Documentation of the Dictionary-based Data Compressing System

Jingmao You

2020-09-05

This is the documentation of the Dictionary-based Data Compressing System. In this documentation I will explain how to use the system, how to test different functionalites by using the main class, and I will also explain each class. The Leaf1.cpp and Leaf1.h are the files that contain all the classes and their implementation. It could be built directly and test different functionalites in the main function. In the final directory, each file contains the related class or implementation. It requires the usage of make file to build the system. Use make clean could clean all the generated files.

```
make
./dictionary
make clean
```

1 Functionality of the system

In order to build the dictionary, we need to have a file which coutains all the neccessary values. The values in the file could have replicates and could be unsorted. We need to read the file from the disk and build the dictionary based on it. Then we could build the prefix encoding index and the decoding index. After that, we could build the suffix dictionary and the suffix encoding index. If we would like to run the tests, then we could just the encoding method or the decoding method. Or else we could generate a file that contain the whole dictionary and indexes.

```
vector<string> strs;
strs = read_csv("sorted result.csv");
Leaves leaves1;
//create the dictionary
int load_result = leaves1.bulkLoad(strs);
//create the prefix encoding index
int encoding = leaves1.encodingIndexcreation();
//create the suffix dictionary
leaves1.bulkLoadSuffix(strs);
//create suffix encoding index
leaves1.encodingIndexcreationSuffix();
//create decoding index
leaves1.decodingIndexcreation();
//generate the file
leaves1.generateFile("test");
//read the file
Leaves leaves2;
leaves2.readFile("test");
//test the serial version of encoding
//exact matching
leaves2.serial_task_encode(strs,0);
leaves2.concurrent_task_encode(strs,0);
```

```
//prefix matching
vector<string> str1;
str1.push_back(a);
leaves2.serial_task_encode(str1,1);
leaves2.concurrent_task_encode(str1,1);
//suffix matching
leaves2.serial_task_encode(str1,2);
leaves2.concurrent_task_encode(str1,2);
//decoding
vector<int> codes;
for(int i=0; i<strs.size(); i++){
  codes.push_back(i);
}
leaves2.serial_task_decode(codes);
leaves2.concurrent_task_decode(codes);
```

2 Class Leaf

This class contains all the parameters in order to define the size of one leaf. It also has the method to print all the values in the leaf, printStrings. BulkLookup is used to do the encoding exact matching in the leaf, bulkLookup_prefix is used to do the encoding prefix matching in the leaf, bulkLookup_code is ued to do the decoding matching in the leaf. Binary seach is used for the binary search in the offset vectors, and sequantial search is used to do the search in a certain interval.

3 Class Leaves

This class is the dictionary class, a dictionary is modeled as the a list of leaves. Some parameters are important here. Fullindex is used to store the prefix encoding index, index_decode is used to store the decoding index, and index_encode_suffix is used to store the suffix encoding index. Decode_result is used to store all the decoding results, and the encode_result is used to store all the encoding results. The bulkLoad method is used to create a new dictionary based on the input strs, and the bulkLoadSuffix method is used to create a suffix dictionary.

4 Class Node

This class is used to create temporary nodes of encoding index.

5 Class Tree

This class is used to create a list to connect all the temporary nodes during the encoding index building process.

6 Class decodingindexNode

This class is about the node object in the decoding index. We could change the predefined size of decoding index node. The indexpointer is used to point to a node. The insertKey method is used to insert a key into the node.

7 Class encodingindexNode

This class is about the node object in the encoding index. We could change the predefined size of encoding index node.