RankRTree

This package contains binary executables of the RankRTree (standalone version). Papers related to this release are:

> [1]. Jianzhong Qi, Yufei Tao, Yanchuan Chang, and Rui Zhang. "Theoretically optimal and empirically efficient R-trees with strong parallelizability." *Proceedings of the VLDB Endowment* 11.5 (2018): 621-634.
>
> [2]. Jianzhong Qi, Yufei Tao, Yanchuan Chang, and Rui Zhang. "Packing R-trees with space-filling curves: Theoretical optimality, empirical efficiency, and bulk-loading parallelizability." *ACM Transactions on Database Systems (TODS)* 45.3 (2020): 14:1-14:47.

Contact: Jianzhong Qi, jianzhong.qi@unimelb.edu.au

Build details

The source code is built on Ubuntu 20.04 (Linux kernel 5.4.0-33-generic) with g++ 8.4.0. Compiler optimization is not used. The released executable should be run in a similar environment. Our code is implemented based on the TPIE library which has been linked statically. Users do not need to install the TPIE library separately.

Released files

ascii2stream

Convert a text-based data file to a TPIE stream file, which is required when working with the TPIE library.

Usage: ./ascii2stream <r|logd|logid> input_file output_file

Argv:

1. <r|logd|logid>: r for H, HR, Z, LH, LHR and LZ; logd for LDZ and LDHR; logid for LIIZ and LIIHR (prepare input for different indexing algorithms: *Appendix A*)
2. input_file: text-based input data file. In an input data file, each line contains the bounding range (four float type numbers) and the ID (a unique integer) of an object, i.e., (x_lo, y_lo, x_hi, y_hi, ID).
3. output_file: TPIE stream data file

buildtree

Build an R-Tree from a TPIE stream file. After index building, the index tree will be dumped into files (named as ${input_stream}.${tree_type}.${blk}) on the hard disk.

Usage: ./buildtree input_stream fanout <H|HR|Z|LH|LHR|LZ|LDZ|LDHR|LIIZ|LIIHR>

Argv:

1. input_stream: TPIE stream data file
2. fanout: the fanout of the built R-tree
3. <H|HR|Z|LH|LHR|LZ|LDZ|LDHR|LIIZ|LIIHR>: tree type -- see *Appendix A*.

testrtree

Load an index file and run queries/insertions/deletions on the index.

Usage: ./testrtree index_stream <H|HR|Z|LH|LHR|LZ|LDZ|LDHR|LIIZ|LIIHR> <query|insert|delete> file

Argv:

1. index_stream: tree index file
2. <H|HR|Z|LH|LHR|LZ|LDZ|LDHR|LIIZ|LIIHR>: tree type -- see *Appendix A*.
3. <query|insert|delete>: type of test operations
4. file: file for the test data. For example, for insertions, this parameter specifies the file that contains the new spatial objects to be inserted.

data files

We provide the following sample datasets:

1. *dataset_uniform_1m*: a uniform dataset with 1 million spatial objects
2. *dataset_skew_9_1m*: a skewed (alpha=9) dataset with 1 million spatial objects
3. *insert_uniform_0.2m*: a uniform dataset with 0.2 million spatial objects for insertion

4. *insert_skew_9_0.2m*: a skewed (alpha=9) dataset with 0.2 million spatial objects for insertion
5. *delete_uniform_0.1m*: a uniform dataset with 0.1 million spatial objects for deletion
6. *delete_skew_9_0.1m*: a skewed (alpha=9) dataset with 0.1 million spatial objects for deletion
7. *query_rects_uniform_10*: uniform rectangle range queries with sizes being 0.1% of the data space
8. *query_rects_uniform_1*: uniform rectangle range queries with sizes being 0.01% of the data space

In dataset_*, insert_* and delete_* files, each line represents a spatial object as <x_lo, y_lo, x_hi, y_hi, id>. In query_* files, each line contains one MBR, which is described as <x_lo, y_lo, x_hi, y_hi>. Note that, spatial objects in deletion files do not necessarily exist in the index.

Examples

We first use *ascii2stream* to convert a text-based data file into a TPIE steam file. Only the data files that are used in index building require this conversion. After that, we can build the index from the TPIE steam data file and write index back into a file onto the hard disk. Then, we can run queries and updates on the built index, e.g., range queries, insertions and deletions. Insertions and deletions are executed in place, i.e., the index file will be modified. We give examples using Hilbert curves.

Convert text-based file to stream file

Create stream file for 'r' trees ('r' for H HR HZ LH LHR LZ. See *ascii2stream* usage above).

```
./ascii2stream r dataset_uniform_1m stream_uniform_1m
```

Create stream file for 'logd' trees.

```
./ascii2stream logd dataset_uniform_1m stream_uniform_1m_logd
```

Create stream file for 'logid' trees.

```
./ascii2stream logid dataset_uniform_1m stream_uniform_1m_logid
```

Index building

Build a *HilbertRank-RTree (HR)* on data *stream_uniform_1m*, and its corresponding index structure is stored in *stream_uniform_1m.hrrtree.blk* and *stream_uniform_1m.hrrtree.info*. Note that its fanout (102) is calculated in advance. The hard disk assumes 4096-byte sectors and we use 40 bytes for each entry in a tree node, so the maximum fanout of a node is 102 (ref. Section 6.1 in [1]). An input fanout larger than the maximum fanout allowed will be replaced by the maximum fanout.

```
./buildtree stream_uniform_1m 102 HR
```

Build a *Hilbert-RTree (H)* on data *stream_uniform_1m*.

```
./buildtree stream_uniform_1m 102 H
```

Build a *LogR-Tree with Hilbert curve on rank space* on *stream_uniform_1m_logd* (update improved index, see paper [2]).

```
./buildtree stream_uniform_1m_logd 85 LDHR
```

Build a *LogR*-Tree with Hilbert curve on rank space* on *stream_uniform_1m_logid* (update improved index, see paper [2]).

```
./buildtree stream_uniform_1m_logid 72 LIIHR
```

Range queries

Run range queries on HR, H, LDHR and LIIHR indices respectively.

```
./testrtree stream_uniform_1m.hrrtree HR query query_rects_uniform_10
./testrtree stream_uniform_1m.hrtree H query query_rects_uniform_10
./testrtree stream_uniform_1m_logd.ldhrrtree LDHR query query_rects_uniform_10
./testrtree stream_uniform_1m_logid.liihrrtree LIIHR query query_rects_uniform_10
```

Insertions

Insert spatial objects stored in *insert_uniform_0.2m* into existing indices respectively. After insertions, we can run range queries on the indices as well.

```
./testrtree stream_uniform_1m.hrrtree HR insert insert_uniform_0.2m
./testrtree stream_uniform_1m.hrtree H insert insert_uniform_0.2m
./testrtree stream_uniform_1m_logd.ldhrrtree LDHR insert insert_uniform_0.2m
./testrtree stream_uniform_1m_logid.liihrrtree LIIHR insert insert_uniform_0.2m
```

Deletions

Delete spatial objects stored in *delete_uniform_0.1m* from existing indices respectively. After deletions, we can run range queries on the indices as well.

```
./testrtree stream_uniform_1m.hrrtree HR delete delete_uniform_0.1m
./testrtree stream_uniform_1m.hrtree H delete delete_uniform_0.1m
./testrtree stream_uniform_1m_logd.ldhrrtree LDHR delete delete_uniform_0.1m
./testrtree stream_uniform_1m_logid.liihrrtree LIIHR delete delete_uniform_0.1m
```

Outputs: Every executable file outputs 'Time' and 'Total IO' to the terminal, where 'Time' indicates the running time (in seconds) of the whole process and 'Total IO' indicates the sum of read and write I/Os. The range queries also output the number of spatial objects that are overlapped with the query rectangles.

Example: Here, we give an example output of range queries. We execute range queries on *stream_uniform_1m.hrrtree*. Range queries are executed based on the rectangle MBRs provided in file *query_rects_uniform_10*. After all queries are run, we output the results to the terminal. 'Time' and 'Total IO' indicate the running time and the sum of read and write I/Os respectively. 'Total output' shows the number of spatial objects that satisfy the query predicate.

```
➜ ./testrtree stream_uniform_1m.hrrtree HR query query_rects_uniform_10

Looking for info file stream_uniform_1m.hrrtree.info
Existing block collection.
Loading HR tree from file stream_uniform_1m.hrrtree
Command : query
Extra file : query_rects_uniform_10
Time : 0.01
Total IO : 2182
Total output : 98933
query succ!
```

Appendix

Appendix A: Index type abbreviation

Abbr.	Index type	Appears in
H	R-Tree + Hilbert curve	[1]
HR	R-Tree + Hilbert curve + rank space	[1]
Z	R-Tree + Z curve + rank space	[1]
LH	LR-Tree + Hilbert curve	[1,2]
LHR	LR-Tree + Hilbert curve + rank space	[1,2]
LZ	LR-Tree + Z curve + rank space	[1,2]
LDHR	LogR-Tree + Hilbert curve + rank space	[1,2]

Abbr.	Index type	Appears in
LDZ	LogR-Tree + Z curve + rank space	[1,2]
LIIHR	LogR*-Tree + Hilbert curve + rank space	[1,2]
LIIZ	LogR*-Tree + Z curve + rank space	[1,2]

H, HR and Z are studied in [Ref. 1] to evaluate the effectiveness of rank space on spatial indices. Further, we apply the logarithmic structure to rank space-based spatial indices [Ref. 2], which correspond to tree types starting with letter 'L'. More details about the LR-Tree, LogR-Tree and LogR*Tree can be found in [Ref. 2].