Encoding Data Structures

Rajeev Raman[1]

University of Leicester, UK.

Abstract. In recent years, there has been an explosion of interest in *succinct* data structures, which store the given data in compact or compressed formats and answer queries on the data rapidly while it is still in its compressed format. Our focus in this talk is to introduce *encoding* data structures. Encoding data structures consider the data together with the queries and aim to store only as much information about the data as is needed to store the queries. Once this is done, the original data can be deleted. In many cases, one can obtain space-efficient encoding data structures even when the original data is incompressible.

1 Introduction

The need for performing complex processing on ever-larger volumes of data has led to the re-evaluation of the space usage of data structures. Whereas in classical data structures, a linear space usage, namely using $O(n)$ words of space, is considered to be optimal, this is often too much for very large data. For example, a suffix tree on a string of n characters from a fixed alphabet requires $\Theta(n)$ words. A usual assumption is that a computer word must be of length $\Omega(\log n)$ bits, so that one can work with numbers such as the input size n, and be able to address enough memory to hold the input. Thus, a suffix tree requires $\Theta(n \log n)$ bits of memory, while the input requires only $O(n)$ bits. This asymptotic blow-up also manifests itself in practice: even a even a highly optimized implementation of a suffix tree requires $20n$ bytes in the worst case [15] to index a string of n bytes. This level of internal memory usage is unacceptable if we wish to index gigabytes or terabytes of string data.

In response to this issue, there has been a great deal of research into *succinct* and *compressed* data structures [2] building upon the early work of Jacobson [12] and contemporaries. In succinct data structures, we view the given instance x of the data on which we wish to build a data structure as coming from a set S of objects, and the aim is to represent x using space as close to the information-theoretic bound of $\lceil \log_2 |S| \rceil$ bits as possible. In compressed data structures, we postulate a probability distribution on S and aim to represent x using as close to the Shannon bound of $\lceil \log_2 1/\Pr(x) \rceil$ bits as possible.

However, there are cases where the succinct approach does not offer any asymptotic improvements. Consider the well-known *range maximum query* problem, which is, given a static array $A[1..n]$, to pre-process A to answer queries:

RMQ(l, r)**:** return $\max_{l \le i \le r} A[i]$.

Assume, for simplicity, that A contains a permutation of $\{1, \ldots, n\}$. Observe that since $\mathsf{RMQ}(i,i)$ queries can be used to reconstruct A, any data structure for answering RMQ on A must contain all the information contained in A, and hence use $\Omega(n \log n)$ bits.

In order to get around this, we modify the RMQ slightly:

$\mathsf{RMQ}(l,r)$: return $\arg\max_{l \le i \le r} A[i]$.

In other words, we only seek the index in the range $\{l, \ldots, r\}$ where the maximum value among $A[l], \ldots, A[r]$ lies. In many applications of the RMQ problem, knowing the index where the maximum value lies is sufficient. With this modified version, it is no longer possible to reconstruct A is by performing RMQs. For example, if $n = 3$ then the arrays $A = (1,3,2)$ and $A' = (2,3,1)$ will give exactly the same answer to any RMQ operation, and the space lower bound of $\Omega(n \log n)$ bits no longer applies.

2 Encoding Data Structures

We now define some terminology regarding encoding data structures.

Effective entropy. Given a set of objects S, and a set of queries Q, consider the equivalence class $\mathcal{C}$ on S induced by Q, where two objects from S are equivalent if they provide the same answer to all queries in Q. We define the quantity $\lceil \log_2 |\mathcal{C}| \rceil$ bits to be the *effective entropy* of S with respect to Q. In case it is possible to reconstruct the given object $x \in S$ by means of the queries Q, we have $|\mathcal{C}| = |S|$ and there is no advantage to be gained. However, if $|\mathcal{C}| \ll |S|$, as is sometimes the case, then there can be substantial savings. For example, it is known [7] that for the RMQ problem, $|\mathcal{C}| \le 4^n \ll n! = |S|$, so the empirical entropy of the class of arrays A containing permutations, with respect to RMQ, is only about $2n$ bits, as opposed to the entropy of the arrays A which is $\sim n \log n$ bits. In what follows, we will abbreviate "the effective entropy of S with respect to Q" as "the effective entropy of Q."

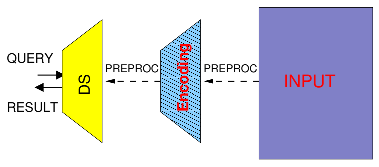

Fig. 1. Schematic illustation of the preprocessing steps in an encoding data structure.

Encoding and Encoding Data Structure. Given an object $x \in S$, instead of storing x directly, we can store a representation of the equivalence class $y \in \mathcal{C}$ that x belongs to. By the definition of $\mathcal{C}$, queries can be answered correctly using y, rather than x; we call y an *encoding* of x. Note that since all queries can be answered using y, there is no need to store x, and it can be deleted, as depicted diagrammatically in Fig 1. The encoding y can then be converted into an *encoding data structure* that not only answers queries correctly but rapidly. Ideally the space usage of this data structure should be $(1 + o(1)) \log_2 |\mathcal{C}|$ bits.

Expected effective entropy. One can also define the *expected* empirical entropy of a class of objects S with respect to Q: postulate a distribution on S, which induces a distribution on $\mathcal{C}$. The effective empirical entropy is then defined as $\sum_{y \in \mathcal{C}} \Pr(y) \cdot \log_2 1/\Pr(y)$. An encoding that aims to achieve the expected effective entropy then tries to represent an encoding y using as close to $\log_2 1/\Pr(y)$ bits as possible.

Minimal Encodings. As stated above, it is assumed that there is an effective characterization of the equivalence class $\mathcal{C}$ induced on S by Q, and that given an input $x \in S$, it is possible to constructively and fairly quickly find a $y \in \mathcal{C}$ to represent x. We will henceforth call such encodings *minimal encodings.* A property of minimal encodings is that the encoding only contains the information about x that could be inferred via queries in Q, and an encoding data structure that is built upon y has the same property, provided the pre-processing does not refer to the input x. Characterizing the $\mathcal{C}$ that leads to a minimal encoding can be a non-trivial enumeration task, leading to objects studied by combinatorial mathematicians such as Baxter permutations [9] and Schröder trees [6].

Minimality is, however, a stringent requirement, often an element from a set E that is larger than $\mathcal{C}$ is used to represent the given input x. Provided that $\log_2 |E| = o(\log_2(|S|))$ we will still consider this to be an encoding.

3 Results on Encoding Data Structures

We now discuss some recent results on encoding data structures.

3.1 Range Statistics on 1D-Arrays

Range Maximum Queries. For the RMQ problem defined above, the non-encoding solution [8] is obtained via the Cartesian tree [17], a binary tree on n nodes. Fischer and Heun [7] observed that the Cartesian tree gives a 1-1 correspondence between binary trees on n nodes and equivalence classes for the RMQ problem, thus giving a minimal encoding for RMQ. Since there are $\frac{1}{2n+1}\binom{2n}{n}$ binary trees on $n + 1$ nodes, the effective entropy works out to be $2n - O(\log n)$ bits, and Fischer and Heun gave a $2n + o(n)$-bit data structure that answers RMQ in $O(1)$ time[1]. Davoodi et al. [4,3] gave alternative $2n + o(n)$-bit data structures.

[1] This result, as do all results in this abstract, use the word RAM model with word size $\Theta(\log n)$ bits.

The expected effective entropy (for uniform random permutations in A) of RMQ is approximately $1.736n$ bits [10]. Davoodi et al. [3] gave an encoding data structure that answers RMQ in $O(1)$ time, using $1.919n + o(n)$ bits on average.

Range Top-k and Range Selection We are given an array $A[1..n]$ that contains a permutation of $\{1, \dots, n\}$ and an integer k specified at pre-processing time, and need to answer the query:

top-k-pos(l, r)**:** return positions of the k largest values in $A[l..r]$.

This is a generalization of the RMQ problem, which is the case $k = 1$. Grossi et al. [11] showed that any encoding must have size $\Omega(n \log k)$ bits and gave an encoding data structure that uses $O(n \log k)$ bits and answers queries in $O(k)$ time. For the case $k = 2$, Davoodi et al. [3] gave a minimal encoding, but were unable to obtain from this a closed-form expression for the empirical entropy of the top-2 problem. They showed that empirical entropy is at least $2.656n$ bits by a computational case analysis, and gave a data structure that took at most $3.272n + o(n)$ bits and answered top-2 queries in $O(1)$ time.

Recently, Gawrychowski and Nicholson [9] gave a different encoding for the top-k problem. Using their encoding, they were able to obtain tight upper and lower bounds of $\frac{1}{k+1}nH(\frac{1}{k+1})$ and $(1 - o(1))\frac{1}{k+1}nH(\frac{1}{k+1})$ bits on the effective entropy for all values of k. Here $H(x) = x \log_2(1/x) + (1-x) \log_2 1/(1-x)$ for any $0 \leq x \leq 1$. For $k = 2$, this gives the encoding complexity of the top-2 problem to be approximately $2.755n$ bits. However, it is not clear that their encoding is minimal, and they do not give an encoding data structure that answers top-k-pos queries rapidly.

The range selection problem is as follows. We are again given an array $A[1..n]$ that contains a permutation of $\{1, \dots, n\}$ and an integer k specified at pre-processing time, and need to answer the query

select(i, l, r)**:** return the position of the i-th largest value in $A[l..r]$, for any $i \leq k$.

Clearly, since by repeated select operations, we can obtain the top-k in a given range, the effective entropy of range selection is no lower than that of the top-k problem. Navarro et al. [16] gave an encoding that takes $O(n \log k)$ bits of space, which is asymptotically optimal, and answers queries in optimal $O(1 + \log i / \log \log n)$ time.

Range Majority. We are given an array $A[1..n]$ that contains (wlog) values from $\{1, \dots, n\}$, and a number $0 < \tau \leq 1/2$, specified at pre-processing time. We wish to answer the following query:

majority$_\tau(l, r)$**:** If some value occurs at least $\tau(r - l + 1)$ times in $A[l..r]$, return any index $i \in \{l, \dots, r\}$ such that $A[i]$ contains this value. If no value occurs with this frequency, return null.

Navarro and Thankachan show that the encoding complexity is $\Omega(\tau \log(1/\tau)n)$ bits and give a data structure that takes $O((n/\tau) \log^* n)$ bits of space and answers queries in $O(\log n)$ time.

Range Maximum-segment Sum. We are given an array $A[1..n]$ that contains positive and negative numbers. We wish to answer the following query:

RMSS(l, r)**:** Return l', r', $l \leq l' \leq r' \leq r$ such that $\sum_{i=l'}^{r'} A[i]$ is maximised.

Nicholson and Gawrychowski [9] showed that an encoding using $\Theta(n)$ bits can be used to answer such queries in $O(1)$ time.

3.2 2D Range Maximum Queries

The input to this problem is a two dimensional $m \times n$ array A, containing a permutation of $\{1, \dots, N\}$ where $N = m \cdot n$. Assume that $m \leq n$. We wish to answer the following query:

RMQ(q)**:** where $q = [i_1 \cdots i_2] \times [j_1 \cdots j_2]$ returns the position of the maximum element in the query range, i.e., $\mathsf{RMQ}(q) = \mathrm{argmax}_{(i,j) \in q} A[i, j]$.

Brodal et al. [1], following on the work of Demaine et al. [5] showed that the encoding complexity must be $\Omega(N \log m)$ bits. Brodal et al. [1] later gave an encoding of size $O(N \log m)$ bits, but this encoding does not yield a fast data structure. Golin et al. [10] showed that the expected effective entropy (assuming a random permutation in A) is $O(N)$ bits and gave a constant-time data structure with this space usage. Finally, for the case $m = 2$, Golin et al. gave a minimal encoding using $5n - O(\log n)$ bits. They also gave a data structure that takes $(5 + \epsilon)n + o(n)$ bits, for any $0 < \epsilon \leq 1$ and answers queries in $O(1/\epsilon)$ time.

3.3 Nearest Larger Values

Again, given an array $A[1..n]$ containing (not necessarily distinct) values from $\{1, \dots, n\}$, we wish to answer the following query:

BNLV(i)**:** return $j > i$ such that $A[j] > A[i]$ and $j - i$ is minimized, and $j' < i$ such that $A[j'] > A[i]$ and $i - j'$ is minimized.

Fischer [6] gave a minimal encoding for this problem that required at most $2.54n$ bits, and gave a corresponding data structure that answers queries in $O(1)$ time. The two-dimensional version of this problem, where A is an $n \times n$ matrix, was recently considered by Jayapaul et al. [13] and Jo et al. [14]. The latter authors gave an asymptotically optimal encoding data structure using $O(n^2)$ bits that answers queries in $O(1)$ time.

4 Conclusion

We have introduced the topic of *encoding* data structures. This topic is recently gaining interest, not only as a way to obtain more space-efficient data structures, but also due to the interesting combinatorial questions that arise. As can be seen even at this early stage, the tight space restrictions of encodings sometimes make it challenging to create efficient data structures with these space bounds. The topic is wide open – any data structuring question can be cast into the encoding framework, provided only that the queries considered do not allow the input to be reconstructed completely, no matter how many queries are asked of the data structure.

References

1. Brodal, G.S., Davoodi, P., Rao, S.S.: On space efficient two dimensional range minimum data structures. Algorithmica 63(4), 815–830 (2012), `http://dx.doi.org/10.1007/s00453-011-9499-0`
2. Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (eds.): Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday, Lecture Notes in Computer Science, vol. 8066. Springer (2013), `http://dx.doi.org/10.1007/978-3-642-40273-9`
3. Davoodi, P., Navarro, G., Raman, R., Satti, S.R.: Encoding range minima and range top-2 queries. Philosphical Transactions of the Royal Society A 372, 20130131 (2014), `http://hdl.handle.net/2381/28856`
4. Davoodi, P., Raman, R., Satti, S.R.: On succinct representations of binary trees. CoRR abs/1410.4963 (2014), `http://arxiv.org/abs/1410.4963`, preliminary version in COCOON 2012, LNCS 7434.
5. Demaine, E.D., Landau, G.M., Weimann, O.: On cartesian trees and range minimum queries. In: Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I. pp. 341–353 (2009), `http://dx.doi.org/10.1007/978-3-642-02927-1_29`
6. Fischer, J.: Combined data structure for previous- and next-smaller-values. Theor. Comput. Sci. 412(22), 2451–2456 (2011), `http://dx.doi.org/10.1016/j.tcs.2011.01.036`
7. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput. 40(2), 465–492 (2011), `http://dx.doi.org/10.1137/090779759`
8. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: DeMillo, R.A. (ed.) Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA. pp. 135–143. ACM (1984), `http://doi.acm.org/10.1145/800057.808675`
9. Gawrychowski, P., Nicholson, P.K.: Optimal encodings for range min-max and top-k. CoRR abs/1411.6581 (2014), `http://arxiv.org/abs/1411.6581`
10. Golin, M.J., Iacono, J., Krizanc, D., Raman, R., Rao, S.S.: Encoding 2-d range maximum queries. CoRR abs/1109.2885 (2011), `http://arxiv.org/abs/1109.2885`, preliminary version in ISAAC 2011, LNCS 7074.
11. Grossi, R., Iacono, J., Navarro, G., Raman, R., Satti, S.R.: Encodings for range selection and top-k queries. In: Bodlaender, H.L., Italiano, G.F. (eds.) Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8125, pp. 553–564. Springer (2013), `http://dx.doi.org/10.1007/978-3-642-40450-4_47`
12. Jacobson, G.: Space-efficient static trees and graphs. In: 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989. pp. 549–554. IEEE Computer Society (1989), `http://doi.ieeecomputersociety.org/10.1109/SFCS.1989.63533`

13. Jayapaul, V., Jo, S., Raman, V., Satti, S.R.: Space efficient data structures for nearest larger neighbor. In: Proc. IWOCA 2014 (2014), to appear.
14. Jo, S., Raman, R., Satti, S.R.: Optimal encodings and indexes for nearest larger value problems. In: WALCOM 2015 (2015)
15. Kurtz, S.: Reducing the space requirement of suffix trees. Softw., Pract. Exper. 29(13), 1149–1171 (1999), `http://dx.doi.org/10.1002/(SICI)1097-024X(199911)29:13<1149::AID-SPE274>3.0.CO;2-O`
16. Navarro, G., Raman, R., Satti, S.R.: Asymptotically optimal encodings for range selection. In: Proc. 34th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2014)
17. Vuillemin, J.: A unifying look at data structures. Commun. ACM 23(4), 229–239 (1980), `http://doi.acm.org/10.1145/358841.358852`