NATIONAL RESEARCH UNIVERSITY

Higher School of Economics
Faculty of Computer Science, School of Software Engineering

SQLITE RDBMS EXTENSION FOR DATA INDEXING USING B-TREE MODIFICATIONS

Anton Rigin, Sergey Shershakov

May 30 2019, SYRCoSE 2019, Saratov State University

OUTLINE

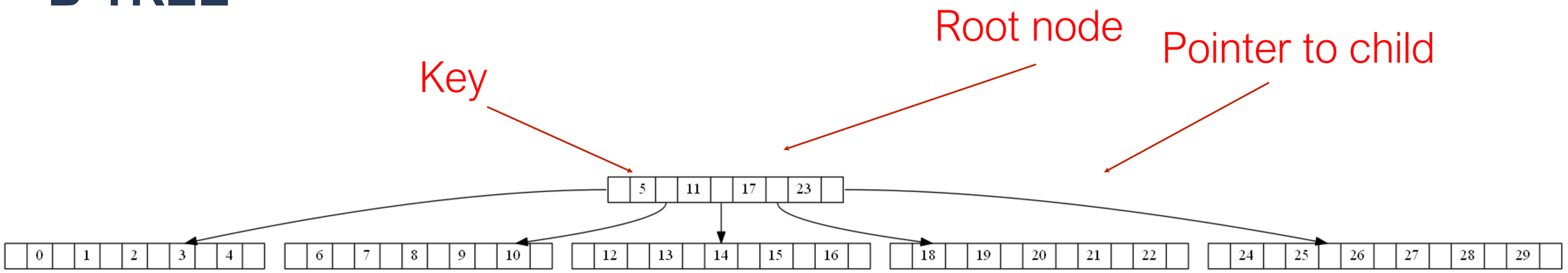
- B-tree and modifications
- Motivation, existing solutions and main goals
- SQLite and extensions
- B-tree modifications C++ library and research conducted
- Algorithm of selecting the index structure
- Implementation details
- Usage example and experiment conducted

B-TREE

- Balanced search tree
- Nodes may contain more than 1 key and more than 2 pointers to the children nodes
- If some node contains k keys than it contains $k + 1$ pointers to the children nodes
- **B-tree order** is such a t number that (k is the count of keys in the node):
  - ✓for each non-root node: $t - 1 \leq k \leq 2t - 1$
  - ✓for root node in the non-empty tree: $1 \leq k \leq 2t - 1$
  - ✓for root node in the empty tree: $k = 0$
- **B-tree height** is $O(log_t n)$, where t is tree order and n is the count of keys in the tree
- Usually used as the data index

[1] R. Bayer and E. McCreight, “Organization and maintenance of large ordered indexes,” *Acta Informatica*, vol. 1, no. 3, pp. 173 – 189, 1972.

B-TREE



The B-tree example, t = 6

B-TREE OPERATIONS

- **Searching**
  - ✓ Time complexity – $O(tlog_t n)$
  - ✓ Memory usage – $O(t)$
  - ✓ Disk operations count – $O(log_t n)$
- **Nodes split (the part of insertion)**
  - ✓ Time complexity – $O(t)$
  - ✓ Memory usage – $O(t)$
  - ✓ Disk operations count – $O(1)$
- **Insertion**
  - ✓ Time complexity – $O(tlog_t n)$
  - ✓ Memory usage – $O(tlog_t n)$ for simple recursion and $O(t)$ for tail recursion or loop
  - ✓ Disk operations count – $O(log_t n)$
- **Deletion**
  - ✓ Time complexity – $O(tlog_t n)$
  - ✓ Memory usage – $O(tlog_t n)$ for simple recursion and $O(t)$ for tail recursion or loop
  - ✓ Disk operations count – $O(log_t n)$

[1] R. Bayer and E. McCreight, “Organization and maintenance of large ordered indexes,” *Acta Informatica*, vol. 1, no. 3, pp. 173 – 189, 1972.

B-TREE MODIFICATIONS

- **B^+-tree**
  - ✓Only leaf nodes contain real keys (real data), other nodes contain router keys [2]
  - ✓Deletion is probably faster than for B-tree and B^*-tree
- **B^*-tree**
  - ✓Each node (except of the root node) is filled at least by 2/3 not 1/2 [3]
  - ✓Keys insertion for B^*-tree is expected to be faster than for B-tree and B^+-tree
- **B^{*+}-tree**
  - ✓Developed by authors of this work [4]
  - ✓Combines the main B^+-tree and B^*-tree features together

[2] K. Pollari-Malmi. (2010). B+-trees [PDF paper]. Available: https://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf
[3] "B^*-tree." NIST Dictionary of Algorithms and Data Structures. Available: https://xlinux.nist.gov/dads/HTML/bstartree.html (accessed Dec. 24, 2018).
[4] A. Rigin, "On the Performance of Multiway Trees in the Problem of Structured Data Indexing," (in Russian), coursework, Dept. Soft. Eng., HSE, Moscow, Russia, 2018.

MOTIVATION AND EXISTING SOLUTIONS

- **Motivation for developing of yet another SQLite extension**
  - ✓ In the current time, the data amounts are continuously growing
  - ✓ It is necessary to develop new efficient approaches to data indexing in the DBMSs
  - ✓ SQLite contains a small number of index structures by default – it is relevant to add new ones
- **Existing solutions**
  - ✓ B-tree is the default index structure in the SQLite
  - ✓ There are some SQLite extensions (for example, the extension adding the R-tree indexing)
  - ✓ No extensions with B^+-tree, B^*-tree, and B^{*+}-tree were found

THE MAIN GOALS OF THE WORK

- To add new indexing engines to SQLite based on B-tree modifications (B^+-tree, B^*-tree and B^{*+}-tree)
- To develop and implement an algorithm that would allow selecting the suitable indexing data structure (B^+-tree, B^*-tree or B^{*+}-tree) according to performed operations for some specific table

SQLITE

- Popular open-source embedded relational DBMS
- Written in the C language
- Uses the B-tree as the default index
- SQLite is expandable by dynamically linked libraries [5]

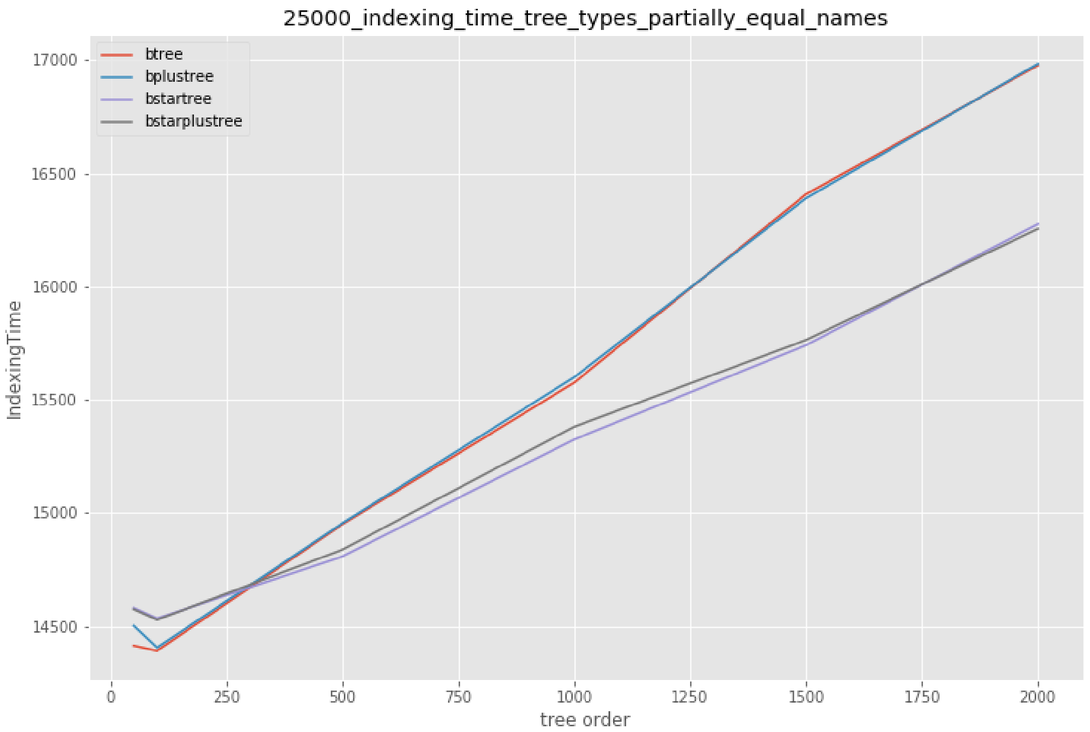
[5] “Run-Time Loadable Extensions.” SQLite.org. Available: https://www.sqlite.org/loadext.html (accessed Jan. 20, 2019).

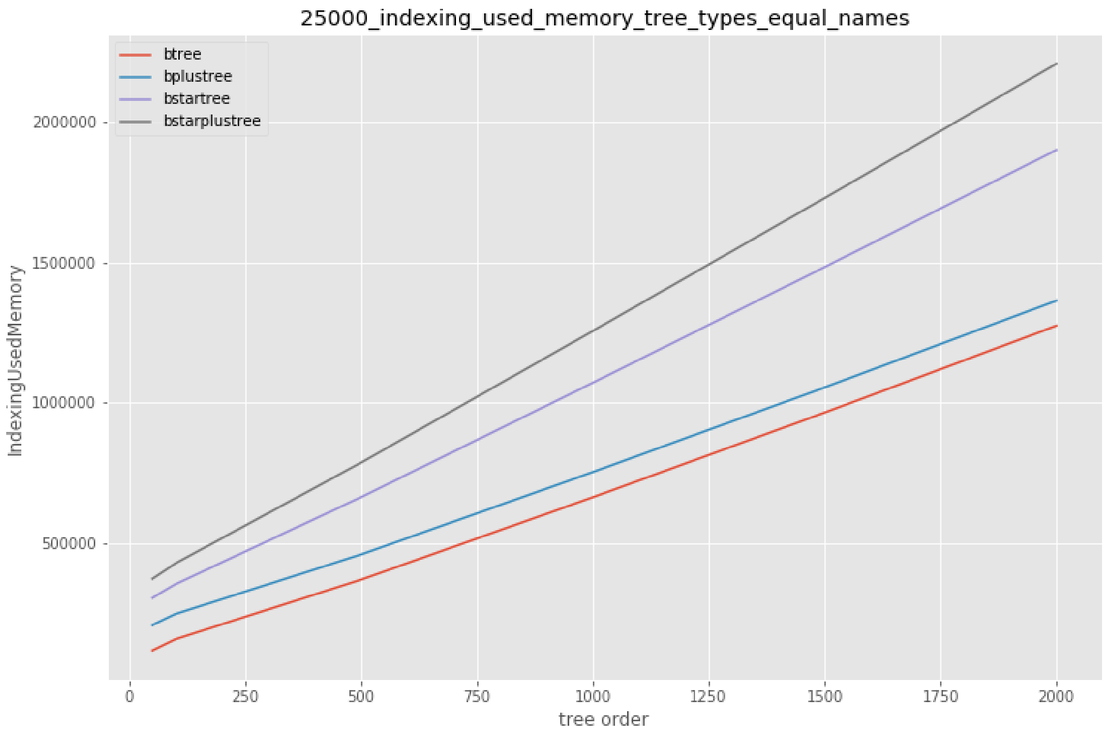
C++ LIBRARY OF B-TREE MODIFICATIONS

- Contains B-tree, B^+-tree, B^*-tree and B^{*+}-tree implementations [4]
- In the current work connected to the SQLite as the **run-time loadable extension**

[4] A. Rigin, “On the Performance of Multiway Trees in the Problem of Structured Data Indexing,” (in Russian), coursework, Dept. Soft. Eng., HSE, Moscow, Russia, 2018.

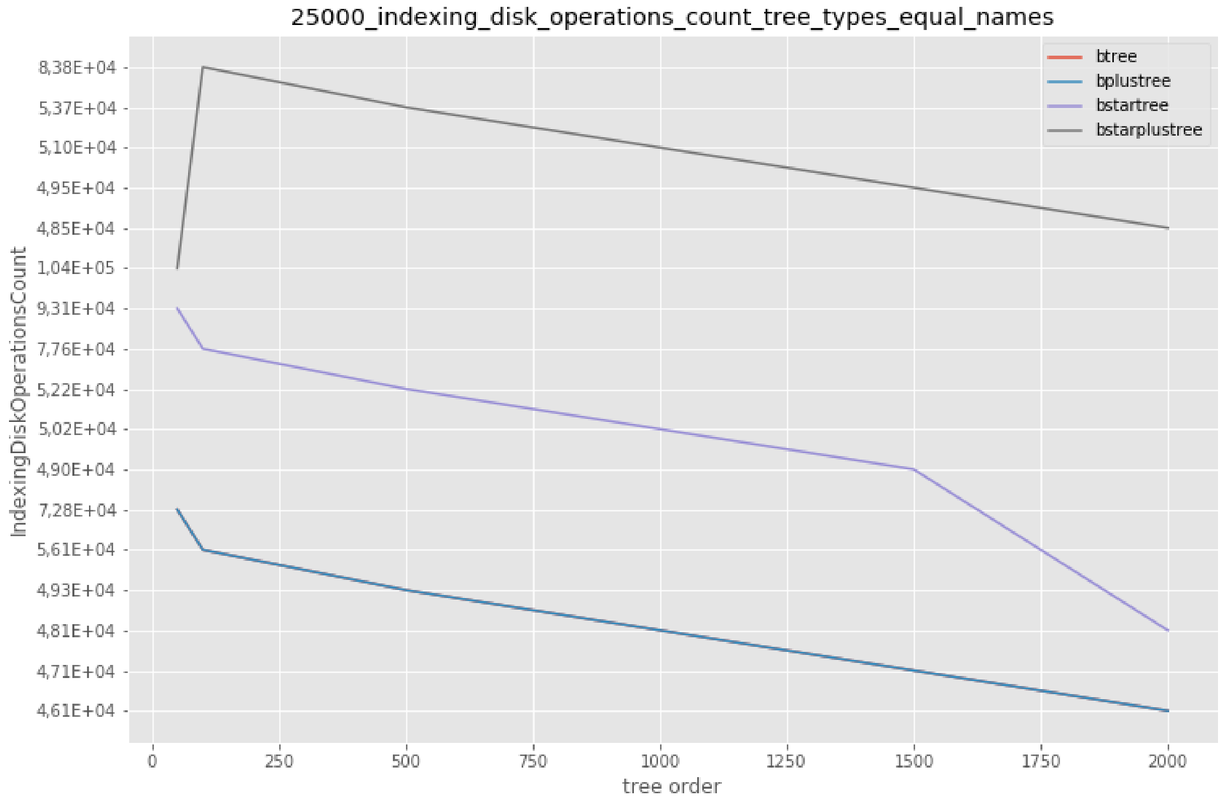
RESEARCH CONDUCTED USING THE B-TREE MODIFICATIONS





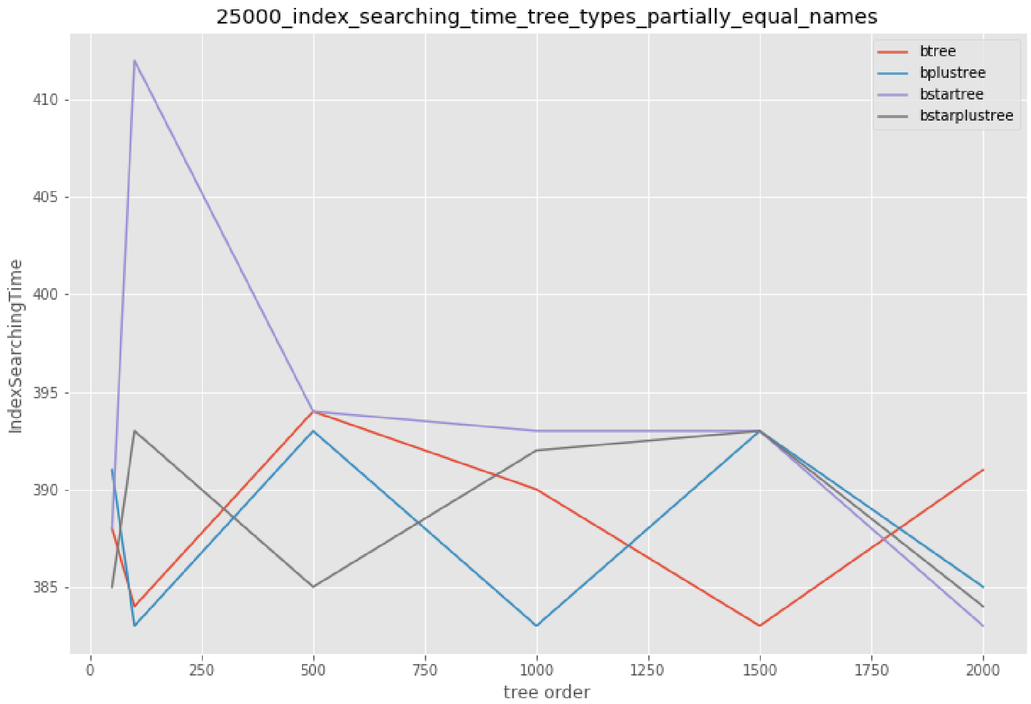
[4] A. Rigin, “On the Performance of Multiway Trees in the Problem of Structured Data Indexing,” (in Russian), coursework, Dept. Soft. Eng., HSE, Moscow, Russia, 2018.

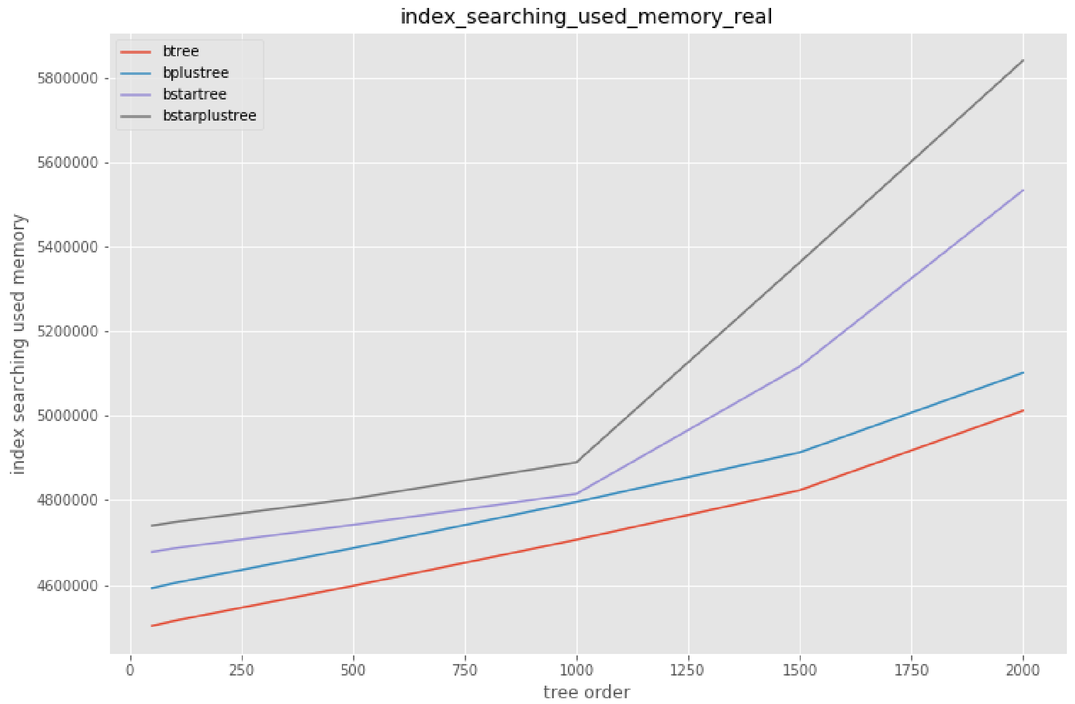
RESEARCH CONDUCTED USING THE B-TREE MODIFICATIONS



[4] A. Rigin, “On the Performance of Multiway Trees in the Problem of Structured Data Indexing,” (in Russian), coursework, Dept. Soft. Eng., HSE, Moscow, Russia, 2018.

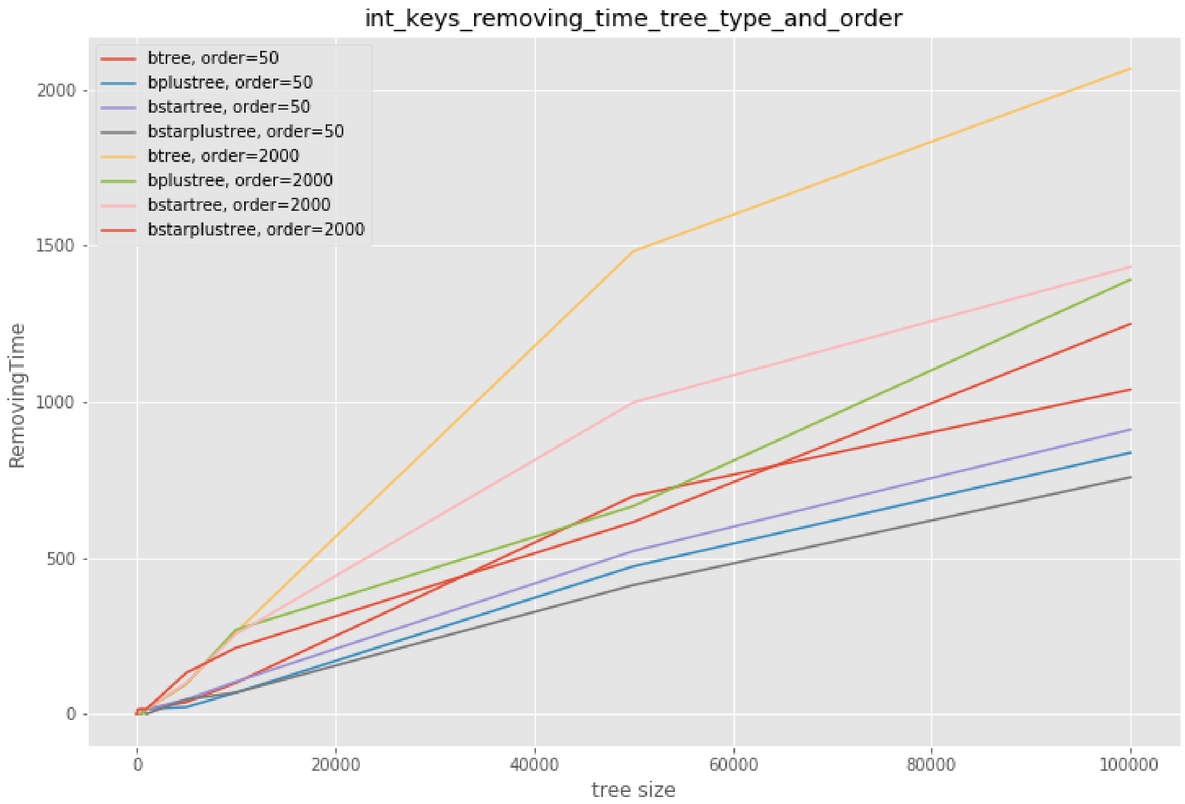
RESEARCH CONDUCTED USING THE B-TREE MODIFICATIONS





[4] A. Rigin, “On the Performance of Multiway Trees in the Problem of Structured Data Indexing,” (in Russian), coursework, Dept. Soft. Eng., HSE, Moscow, Russia, 2018.

RESEARCH CONDUCTED USING THE B-TREE MODIFICATIONS



[4] A. Rigin, “On the Performance of Multiway Trees in the Problem of Structured Data Indexing,” (in Russian), coursework, Dept. Soft. Eng., HSE, Moscow, Russia, 2018.

SELECTING THE INDEX STRUCTURE

- The index structure is selected from the B-tree modifications (B^+-tree, B^*-tree and B^{*+}-tree)
- Selecting is executed at the start of each table operation (search, insertion, updating, deletion)
- The index structure rebuilding is performed only at each 1000-th operation and only for the first 10000 operations
- Index structure selection depends on the ratios of the numbers of operations of different types (search, insert, delete), applied to a tree
- The tree order of the B-tree and its modifications used in the SQLite extension developed in this work is chosen to be equal to 750
- The B^+-tree is used by default in the developed SQLite extension

ALGORITHM OF SELECTING THE INDEX STRUCTURE

1. If the current total number of the operations on a tree is equal to 0, or more than 10000, or not a multiple of 1000, then the algorithm stops, otherwise it goes to step 2.

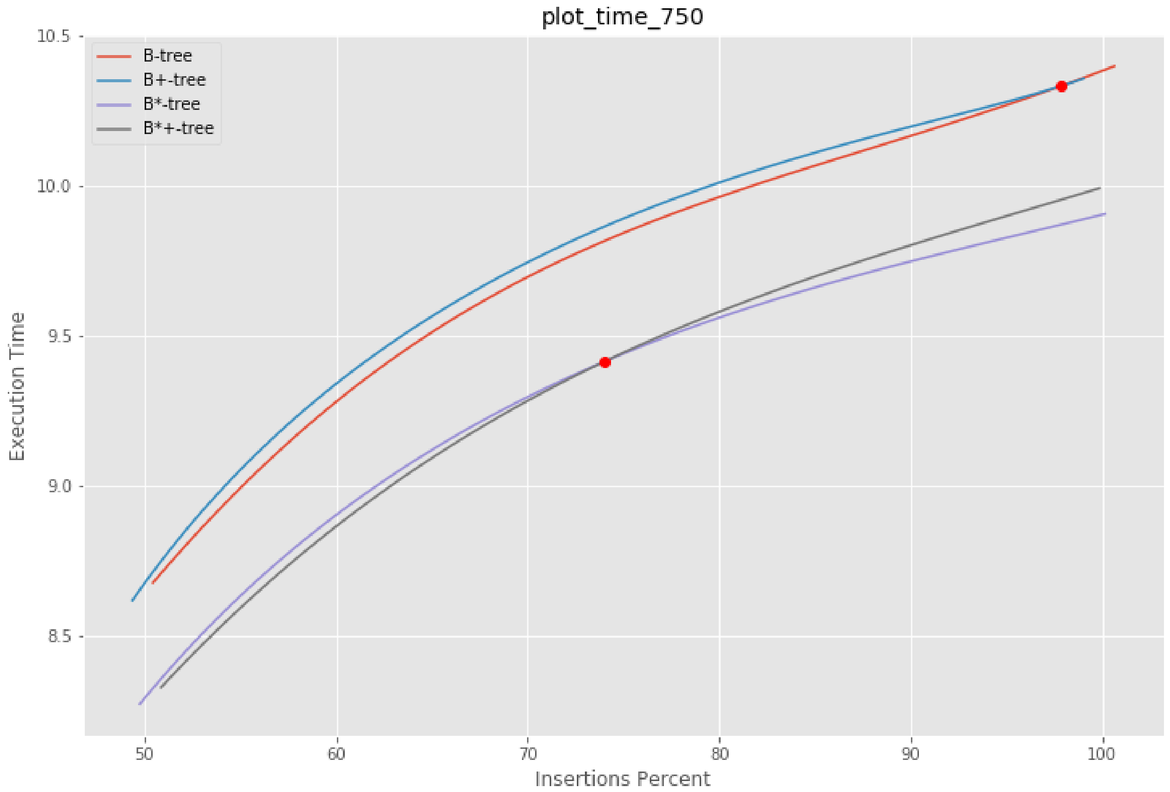
2. If the current number of the modifying operations (key insertions, key deletions) on the tree is less than 10 % of the current total number of the operations on the tree, then the algorithm stops, otherwise it goes to step 3.

3. If the current number of the key insertion operations is more than $p = 73.97\ \%$ of the total number of the modifying operations on the tree, then the algorithm selects the B*-tree as the index structure and goes to step 5, otherwise it goes to step 4.

4. The algorithm selects the B*+-tree as the index structure and goes to step 5.

5. If the new index structure has been selected at the steps 3 – 4, then the algorithm rebuilds the existing index structure replacing it by the new selected index structure and copies all the data stored in the previous index structure to the new index structure.

SELECTING THE INDEX STRUCTURE



TREE KEY STRUCTURE

IMPLEMENTATION DETAILS

- The C API for the B-tree modifications library is implemented as dynamically shared library using the *extern “C” { … }* C++ convention
- The SQLite extension registers *btrees_mods* module for creating the virtual tables
- A virtual table is any table created using such a module
- The extension also registers *btreesModsVisualize*, *btreesModsGetTreeOrder*, *btreesModsGetTreeType* functions

IMPLEMENTATION DETAILS

Method	Purpose
btreesModsCreate(sqlite3*, void*, int, const char* const*, sqlite3_vtab**, char**)	Creates a new table.
btreesModsUpdate(sqlite3_vtab*, int, sqlite3_value**, sqlite_int64*)	Inserts, deletes or updates a value of a row in the table.
btreesModsFilter(sqlite3_vtab_cursor*, int, const char*, int, sqlite3_value**)	Searches for a row in the table.

IMPLEMENTATION DETAILS

Method	Purpose
btreesModsVisualize(sqlite3_context*, int, sqlite3_value**)	Outputs the graphical representation of the table's index structure (tree) into the GraphViz DOT file.
btreesModsGetTreeOrder(sqlite3_context*, int, sqlite3_value**)	Outputs the order of the tree used as the table's index structure.
btreesModsGetTreeType(sqlite3_context*, int, sqlite3_value**)	Outputs the type of the tree (1 – B-tree, 2 – B^+-tree, 3 – B^*-tree, 4 – B^{*+}-tree) used as the table's index structure.

USAGE EXAMPLE

```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods(id INTEGER PRIMARY KEY, a INTEGER, b TEXT);
sqlite> INSERT INTO btt VALUES (4, 2, "ABC123");
sqlite> INSERT INTO btt VALUES (7, 3, "def");
sqlite> SELECT * FROM btt WHERE id = 4;
4|2|ABC123
sqlite> SELECT * FROM btt WHERE id = 7;
7|3|def
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|ABC123
7|3|def
sqlite> .tables
btrees_mods_idxinfo  btt                  btt_real
sqlite> SELECT * FROM btt_real;
4|2|ABC123
7|3|def
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|1|0|id|INTEGER|4|tree_18291557263097.btree
sqlite> DROP TABLE btt;
sqlite> .tables
btrees_mods_idxinfo
sqlite> SELECT * FROM btrees_mods_idxinfo;
sqlite> .exit
```

EXPERIMENT CONDUCTED USING THE SQLITE EXTENSION

Operation on the table	Total execution time (ms)	Mean execution time per row (ms)
Table creation	20	-
First 500 rows insertion	10301	20.6
Next 500 rows insertion	10322	20.6
1001st row insertion (including the B^+-tree into the B^*-tree rebuilding)	40	40
Next 499 rows insertion	9386	18.8
Last 500 rows insertion	9032	18.1

EXPERIMENT CONDUCTED USING THE SQLITE EXTENSION

Operation on the table	Total execution time (ms)	Mean execution time per row (ms)
First 500 rows deletion	11558	23.1
Next 500 rows deletion	10708	21.4
1001st row insertion (including the B^*-tree into the B^{*+}-tree rebuilding)	62	62
Next 499 rows deletion	9418	18.9
Last 500 rows deletion	8863	17.7
1000 rows insertion	18890	18.9
Next 5000 rows insertion (including the B^{*+}-tree into the B^*-tree rebuilding)	92395	18.5

EXPERIMENT CONDUCTED USING THE SQLITE EXTENSION

- The key insertion into the B*-tree was *faster* than into the B^{+}-tree
- The key deletion from the B^{*+}-tree was *faster* than from the B*-tree
- The key insertion into the B*-tree was *slightly faster* than into the B^{*+}-tree
- The search in a table took about 1 ms on all the B-tree modifications considered in this work

SUMMARY

- The developed B^{*+}-tree has smaller running time for keys insertion and deletion than B-tree, however it has greater memory usage
- The B-tree modifications library is connected to the SQLite as an extension using C API developed in this work
- The experiments were conducted using the developed library
- The algorithm of automatic selection of the suitable index structure is developed and implemented
- Experiments calculating performance of indexing table data were conducted using the developed SQLite extension

REFERENCES

[1] R. Bayer and E. McCreight, “Organization and maintenance of large ordered indexes,” *Acta Informatica*, vol. 1, no. 3, pp. 173 – 189, 1972.
[2] K. Pollari-Malmi. (2010). “B$^+$-trees,” *University of Helsinki*. [PDF paper]. Available: https://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf
[3] “B*-tree.” NIST Dictionary of Algorithms and Data Structures. Available: https://xlinux.nist.gov/dads/HTML/bstartree.html (accessed Dec. 24, 2018).
[4] A. Rigin, “On the Performance of Multiway Trees in the Problem of Structured Data Indexing,” (in Russian), coursework, Dept. Soft. Eng., HSE, Moscow, Russia, 2018.
[5] “Run-Time Loadable Extensions.” SQLite.org. Available: https://www.sqlite.org/loadext.html (accessed Jan. 20, 2019).

NATIONAL RESEARCH
UNIVERSITY

Thank you for your attention!

amrigin@edu.hse.ru
anton19979@yandex.ru
anton19979@yandex-team.ru