Topic: Implementing a disk-resident spatial index structure (Quadtree)

This project gives an in depth understanding of file management and index structures, in the context of spatially extended DBMS. Spatial DBMS are database systems that are optimized to store and query data that represents objects defined in a geometric space (e.g., a point in a Cartesian coordinate system, geometry of Zurich on a world map). For spatial DBMS specific kind of indexes (called spatial indexes) are used that can handle two (or more) dimensional data. The most commonly used spatial indexes are: Quadtree, R-Tree, Spatial hash index. The goal of this project is to implement a disk-resident spatial index. Namely, a Quadtree. A quadtree is an adaptation of a binary tree to represent two-dimensional data, proposed by R. A. Finkel and J. L. Bentley [1]. This projects consists of 3 tasks.

1. In order to implement a disk-resident index, first a memory **data buffer** should be implemented that manages files in terms of blocks and memory pages. This data buffer consists of an array of memory pages (e.g., an array of 20 memory pages each one able to store a block of 1KB data). Using this structure a file block should first be loaded in memory in order to access its data, change it, and then write it back to the disk. This strategy, called blocking, minimizes the number of required disk I/Os and it is a standard technique used in DBMS. The data buffer should implement at least the following functions:

- **void init():** Initialization function
- **void createFile(char* filename):** Creates an empty file.
- **int openFile(char* filename):** Opens an existing file. Returns a file descriptor
- **void closeFile(int fileDescriptor):** Closes an open file.
- **void deleteFile(int fileDescriptor):** Deletes a closed file.
- **int getBlockCounter(int fileDescriptor):** Returns the number of blocks of an open file.
- **void* readBlock(int fileDescriptor, int blockNo):** Loads a block in a memory page.
- **void writeBlock(int fileDescriptor, int blockNo, void* blockBuffer):** Writes a block to disk.
- **void* addBlock(int fileDescriptor):** Adds a new empty block in a file and load it in a memory page.

When there is no any free space in memory, buffer manager should perform one replacement policy (e.g., LRU or MRU) and replace an block in memory with a new one from disk.

2. The index structure should be implemented using the memory buffer. The **quadtree** index implements the insertion algorithm and the region search algorithm described in [1]. The quadtree index should implement at least the following functions:

- **void init():** Initialization function.
- **void createIndex(char* filename):** Creates a empty index file.
- **int openIndex(char* filename):** Opens an existing index file. Returns a file descriptor.
- **void closeIndex(int fileDescriptor):** Closes an open index file.
- **void deleteIndex(int fileDescriptor):** Deletes a closed index file.
- **void insertEntry(int fileDescriptor, void* tuple):** Inserts a new record in an index file.
- **int startRegionSearch(int fileDescriptor, Point southWestPoint, Point northEastPoint):** start a region search scan on an index. Returns a searchDescriptor.
- **Point findNextEntry(int searchDescriptor):** Returns the next entry according to an already opened region search.
- **closeRegionSearch(int searchDescriptor):** Closes an opened region search.

3. The implementation should be demonstrated by performing indexing and searching on given data

that demonstrates good and bad scenarios of using a Quadtree.

The preferred programming language for this project is C, but C++ or Java can also be used.

References

[1] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, no. 1, pp. 1–9, 1974.