Programming Assignment 3
Due Monday December 1st, 2014 11:59pm
Multidimensional Indexing (20 points)

The scope of this project is Multidimensional Indexing and querying. You will implement a multidimensional access method, namely an R-tree and then you will design and implement search algorithms on top of your R-tree.

Your code should include:

1. Implementation of a two-dimensional R-tree structure. For this assignment you do not have to implement update, insert or delete operations. Your implementation should store the data points on disk and retrieve disk pages as needed during query processing.
2. Initialization of the R-tree with the data points attached with the project. The data set *D* contains 30,000 points with values in both *x* and *y* dimensions contained in [0,10000]. To make the project more interesting and increase the size of the tree, **you are asked to add a "description" attribute to each tuple, of size 500 char**. You can fill it with random characters. Tuples should be stored in Alternative 1 and the R-tree should only index on the *x* and *y* dimensions. You can implement a bulk-loading technique, as discussed in class (recommended) or implement the insert operation and insert points one by one.
3. Search query on the R-tree: you should support **simple point queries and range queries** on the R-tree. Your implementation should prompt the user for query values. Range queries will consist of entering the coordinates of two corners of the corresponding bounding box. A point query is either just the point coordinates or a range query with both corner points being equal.

To make testing and grading more efficient, you should separate the data initialization process from the querying process so that it is easy to run several queries on your R-tree once it is loaded with data points.

Requirements Details:

- You do not have to parameterize your code for variable *d* (dimension). Supporting the test data set (i.e., two dimensions) is enough.
- You have to design the search routine with an interface accepting the query values and reporting the data results as well as cost in terms of disk access.
- Both data pages and index pages should be stored on disk and retrieved on demand as needed. Assume a 4K (i.e. 4096 bytes) data page. You can ignore the impact of the cache and the memory buffer in your cost computation.
- C++ or Java are both acceptable
- You should provide a README file with code documentation and compilation instruction. Your code should compile on the cluster machines.

Graduate Student required implementation or Extra Credit for Undergraduate Students:

- Implement a search routine that identifies the point maximizing a function *f* in D, the data space. *f* is of the form *ax+by*, where *a* and *b* are positive real numbers such that *a+b=1*, *a>0* and *b>0*. *a* and *b* are input values provided by the user.

Hint: The bounding box that maximizes *f* does not necessarily contain the point that maximizes *f*.

Extra Credit for Graduate Students:

- Implement R-tree insertion and deletion. You can make your own choces as to the heuristics used, but you need to explain and justify them in the README.

FAQ

1. Do we have to implement disk accesses?
    - No, you can keep everything in memory. You will have to keep track of the number of pages traversed to estimate the IO cost.
2. What value should we set BITS_PER_DIM to in the getHilbertValue method?
    - x1 and x2 are the dimensions of the data point for which you want to compute the hilbert value.
    - BITS_PER_DIM represents how many bits from each dimension (x and y) you are using to compute the hilbert value (it can be fewer than the bits of x and y)
3. If we are coding in Java, how do we evaluate the size of a pointer in determining how many sets of regions/pointers can be fit on a 4 KB page?
    - A java object reference is 32bits. Since you are not keeping actual references, you can also keep a pageID as an int instead of a pointer.
4. Are there duplicates in the data?
    - Yes.
5. Are you providing tests?
    - No.