System Design Document

Micro DB

COMS 4995 : DESIGN USING C++

Group Members:

Lulu, Chengrui, Shubham

Introduction

Database frameworks in C++ are not new, we noticed that there are many C++-based SQL databases frameworks like SQLAPI++, SOCI, etc. What we observed was that these have complex instruction set and needs other dependent libraries to function properly. This makes it very difficult for users to use such frameworks.

Our target users are developers who want a key-value pair datastore. We wanted to provide a solution that is intuitive and easy to use. We decided to build a self-contained library that can support storing key-value pairs in the filesystem.

System Overview

When the library runs it needs a database to attach itself to.

1. If the IDX and DAT files with provided name is already present in the directory then it opens it else it creates a set of IDX and DAT files to store the information
    a. IDX file is used to keep track of all keys entered into the DAT file along with index positions
2. We use BKDR(Brian Kernighan and Dennis Ritchie) hash function for maintaining the IDX file
3. DAT file contains the corresponding value for each key written sequentially
    a. Values are stored in form of a vector of string
4. Exceptions are handled

List of functionalities supported by the library:

1. INSERT: to add a key-value pair into the database
2. FIND: to get the value of any key passed
3. DELETE: To delete a key-value pair from the database
4. QUERY: to search through the database based on the condition of key or value at a particular index
5. IMPORT: import data from a CSV or TSV file into the database

File List:

File Name	Purpose
CMakeLists.txt	CMake file to build the project
Manual	Instruction Manual
data.tsv	Test dataset from IMDB
db.cpp	Function definitions
db.h	Function declarations
exception.h	Exception declarations
idx.h	Index file declarations
main.cpp	Application file which provides command line interface to test
test.cpp	Test application
tutorial.cpp	Tutorial file

Design Considerations

Assumptions and Dependencies

1. Since we do not know the data type, we had to keep all data types as strings.
2. At a time only one user is trying to use this database

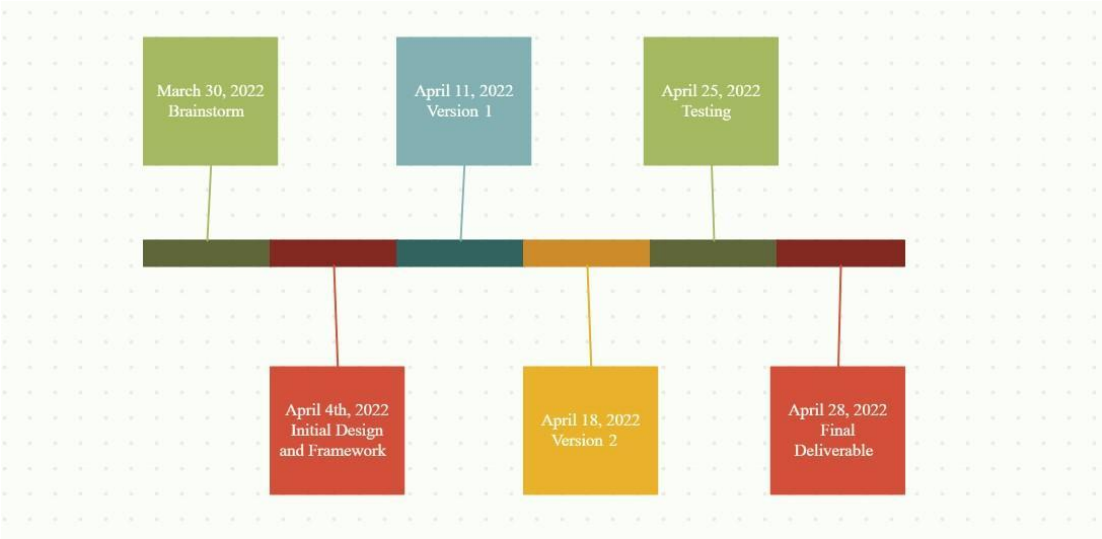
General Constraints

1. By default, the max key length supported is 20(can be modified in idx.h file)
2. By default, the size of the Hash table used is 20000003(can be modified in db.h)

Development Methods

Project development with Agile model of development.

Milestones and Timeline



Brainstorm: March 30th to 4th April

Used design frameworks to brainstorm and narrowed down the set of functionalities that the project will support based on the needs of users and their pain points.

Initial Design and Framework: April 4th to April 11th

Designed the hashing function and storage architecture

Version 1: April 4th to April 11th

Developed the insert, find, delete and create database functionality

Version 2: April 11th to April 18th

Added support to insert values into the database on file uploads. Added support for query functionality

Testing: April 18th to April 25th

Added performance test application and tested with datasets from IMDB

Final Deliverables: April 25th to April 28th

Created tutorial, demo, and sample docs.

System Architecture

- Library implemented as a class. All supported functions are defined inside this class "DB". The declaration of DB class can be found in the db.h file

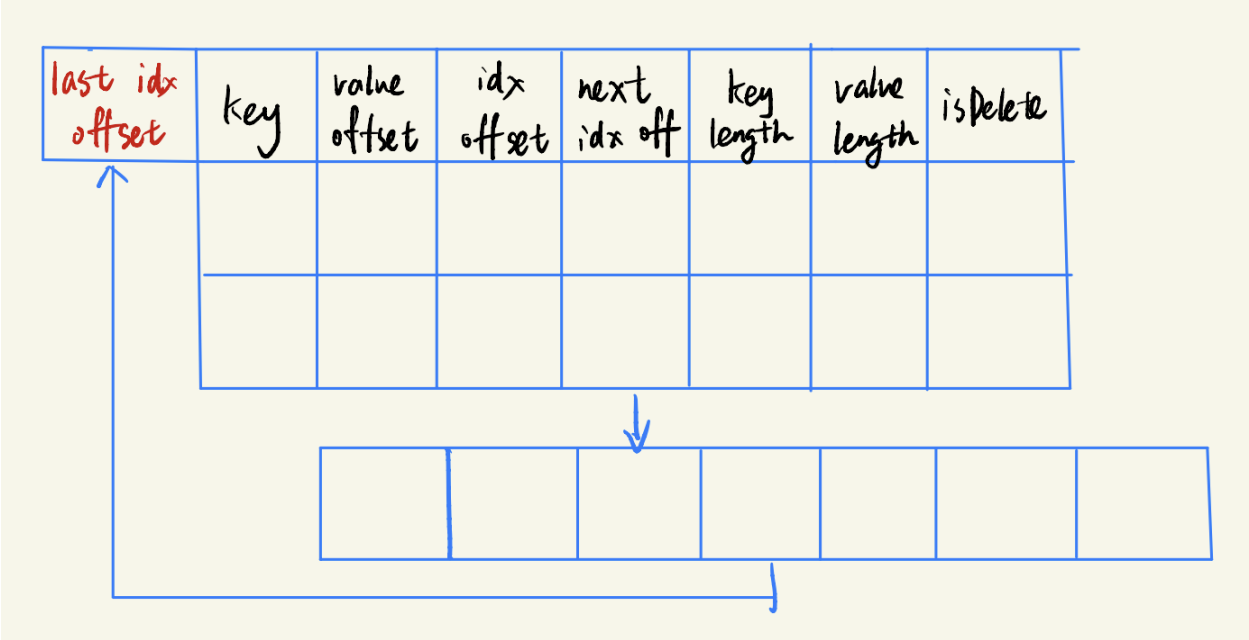
```
class DB {
public:
    DB(string fileName);                                    //initialize
db, create files
    unsigned int hash(const char* key);                 //hash function
    bool open();
            //open db files. Load last_idx_off and last_value_off
    bool close();                                            //close db
files. store last_idx_off and last_value_off.
    bool insert_file(string file_name, int num);
//insert data with csv/tsv files, num is the max rows
    vector<string> find(const char* key);                   //look for the
value. return null if not found
    bool del(char* key);
//delete key
    bool insert(char* key, vector<string> val);           //insert
key-value. if already exists, return false.
    bool replace(char* key, vector<string> val);          //update value
    vector<string> query(int k, string val);               //use given
value to search for key.
    void clear();
//delete db files.
private:
    Idx* find_key(const char*key);                  //find corresponding
idx struct of key
    unsigned int last_idx_off;                      //total offset of idx
file
    unsigned int last_dat_off;                      //total offset of data
file
    string fileName;                                //input csv/tsv file
name
    string idxName, datName;                        //name of idx file and
data file
    FILE* fp1;                                      //idx file stream
    FILE* fp2;                                       //data file stream
};
```

- Definition of each of these functions can be accessed in db.cpp file
- Other major component includes the test application, which benchmarks the performance for all the functions and does sanity check of the database.

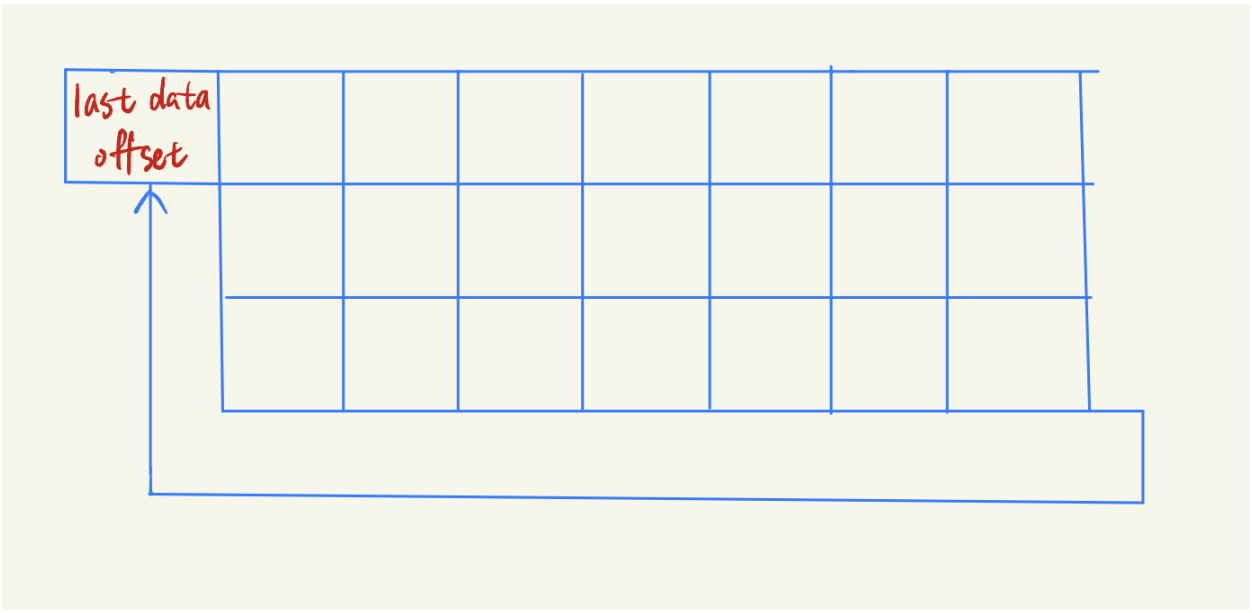
Detailed System Design

The library stores the data using IDX and DAT file

- Writing data into index file
- When key value conflicts happens, append the new idx struct to the end of the file.
- Set the next_idx_off of the last conflict key's idx struct as the offset of the new idx struct. Thus, the hash table is organised as a Linked list.
- When a key's idx struct is deleted, we do not truly delete it. Only set "isDelete" to be true. We can directly insert a new idx struct to the position where "isDelete" == true.

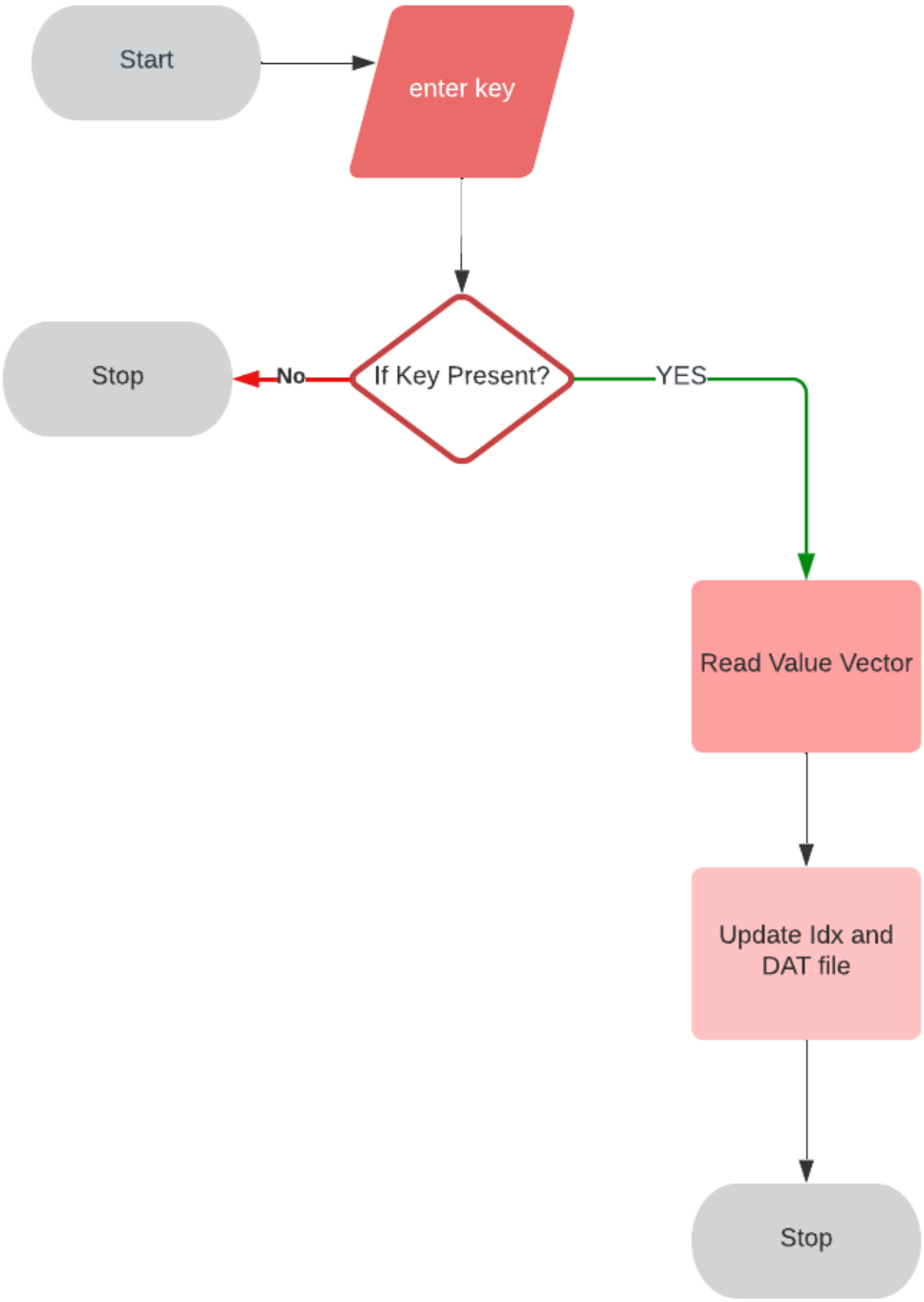


- Storing data into the DAT file
- **The value data of each key is stored sequentially in the dat file.**
- On read, the library reads based on the stored index value offset obtained from Idx file pointer

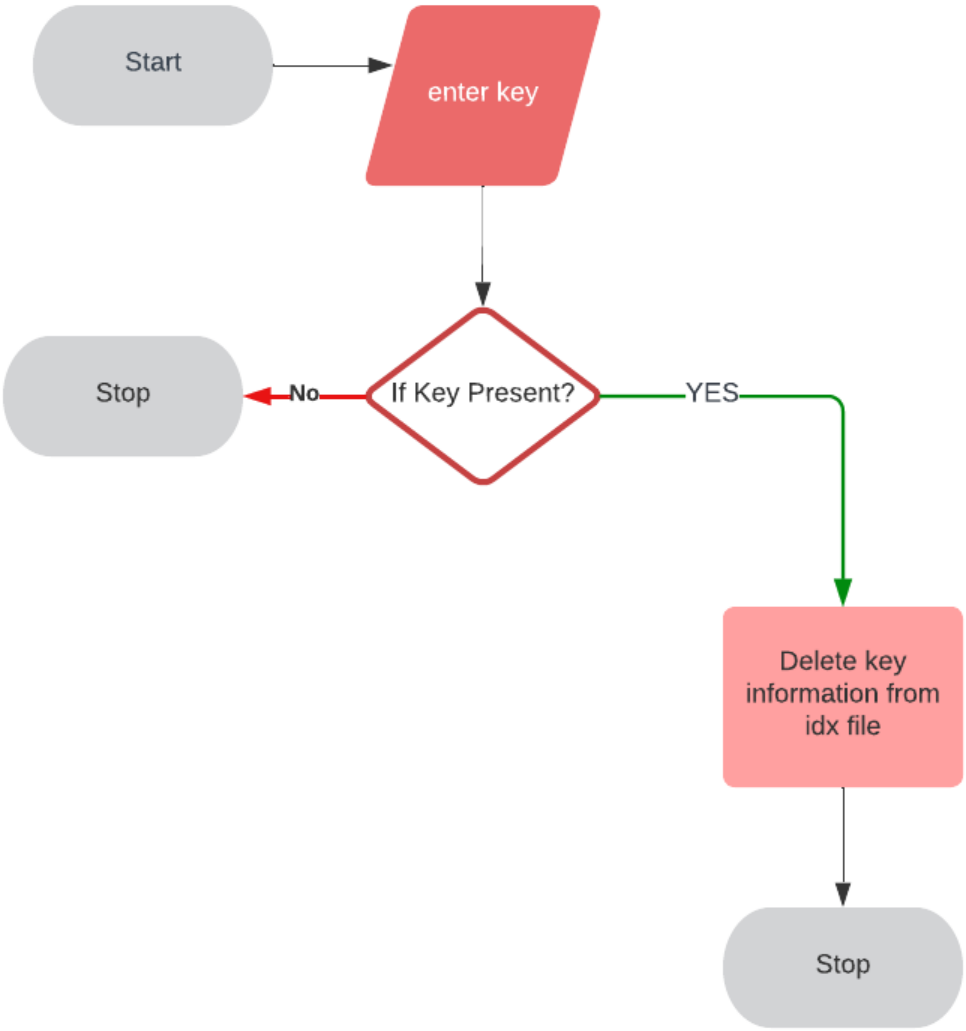

last data
offset

Use Cases

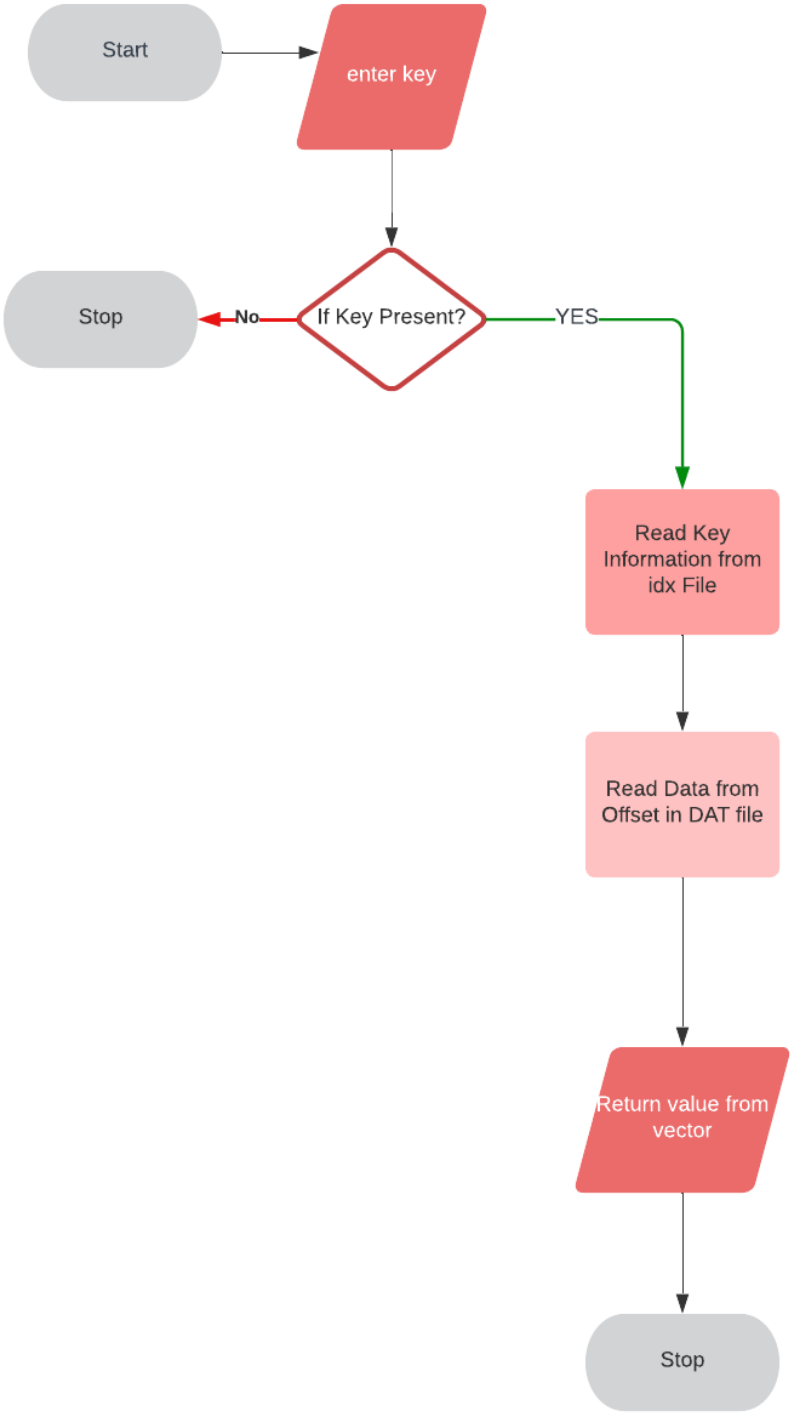
1. **Insert:**



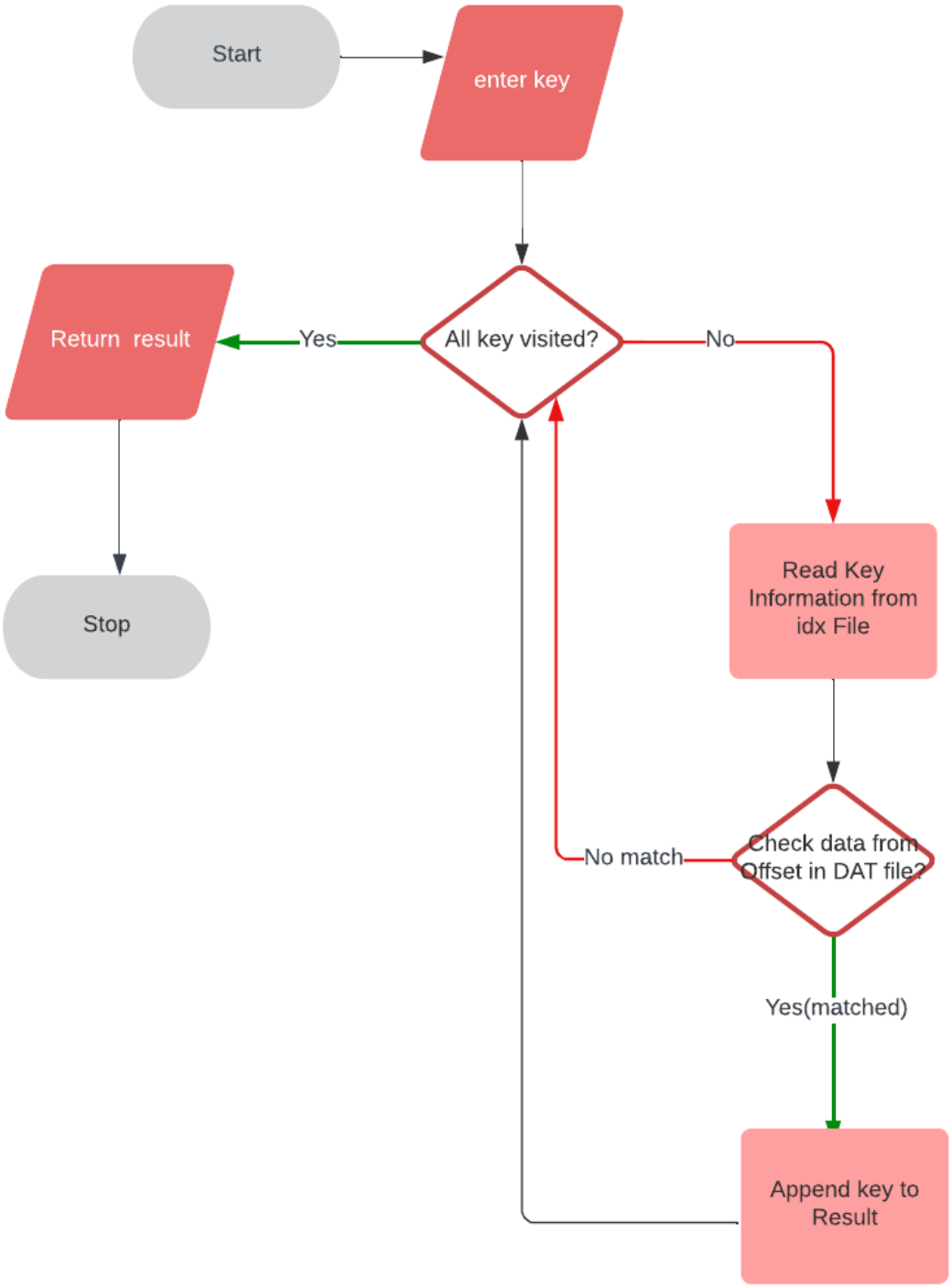
Delete:


Start
enter key
If Key Present?
No
Stop
YES
Delete key information from idx file
Stop

Find:


Start
enter key
If Key Present?
No
Stop
YES
Read Key Information from idx File
Read Data from Offset in DAT file
Return value from vector
Stop

Query:


Start
enter key
All key visited?
Yes
No
Return result
Stop
Read Key Information from idx File
Check data from Offset in DAT file?
No match
Yes(matched)
Append key to Result

Test Performance Log

- We tested with two mechanisms
    - First, through the command-line interface
        - User can perform operations on any database file through interface
        - Sample log from command-line interface Link
    - Second, through test_performance application
        - Test app benchmarks all operations for different size of the input file and prints log
        - Test checks correctness of the database by generating random string and testing read/write values based on it.
        - Sample test_performance log file Link.
- Databases Used for testing:
    - https://datasets.imdbws.com/
        - Title.ratings.tsv.gz
        - title.akas.tsv.gz