ASSIGNMENT 3 REPORT

Álvaro Castillo García
Pablo Ernesto Soëtard

Universidad Autónoma de Madrid

Escuela Politécnica Superior

Ingeniería Informática bilingüe, group 1251

Data Structures

INTRODUCTION

In this assignment we where asked to create a primitive DBMS, which could handle table functions such as creation, opening, closing, insertion… And insert data of type INT, STR, LLNG, or DBL.

We also had to create a way to index the data that was stored in our tables, thus we programmed the functions of index.c that can handle and index file to access the data from the tables.

TABLES

The file “tables.c” contains all the functions to deal with table insertion and retrieval of data from tables. First we had to decide how the structure of our tables file would be, we decided to include a header at the start of that file which will contain the following information: number of columns, type of data of each column and number of rows.

Then we had to decide what would our table data structure contain, and we arrived to this result:

Where fp stands for the File Pointer, columns for the number of columns, row for the number of rows for number of rows, types is an array of type_t which contains at each index the datatype corresponding to that column, fpos holds the first position of the file and last_row is an general type array which contains in each index a general pointer to each column corresponding to the last row read.

Regarding the length, we opted to store it at the beginning of each row, then when we want to read a record we just need to allocate the amount of memory that we read from the initial 8 bits of the record, that will be the length of the data stored there.

While creating the functions of table.c we went through several problems. One of them was that we where able to correctly write the table with our custom header and data to a file but we were not able to retrieve the information from the file. The reason for this issue was a pointer that was shifted by one bit, hence we started reading from the wrong address and thus the data types were not interpreted correctly and we were getting wrong values.

We had also some valgrind errors while allocating space for the buffer to read the file and storing it in the last_row variable. We were allocating space in a variable already allocated and we were not freeing correctly the memory, thus getting memory leaks. Therefore, we came up with a solution, we just had to free last_row[0] before allocating space for a new buffer.

After that we implemented the other data types that were asked, LLNG (long long integers) and DBL (double precision numbers). That was not a difficult task, since all our pointer were general pointers (void *) and we just needed to do a few modifications to the code. One of them was to include those data types to the type.c and type.h functions.

INDEXES

The file “index.c” contains all the functions to deal with integer indexes insertion and retrieval from files. First we had to decide how the structure of our index file would be, we decided to include a header at the start of that file which will contain the following information: number of records and type of record. Then all the record’s keys, the number of records of that key and those records will be saved.

Then we had to decide what would our index and record data structures contain, and we arrived to this result:

```
struct index_ {
    int n_records;
    irecord_t **recs;
    FILE *fp;
    type_t type;
};
```

```
struct irecord_ {
    void * key;
    int n_records;
    long *rec;
};
```

On the one hand the index data structure holds the number of records, an array of pointer to those records, the file pointer and the type of the records.
On the other hand the irecord data structure holds the key of that record group, the number of records that are assigned to that key and an array that has the positions of that records.

After that we started to implement the rest of the functions and when we came across the index_put function we realized that we had to implement a binary_search like function, so we looked up on the assignment and we implemented a binary search algorithm based on the one that was given and one from our algorithmic class, the result is the following:

```c
int binary(irecord_t **array, void *key, int n, type_t type) {
  int low = 0;
  int high = n-1;
  int middle;
  if(array==NULL || key==NULL || n<1){
    return -1;
  }
  while(low <= high) {
    middle = (low+high) >> 1;
    if (value_cmp(type, array[middle]->key, (void*) key) == 0){
      return middle;
    }
    else if(value_cmp(type, array[middle]->key, (void*) key) > 0){
      high = middle - 1;
    }
    else{
      low = middle + 1;
    }
  }
  return -(middle+1); // Key not found.
}
```

We had some problems while inserting records in the structure as they had to be ordered. We were inserting them in the wrong place because index_put did not work as we expected and it was pushing the elements in the position that binary returns. When a record with a new key is inserted we were not checking if we had to move all the elements until the position marked by binary or that position + 1. Having fixed that, everything in index.c worked perfectly.

CONCLUSION

With this assignment we have learned how DBMS work more deeply and understood their structure and data flows. We have implemented a rudimentary DBMS which is able to handle tables and relations between records. Regarding the optional part of the assignment, it would not be a very difficult task since we have implemented our record data structure in a way that the key is a general pointer (void *) so we would only have to change some functions headers and modify a couple of lines of code to get it to work with STR key types, but due to the lack of time we were not able to implement that feature.