Improved Bounds for Range Mode and Range Median Queries

Holger Petersen

Univ. Stuttgart
FMI
Universitätsstr. 38
D-70569 Stuttgart
petersen@informatik.uni-stuttgart.de

Abstract. We investigate the following problem: Given a list of n items and a function defined over lists of these items, generate a bounded amount of auxiliary information such that range queries asking for the value of the function on sub-lists can be answered within a certain time bound.

For the function "mode" we improve the previously known time bound $O(n^\varepsilon \log n)$ to $O(n^\varepsilon)$ with space $O(n^{2-2\varepsilon})$, where $0 \leq \varepsilon < 1/2$. We improve the space bound $O(n^2 \log\log n/\log n)$ for an $O(1)$ time bounded solution to $O(n^2/\log n)$.

For the function "median" the space bound $O(n^2 \log\log n/\log n)$ is improved to $O(n^2 \log^{(k)} n/\log n)$ for an $O(1)$ time solution, where k is an arbitrary constant and $\log^{(k)}$ is the iterated logarithm.

1 Introduction

In this work we investigate the complexity of the following problem: Let $A = (a_1, \ldots, a_n)$ be a list of elements chosen from some set S and let f be a function defined for lists over S. After possibly computing auxiliary information about A in advance, a sequence of *range queries* asking for $f(a_p, \ldots, a_q)$ with varying p and q has to be answered. We simultaneously bound the size of the additional data stored by the preprocessing (the space) and the time for each query. The computational model is a unit-cost RAM with $\Theta(\log n)$ word length.

The obvious solution of storing answers for all sub-lists uses $\Theta(n^2)$ space and has constant time complexity. Our goal is to improve the space-time product to sub-quadratic bounds.

The problem of range queries is particularly easy if S is a group with a constant time computable operation and $f(a_p, \ldots, a_q) = a_p \cdots a_q$ is the product of all elements in the range. In this case all partial products $m_i = a_1 \cdots a_i$ and their inverses can be precomputed and stored in $O(n)$ space. The computation of $f(a_p, \ldots, a_q) = m_{p-1}^{-1} m_q$ is possible in constant time. Since the list A itself requires space n these bounds are asymptotically optimal.

Among the operators of interest that do not admit the computation of inverses are min and max where S is an ordered set. Nevertheless an optimal $O(n)$ space

V. Geffert et al. (Eds.): SOFSEM 2008, LNCS 4910, pp. 418–423, 2008.

and $O(1)$ query time solution has been given by Gabow, Bentley, and Tarjan [GBT84], which is based on the solution of the nearest common ancestor problem by Harel and Tarjan [HT84]. A simplified version is due to Bender and Farach-Colton [BFC00, BFCP$^+$05].

A *median* is an element of a sorted multi-set dividing the higher half from the lower half. In comparison to the average (which usually is not a member of the multi-set), the median often more accurately captures the concept of a typical element. As an example of an area where median computations arise we mention the definition of poverty: In the EU a person with an income below 60 % of the median in a country is considered to be at risk of poverty [SCFF05, p. 125]. Another (not necessarily unique) parameter of a multi-set is the *mode*, a value with maximum frequency.

The problem of computing the median and mode for ranges of lists has been investigated by Krizanc, Morin and Smid [KMS05]. For mode we improve their time bound $O(n^\varepsilon \log n)$ with space $O(n^{2-2\varepsilon})$ by a logarithmic factor and for the $O(1)$ time solution we improve the space bound $O(n^2 \log\log n/\log n)$ to $O(n^2/\log n)$.

For the function median we improve the previous space bound $O(n^2 \log\log n/\log n)$ to $O(n^2 \log^{(k)} n/\log n)$ for the $O(1)$ time solution, where $k \geq 1$ is an arbitrary constant and $\log^{(k)}$ is the iterated logarithm.

Algorithms for approximate range mode and median queries are due to Bose, Kranakis, Morin and Tang [BKMT05]. For computing an approximate range mode they also present an $\Omega(n \log n)$ lower bound on the time necessary for preprocessing and answering the query, where the model is that of algebraic decision trees. Notice that the algorithms of [KMS05] and the present work are designed for random access machines.

The new bounds obtained and some previous results are summarized in the following tables (for restrictions on ε see the references):

Range Mode			
space	time	space × time	ref.
$O(n^{2-2\varepsilon})$	$O(n^\varepsilon)$	$O(n^{2-\varepsilon})$	Theorem 1
$O(n^2/\log n)$	$O(1)$	$O(n^2/\log n)$	Theorem 2

Range Median			
space	time	space × time	ref.
$O(n)$	$O(n^\varepsilon)$	$O(n^{1+\varepsilon})$	[KMS05]
$O(n \log^2 n/\log\log n)$	$O(\log n)$	$O(n \log^3 n/\log\log n)$	[KMS05]
$O(n^2 \log^{(k)} n/\log n)$	$O(1)$	$O(n^2 \log^{(k)} n/\log n)$	Theorem 3

2 Results

We need the following observation from [KMS05]:

Lemma 1. *Let A, B, C be multi-sets. If a mode of $A \cup B \cup C$ is not a member of $A \cup C$, then it is a mode of B.*

In the following proofs we often use non-integer values when an integer is required. We assume that these values are appropriately rounded.

Theorem 1. *For every $0 \leq \varepsilon < 1/2$ there is a data structure of size $O(n^{2-2\varepsilon})$ that can answer range mode queries in $O(n^{\varepsilon})$ time.*

Proof. We first notice, that each element can be represented by an integer from $\{1, \ldots, n\}$, since a translation table can be stored within the space bound. We will work only with these numbers and identify them with the original elements.

Let $r = \lceil \varepsilon/(1-2\varepsilon) \rceil$. Notice that r is a constant with $\varepsilon \leq r/(2r+1)$. We divide the list into nested intervals, where each level ℓ interval has length $n^{(r+\ell)\cdot\varepsilon/r}$ for $0 \leq \ell \leq r$. A level ℓ interval thus contains $n^{\varepsilon/r}$ level $\ell - 1$ intervals. For each level r interval i we pre-compute a table f_i of size n that stores the frequency of each element in the prefix of the list up to and including interval i. Notice that the frequency of element e in the range from interval j to k is $f_k[e] - f_{j-1}[e]$ (f_0 is always 0).

For each level ℓ interval i with $\ell > 0$ we choose a constant time computable hash function h_i that is perfect for the elements occurring in the interval and mapping into a set of size $O(n^{(r+\ell)\cdot\varepsilon/r})$ [FKS84]. We store each element e in a table s at position $h_i(e)$. All entries of s that are not mapped to by h_i are filled with an arbitrary element from the interval. In addition we form a table $t(g, j)$ of size $O(n^{(r+\ell+1)\cdot\varepsilon/r})$ that stores for every hash value g and every level $\ell - 1$ interval j the frequency of the hashed element in the prefix of the list up to and including that level $\ell - 1$ interval.

In addition a table of size $O(n^{2-2\varepsilon})$ is set up that stores for each sorted pair of (possibly identical) level 0 intervals the mode of the list between them (including both intervals) and the frequency of each of these modes.

Finally a table c with n entries will be initialized with zeroes. While processing a query this table will record the frequency of certain elements from the selected range. After a query has been processed, the table will be reset to its initial state. An easy way to do this without increasing the time complexity by more than a constant factor is to record all modified entries in a linked list. After processing the query all recorded entries are set to zero.

Suppose a range mode query for the list from position p to position q has to be answered. If p and q are in the same level 0 interval, then in one scan for every element e in the range $c[e]$ is incremented. In a second scan of the interval the element with the maximum count is selected.

In the following we assume that p and q are in different level 0 intervals. Let m be the mode of the pair of (possibly identical) level 0 intervals properly between p and q, or an arbitrary element of the range if p and q are in neighboring level 0 intervals. By Lemma 1 the mode of the sub-list from p to q is m or one of the $O(n^{\varepsilon})$ elements in the prefix resp. suffix of the level 0 intervals containing p resp. q.

By the remarks in the introduction it is sufficient to compute the frequency from the start of the list up to position p (resp. $q - 1$) for each element e which could be the mode in constant time. The frequency can be computed as the difference of the two values. We describe the frequency computation for position p. Let ℓ be the minimal level of intervals containing p and a position at which e

occurs, or $\ell = r+1$ if no such interval exists. This ℓ can be computed in constant time by performing the test $e = s(h_i(e))$ for each of the at most $r+1$ intervals involved. If $\ell = r+1$ the frequency up to the preceding level r interval i (which is the result) can be looked up in f_i. If $r \geq \ell > 0$, then the value $t(h_i(e), j)$ for the level ℓ interval i and the level $\ell - 1$ interval j containing p is the result. If finally $\ell = 0$, then an approximate frequency up to the preceding level 0 interval j can be fetched from $t(h_i(e), j)$, where i is the level 1 interval containing j. All these approximate frequencies are stored in table c. Also the frequency of m is stored in c. Now in a second pass for each occurrence of an element e in the prefix of the level 0 interval containing p entry $c[e]$ is incremented. This can be done in $O(n^{\varepsilon})$ steps, thus in constant time per element. In this way the frequency of each element in a prefix is computed in time $O(1)$. From the non-zero entries of c the maximum frequency is selected and returned as the answer to the query.

Each level r interval is of length $n^{2\varepsilon}$, therefore there are $n^{1-2\varepsilon}$ of them. Hence the tables f_i with n entries require space $O(n^{2-2\varepsilon})$ in total. The space used by each level is dominated by the tables t which are of size $O(n^{1+\varepsilon/r})$. We have $1 + \varepsilon/r \leq 1 + 1/(2r+1) = 2 - (2r/(2r+1)) \leq 2 - 2\varepsilon$ and therefore $O(n^{1+\varepsilon/r}) = O(n^{2-2\varepsilon})$. The table of modes of pairs of level 0 intervals can be stored in space $O((n^{1-\varepsilon})^2) = O(n^{2-2\varepsilon})$. □

Theorem 2. *There is a data structure of size $O(n^2/\log n)$ that can answer range mode queries in $O(1)$ time.*

Proof. In order to simplify computations we assume that the n elements are stored in $A[0, \ldots, n-1]$. For every interval of the list of the form $A[i, j \cdot \log n]$ for $0 \leq i \leq n-1$ and $i \leq j \log n \leq n-1$ its mode is stored in the array m at position $m[i, j]$. For each of the $\log n - 1$ positions $k = j \cdot \log n + 1, j \cdot \log n + 2, \ldots, (j+1) \cdot \log n - 1$ the mode of $A[i, k]$ is the mode of $A[i, k-1]$ or element $A[k]$ by Lemma 1 (notice that one of the sets is empty). This information can be encoded into a single bit and all $\log n - 1$ bits thus determined can be encoded into a number $r[i, j]$ stored in an array r. A systematic way to do this is to store the bit for $k = j \cdot \log n + 1$ as the least significant bit and proceed to more significant bit positions in $r[i, j]$.

Arrays m and r clearly require space $O(n^2/\log n)$. The information stored can be accessed with the help of an auxiliary array b. Entry $b[v, \ell]$ selects for value v the least significant ℓ bits and returns the position of the most significant 1 among those bits, or 0 if all selected bits are 0. Array b is of size $O(n \log n)$. If the mode of $A[i, p]$ has to be determined, then first $j = p \operatorname{div} \log n$ and $\ell = p - j \cdot \log n$ are computed. If $b[r[i, j], \ell] = 0$ then $m[i, j]$ is returned. Otherwise the result is element $A[j \cdot \log n + b[r[i, j], \ell]]$.

Since the number of array accesses is fixed the running time is $O(1)$. □

We define the iterated logarithm function by letting $\log^{(1)} n = \log n$ and $\log^{(k+1)} n = \log \log^{(k)} n$.

Theorem 3. *There is a data structure of size $O(n^2 \log^{(k)} n / \log n)$ that can answer range median queries in $O(1)$ time for every integer $k \geq 1$.*

Proof. For $k = 1$ the statement holds, since the trivial solution of storing answers to all possible queries suffices. In the following we assume that $k \geq 2$.

We construct k levels of data structures. The level 1 structure contains for every pair of blocks of length $b_1 = \log n$ of the list the at most $4b_1$ elements that can possibly be a median for pairs of indices within the blocks (each element in the blocks can be a median, and the $2b_1$ elements in the center of the sorted sequence of blocks properly in between, if any). The level 1 structure requires space $O(n^2/\log n)$.

Suppose level $\ell-1$ has been constructed for $2 \leq \ell \leq k-1$. Then level ℓ contains for every pair of blocks B_i^ℓ and B_j^ℓ of length $b_\ell = \log^{(\ell)} n$ pointers to the at most $4b_\ell$ elements that can be a median for every pair of indices within the blocks. These pointers are relative to the start of the data of the pair of level $\ell-1$ blocks $B_{i'}^{\ell-1}$ and $B_{j'}^{\ell-1}$ containing B_i^ℓ and B_j^ℓ. Therefore $O(\log^{(\ell)} n)$ bits per pointer are sufficient, leading again to a space complexity of $O(n^2/\log n)$.

The construction for level k stores a pointer to the level $k-1$ structure of length $O(\log^{(k)} n)$ for *every* pair of indices, thus the level k structure requires space $O(n^2 \log^{(k)} n/\log n)$.

In order to meet the space bounds, the pointers have to be packed as fields into numbers of $O(\log n)$ bits each. Accessing a field requires shift and bit-mask operations which can be implemented efficiently with the help of tables as explained below (if these operations are assumed not to be included into the instruction set of a RAM).

A query is answered by following pointers starting from level k. If the median of $(a_p, \ldots, a_q)$ has to be computed, the algorithm first fetches $\log^{(k)} n + 2$ bits from a three-dimensional array m_k with $n \times (n \operatorname{div} \log n) \times (\log^{(k)} n + 2)$ entries. It reads $m_k[p, (q \operatorname{div} \log n), ((q \bmod \log n) \cdot (\log^{(k)} n + 2)) \operatorname{div} \log n]$ and extracts $\log^{(k)} n + 2$ bits starting at position $((q \bmod \log n) \cdot (\log^{(k)} n + 2)) \bmod \log n$. Here we start counting at the least significant position and allow for at most $\log n + \log^{(k)} n + 1$ bits per stored word in order to have only one access to m_k. The bits are shifted to the least significant positions, which can be accomplished with the help of a table of size $O(n \log^{(k-1)} n \log n)$, since the number of bits stored is $O(\log n)$. Then the lower $\log^{(k)} n + 2$ bits are selected with the help of a table of size $O(n \log^{(k-1)} n)$. Now the algorithm has computed a pointer i with $0 \leq i < 4 \log^{(k-1)} n$. Suppose a pointer i into the data structure for level $\ell < k$ has been determined. If $\ell \geq 2$ then the algorithm computes $p' = p \operatorname{div} \log^{(\ell)} n$, $q' = q \operatorname{div} \log^{(\ell)} n$, reads $m_\ell[p', (q' \operatorname{div} \log n), ((q' \bmod \log n) \cdot (\log^{(\ell)} n + 2)^2 + i \cdot (\log^{(\ell)} n + 2)) \operatorname{div} \log n]$ and extracts $\log^{(\ell)} n + 2$ bits starting at position $((q' \bmod \log n) \cdot (\log^{(\ell)} n + 2)^2 + i \cdot (\log^{(\ell)} n + 2)) \bmod \log n$. Here the tables have sizes at most $O(n \log^2 n)$ and $O(n \log n)$, which is within the claimed space bound. For level $\ell = 1$ the access is to $m_1[(p \operatorname{div} \log n), (q \operatorname{div} \log n), i]$, where the index of an element is stored directly.

Since the number of levels is fixed and every level requires a constant number of operations or memory accesses, we get time complexity $O(1)$. □

Remark 1. By the method from the proof of Theorem 3 we can also obtain the space bound $O(n^2 \log^* n / \log n)$ with time bound $O(\log^* n)$.

3 Summary

We have improved previous bounds for range mode and range median queries on lists. No non-trivial lower bounds for these problems are known on the RAM. Thus it is not clear how far these solutions are from being optimal and algorithmic improvements as well as lower bounds are problems left open.

Acknowledgments. The author is grateful to Benjamin Hoffmann, Jürn Laun, and the referees for useful comments.

References

[BFC00] Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)

[BFCP+05] Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest common ancestors in trees and directed acyclic graphs. J. Algorithms 57, 75–94 (2005)

[BKMT05] Bose, P., Kranakis, E., Morin, P., Tang, Y.: Approximate range mode and range median queries. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 377–388. Springer, Heidelberg (2005)

[FKS84] Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. Journal of the ACM 31, 538–544 (1984)

[GBT84] Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: STOC 1984. Proceedings of the sixteenth annual ACM Symposium on Theory of Computing, pp. 135–143. ACM Press, New York (1984)

[HT84] Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM Journal on Computing 13, 338–355 (1984)

[KMS05] Krizanc, D., Morin, P., Smid, M.: Range mode and range median queries on lists and trees. Nordic Journal of Computing 12, 1–17 (2005)

[SCFF05] Schäfer, G., Cervellin, S., Feith, M., Fritz, M. (eds.): Europe in figures — Eurostat yearbook 2005. Office for Official Publications of the European Communities, Luxembourg (2005)