Weekplan: External Memory II

Philip Bille Inge Li Gørtz Eva Rotenberg

References and Reading

[1] An Introduction to B^ε-Trees and Write-Optimization. M. A. Bender, M. Farach-Colton, W. Jannen, R. Johnson, B. C. Kuszmaul, D. E. Porter, J. Yuan, and Y. Zhan, ;login: USENIX Magazine, 2015.

[2] Lower bounds for external memory dictionaries, G. S. Brodal and R. Fagerberg, SODA 2003.

[3] The string B-tree: a new data structure for string search in external memory and its applications, P. Ferragina and R. Grossi, JACM 1999.

We recommend reading [1] and [3] in detail. [2] is the original paper introducing B^ε-trees.

1 [w] B^ε-tree Example Solve the following exercises.

1.1 Consider a $\sqrt{B}$-tree with degree 3 and buffer size 3. Insert the sequence $S = 1, 2, 3, \ldots, 10$. At the end show the action of searching for each element in S.

1.2 Consider inserting and deleting the same element multiple times in a B^ε-tree. How will the buffers in the tree reflect the current state of the element? How can we correctly perform search?

2 [w] B^ε-tree Analysis Analyze I/O bounds for searches and updates on a B^ε-tree. What happens if we set $\varepsilon = 1$?

3 [w] B^ε-tree Node Implementation Give an efficient internal memory implementation of a B^ε-tree node. *Hint:* what operations are needed and what data structure supports these efficiently?

4 Shared Buffers in B^ε-trees Each node in a B^ε-tree stores B^ε separate buffers (one for each child) of size $B^{1-\varepsilon}$. A friend suggest to store all the buffers together in a single buffer of size B. We only flush the buffer if it is completely full and hence potentially avoid some of the flushing if updates are uneven. Will this modification work and how will it affect the performance of B^ε-trees?

5 Advanced Updates in B^ε-Trees Consider a B^ε tree storing a set of keys S. Each key x also stores an integer x.data. Consider the following update operations:

- increment(x): If $x \in S$, add 1 to x.data. Otherwise, do nothing.
- delete-predecessor(x): If $x \in S$, delete the predecessor of x.
- delete-if-negative(x): If $x \in S$ and $x.\text{data} < 0$ delete x. Otherwise, do nothing.

Solve the following exercises.

5.1 Consider implementing the above operations. For each of them either give an efficient buffered implementation or argue why they cannot be implemented with the buffering technique.

5.2 In general, what operations can be efficiently implemented with the buffering technique?

6 Range Searching Suppose we want to extend B^ε-trees to support the following range searching operations:

- report(i, j): Report all elements with keys k, such that $i \leq k \leq j$.
- count(i, j): Return the number of elements with keys k, such that $i \leq k \leq j$.

Solve the following exercises.

6.1 $[w]$ Recall (or reproduce) the bounds for these operations on B-trees.

6.2 Can you match these bounds with a B^ε-tree?

7 String Dictionaries on a RAM Consider your favourite data structure for string searching on a RAM. What is the I/O complexity of this algorithm if implemented directly in external memory? Compare the result with a good external data structure.

8 Blind Trie Space Argue that the space of blind trie with K strings of total length N is $O(K)$.

9 Blind Trie Search Consider searching for a string P in a blind trie T for a set of strings S. Let X be a longest matching string in T, i.e., a string in T that agrees with P on the maximum number of branching character. Solve the following exercises.

9.1 Show that longest common prefix of P and X is a longest common prefix of P and any string in T.

9.2 Conclude that the traverse and verify algorithm in a blind trie correctly implements the search operation.

9.3 Consider the complexity of the traverse and verify algorithm. Argue that it can be implemented in $O(|P|)$ time and $O(K)$ space in addition to the strings on a word RAM. *Hint:* how should we implement navigation at branching nodes?

10 $[w]$ **String B-trees Warm Up** Consider the set S of 15 strings and the sketch of the string B-tree. Explicitly construct the full string B-tree (or a large part of it) including each of the blind tries stored at nodes. Show the action of searching for various query strings in string B-tree.

11 Queries on String B-trees Consider the following operations on String B-trees.

- predecessor(P): Return the lexicographic predecessor of P.
- report(P): Return all strings for which P is a prefix.
- count(P): Return the number of strings for which P is a prefix.

Show how to implement the above operations efficiently.

12 Dynamic String B-trees Show how to implement insertions and deletions on string B-trees in $O(\log_B N + |P|/B)$ I/Os.