GIS: PR Quadtree Stored on Disk

In this project, you will re-implement the Geographic Information System for storing point data using a PR Quadtree. However, this time the PR Quadtree will reside on disk. A buffer pool (using the LRU replacement strategy) will mediate access to the disk file, and a memory manager (similar to the one implemented for Project 1) will decide where to store the PR Quadtree nodes. Note that the BST from Project 2 will remain unchanged in this project (that is, it will stay in main memory, you need not convert it to be stored on disk.)

Input and Output:

Your program will be named "Bindisk", and it will take three command-line arguments. The first is the name of the command input file. The second is the number of buffers in the buffer pool, and will be in the range 1–20. The third is the size for a block in the file (which therefore determines the buffer size as well).

The input for this project will consist of a series of commands (some with associated parameters, separated by spaces), one command for each line. The commands will be read from the file given in command line parameter 1, and the output from the command will be written to standard output. The format and interpretation for the commands will be identical to Project 2, with the following exceptions.

In addition to listing the nodes of the PR Quadtree, the "debug" command will also list the following: (1) Block ID's of the blocks currently contained in the bufferpool in order from most recently to least least recently used; and (2) a listing of the memory manager's free blocks, in order of their occurrence in the freeblock list.

The output for the PR quadtree in the "debug" command will be a little different from that used in Project 2. An internal node should be represented by parentheses (and) around the contents of that node. A leaf node should print its contents, then the "bar" or "pipe symbol" `|`. An empty leaf node should print as its contents an asterisk `*`. A non-empty leaf node should print as its contents the records it contains. Each record should be printed as "X,Y,NAME:" The Project 2 sample data file at a certain point produces the following debug output:

```
I0,0,Floyd5001,5012,Blacksburg5001,6213,Blacksburg|E5001,8414,Christiansburg|16383,
16383,Virginia_Beach|
```

In Project 4, that same tree would look like this:

```
(0,0,Floyd:5001,5012,Blacksburg:5001,6213,Blacksburg:|*|5001,8414,Christiansburg:|
16383,16383,Virginia_Beach:|)
```

You will need to create and maintain a disk file which the buffer pool is acting as the intermediary for. The name of this disk file must be "p4bin.dat". After completing all commands in the input file, all unwritten blocks in the buffer pool should be written to disk, and the disk file should be closed, **not deleted**.

Note that "Block ID" simply refers to the block number, starting with 0. Thus, if the block size is 1024 bytes, then bytes 0-1023 are in Block 0, bytes 1024-2047 are in Block 1, and so on.

Implementation:

The implementation rules for the PR Quadtree from Project 2 are still in place. That is, all operations that traverse or descend the PR Quadtree structure **MUST** be implemented recursively, and the PR Quadtree nodes **MUST** be implemented with separate classes for the internal nodes and the leaf nodes, both of which inherit from some base node type.

The PR Quadtree itself will be stored on disk, not in main memory. This is the primary difference from Project 2. The nodes will be of variable length, and where a node is stored on disk will be determined by a memory manager. When implementing the PR Quadtree nodes, access to a node (by calling the node's "getchild" method) will mean a request to the memory manager to return the node contents from the memory pool, and creation or alteration of a node will require writing to the memory pool. From the point of view of the memory manager and the buffer pool, communications are in the form of variable-length "messages" that must be stored.

The memory manager should follow the same definition as in Project 1. In particular, the location for placing the next message within the memory pool should be determined using worst fit; a list of the free blocks may be maintained in main memory, and adjacent free blocks should be merged together.

Initially, the memory pool (and the file) should have length 0. Whenever a request is made to the memory manager that it cannot fulfill with existing free blocks, the size of the memory pool should grow by one (or more) disk blocks to meet the request.

The memory manager will be managing data residing in the memory pool, and this memory pool will reside on disk, but the memory manager does not actually have direct access to the disk. All disk access is through the buffer pool. Thus, the flow of control for a node access will be that the Bintree will request a "message" from the memory manager via a handle, the memory manager will ask the buffer pool for the data at a physical location, the buffer pool will hand the contents of the "message" to the memory manager, which will in turn hand the contents of the "message" back to the Bintree.

The layout of the PR Quadtree node messages sent to the memory manager **MUST** be as follows. (Note that, as in Project 1, the memory manager will actually add to the message the length of the message as stored in the memory pool.) Internal nodes will store 17 bytes: a one-byte field used to distinguish internal from leaf nodes, followed by four 4-byte fields to store handles for the four children. Leaf nodes that contain one or more city records will store the following fields: a one-byte field used to distinguish internal from leaf nodes; a one-byte field to indicate the number of city records stored; and then a series of city records that each contain two 4-byte fields for the city x- and y-coordinates, respectively, and a 4-byte handle for the city name. The city name will be sent to the memory manager as a separate message. This message will contain the length of the name in the first byte, followed by the characters for the name. The null character at the end of the city name string should not be included in the message, nor stored on disk. Empty nodes will be represented by storing in the empty node's parent a handle value that is recognized by the node class "getchild" method as representing an empty leaf node (the flyweight). The flyweight may be actually represented as a physical node on disk, or you may use a special handle value that is simply recognized as the flywight.