Predecessor problem on smooth distributions

Vladimír Čunát[a,b,1]

[a] *Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic*
[b] *Institute of Computer Science, The Czech Academy of Sciences*

Abstract

We follow a research thread studying the predecessor problem on "smooth" distribution families. We propose a conceptually simpler solution utilizing well-known results from much better studied variant of the problem that assumes nothing about the input. As a side effect, we are able to extend the range of handled input distributions for the most studied case needing expected $\mathcal{O}(\log\log n)$ time, and we provide better insight into why the related methods are faster on smooth inputs.

Keywords: data structures, computational complexity, search tree

---

1. Definitions

In this section we state the predecessor problem and the input model that we use to study it. In Section 2 we formalize and prove a key property of smooth distributions that seems implicitly used in all related work (in weaker forms) as the most important step to achieve the stated performance. In Section 3 we show how to utilize this property directly with well-known distribution-independent structures for the predecessor problem to get better performance in a simpler way, essentially by converting to the case of polynomial-sized universe.

The predecessor problem consists of maintaining a set of linearly ordered keys. The basic variant allows insertions, deletions, and *predecessor* queries which find the greatest contained key that is less than a given value. We assume the usual word-RAM [1] as our computational model, and we restrict keys to word-sized integers. That can be shown not to be a significant limitation, as many other key types can be converted to the integer case cheaply, for example standard floating-point [2, sec. 2.1.3] and string keys [3].

We study the behavior of the predecessor problem on specific input key distributions, which requires us to specify how we model the input:

Email address: `vcunat@gmail.com` (Vladimír Čunát)
URL: `http://vcunat.matfyz.cz` (Vladimír Čunát)

[1] This research was supported by the Czech Science Foundation under grant GA14-10799S and by SVV project number 260 104.

Definition 1 (input model)**.** Insertions take keys distributed according to a density μ that does not change during the whole life of the structure. Deletions remove uniformly from the set of *contained* elements. All the operations are arbitrarily intermixed, and the keys are chosen independently of each other.

This model is convenient because it preserves distribution of the stored set – at any point of execution the set appears to be created by independent μ-insertions only. That is usually essential for complexity analyses in related structures. Various random deletion models were thoroughly studied by Knuth [4].

Most of related papers work with key distributions that can be described as $\left(s^\alpha, s^{1-\delta}\right)$*-smooth* for some constants $\alpha, \delta > 0$. Typically it is assumed that the particular density of the distribution is not known, but α and δ are fixed parameters.

Remark. If the distribution *was* known and its inverse cumulative distribution function F^{-1} was cheaply computable (or approximable), we could convert the problem to the uniform case (or another bounded density). We could simply store keys transformed by F^{-1} which would preserve the order. With any bounded input distribution, all operations can be easily handled in constant time by simply splitting the the key domain into $\Theta(n)$ intervals and mapping each to an element of an array. This solution for bounded distributions was pointed out already by Andersson and Mattsson [5, sec. 5.2].

Definition 2 ([6])**.** Let μ be the density of a probability distribution over an interval $\langle a, b\rangle$. Given two functions f_1 and f_2, μ is $(\mathbf{f_1}, \mathbf{f_2})$**-smooth** if

$$\exists\beta\ \forall c_1, c_2, c_3 \quad a \le c_1 < c_2 < c_3 \le b \quad \forall s \in \mathbb{N}$$
$$\Pr_{X\sim\mu}\left[X \in \left\langle c_2 - \frac{c_3 - c_1}{f_1(s)}, c_2 \right\rangle \,\middle|\, X \in \langle c_1, c_3\rangle\right] \le \frac{\beta f_2(s)}{s}$$

There is quite a simple intuition behind this complicated condition. It implies that if we cut some interval $\langle c_1, c_3\rangle$ into $f_1(s)$ subintervals and μ-generate s values from $\langle c_1, c_3\rangle$, every subinterval is expected to get $\mathcal{O}(f_2(s))$ elements.

2. Static analysis of bucketing

In this section we show how smoothness implies that an arbitrary sufficiently short interval is expected to get only constant number of input keys.

Lemma. *Given $\alpha, \delta > 0$ and a positive integer n, let us generate n keys from a $\left(s^\alpha, s^{1-\delta}\right)$-smooth distribution, and split the whole domain into at least $n^{\alpha/\delta}$ equally long intervals. Then the expected number of keys in an interval is $\mathcal{O}(1)$.*

Proof. The smoothness (Definition 2) over the domain $\langle a, b\rangle$ gives us:

$$\exists\beta\ \forall c_1, c_2, c_3 \quad a \le c_1 < c_2 < c_3 \le b \quad \forall s \in \mathbb{N}$$
$$\Pr\left[X \in \left\langle c_2 - \frac{c_3 - c_1}{s^\alpha}, c_2 \right\rangle \,\middle|\, X \in \langle c_1, c_3\rangle\right] \le \frac{\beta s^{1-\delta}}{s} = \beta s^{-\delta}.$$

We choose to cover the whole domain $\langle c_1, c_3\rangle := \langle a, b\rangle$. The conditioning can be removed because it is always fulfilled.

$$\exists\beta\ \forall c_2 \quad a < c_2 < b \quad \forall s \in \mathbb{N} \quad \Pr\left[X \in \left\langle c_2 - \frac{b-a}{s^\alpha}, c_2\right\rangle\right] \;\le\; \beta s^{-\delta}$$

Now we consider splitting the domain into $k \ge n^{\alpha/\delta}$ equally long intervals. Let us choose c_2 as the endpoint of an arbitrary[2] interval I and choose $s := \lfloor n^{1/\delta}\rfloor$, so $s^\alpha \le n^{\alpha/\delta} \le k$ and thus the above probability covers at least the whole interval I. That gives us:

$$\Pr[X \in I] \;\le\; \beta s^{-\delta} \;=\; \beta\lfloor n^{1/\delta}\rfloor^{-\delta} \;\le\; \beta\left(n^{1/\delta} - 1\right)^{-\delta}.$$

Since $\lim\limits_{n\to\infty} \dfrac{(n^{1/\delta})^{-\delta}}{(n^{1/\delta}-1)^{-\delta}} = 1$, we conclude that $\Pr[X \in I]$ is $\mathcal{O}(1/n)$.

The input keys are chosen independently, so the number of keys in I is given by the binomial distribution, and its expected value $n\Pr[X \in I]$ is in $\mathcal{O}(1)$. $\square$

Remark. We showed that the number of keys in an interval is expected to be constant, but the tail of binomial distribution can be bounded even stronger, e. g. by Chernoff bounds [7, chapter 4.1], to guarantee that high values are exponentially rare. We do not elaborate on a finer analysis, because we feel there is a more pressing problem that we do not solve in this paper – the unrealistic (unmotivated) character of the used input model. We use it for our analysis to enable comparison with known structures, as we have only found one published structure that uses a different model [8] and moreover the bounds implied in that case seem relatively weak.

3. Combining with known results and comparing to related work

We propose to split the bit representation of every key into two parts, exactly as one step of of decomposition from van Emde Boas trees, except that we split asymmetrically. In both parts we propose to use standard structures that do not utilize distribution properties of the input. The speed on the more significant bits will be due to the keys being short, and for the less significant bits the number of keys will be small in expected case due to smoothness.

[2] There is a technical difficulty with the last interval, because Definition 2 (taken from referred papers) does not allow us to choose $c_2 = b$ for some unknown reason. However, we can choose $c_2 := b - 1$, so only the maximal value is not covered. Adding a single key to the last interval separately then does not affect the validity of the implied Lemma.

3.1. Reviewing standard vEBT decomposition

Standard van Emde Boas trees solve the predecessor problem with all operations needing $\mathcal{O}(\log l)$ time when working with keys of l bits. That assumes l is not asymptotically larger than the word length of the used word-RAM.

The well-known decomposition can be viewed as performing binary search for the longest prefix of bit-representation that is shared by the searched key and some of the stored keys. The structure stores a mapping from the more significant halves to corresponding subsets of the less significant halves of the stored keys. These subsets are stored recursively in the same way, and also the set of occurring more significant halves is stored recursively. The operations on vEBT are carefully designed to perform at most one nontrivial recursive call on each recursion level, and all the rest are constant-time operations.

To keep the space linear, some modifications to the original structure are required. The mappings can be represented by hash tables instead of simple arrays, and indirection can be used: the whole set is split into $\Theta(\log l)$-sized consecutive clusters in a fashion similar to B-trees, and the actual vEBT construction is only applied to the whole clusters joined together by a doubly linked list. These changes make the complexity of insertions and deletions expected and amortized where the expectation is only over random bits.

A possible way of doing these modifications is described in more details e.g. in [9, chapter 4], including the algorithms for operations which can also be found in textbooks [10, chapter 20].

3.2. Modifying the vEBT decomposition

To cover the cases most studied in previous work, we focus on using linear space and work with $\left(s^{\alpha}, s^{1-\delta}\right)$-smooth distributions. To use the Lemma, we need to split away at least $(\alpha/\delta)\log n$ bits. If the values of parameters are unknown, we can split away more bits to be asympotically certain, e.g. $\log^2 n$.

These bounds on the number of bits change during the life of the structure, but we can easily ensure that we cut at least that many by performing a full rebuild after every $n_0/4$ modifying operations where n_0 denotes the size of the stored set during the last rebuild. It would also be possible to deamortize that process by standard technique of global rebuilding [11]. Also note that $(\alpha/\delta)\log n > \log^2 n$ only for $n < 2^{\alpha/\delta}$ which is an unknown constant; thus the performance will be constant in that base case.

We can use standard hashed vEBT on the split more significant bits to get $\mathcal{O}\left(\log\left(\log^2 n\right)\right) = \mathcal{O}\left(\log\log n\right)$ expected amortized time per operation. Each substructure for the less significant bits is expected to store $\mathcal{O}(1)$ keys. The resulting performance is essentially the same as that of Kaporis et al. [6], except that unlike all previous work we achieve linear space without requiring $\alpha \leq 1$, and our approach is conceptually much simpler.

Theorem (main result)**.** *There is a structure solving the predecessor problem in $\mathcal{O}\left(\log\log n\right)$ expected time per operation and linear space on any $\left(s^{\alpha}, s^{1-\delta}\right)$-smooth distribution for arbitrary $\alpha, \delta > 0$. The distribution and parameters do not need to be known.*

3.3. Concluding remarks

If we wanted to cut down time on unfavorable input, we can use the same approach as Kaporis et al. There are standard structures for the predecessor problem needing time $\mathcal{O}(\sqrt{\log n/\log\log n})$ per operation on arbitrary input (considering the less precise bounds), so these can be used for the less significant bits. Note that when handling the more significant bits, the expected amortized time $\mathcal{O}(\log\log n)$ remains independent of the input, as it only comes from randomization during hashing.

If we allowed slightly more space of $\mathcal{O}\left(n^{1+1/o(\log n)}\right) \supset \mathcal{O}\left(n^{1+\epsilon}\right)$ words, we would immeditely get $\mathcal{O}(1)$ expected time due to the way van Emde Boas trees speed up in that case [12, Sec. 1.3.2]. That covers the cases studied in an unpublished work of Bellazougui et al. [13], also in a simpler way.

We feel that it is good to decouple most of the analysis from complex conditions on the input, as especially in practice we can rarely assume that all operations on the structure are independent and identically distributed. For the approach of our paper to work, it is enough (informally) that the information is only carried by a sufficiently short bit prefix of the keys.

Acknowledgements. Most of results in this paper were included in the author's master thesis [9]. Many thanks belong to his supervisor Václav Koubek for helpful discussions during our research into this problem.

References

[1] T. Hagerup, Sorting and searching on the word RAM, in: M. Morvan, C. Meinel, D. Krob (Eds.), STACS 98, Vol. 1373 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1998, pp. 366–398. URL `http://www.springerlink.com/content/w53xp778k50458n6/abstract/`

[2] D. Goldberg, What every computer scientist should know about floating-point arithmetic, ACM Comput. Surv. 23 (1) (1991) 5–48. `doi:http://doi.acm.org/10.1145/103162.103163`.

[3] A. Andersson, M. Thorup, Dynamic string searching, in: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '01, Society for Industrial and Applied Mathematics, 2001, pp. 307–308. URL `http://dl.acm.org/citation.cfm?id=365411.365465`

[4] D. E. Knuth, Deletions that preserve randomness 3 (5) (1977) 351–359. URL `http://dx.doi.org/10.1109/TSE.1977.231160`

[5] A. Andersson, C. Mattsson, Dynamic interpolation search in o(log log n) time, in: Automata, Languages and Programming, Springer, 1993, pp. 15–27. URL `http://link.springer.com/content/pdf/10.1007/3-540-56939-1_58.pdf`

[6] A. Kaporis, C. Makris, S. Sioutas, A. Tsakalidis, K. Tsichlas, C. Zaroliagis, Dynamic interpolation search revisited, in: 33rd International Colloquium for Automata, Languages and Programming (ICALP 2006), Vol. 4051 of Lecture Notes in Computer Science, Springer, 2006, pp. 382–394.
URL `http://icalp06.dsi.unive.it/`

[7] R. Motwani, P. Raghavan, Randomized algorithms, Cambridge University Press, New York, NY, USA, 1995.

[8] E. D. Demaine, T. Jones, M. Pătraşcu, Interpolation search for non-independent data, in: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '04, Society for Industrial and Applied Mathematics, 2004, pp. 529–530.
URL `http://dl.acm.org/citation.cfm?id=982792.982870`

[9] V. Čunát, Data structures for various distributions of data (2010).
URL `https://is.cuni.cz/webapps/zzp/detail/78657/11784999/`

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Third Edition, 3rd Edition, The MIT Press, 2009.

[11] M. H. Overmars, The Design of Dynamic Data Structures, Vol. 156 of Lecture Notes in Computer Science, Springer-Verlag, 1983, Ch. 5.

[12] M. Pătraşcu, M. Thorup, Time-space trade-offs for predecessor search, in: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, STOC '06, ACM, 2006, pp. 232–240. `doi:10.1145/1132516.1132551`.
URL `http://doi.acm.org/10.1145/1132516.1132551`

[13] D. Belazzougui, A. C. Kaporis, P. G. Spirakis, Random input helps searching predecessors, CoRR abs/1104.4353.
URL `http://arxiv.org/abs/1104.4353`