Tutorial - 2

Name:- Sairaj P Rajput

Division:- TY-A

Roll no – 331054

PRN - 22110171

Aim - Investigating and evaluating the Lempel-Ziv-Welch (LZW) text compression algorithm.

Required Knowladge:

- the fundamentals of data structures like linked lists, dictionaries, and arrays.
- familiarity with binary representation and string manipulation.
- Programming knowledge in languages such as Python, C++, or Java.
- Knowledge of Compression Principles

Theory:-

- Abraham Lempel, Jacob Ziv, and Terry Welch developed the lossless data compression algorithm known as Lempel-Ziv-Welch (LZW) compression. It is extensively utilized in applications like network protocols (like Unix compress) and file compression (like GIF images).
- Dictionary-Based Compression: An algorithm for dictionary-based compression is called LZW compression. In order for it to function, a dictionary of substrings found in the input data is constructed, and repeated substrings are replaced with shorter codes.

- Dynamic Dictionary: LZW compression dynamically creates and updates the dictionary during the compression process, in contrast to static dictionary-based techniques. As a result, it can adjust to the unique patterns and attributes of the incoming data.

- Variable-Length Codes: Substrings are given variable-length codes by LZW compression; shorter codes are given to substrings that occur more frequently. This makes it possible to encode the input data efficiently because shorter codes are used to represent common substrings.

Algorithm:

- Initialize a dictionary with all possible single-character strings.
- Scan the input text from left to right.
- At each step, find the longest prefix in the dictionary that matches the current input sequence.
- Output the code corresponding to the matched prefix.
- Add the new sequence (prefix + next character) to the dictionary.
- Repeat steps 3-5 until the entire input text is processed.

Example:

Examine the following text input: "abababcabababcabcdeabcde" The following steps are involved in the LZW compression process:

• Fill a dictionary to the brim with every possible string of one character.

• Go from left to right through the input text.

• Find the longest prefix in the dictionary that corresponds to the current input sequence at each step.

• Produce the code that matches the prefix that was matched.

• Update the dictionary with the new sequence (prefix + next character).

• Continue from steps 3-5 until all of the input text has been handled. The following dictionary and output are produced by the LZW compression process for the provided example text:

Dictionary:

0: a

1: b

2: ab

3: ba

4: c

5: abc

6: cab

7: abcab

8: cde

Output:

0 1 2 4 6 7 8

Thus, the compressed representation of the input text "abababcabababcabcdeabcde" would be "0124678".

Conclusion:

In conclusion, by substituting shorter codes for frequently occurring substrings, the LZW compression algorithm provides a potent way to compress text data.

Code:

```
#include<bits/stdc++.h>

using namespace std;
unordered_map<string,int> dictionary;
map<int,string> dict;
vector<int> Encode;

int main(){
// enter the project name
// cout << "I am batman " << endl;

string input = "wabbabwabbabwabbabwabbabw";
string current = "";

for(int i = 0; i < input.size(); i ++){
```

```
char c = input[i];
string temp = c + current;
if(dictionary.find(temp) != dictionary.end()){
// i got the string
current = temp;
}
else{
Encode.push_back(dictionary[current]);
dictionary[temp] = dictionary.size();
current = string(1,c);
}
}

if(!current.empty()){
dictionary[current] = dictionary.size();
}

for(auto it : dictionary){
dict[it.second] = it.first;
}

cout << "Encoded data : " << endl;
for(int m : Encode){
cout << m << " ";
}
cout << endl;

string decoded;
for(int i = 0; i < Encode.size(); i++) {
int code = Encode[i];
if(dict.find(code) != dict.end()) {
decoded += dict[code];
}
}

cout << "Decoded data : " << endl;
cout << decoded << endl;


return 0;
}
```

Output:

```
Encoded data :
0 1 0 0 0 4 2 6 8 4 7 0 9 13
Decoded data :
bbbbbaawbbwbabaabbbawbab
```