Homework 2
Extendable Hashing

Due April 8

Homework 2

- Implement the DBTable and the ExtHash classes and a driver class to test the classes. The DBTable class implements the basic parts of a database table and the ExtHash class implements an extendable hash index. Shown on the following slides are the methods for the DBTable and the ExtHash classes.
- The DBTable must use the ExtHash to find rows in the table. Every DBTable method will use ExtHash.

DBTable

- The rows in the DBTable will include an integer key and one or more fixed length strings.
- The DBTable maintains a list of free slots (space in the file previously used for rows that have been removed)
- When a new row is inserted the DBTable should reuse a free slot if one is available. Otherwise the new row is inserted at the end of the file.

Homework 2 (DBTable Example)

	Addr	Contents		
numOtherFields	0	2		
Length 1	4	10		
Length 2	8	20		
Free	12	276		
	20	50	Anton	Chekhov
	84	468		
	148	10	Vladimir	Nabokov
	212	60	Alonzo	Church
	276	84		
	340	20	Mark	Twain
	404	70	Gottlob	Frege
	468	0		
	532	40	Hannah	Arendt
	568	30	George	Eliot

Hash Values

Key	Hash Value
10	1000
20	0111
30	1110
40	1001
50	0101
60	1100
70	0010

Bucket File (does not include overflow information)

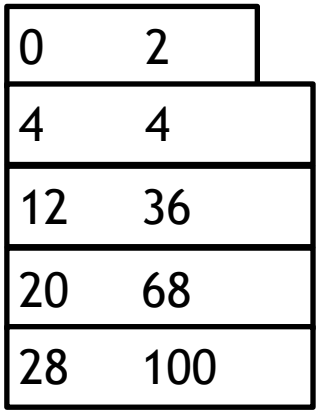
Bucket Size

0	2			
4	2	2	60 10	212 148
36	2	2	40 50	532 20
68	2	2	70 30	404 568
100	2	1	20	340

Addr nBits nKeys Keys Table Addrs

Directory

Hash Bits



DBTable

```
public class DBTable {

    RandomAccessFile rows; //the file that stores the rows in the table
    long free; //head of the free list space for rows
    int numOtherFields;
    int otherFieldLengths[];
    //add other instance variables as needed
```

DBTable

```
private class Row {
      private int keyField;
      private char otherFields[][];
      /*
      Each row consists of unique key and one or more character array fields.

      Each character array field is a fixed length field (for example 10
            characters).

     Each field can have a different length.

      Fields are padded with null characters so a field with a length of
            of x characters always uses space for x characters.
      */

      //Constructors and other Row methods

}
```

DBTable

```
public DBTable(String filename, int fL[], int bsize ) {
/*
      Use this constructor to create a new DBTable.

      filename is the name of the file used to store the table
      fL is the lengths of the otherFields
      fL.length indicates how many other fields are part of the row
      bsize is the bucket size used by the hash index

      A ExtHash object must be created for the key field in the table

      If a file with name filename exists, the file should be deleted before the
      new file is created.
*/

}
```

DBTable

```
public DBTable(String filename) {
//Use this constructor to open an existing DBTable
}

public boolean insert(int key, char fields[][]) {
//PRE: the length of each row in fields matches the expected length
/*
      If a row with the key is not in the table, the row is added and the method
      returns true otherwise the row is not added and the method returns false.

      The method must use the hash index to determine if a row with the key
       exists.

      If the row is added the key is also added into the hash index.
*/

}
```

DBTable

```
public boolean remove(int key) {
/*
      If a row with the key is in the table it is removed and true is returned
      otherwise false is returned.

      The method must use the hash index to determine if a row with the key
       exists.

      If the row is deleted the key must be deleted from the hash index
*/

}
```

DBTable

```
    public LinkedList<String> search(int key) {
    /*
          If a row with the key is found in the table return a list of the other fields in
          the row.

          The string values in the list should not include the null characters used for
          padding.

          If a row with the key is not found return an empty list

          The method must use the hash index index to determine if a row with the
          key exists
    */

    public void close() {
    //close the DBTable. The table should not be used after it is closed
    }
}
```

ExtHash

```
public class ExtHash {

    RandomAccessFile buckets;
    RandomAccessFile directory;
    int bucketSize;
    int directoryBIts;   //indicates how many bits of the hash function are
                             used by the directory
    //add instance variables as needed.

    private class Bucket {
        private int bucketBits; //the number of hash function bits used
                                    by this bucket
        private int count; // the number of keys are in the bucket
        private int keys[];
        private long rowAddrs[];
        //overflow bucket?

        //constructors and other method
```

ExtHash

```
public ExtHash(String filename, int bsize {
//bsize is the bucket size.
//creates a new hash index
//the filename is the name of the file that contains the table rows
//the directory file should be named filename+”dir”
//the bucket file should be named filename+”buckets”
//if any of the files exists the should be deleted before new ones are made

}

public ExtHash(String filename) {
//open an existing hash index
//the associated directory file is named filename+”dir”
//the associated bucket file is named filename+”buckets”
//both files should already exists when this method is used


}
```

ExtHash

```
public boolean insert(int key, long addr) {
/*
      If  key is not a duplicate add key to the hash index
      addr is the address of the row that contains the key
      return true if the key is added
      return false if the key is a duplicate
*/
}

public long remove(int key) {
/*
      If  the key is in the hash index, remove the key and return the address of
       the row.
      return 0 if the key is not found in the hash index
*/
}
```

ExtHash

```
    public long search(int k) {
    /*
          If the key is found return the address of the row with the key
          otherwise return 0
    */

    }
    public int hash(int key) {
    //return the hash value

    }

    public void close() {
    //close the hash index. The tree should not be accessed after close is called
    }
}
```

Homework 2: Other Issues

- Inserting a new key might require modifying the directory
- Removing a key might require modifying the directory
- Because of the possibility of bucket overflow your code should be able to manage space for overflow buckets.
- Undergraduates can work in pairs but graduate students must work alone

Homework 2

- You will demonstrate your program to me. I will give you a driver to use for the demonstration but you must develop your own driver to test your implementation before you demo.
- Plan to demonstrate during the week of April 8