Carlos
Ben
Mitchell
Pavan
CS 454 Final Project

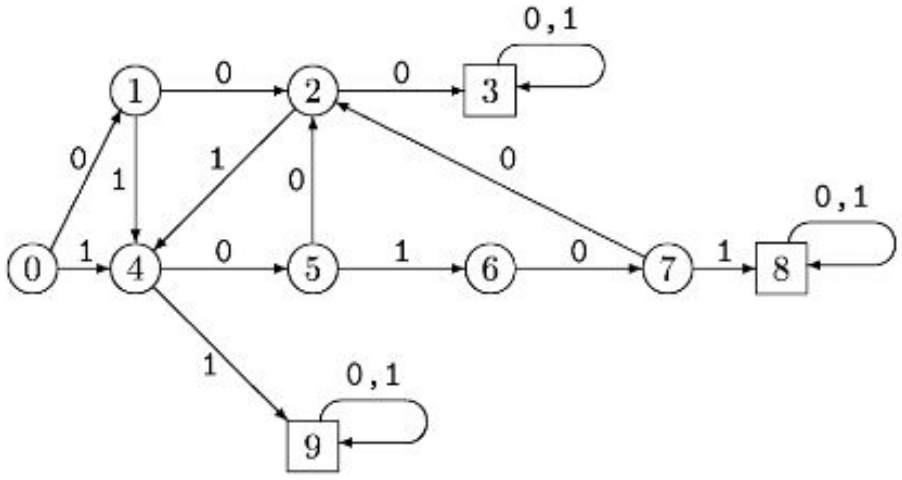
Data Compression Using Finite Automaton

Problem statement

Our goal was to compress an input string specified by the user using finite automatons and forbidden words within anti dictionaries. We needed to build an automaton that would accept forbidden words and do this efficiently in linear time and space. For even greater efficiency, we also intended to build an automaton that would accept minimal forbidden words, so that it could be transformed into a transducer that would compress and decompress the input string in a single pass.

Main ideas behind solution

The idea was to create an algorithm to build an anti dictionary made up of forbidden words from a given string of arbitrary length. Then we needed to implement a compression algorithm as described in Data Compression using Algorithms by M. Crochemore using the anti dictionary provided by the first function. From there we wrote decompression algorithms to reverse the compression algorithm and return data to an uncompressed format.



Shown above is an example of a finite automata that accepts the forbidden words of the set {11, 000, 1010}. The squares represent “sink” states, ones that have accepted one of the strings in the set.

General steps of solution:

Take input string

Create DFA “Fdawg” that accepts all substrings of input string

From Fdawg, Create MFW that accepts all minimal forbidden words of the input string

Use Fdawg to compress input string

Implementation details/Time complexity Analysis

The Minimal Forbidden Word algorithm takes an input string and returns all the minimal forbidden words of that string. In our implementation, we built a DFA that accepts all the minimal forbidden words. The algorithm iterates through states of the DFA using BFS. Forbidden refers to substrings that are not contained in the input string and minimal refers to the smallest possible instances of such forbidden words. The complexity of the algorithm to build the automaton that accepts the minimal forbidden words is O(N) in time and O(N) in space. Using the automaton, we built a finite-state transducer to handle the encode and decode algorithms.

Sample test cases with outputs

```
> testAndPrint('test', 8)
String used:
test
Original length (of binary string): 31
Compressed binary string: 1111
Compressed length: 4
Anti-dict length: 231
Total encoded length (with AD): 251
Compression value WITHOUT AD: 7.75
Compression value WITH AD: 0.12350597609561753
Compression time: 0.08894777297973633
Decoded string result:
test
Decoded bits match original bits: True
Decompress time: 0.001957416534423828
```

Shown here are a few sample test cases where we test the initial, slower, version of our compression algorithm. The string to be compressed is the first argument passed to the function and the other argument passed to ‘testAndPrint’ is

```
> testAndPrint('computer', 24)
String used:
computer
Original length (of binary string): 63
Compressed binary string: 110001
Compressed length: 6
Anti-dict length: 613
Total encoded length (with AD): 637
Compression value WITHOUT AD: 10.5
Compression value WITH AD: 0.0989010989010989
Compression time: 0.13821673393249512
Decoded string result:
computer
Decoded bits match original bits: True
Decompress time: 0.015933990478515625
```

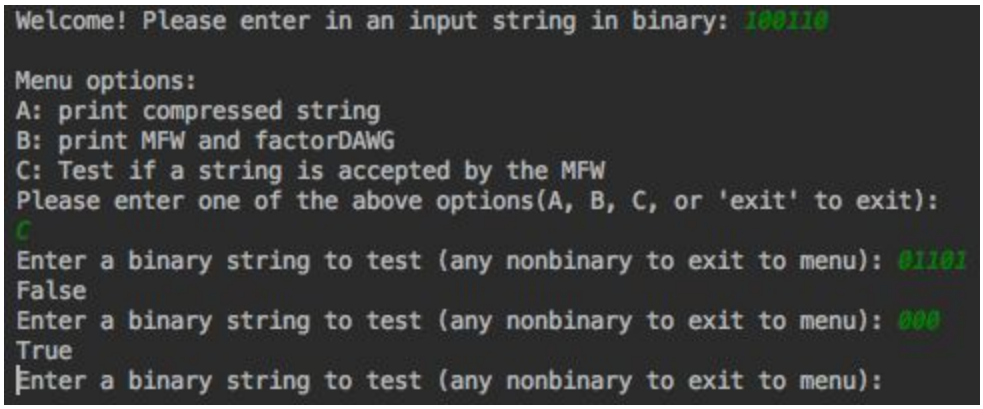
maxWordLength which indicates how many bits we want to look at, at a time. Our test function displays the original length of our string in binary then the compressed version of our string along with its new length. Once the string is compressed the new length of the string is shown in "Total encoded length" and as you can see it's greater than the original length. The reason for this is because the length of the antidictionary is so long but without it, the compressed string would be "compression value without AD" which is much smaller. Then we finish the test by decoding the string and checking to make sure that the decoded string as well as the decoded bits match our original bits.

Output of minimal forbidden words and factorDAWG for binary input string

```
Welcome! Please enter in an input string in binary: 11010

Menu options:
A: print compressed string
B: print MFW and factorDAWG
C: Test if a string is accepted by the MFW
Please enter one of the above options(A, B, C, or 'exit' to exit):
B
Each pair represents a state. First number in each pair is the next state on input 0, second number is next state on input 1. -1 means there is no edge for this input.
MFW :  [[3, 1], [6, 2], [3, 7], [-1, 4], [5, 7], [-1, 7], [7, 4], [-2, -2]]
factorDAWG :  [[3, 1], [6, 2], [3, -1], [-1, 4], [5, -1], [-1, -1], [-1, 4]]
```

Output of whether test input string is accepted by the MFW



To test, run main.py, then follow the instructions in the main menu.

All developed in Python 3.5

Sources:

http://www.cs.au.dk/~cstorm/students/Christensen_May2016.pdf

http://poincare.matf.bg.ac.rs/~azrael/antidict.pdf__