Projects for the Course of Web and Document Databases

(Deadline: TBD)

For all the projects, the students are required to give a 20 minutes presentation (with extra 5 minutes for discussion). Also, a technical report is submitted, with theoretic description, program design details, and C++ code.

1. Tree encoding (selected by MD M. Hassan, March 24, 2025; first presentation)
   (a) C++ program to load a real XML data (such as DPLP) into main memory and transform it to a tree structure *T*.
   (b) We associate each node *v* in *T* with a quadruple $\alpha(v) = (d, l, r, ln)$, where *d* is the document *identifier* (*DocId*), l = *LeftPos*, r = *RightPos* and ln = *LevelNum*. (*LeftPos* and *RightPos* are generated by counting word numbers from the beginning of the document until the start and end of the element, respectively.)
   By using such a data structure, the structural relationship between the nodes in an XML database can be simply determined:
   i. *Ancestor-descendant*: A node v_1 associated with (d_1, l_1, r_1, ln_1) is an ancestor of another node v_2 with (d_2, l_2, r_2, ln_2) iff $d_1 = d_2$, $l_1 < l_2$ and $r_1 > r_2$
   ii. *Parent-child*: A node v_1 associated with (d_1, l_1, r_1, ln_1) is the parent of another node v_2 with (d_2, l_2, r_2, ln_2) iff $d_1 = d_2$, $l_1 < l_2$, $r_1 > r_2$ and $ln_2 = ln_1 + 1$
   iii. *From left to right*: A node v_1 associated with (d_1, l_1, r_1, ln_1) is to the left of another node v_2 with (d_2, l_2, r_2, ln_2) iff $d_1 = d_2$, $r_1 < l_2$.

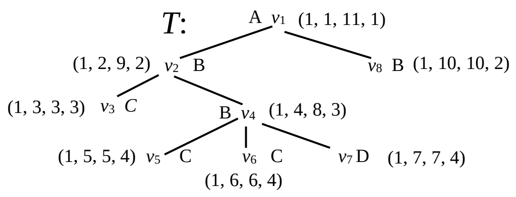

Fig. 1: Illustration for tree encoding

   In Fig. 1, v_2 is an ancestor of v_6 and we have v_2.*LeftPos* = 2 < v_6.*LeftPos* = 6 and v_2.*RightPos* = 9 > v_6.*RightPos* = 6. In the same way, we can verify all the other relationships of the nodes in the tree. In addition, for each leaf node *v*, we set *v*.*LeftPos* = *v*.*RightPos* for simplicity, which still work without downgrading the ability of this mechanism.
   (c) For the purpose of experiments, download an XML document from http://www.cs.washington.edu/research/xmldatasets, and then transform it to a tree structure.
   (d) Report: algorithm description, implementation details, C++-code, test results.

2. Web crawler ((selected Chowdhury, Md Sahadatunn, March 26, 2025; third presentation)
   (a) C++ program to get web pages for a set of URLs (programs are available).
   (b) Each time adding a web page to the set of web pages, it will be checked whether the new is already in the set or a fake page.

(c) Experiment
(d) Report: algorithm description, implementation details, C++-code, test results (such as the number of different words in all documents, sizes of established inverted files and signature trees).

3. Quad-tree (selected by Dharanidharan, April 02, 2025; first presentation)
   (a) C++ program to generate a quad-tree for a data file over two numeric attributes. The file contains 1 million fake records.
   (b) Store the quad-tree in a file.
   (c) Experiment: quad-tree built for a data file over two numeric attributes. The file contains 1 million fake records. Try 20 queries with the quad-tree used and not used.
   (d) Report: algorithm description, implementation details, C++-code, test results.

4. XML document storage (selected by Barua, March 31, second presentation)
   (a) C++ program to store XML documents in four files (tables) with the following formats:
   DocRoot(docID, rootElmentID)
   SubElement(parented, childID, position)
   ElementAttribute(elementID, name, value)
   ElementValue(elementID, value)
   (b) Work on a database system such as Access, or SQL server, and transform some simple XPath expressions to SQL queries, evaluate them against your database.
   (c) Report: algorithm description, implementation details, C++-code, test results.

5. Reachability checking (selected by Prerit Bhandari, March 24, 2025; third presentation)
   (a) Read paper: Sebastiaan J. van Schaik and Oege de Moor. A memory efficient reachability data structure through bit vector compression. In SIGMOD '11: Proceedings of the 37th SIGMOD international conference on Management of data, pages 913-924, New York, NY, USA, 2011. Summarize its main idea.
   (b) The code of the method discussed in the paper can be found in: https://github.com/sj/PWAHStackTC.git (http://www.sjvs.nl/?p=72).
   (c) Run the code against some graphs, which can be found in web sites listed in the paper.
   (d) Report: algorithm description, test results.

6. Bitmap (selected by Anika, March 24, 2025; second presentation)
   (a) C++ program to generate bit-vectors for a data file over two numeric attributes. The file contains 1 million fake records.
   (b) Compressing bit vectors.
   (c) Experiment on the query time and space overhead.
   (d) Report: algorithm description, implementation details, C++-code, test results.

7. Set intersection (1)

(a) The set intersection operation is a core operation in any search engine, such as Google and Yahoo. Implement this operation by using the Skip-List data structure in C++.
(b) Experiment
(c) Report: algorithm description, time analysis, implementation details, C++-code, test results

Reference: W. Pugh. A skip list cookbook, technical report, UMIACS-TR-89-72, University of Maryland, 1990.
http://nlp.stanford.edu/IR-book/html/htmledition/faster-postings-list-intersection-via-skip-pointers-1.html

8. B+-tree (selected by Kahn, March 26, 2025; first presentation)
(a) Implementation of the algorithm for constructing a B+-tree in C++.
(b) Store the B+-tree in a file.
(c) Experiment: B+-tree built for a data file over a numeric key. The file contains 1 million fake records. Try 20 queries with the B+-tree used and not used.
(d) Report: algorithm description, implementation details, C++-code, test results.

13. Linear Hashing (selected by Rony, March 31, third presentation)
(a) The linear hasing is a dynamic hashing method to store a data file so that the records in it can be found very fast. The description of the method can be found on one of my lecture notes (on home page). Implement this method in C++.
(b) Experiment
(c) Report: algorithm description, time analysis, implementation details, C++-code, test results

14. Hash-based method for set intersection (selected Sotonye Charls, March 26, 2025; sec ond pres-entation)
(a) Implement the set intersection operation by using the hashing technique.
(b) Experiment
(c) Report: algorithm description, time analysis, implementation details, C++-code, test results

Reference: B. Ding, A.C. König, Fast set intersection in memory, *Proc. of the VLDB Endowment, v.4 n.4*, p.255-266, January 2011.

15. KD-tree construction and search
(a) Implement the construction of a *kd*-tree over *k* attributes in a table. Also, implement the search of a *kd*-tree.
(b) Experiment
(c) Report: algorithm description, time analysis, implementation details, C++-code, test results

16. R-tree
(a) Implementation of the algorithm for constructing an R-tree in C++.
(b) Store the R-tree in a file.

(c) Experiment: R-tree built for a data file containing a set of rectangles. The file contains 1 million fake records. Try 20 queries with the R-tree used and not used.
(d) Report: algorithm description, implementation details, C++-code, test results (searching time, number of accessed pages by each index tree searching).

17. Evaluation of word-oriented queries based on inverted files
    (a) Implementation of the algorithm for constructing an inverted file in C++.
    (b) Query evaluation based on bitwise logic operations
    (c) Experiment
    (d) Report: algorithm description, implementation details, C++-code, test results.

18. Trie-based method to evaluate word-oriented queries
    (a) Implementation of the algorithm for constructing a trie in C++.
    (b) Query evaluation based on trie searching
    (c) Experiment
    (d) Report: algorithm description, implementation details, C++-code, test results.

19. Page ranker based on transition matrices
    (a) Implementation of the algorithm for constructing transition matrices in C++.
    (b) Estimate page importance by using transition matrices.
    (c) Experiment
    (d) Report: algorithm description, implementation details, C++-code, test results.

20. Index over intervals
    (a) Implementation of an algorithm for constructing a variant of the B+-tree over intervals in C++.
    (b) The intervals can be created as follows:
        - First, create a tree *T*.
        - Then, generate intervals for the nodes in *T* as below.

We can assign each node *v* in *T* an interval $[\alpha_v, \beta_v)$, where α_v is *v*'s preorder number (denoted *pre*(*v*)) and β_v - 1 is equal to the largest preorder number among all the nodes in *T*[*v*]. So another node *u* labeled $[\alpha_u, \beta_u)$ is a descendant of *v* (with respect to *T*) iff $\alpha_u \in [\alpha_v, \beta_v)$, as illustrated in Fig. 2. If $\alpha_u \in [\alpha_v, \beta_v)$, we say, $[\alpha_u, \beta_u)$ is subsumed by $[\alpha_v, \beta_v)$. This method is called the *tree labeling*.

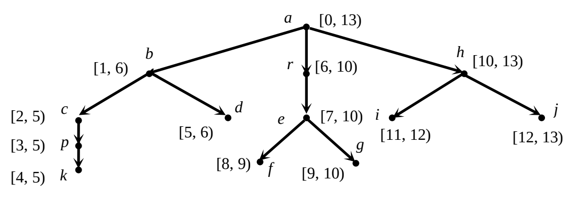

Fig. 2: A spanning tree and intervals

A B+-tree over intervals can be established in a way similar to B+-tree trees, but over intervals, as shown in Fig. 3.

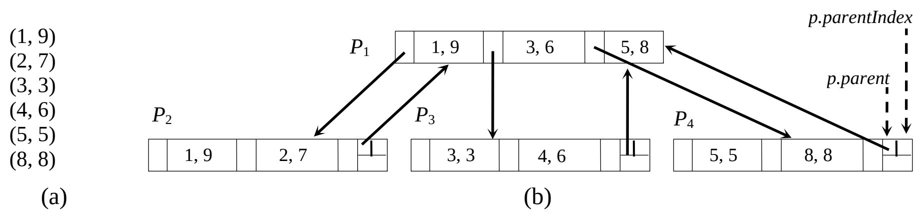

Fig. 3: An interval sequence and XB-tree over it

In such an index structure, each entry in a page is a pair *a* = (LeftPos, RightPos) (referred to as a bounding segment) such that any entry appearing in the subtree pointed to by the pointer associated with *a* is subsumed by *a*. In addition, all the entries in a page are sorted by their LeftPos values.

In each page *P* of an XB-tree, the bounding segments may partially overlap, but their LeftPos positions are in increasing order. Let a_1 and a_2 be two consecutive entries. Then the leftPos value of any entry in the subtree rooted at a_1 is smaller than the leftPos value of a_2. For instance, in the XB-tree shown in Fig. 3(b), the first entry of P_1: [1, 9] overlaps the second entry: [3, 6]. But the leftPos value of any entry in the subtree rooted at [1, 9] is smaller than 3. Besides, it has two extra data fields: *P.parent* and *P.parentIndex*. *P.parent* is a pointer to the parent of *P*, and *P.parentIndex* is a number *i* to indicate that the *i*th pointer in *P.parent* points to *P*. For instance, in the XB-tree shown in Fig. 3(b), P_3*.parentIndex* = 2 since the second pointer in P_1 (the parent of P_3) points to P_3.

(c) For the purpose of experiments, download an XML document from http://www.cs.washington.edu/research/xmldatasets, and then transform it to a tree structure.
(d) Report: algorithm description, implementation details, C++-code, test results (searching time, number of accessed pages by each index tree searching).

20. Unordered tree matching
   (a) Implementation of an algorithm for evaluating unordered tree pattern queries in C++.
   (b) Experiments on some tree patterns represented as XPath expressions.
   (c) Report: algorithm description, implementation details, C++-code, test results.

21. Data mining (selected by Devkumar Barot, March 31, 2025; first presentation)
   (a) Implementation of the algorithm for finding a single package which can satisfy a maximum group of customers in terms of a given query log in C++.
   (b) Experiment
   (c) Report: algorithm description, time analysis, implementation details, C++-code, test results

22. SvS algorithm and adaptive algorithm for set intersection (4)
   (a) Implement the set intersection operation by using the SvS algorithm and the adaptive algorithm.
   (b) Experiment

(c) Report: algorithm description, time analysis, implementation details, C++-code, test results

Reference: J. Barbay, A. López-Ortiz, T. Lu, A. Salinger: An experimental investigation of set intersection algorithms for text searching, *ACM Journal of Experimental Algorithmics 14*: (2009).

23. Tree inclusion
    (a) Implement an algorithm to check tree inclusion.
    (b) Experiment
    (c) Report: algorithm description, time analysis, implementation details, C++-code, test results

    Reference: Y. Chen and Y.B. Chen, Tree Inclusion Checking Revisited, 5th International Conference on Data Management Technologies and Applications (DATA), July 24 - 26, 2015, Lisbon, Portugal, pp. 301 - 308.

24. Barbay-Kenyon algorithm for set intersection
    (b) Implement the set intersection operation by using Barbay-Kenyon algorithm.
    (c) Experiment
    (d) Report: algorithm description, time analysis, implementation details, C++-code, test results

    Reference: J. Barbay, A. López-Ortiz, T. Lu, A. Salinger: An experimental investigation of set intersection algorithms for text searching, *ACM Journal of Experimental Algorithmics 14*: (2009).

25. Implementing RRR data structure to support the rank operation. By the rank operation, we will find the number of the appearances of a certain character up to a specific position in a string.
    (a) Implementation of the algorithm in C++.
    (b) Experiment
    (c) Report: algorithm description, time analysis, implementation details, C++-code, test results.

References:

A, Bowe, Multiary Wavelet Trees in Practice, Master thesis, School of Computer Science and Information Technology, RMIT University, Melbourne, Australia, 2010. (http://alexbowe.com/rrr/)

26. Set union (~~selected Chowdhury, Md Sahadatunn, March 31, 2025; first presentation~~)
    a. The set union operation will merge two sets A and B. Implement this operation using the Hwang-Lin algorithm.
    b. Experiment
    c. Report: algorithm description, time analysis, implementation details, C++-code, test results

    Reference: F.K. Hwang and S. Lin, A Simple Algorithm for Merging Two Distinct Linear Ordered Sets, *SIAM J. Comput.*, Vol. 1. No. 1, March 1972.

27. Implementing Wavelet trees to support the rank operation
    (a) Implementation of the algorithm in C++.
    (b) Experiment
    (c) Report: algorithm description, time analysis, implementation details, C++-code, test results.

References:

https://en.wikipedia.org/wiki/Wavelet Tree

A, Bowe, Multiary Wavelet Trees in Practice, Master thesis, School of Computer Science and Information Technology, RMIT University, Melbourne, Australia, 2010. (http://alexbowe.com/rrr/)

28. Set Intersection (2)
    (a) Implement Baeza-Yates algorithm for doing set intersection.
    (b) Experiment
    (c) Report: algorithm description, time analysis, implementation details, C++-code, test results

    Reference: R.A. Baeza-Yates, and A. Salinger: Experimental analysis of a fast intersection algorithm for sorted sequences, in *Proc. 12th Intl. Conf. on String Processing and Information*, Springer, Berlin, 13-24, 2005..

Guidance to project reports:

1. Introduction (including the problem description, motivation – its significance and application in the computer engineering and industry)
2. Related work (describe some important techniques related to the problem to be addressed)
3. Main thrust (detailed description of the method, formal algorithm, analysis of computational complexities: time and space overhead)
4. Experiments (main data structures used for implementation, description of the data used for tests, test results: charts, histogram, or tables)
5. Future work (discussion on the possible improvements, or possible extension)
6. References

Project evaluation criteria:

Presentation: 30%
Technical report: 30%
Test and experiment: 30%
Participation of seminar: 10%