Project 2 (Index Library) Sample Application Guide
May Zhai
Bill Ge

Note: Our BTree accepts three generic types of primary keys. A key class hierarchy was used in order to represent the various keys that could be used. These generic keys were chosen specifically since they were most relevant to our set of data.

- GenericKey
  - Used when the user chooses a seemingly "random" tuple to be used as their primary key
  - Composed of an array of objects, which is an array of the fields the user chose
  - Comparison between the objects is done by simply comparing the toString() of each object
- RankKey
  - Composed of rank of the song and year of the song
  - Comparison between objects is primarily determined by the billboardYear and secondarily by the rank of the song
- SongKey
  - Composed of songName, artistName, and billboardYear
  - Comparison between the objects is done through a String comparison of the concatenation of the above data

Jars for sample applications can be found in the Index Library directory

a) Sample Application 1 - BuildTreeSampleApplication
This application tries to demonstrate that our BTree can be built from any .csv file with a format similar to our original csv file. It allows the user to specify a primary key to be used for the BTree. Our application will also check that the given key is valid. If the key is valid, then the tree will be built. Once the tree is built, the application will print out the tree with a breadth-first-search traversal. It will also print the leaf nodes of the tree in order.

b) Sample Application 2 – ConcurrencySampleApplication
This application demonstrates that our BTree and datastore do not crash when multiple processes/threads are interacting with it. The application starts with our sample data of about 500 unique data entries. There also exists a dataset of 1 million entries. Four threads will be generated that run concurrently in order to insert the million entries into the BTree. The application is proven to be correct when it is found that there are a total of 1 million 500 hundred entries at the end of runtime.

We were able to allow for concurrent use of our application by implementing a mutex for our BTree as well as Datastore. We ensure that only 1 thread is allowed to access the critical parts of our application at a time.

b) Sample Application 3 - FunctionalitySampleApplication

For this application, it was assumed that the application would be using RankKey. This application demonstrates the following functions that can be executed in our BTree:

- Insert
    - For the purposes of ease and convenience of the user, random samples of data are generated to be put into the BTree. If the user tries to insert a duplicate entry, nothing will occur. If the data is unique, both the BTree and datastore will be updated with the new data.
- Search
    - Given a primary key composed of rank and year, the application searches through the BTree to check if the tree contains the given primary key. If an entry with the given key is found, then the corresponding data to the key will be returned.
- Delete
    - Given a primary key composed of rank and year, the application tries and deletes the entry with the primary key specified. If the primary key is not found, nothing will happen. It if it found, it will be removed.
- SearchBetween
    - Range search of the tree: A lower bound and an upper bound to be searched for will be specified by the user. The tree will then search for any keys that are within the range specified by the lower and upper bound. The search is inclusive of the boundaries. The data corresponding to all the primary keys found will be printed.

Caching is implemented in this particular sample application through the fetching of the data. The BTree leaf nodes only contain data containing information about the offset into the datastore in order to find the data that corresponds to the given offset. Whenever this is done, the data found is cached. This way, if a particular primary key is used multiple times, the application will reference the cached value rather than reading it again from disk.