Applications

Applications of snapshots include distributed databases, storing checkpoints or backups for error recovery, garbage collection, deadlock detection, debugging distributed programmes and obtaining a consistent view of the values reported by several sensors. Snapshots have been used as building blocks for distributed solutions to randomized consensus and approximate agreement. They are also helpful as a primitive for building other data structures. For example, consider implementing a counter that stores an integer and provides increment, decrement and read operations. Each process can store the number of increments it has performed minus the number of its decrements in its own component of a single-writer snapshot object, and the counter may be read by summing the values from a scan. See [10] for references on many of the applications mentioned here.

Open Problems

Some complexity lower bounds are known for implementations from registers [9], but there remain gaps between the best known algorithms and the best lower bounds. In particular, it is not known whether there is an efficient wait-free implementation of snapshots from small registers.

Experimental Results

Riany, Shavit and Touitou gave performance evaluation results for several implementations [16].

Cross References

- ► Implementing Shared Registers in Asynchronous Message-Passing Systems
- ► Linearizability
- ► Registers

Recommended Reading

See also Fich's survey paper on the complexity of implementing snapshots [11].

1. Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. J. Assoc. Comput. Mach. **40**, 873–890 (1993)
2. Anderson, J.H.: Composite registers. Distrib. Comput. **6**, 141–154 (1993)
3. Anderson, J.H.: Multi-writer composite registers. Distrib. Comput. **7**, 175–195 (1994)
4. Aspnes, J., Herlihy, M.: Wait-free data structures in the asynchronous PRAM model. In: Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures, Crete, July 1990. pp. 340–349. ACM, New York, 1990
5. Attiya, H., Fouren, A.: Adaptive and efficient algorithms for lattice agreement and renaming. SIAM J. Comput. **31**, 642–664 (2001)
6. Attiya, H., Fouren, A., Gafni, E.: An adaptive collect algorithm with applications. Distrib. Comput. **15**, 87–96 (2002)
7. Attiya, H., Herlihy, M., Rachman, O.: Atomic snapshots using lattice agreement. Distrib. Comput. **8**, 121–132 (1995)
8. Attiya, H., Rachman, O.: Atomic snapshots in $O(n \log n)$ operations. SIAM J. Comput. **27**, 319–340 (1998)
9. Ellen, F., Fatourou, P., Ruppert, E.: Time lower bounds for implementations of multi-writer snapshots. J. Assoc. Comput. Mach. **54**(6) article 30 (2007)
10. Fatourou, P., Kallimanis, N.D.: Single-scanner multi-writer snapshot implementations are fast! In: Proc. 25th ACM Symposium on Principles of Distrib. Comput. Colorado, July 2006 pp. 228–237. ACM, New York (2006)
11. Fich, F.E.: How hard is it to take a snapshot? In: SOFSEM 2005: Theory and Practice of Computer Science. Liptovský Ján, January 2005, LNCS, vol. 3381, pp. 28–37. Springer (2005)
12. Guerraoui, R., Ruppert, E.: Anonymous and fault-tolerant shared-memory computing. Distrib. Comput. **20**(3) 165–177 (2007)
13. Jayanti, P.: An optimal multi-writer snapshot algorithm. In: Proc. 37th ACM Symposium on Theory of Computing. Baltimore, May 2005, pp. 723–732. ACM, New York (2005)
14. Kirousis, L.M., Spirakis, P., Tsigas, P.: Simple atomic snapshots: A linear complexity solution with unbounded time-stamps. Inf. Process. Lett. **58**, 47–53 (1996)
15. Mostéfaoui, A., Rajsbaum, S., Raynal, M., Roy, M.: Condition-based consensus solvability: a hierarchy of conditions and efficient protocols. Distrib. Comput. **17**, 1–20 (2004)
16. Riany, Y., Shavit, N., Touitou, D.: Towards a practical snapshot algorithm. Theor. Comput. Sci. **269**, 163–201 (2001)

Sojourn Time

- ► Minimum Flow Time
- ► Shortest Elapsed Time First Scheduling

Sorting of Multi-Dimensional Keys

- ► String Sorting

Sorting Signed Permutations by Reversal (Reversal Distance)

2001; Bader, Moret, Yan

David A. Bader
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Keywords and Synonyms

Sorting by reversals; Inversion distance; Reversal distance

Problem Definition

This entry describes algorithms for finding the minimum number of steps needed to sort a signed permutation (also known as: inversion distance, reversal distance). This is a real-world problem and for example is used in computational biology.

Inversion distance is a difficult computational problem that has been studied intensively in recent years [1,4,6,7,8,9,10]. Finding the inversion distance between unsigned permutations is NP-hard [7], but with signed ones, it can be done in linear time [1].

Key Results

Bader et al. [1] present the first worst-case linear-time algorithm for computing the reversal distance that is simple and practical and runs faster than previous methods. Their key innovation is a new technique to compute connected components of the overlap graph using only a stack, which results in the simple linear-time algorithm for computing the inversion distance between two signed permutations. Bader et al. provide ample experimental evidence that their linear-time algorithm is efficient in practice as well as in theory: they coded it as well as the algorithm of Berman and Hannenhalli, using the best principles of algorithm engineering to ensure that both implementations would be as efficient as possible, and compared their running times on a large range of instances generated through simulated evolution.

Bafna and Pevzner introduced the cycle graph of a permutation [3], thereby providing the basic data structure for inversion distance computations. Hannenhalli and Pevzner then developed the basic theory for expressing the inversion distance in easily computable terms (number of breakpoints minus number of cycles plus number of hurdles plus a correction factor for a fortress [3,15]—hurdles and fortresses are easily detectable from a connected component analysis). They also gave the first polynomial-time algorithm for sorting signed permutations by reversals [9]; they also proposed a $O(n^4)$ implementation of their algorithm which runs in quadratic time when restricted to distance computation. Their algorithm requires the computation of the connected components of the overlap graph, which is the bottleneck for the distance computation. Berman and Hannenhalli later exploited some combinatorial properties of the cycle graph to give a $O(n\alpha(n))$ algorithm to compute the connected components, leading to a $O(n^2\alpha(n))$ implementation of the sorting algorithm [6], where α is the inverse Ackerman function. (The later Kaplan–Shamir–Tarjan (KST) algorithm [10] reduces the time needed to compute the shortest sequence of inversions, but uses the same algorithm for computing the length of that sequence.)

No algorithm that actually builds the overlap graph can run in linear time, since that graph can be of quadratic size. Thus, Bader's key innovation is to construct an *overlap forest* such that two vertices belong to the same tree in the forest exactly when they belong to the same connected component in the overlap graph. An overlap forest (the composition of its trees is unique, but their structure is arbitrary) has exactly one tree per connected component of the overlap graph and is thus of linear size. The linear-time step for computing the connected components scans the permutation twice. The first scan sets up a trivial forest in which each node is its own tree, labeled with the beginning of its cycle. The second scan carries out an iterative refinement of this first forest, by adding edges and so merging trees in the forest; unlike a Union-Find, however, this algorithm does not attempt to maintain the trees within certain shape parameters. This step is the key to Bader's linear-time algorithm for computing the reversal distance between signed permutations.

Applications

Some organisms have a single chromosome or contain single-chromosome organelles (such as mitochondria or chloroplasts), the evolution of which is largely independent of the evolution of the nuclear genome. Given a particular strand from a single chromosome, whether linear or circular, we can infer the ordering and directionality of the genes, thus representing each chromosome by an ordering of oriented genes. In many cases, the evolutionary process that operates on such single-chromosome organisms consists mostly of inversions of portions of the chromosome; this finding has led many biologists to reconstruct phylogenies based on gene orders, using as a measure of evolutionary distance between two genomes the inversion distance, i. e., the smallest number of inversions needed to transform one signed permutation into the other [11,12,14].

The linear-time algorithm is in wide-use (as it has been cited nearly 200 times within the first several years of its publication). Examples include the handling multichromosomal genome rearrangements [16], genome comparison [5], parsing RNA secondary structure [13], and phylogenetic study of the HIV-1 virus [2].

Open Problems

Efficient algorithms for computing minimum distances with weighted inversions, transpositions, and inverted transpositions, are open.

Experimental Results

Bader et al. give experimental results in [1].

URL to Code

An implementation of the linear-time algorithm is available as C code from www.cc.gatech.edu/~bader. Two other dominated implementations are available that are designed to compute the shortest sequence of inversions as well as its length; one, due to Hannenhalli that implements his first algorithm [9], which runs in quadratic time when computing distances, while the other, a Java applet written by Mantin (http://www.math.tau.ac.il/~rshamir/GR/) implements the KST algorithm [10], but uses an explicit representation of the overlap graph and thus also takes quadratic time. The implementation due to Hannenhalli is very slow and implements the original method of Hannenhalli and Pevzner and not the faster one of Berman and Hannenhalli. The KST applet is very slow as well since it explicitly constructs the overlap graph.

Cross References

For finding the actual sorting sequence, see the entry:

- ► Sorting Signed Permutations by Reversal (Reversal Sequence)

Recommended Reading

1. Bader, D.A., Moret, B.M.E., Yan, M.: A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. J. Comput. Biol. **8**(5), 483–491 (2001) An earlier version of this work appeared In: the Proc. 7th Int'l Workshop on Algorithms and Data Structures (WADS 2001)
2. Badimo, A., Bergheim, A., Hazelhurst, S., Papathanasopolous, M., Morris, L.: The stability of phylogenetic tree construction of the HIV-1 virus using genome-ordering data versus env gene data. In: Proc. ACM Ann. Research Conf. of the South African institute of computer scientists and information technologists on enablement through technology (SAICSIT 2003), vol. 47, pp. 231–240, Fourways, ACM, South Africa, September 2003
3. Bafna, V., Pevzner, P.A.: Genome rearrangements and sorting by reversals. In: Proc. 34th Ann. IEEE Symp. Foundations of Computer Science (FOCS93), pp. 148–157. IEEE Press (1993)
4. Bafna, V., Pevzner, P.A.: Genome rearrangements and sorting by reversals. SIAM J. Comput. **25**, 272–289 (1996)
5. Bergeron, A., Stoye, J.: On the similarity of sets of permutations and its applications to genome comparison. J. Comput. Biol. **13**(7), 1340–1354 (2006)
6. Berman, P., Hannenhalli, S.: Fast sorting by reversal. In: Hirschberg, D.S., Myers, E.W. (eds.) Proc. 7th Ann. Symp. Combinatorial Pattern Matching (CPM96). Lecture Notes in Computer Science, vol. 1075, pp. 168–185. Laguna Beach, CA, June 1996. Springer (1996)
7. Caprara, A.: Sorting by reversals is difficult. In: Proc. 1st Conf. Computational Molecular Biology (RECOMB97), pp. 75–83. ACM, Santa Fe, NM (1997)
8. Caprara, A.: Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J. Discret. Math. **12**(1), 91–110 (1999)
9. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In: Proc. 27th Ann. Symp. Theory of Computing (STOC95), pp. 178–189. ACM, Las Vegas, NV (1995)
10. Kaplan, H., Shamir, R., Tarjan, R.E.: A faster and simpler algorithm for sorting signed permutations by reversals. SIAM J. Comput. **29**(3), 880–892 (1999) First appeared In: Proc.8th Ann. Symp. Discrete Algorithms (SODA97), pp. 344–351. ACM Press, New Orleans, LA
11. Olmstead, R.G., Palmer, J.D.: Chloroplast DNA systematics: a review of methods and data analysis. Am. J. Bot. **81**, 1205–1224 (1994)
12. Palmer, J.D.: Chloroplast and mitochondrial genome evolution in land plants. In: Herrmann, R. (ed.) Cell Organelles, pp. 99–133. Springer, Vienna (1992)
13. Rastegari, B., Condon, A.: Linear time algorithm for parsing RNA secondary structure. In: Casadio, R., Myers, E.: (eds.) Proc. 5th Workshop Algs. in Bioinformatics (WABI'05). Lecture Notes in Computer Science, vol. 3692, pp. 341–352. Springer, Mallorca, Spain (2005)
14. Raubeson, L.A., Jansen, R.K.: Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. Science **255**, 1697–1699 (1992)
15. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS, Boston, MA (1997)
16. Tesler, G.: Efficient algorithms for multichromosomal genome rearrangements. J. Comput. Syst. Sci. **63**(5), 587–609 (2002)

Sorting Signed Permutations by Reversal (Reversal Sequence)

2004; Tannier, Sagot

Eric Tannier
NRIA Rhone-Alpes, University of Lyon, Lyon, France

Keywords and Synonyms

Sorting by inversions

Problem Definition

A *signed permutation* π of size n is a permutation over $\{-n, \ldots, -1, 1 \ldots n\}$, where $\pi_{-i} = -\pi_i$ for all i.

The *reversal* $\rho = \rho_{i,j}$ $(1 \leq i \leq j \leq n)$ is an operation that reverses the order and flips the signs of the elements