_1 < c_2 < c_3 \leq b$, and all integers n, it holds that we have that

$$\int_{c_2 - \frac{c_3 - c_1}{f_1(n)}}^{c_2} \mu[c_1, c_3](x)dx \leq \frac{\beta \cdot f_2(n)}{n}$$

where $\mu[c_1, c_3](x) = 0$ for $x < c_1$ or $x > c_3$, and $\mu[c_1, c_3](x) = \mu(x)/p$ for $c_1 \leq x \leq c_3$, where $p = \int_{c_1}^{c_3} \mu(x)dx$.

Intuitively, function f_1 partitions an arbitrary subinterval $[c_1, c_3] \subseteq [a, b]$ into f_1 equal parts, each of length $\frac{c_3 - c_1}{f_1} = O(\frac{1}{f_1})$; that is, f_1 measures how fine is the partitioning of an arbitrary subinterval. Function f_2 guarantees that no part, of the f_1 possible, gets more probability mass than $\frac{\beta \cdot f_2}{n}$; that is, f_2 measures the sparseness of any subinterval $[c_2 - \frac{c_3 - c_1}{f_1}, c_2] \subseteq [c_1, c_3]$. The class of (f_1, f_2)-smooth distributions (for appropriate choices of f_1 and f_2) is a superset of both regular [11] and uniform classes of distributions, as well as of several non-uniform classes ([2],[5]). Actually, *any* probability distribution is $(f_1, \Theta(n))$-smooth, for a suitable choice of β.

Herewith it is worthwhile to note that the data structure used can reflect the distribution function in some places in order to guide the searching properly and to the fact that a random IST has subtrees of rootic size with high probability.

In [2], a technique is presented which extends the technique of [8] to a larger class of distributions and better bounds on searches and updates.

In [6], the *IS-Tree*, a dynamic data structure based on interpolation search is presented, which consumes worst case linear space and can be updated in $O(1)$ time worst case when the update position is given. Furthermore, the elements can be searched in $O(\log \log n)$ time expected with high probability, given that they are drawn from a $(n^\alpha, n^{1/2})$-smooth distribution, for constant $1/2 < \alpha < 1$. The worst case search time is $O(\log^2 n)$.

The externalization [10] of this data structure, called *ISB-tree*, was introduced in [4]. It supports search operations in $O(\log_B \log n)$ expected I/Os and update operations in $O(1)$ worst-case I/Os provided that the update position is given and B is the block size. The expected search bound holds with high probability, if the elements are drawn by a $(n/(\log \log n)^{1+\epsilon}, n^{1-\delta})$-smooth distribution, where $\epsilon > 0$ and $\delta = 1 - \frac{1}{B}$ are constants. The worst case search bound is $O(\log_B n)$ block transfers.

AVL-trees were introduced by Adel'son-Velskii and Landis in 1962 [1]. A binary search tree is AVL if the height of the subtrees at each node differ by at most one. In [7] we analyse the amortized behavior of AVL-trees under a sequence of insertions. We show that the total rebalancing cost for a sequence of n arbitrary insertions is at most $2.618n$. For random insertions, the bound is improved to $2.26n$. We show that the probability that t or more balance changes are required decreases exponentially with t. Mixed insertions and deletions do not have amortized constant complexity. In [9] is shown that the total rebalancing cost for a sequence of only arbitrary deletions is $1.618n$.

References

1. Adel'son-Velskii, G.M., Landis, E.M.: An Algorithm for the Organization of Information. Dokl. Akad. Nauk SSSR 146, 263–266 (1962) (in Russian); English translation in Soviet. Math. 3, 1259–1262 (1962)
2. Andersson, A., Mattson, C.: Dynamic Interpolation Search in $o(\log \log n)$ Time. In: Lingas, A., Carlsson, S., Karlsson, R. (eds.) ICALP 1993. LNCS, vol. 700, pp. 15–27. Springer, Heidelberg (1993)
3. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making Data Structures Persistent. J. Comput. System Sci. 28, 86–124 (1989)
4. Kaporis, A., Makris, C., Mavritsakis, G., Sioutas, S., Tsakalidis, A., Tsichlas, K., Zaroliagis, C.: ISB-tree: A new indexing scheme with efficient expected behaviour. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 318–327. Springer, Heidelberg (2005)

5. Kaporis, A., Makris, C., Sioutas, S., Tsakalidis, A., Tsichlas, K., Zaroliagis, C.: Improved Bounds for Finger Search on a RAM. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 325–336. Springer, Heidelberg (2003)
6. Kaporis, A., Makris, C., Sioutas, S., Tsakalidis, A., Tsichlas, K., Zaroliagis, C.: Dynamic Interpolation Search Revisited. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 382–394. Springer, Heidelberg (2006)
7. Mehlhorn, K., Tsakalidis, A.K.: An Amortized Analysis of Insertions into AVL-Trees. SIAM J. Comput. 15(1), 22–33 (1986)
8. Mehlhorn, K., Tsakalidis, A.: Dynamic Interpolation Search. Journal of the ACM 40(3), 621–634 (1993)
9. Tsakalidis, A.K.: Rebalancing Operations for Deletions in AVL-Trees. RAIRO Inform. Theor. 19(4), 323–329 (1985)
10. Vitter, J.: External Memory Algorithms and Data Structures: dealing with massive data. ACM Computing Surveys 33(2), 209–271 (2001)
11. Willard, D.E.: Searching unindexed and nonuniformly generated files in log log N time. SIAM J. Comput. 14(4), 1013–1029 (1985)

3 List Manipulation

In [10], we give a representation for linked lists which allows to efficiently *insert* and *delete* objects in the list and to quickly determine the order of two list elements. The basic data structure, called an *indexed BB[a]-tree* ([9], [2]), allows to do n insertions and deletions in $O(n \log n)$ steps and determine the order in constant time, assuming that the locations of the elements worked at are given. The improved algorithm does n insertions and deletions in $O(n)$ steps and determines the order in constant time. An application of this provides an algorithm which determines the *ancestor relationship* of two given nodes in a dynamic tree structure of bounded degree in time $O(1)$ and performs n arbitrary insertions and deletions at given positions in time $O(n)$ using *linear space.*

The amortized analysis of our algorithm is substantiallly based on the *weight−property* of BB[a]-trees, which is proved in [2]. The weight-property can be stated as follows: A node of weight w (i.e. w descendants) participates in only $O(n/w)$ structural changes of the tree when a sequence of n insertions and deletions is processed. This result improves the bounds given in [6], where only insertions were allowed. In [7], two algorithms are given. The first algorithm matches the $O(1)$ amortized time per operations of [10] and is simpler. The second algorithm permits all operations in $O(1)$ *worst-case* time. In [1], simpler solutions are given that match the bounds of [7].

The results of [10] are well used in the theory of persistent data structures [3–5, 8].

References

1. Bender, M.A., Cole, R., Demaine, E.D., Farach-Colton, M., Zito, J.: Two Simplified Algorithms for Maintaining Order in a List. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 219–223. Springer, Heidelberg (2002)
2. Blum, N., Mehlhorn, K.: On the average number of Rebalancing Operations in weight-balanced trees. Theor. Comput. Sci. 11, 303–320 (1980)

3. Buchsbaum, A., Tarjan, R.E.: Confluently Persistent Deques via Data Structural Bootstrapping. In: Proc. of the fourth annual ACM-SIAM Symposium on Discrete Algorithms, pp. 155–164 (1993)
4. Dietz, P.F.: Fully Persistent Arrays (Extended Abstract). In: Proc. of the Workshop on Algorithma and Data Structures, pp. 67–74 (1989)
5. Dietz, P.F.: A Space Efficient Variant of Path Copying for Partially Persistent Sorted Sets. J. Comput. System Sci. 53, 148–152 (1996)
6. Dietz, P.F.: Maintaining Order in a Linked List. In: Proc. 14th ACM STOC, pp. 122–127 (1982)
7. Dietz, P.F., Sleator, D.D.: Two Algorithms for Maintaining Order in a List. In: Proc. 19th ACM STOC, pp. 365–372 (1987)
8. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making Data structures Persistent. J. Comput. System Sci. 28, 82–124 (1989)
9. Nievergelt, I., Reingold, E.M.: Binary Search Tress of Bounded Balance. SIAM J. Comput. 2, 33–43 (1973)
10. Tsakalidis, A.K.: Maintaining Order in a generalized Linked List. Acta Informatica 21(1), 101–112 (1984)

4 Priority Queues

A min-max priority queue is a priority queue that structures the elements with respect to the maximum element as well the minimum element. In [2], we present a simple and efficient implementation of a min-max priority queue, reflected *min-max priority queues.* The main merits of our construction are threefold. First, the space utilization of the reflected min-max heaps is much better than the naive solution of putting two heaps back-to-back [3]. Second, the methods applied in this structure can be easily used to transform ordinary priority queues into min-max priority queues. Third, when considering only the setting of min-max priority queues, we support merging in constant worst-case time which is a clear improvement over the best worst-case bound achieved [1].

References

1. Hoyer, P.: A General Technique for Implementation of Efficient Priority Queues. In: Proc. of the 3rd Israel Symposium on Theory of Computing systems (ISTCS) (1995)
2. Makris, C., Tsakalidis, A., Tsichlas, K.: Reflected Min-Max Heaps. Inf. Proc. Letters 86, 209–214 (2003)
3. Williams, J.W.J.: Algorithm 232. Comm. ACM 7(6), 347–348 (1964)

5 Temporal Precedence

In this section we refer to the *Temporal Precedence Problem* on PPM-machine. This problem asks for the design of a data structure, maintaining a set of stored elements and supporting the following two operations: *insert* and *precedes*. The Operation $insert(a)$ introduces a new element a in the structure, while the operation $precedes(a, b)$ returns true iff element a was inserted before element b

temporally. In [4], a solution is provided to the problem with worst-case time complexity $O(\log\log n)$ per operation and $O(n\log\log n)$ space, where n is the number of elements inserted. It was demonstrated that the *precedes* operation has a lower bound of $\Omega(\log\log n)$ for the *Pure Pointer Machine* model of computation. In [1] two simple solutions are presented with *linear* space and worst-case *constant* insertion time. In addition, two algorithms are described that can handle the *precedes*(a,b) operation in $O(\log\log d)$ time, where d is the temporal distance between the elements a and b. In [2], solutions are given, which match the same time and space bounds in simpler manner. The *Temporal Precedence Problem* is related to a very concrete problem that arises in parallel implementation of logic programming languages [3]. More specifically, in the And-Parallelism problem the problem of correct binding and assignment of variables can be reduced to the *insert* and *precedes* operations of the Temporal Precedence problem.

References

1. Brodal, G.S., Makris, C., Sioutas, S., Tsakalidis, A., Tsichlas, K.: Optimal Solution for the Temporal Precedence Problem. Algorithmica 33(4), 494–510 (2002)
2. Pontelli, E., Ranjan, D.: A Simple Optimal Solution for the Temporal Precedence Problem on Pure Pointer Machines. Theory of Computing Systems 38, 115–130 (2005)
3. Pontelli, E., Ranjan, D., Gupta, G.: On the Complexity of Parallel Implementation of Logic Programs. In: Ramesh, S., Sivakumar, G. (eds.) FST TCS 1997. LNCS, vol. 1346, pp. 123–137. Springer, Heidelberg (1997)
4. Ranjan, D., Pontelli, E., Gupta, G., Longpre, L.: The Temporal Precedence problem. Algorithmica 28(3), 288–306 (2000)

6 Finger Search

Finger search trees represent ordered lists into which pointers can be maintained, called fingers, from which searches can start; the time for a search, insertion or deletion is $O(\log d)$, where d is the number of items between the search starting point and the accessed item. The $O(\log d)$ bound can either be achieved in the amortized sence [3, 10] or in the worst case [9, 11, 14–16]. Finger search leads to optimal algorithms for the basic operations *union*, *intersection*, *difference*, which can support the development of efficient algorithms for sorting presorted files ([8], [7]) and for locally adaptive data schemes [2].

In [4], a general solution is proposed for the persistence problem. The authors develop simple, systematic efficient techniques for making different linked data structures persistent. They show first that if an ephemeral structure has nodes of bounded in-degree, then the structure can be made partially persistent at an amortized space cost of $O(1)$ per update step and a constant-factor increase in the amortized cost of access and update operations. Second, they present a method which can make a linked stucture of bounded in-degree fully persistent at an amortized time and space cost of $O(1)$ per update step and a worst-case time of $O(1)$ per access step. Finally, the authors present a partial persistent implementation of balanced search tree with worst-case time per operation of $O(\log n)$ and an amortized space cost of $O(1)$ per insertion or deletion. Combining this result with a *delayed updating technique* of Tsakalidis [16], we obtain a fully persistent form of balanced search trees with the same time and space bounds as in the partially persistent case. The technique employed in [4] is strongly related to *fractional cascading.* This relation can be used to support a $forget$ operation which permits to explicitly delete versions and thus improves the space requirement [12].

The results of [15, 16] are well used in the theory of the persistent data structures [4, 5, 13] and in multidimensional search [6].

In [1], a new finger search tree is developed with worst-case constant update time in the PM-machine. This was a major problem in the field of Data Structures and was tantalizigly open for over 25 years [9], while many attempts by researchers were made to solve it. The result is a consequence of the innovative mechanism that guides the rebalancing operations, combined with incremental multiple splitting and fusion techniques over nodes.

References

1. Brodal, G.S., Lagogiannis, G., Makris, C., Tsakalidis, A.K., Tsihlas, K.: Optimal Finger Search Trees in the Pointer Machime. J. of Comput. System Sci. 67, 381–418 (2003)
2. Bentley, J.L., Sleator, D.D., Taran, R.E., Wei, V.K.: A locally adaptive Data Compression Scheme. Com. of ACM, 320–330 (1986)
3. Brown, M.R., Tarjan, R.E.: Design and Analysis of a Data Structure for representing Sorted Lists. SIAM J. Comput. 9, 594–614 (1980)
4. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making Data structures Persistent. J. Comput. System Sci. 28, 82–124 (1989)
5. Driscoll, J.R., Sleator, D.D., Tarjan, R.E.: Full Persistent Lists with Catenation. J. ACM 41(5), 943–959 (1994)
6. Duch, A., Martinez, C.: Improving the Performance of Multidimensional Search using Fingers. ACM Journal of Expermental Algorithmics 10, Art. no 2.4, 1–23 (2005)
7. Elmasry, A.: Adaptive Sorting with AVL Trees. IFIP TCS, 307–316 (2004)
8. Estivill-Castro, V., Wood, D.: A survey of Adaptive Sorting Algorithms. ACM Computing Surveys 24, 441–475 (1992)
9. Guibas, L.J., McCreight, E.M., Plass, M.F., Roberts, J.R.: A new represenation for Linear Lists. In: Proc. 9th Ann. ACM Symp. on Theory of Computing, pp. 49–60 (1978)
10. Huddleston, S., Mehlhorn, K.: A new Data Structure foe representing Sorted Lists. Acta Inform. 17, 157–184 (1982)
11. Kosaraju, S.R.: Localized Search in Sorted Lists. In: Proc. 13th Ann. ACM Symp. on Theory of Computing, pp. 62–69 (1981)
12. Mehlhorn, K., Näher, S., Uhrig, C.: Deleting Versions in Persistent Data Structures. Tech. Report, Univ. of Saarland, Saarbrücken (1989)
13. Sarnak, N., Tarjan, R.E.: Planar Point Location using Persistent Search Trees. Comm. ACM 29, 669–679 (1986)
14. Tarjan, R.E.: Private communication (1982)

15. Tsakalidis, A.K.: AVL-trees for Lícalized Search. Inform. and Control 67(1-3), 173–194 (1985)
16. Tsakalidis, A.K.: A Simple Implementation for Localized Search. In: Proc. WG 1985, Internat. Workshop on Graph-theoretical Concepts in Computer Science, pp. 363–374. Trauner-Verlag (1985); An optimal Implementation for localized seach. Tech. Rep. A84/06, Fachbereich Angewandte Mathematik and Informatik. Universität des Saarlandes, Saarbrücken, West Germany (1984)

7 Nearest Common Ancestors

Considering an arbitrary tree, the problem is to compute the nearest common ancestor of two given nodes x and y, denoted by $nca(x, y)$, which is defined as the lowest common node of the two paths from node x and node y to the root of the tree. In [1], a RAM-algorithm is presented running in $O(n)$ preprocessing time, $O(n)$ space and answering a query in $O(1)$ time. In [2], a PM-algorithm is given which requires $O(n \log \log n)$ preprocessing time, $O(n \log \log n)$ space and $O(\log \log n)$ *optimal* query time. In [1], the optimality of this query time is proved, and it is claimed that the algorithm of [2] can be modified to run on a PM-machine in linear time and space. Another *optimal* PM-algorithm with $O(n)$ preprocessing time, $O(n)$ space and $O(\log \log n)$ query time is described in [3].

Considering the dynamic case of one arbitrary tree, where the tree can be updated by insertions on the leaves or deletions of nodes, we get in [4] a PM-algorithm which needs $O(n)$ space, performs m arbitrary insertions on an initially empty tree in time $O(m)$, and allows to determine the nearest common ancestor of nodes x and y in time $O(\log(min\{depth(x), depth(y)\}) + a(k, k))$, where the second term is amortized over the k queries and $depth(x)$ is the distance from the node x to the root, and $a(k, k)$ is the inverse Ackerman function.

References

1. Harel, D., Tarjan, R.E.: Fast Algorithms for finding Nearest Common Ancestors. SIAM J. Comp. 13, 338–355 (1984)
2. van Leeuwen, J.: Finding Lowest Common Ancestors in less than Logarithmic Time. Unpublished report, Amherst, NY (1976)
3. van Leeuwen, J., Tsakalidis, A.K.: An optimal Pointer Machine Algorithm for Nearest Common Ancestors. Tech. Report, UU-CS-88-17, dept. of Computer Science, Univ. of Utrecht, Utrecht (1988)
4. Tsakalidis, A.K.: The Nearest Common Ancestor in a Dynamic Tree. Acta Informatica 25, 37–54 (1988)

8 Negative Cycle

The negative cycle problem is the problem of finding a negative length cycle in a directed graph with positive and negative edge-costs or proving that there are none. In [2] is shown that a negative cycle in a directed weighted graph with n

nodes and e edges can be computed in $O(n+e)$ time and $O(n+e)$ space. Assuming that the input of the algorithm is a weighted random digraph, it is proved in [1] that its average time complexity for dense graphs lies between $O(n \log n)$ and $O(min\{n^2/log^2 n, e\})$, the exact value depending on the probability with which an edge is present in the random graph, and for sparse random graph is $\Theta(n^2)$.

References

1. Spirakis, R., Tsakalidis, A.: A very fast practical algorithm for finding a negative cycle in a digraph. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 397–406. Springer, Heidelberg (1986)
2. Tsakalidis, A.: Finding a Negative Cycle in a Directed Graph. Techn. Report A85/05, Angewandte Mathematik und Informatik, FB-10, Univ. des Saarlandes, Saarbrücken (1985)

9 Three-Sided Queries

Let S be a set of n points in a two-dimensional space. A three-sided range query takes as arguments three coordinates x_1, x_2, y_1 and reports the set K of all points (x, y) of S with $x_1 \leq x \leq x_2$ and $y \leq y_1$. In [2] we consider 3-sided range queries on n points for a universe of $[N] \times \Re$, where N is the set of integers $\{0, ..., N-1\}$. We achieve $O(\log \log n + k)$ time, usings $O(N+n)$ space and preprocessing time, where k denotes the size of the output. This was later improved in [1] to $O(k)$ time, but with expected linear preprocessing time. The only dynamic sublogarithmic bounds on this problem can be found in [3], where it is attained $O\left(\frac{\log n}{\log \log n}\right)$ worst case or $O(\log n)$ randomized update time and $O\left(\frac{\log n}{\log \log n} + k\right)$ query time in linear space.

References

1. Alstrup, S., Brodal, G.S., Rauhe, T.: New Data Structures for Orthogonal Range Searching. In: FOCS 2000, pp. 198–207 (2000)
2. Fries, O., Mehlhorn, K., Näher, S., Tsakalidis, A.K.: A loglogn Data structure for Three-Sided Range Queries. Inf. Process. Lett. 25(4), 269–273 (1987)
3. Willard, D.E.: Examining Computational Geometry, Van Emde Boas Trees and Hashing from the perspective of the Fussion Tree. SIAM J. Comp. 29(3), 1030–1049 (2000)

10 Rectangle Enclosure

We consider two versions of the rectangle enclosure problem. Given a set S of rectangles in the plane, in the first version we report all the rectangles, which enclose a given query rectangle. In [1], a solution is given for the first version of the problem generalized in d-space, which needs $O(\log^{2d-2} \log \log n + k)$ query time, where k is the size of the answer in the case that the data structure used is static. In the dynamic case, the query time is $O(\log^{2d-1} n + k)$, and an update operation costs $O(\log^{2d-1} n)$. In both cases, the space used is $O(n \log^{2d-2} n)$. The query time in the static case is improved to $O(\log^{2d-2} n + k)$ in [3].

In the second version, we report all the pairs of the rectangles (R, R'), where $R, R' \in S$ and R' encloses R. In [6] a solution was given that needs $O(n)$ space and runs in $O(n \log^2 n + k)$ time. It has been an open problem for more than ten years how the $O(n \log^2 n)$ term of the reporting time could be reduced. In [4], an algorithm was given that solved this problem in $O(n + k)$ space and $O(n \log n \log\log n + k \log\log n)$ time. In [2] a subroutine of the previous algorithm is modified using persistence and periodic rebuilding of list structures and the space required is reduced to linear, while retaining the same time complexity. In [5], a simple solution is presented with the same time and space bounds.

References

1. Bistiolas, V., Sofotassios, D., Tsakalidis, A.: Computing Rectangle Enclosures. Comput. Geometry: Thery & Appl. 2(6), 301–308 (1993)
2. Bozanis, R., Kitsios, N., Makris, C., Tsakalidis, A.: The Space-Optimal Version of a known Rectangle Enclosure Reporting Algorithm. Inf. Process. Letters 61, 37–41 (1997)
3. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: New Results on Intersection Query Problems. Computer J. 40(1), 22–29 (1997)
4. Gupta, P., Janardan, R., Smid, M., Dasgupta, B.: The Rectangle Enclosure and Point-Dominance Problems. Intern. J. Comput. Geom. Applic. 7(5), 437–457 (1997)
5. Lagogiannis, G., Makris, C., Tsakalidis, A.: A new Algorithm for Rectangle Enclosure Reporting. Inf. Process. Letters 72, 177–182 (1999)
6. Lee, D.T., Preperata, F.P.: An improved Algorithm for the Rectangle Enclosure Problem. J. Algorithms 3, 218–224 (1982)

11 Dominance Searching

In [1], several data structures are presented for the 3-dominance searching problem: store a set S of n points in $\Re^3$ in a data structure, such that the points in S dominating a query point can be reported efficiently. We say that a point $p = (p_1, p_2, p_3)$ dominates a point $q = (q_1, q_2, q_3)$, if and only if $p_i \geq q_i$ for all $1 \leq i \leq 3$ and $p \neq q$. All our data structures use *linear* space. The first data structure works for the restricted case where the coordinates of the points in S and of the query points are integers in the range $[0, N-1]$. In this case, we achieve a query time of $O\left((\log\log N)^2 \log\log\log N + k \log\log N\right)$, where k is the number of answers to the query. The second and third data structure both work for the unrestricted case, where the coordinates are arbitrary reals. We achieve $O(\log n \log\log n + k)$ query time for pointer machines and $O(\log n + k)$ query time for random access machines. These results are improved in [2].

References

1. Makris, C., Tsakalidis, A.: Algorithms for three-dimensional Dominance Searching in linear space. Inf. Process. Letters 66, 277–283 (1998)
2. Afshani, P.: On dominance reporting in 3D. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 41–51. Springer, Heidelberg (2008)

12 Intersection Queries

Generalized intersection searching problems are a class of problems that constitute an extension of their standard counterparts. In such problems, we are given a set of collored objects and we want to report or count the distinct colors of the oblects intersected by a query object. Many solutions have appeared for both iso-oriented and non-oriented objects. In [1, 2], it is shown how to improve the bounds of several generalized inresection searching problems as well as how to obtain upper bounds for some problems like arbitrary line segment and generalized triangle stabbing, which were not treated before.

In [3], efficient solutions are given for the following problems: the Static d-dimensional rectangle enclosure problem, with $O(n \log^{2d-2} n)$ space and $O(\log^{2d-2} n + k)$ query time, the generalized c-oriented polygonal intersection searching, with $O(n \log^2 n)$ space and $O(\log n + k)$ query time, the generalized rectangular point enclosure problem, with $O(n \log n)$ space and $O(\log n + k)$ and the two-dimensional dominance searching problem with respect to a set of obstacle points, with $O(n \log n)$ space and $O(\log n + k)$ query time.

References

1. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: New upper Bounds gor Generalized Intersection Searching Problems. In: Fülöp, Z., Gecseg, F. (eds.) ICALP 1995. LNCS, vol. 944, pp. 464–474. Springer, Heidelberg (1995)
2. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: Red-Blue Inrersection Reporting for Objects of Non-Constant Size. Computer J. 39(6), 5410546 (1996)
3. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: New Results on Intersection Query Problems. Computer J. 40(1), 22–29 (1997)

13 Hidden Line Elimination

In a hidden line (or Surface) elimination problem we are given a set of objects in 3D space and a view-point and ask for the parts of the objects that are visible from the viewpoint. In a hidden line problem, the reported parts are line segments while in hidden surface problem, they are regions of surfaces. In [3], an algorithm is presented with optimal $O(n)$ space and worst case time $O(n \log n + k \log(n^2/k))$. In [4], a simple intersection sensitive algorithm is presented which solves the hidden line elimination problem in optimal $O(n)$ space and time complexity of $O((n + I)\log n)$, where I is the number of the intersections of the edges on the projection plane. An extension of this algorithm can solve the hidden surface removal problem in $O((n+I)\log n)$ time and $O(n+k)$ space, where k is the output size.

We consider the following problem as defined in [1]. Given a set of n isothetic rectangles in 3D space determine the subset of rectangles, that are not completely hidden. In [2], we present an optimal algorithm for this problem that runs in $O(n \log n)$ time and $O(n)$ space. Our results are an improvement over the one in [1] by a logarithmic factor in storage and are achieved by using a different approach. An analogous approach gives non-trivial solutions for other kinds of objects, too.

References

1. Grove, E.F., Murali, T.M., Vitter, J.S.: The object complexity model for hidden elimnination. Internat. J. Comput. Geom. Appl. 9, 207–217 (1999)
2. Kitsios, N., Makris, C., Sioutas, S., Tsakalidis, A., Tsaknakis, J., Vasiliadis, B.: An Optimal Algotithm for reporting Visible Rectangles. Inf. Process. Letters, 283–288 (2002)
3. Kitsios, N., Tsakalidis, A.: Space-Optimal Hidden Line Elimination for Rectangles. Inf. Process. Letters 60, 195–200 (1996)
4. Kitsios, N., Tsakalidis, A.: Space Reduction and an Extension for a Hidden Line Elimination Algorithm. Comp. Geom: Theory & Applic., 397–404 (1996)

14 String Manipulation

In [3], we consider several new versions of approximate string matching with gaps. The main characteristic of these new versions is the existence of gaps in the matching of a given pattern in a text. Algorithms are devised for each version, and their time and space complexities are stated. These specific versions of approximate string matching have various applications in computerized music analysis. In [5], we describe algorithms for computing typical regularities in strings that contain *don't care symbols*. We show also how our algorithms can be used to compute other string regularities, specifically the covers of both ordinary and circular strings.

Biological weighted sequences are used extensively in molecular biology as profiles for protein families, in the representation of binding sites and often for the representation of sequences produced by shotgun sequencing strategy. In [4], we introduce the *Weighted Suffix Tree*, an efficient data structure for computing string regularities in weighted sequences for molecular data. Repetitions, pattern matching and regularities in biological weighted sequences are also considered in [2]. In [6], we present algorithms for the *Motif Identification Problem* in Biological Weighted Sequences. The first algorithm extracts *repeated motifs*, the second algorithm extracts *common motifs* and the third alorithm extracts *maximal pairs.*

A multirepeat in a string is a substring that appears a predefined number of times. A multirepeat is maximal if it cannot be extended either to the right or to the left and produce a multirepeat. In [1], we present algorithms for two different versions of the problem of finding maximal multirepeats in a set of strings. In the first version we consider the case of arbitrary gaps and in the second version the case that the gap is bounded in a small range.

References

1. Bakalis, A., Iliopoulos, C.S., Makris, C., Sioutas, S., Theodoridis, E., Tsakalidis, A., Tsichlas, K.: Locating Maximal Multirepeats in Multiple Strings under various Constraints. Comput. J. 50(2), 178–185 (2006)
2. Christodoulakis, M., Iliopoulos, C., Mouchard, L., Tsakalidis, A., Tsichlas, K.: Computations of repetitions and Regularities of Biological Weighted Sequences. J. Comput. Biology 13(6), 1214–1231 (2006)
3. Crochemore, M., Iliopoulos, C., Makris, C., Rytter, W., Tsakalidis, A., Tsichlas, K.: Approximate String matching with Gaps. Nordic J. Comput. 9(1), 54–65 (2002)
4. Iliopoulos, C., Makris, C., Panagis, Y., Perdikuri, K., Theodoridis, E., Tsakalidis, A.: The Weighted Suffix Tree: An efficient Data Structure for handling Molecular Weighted Sequences and its Applications. Fundamenta Informaticae 71, 259–277 (2006)
5. Iliopoulos, C., Mohamed, M., Mouchard, L., Perdikuri, K., Smyth, W.F., Tsakalidis, A.: String Regularities with don't Cares. Nordic J. Comput. 10(1), 40–51 (2003)
6. Iliopoulos, C., Perdikuri, K., Theodoridis, E., Tsakalidis, A., Tsichlas, K.: Algorithms for extracting Motifs from Biological Weighted Sequences. J. Discrete Algorithms 5, 229–242 (2007)

Acknowledgement

We have presented some results derived since 1983 in collaboration with my students, my colleagues and my advisor Kurt Mehlhorn. I met Kurt in 1976. He was the youngest Professor in Germany and I was the oldest undergraduate student in our department. I was honored to live in his *scientific neighborhood* for 13 years as undergraduate student, as graduate student and as collaborator. Kurt was and remains the Professor, the Researcher and the Head of the Family. In 1979 I made Kurt's portait. I am grateful for everything that he taught me. I would like to say in public "Kurt, ich danke Dir, dass ich Dich in meinem Leben getroffen habe" (Kurt, I thank You that I have met You in my life).