Mihir Jham
CS251
Project 3 Report
04/14/2014

CS251 – Project 3 Report

1. The brute force algorithm goes through every possible combination that the ALPHABET[] array can create. Depending on the value of C, the algorithm starts from 'a' in every position and goes to '5' in every position and it encrypts each possibility. If the encrypted possibility equals the encrypted password that is input, it displays it.

> The complexity is 2^{B*C}, where B is the number of bits per character and C is the length of the password.
>
> The following tests were run on data.cs.purdue.edu, and all values are in seconds.

Char	Brute
4	0.500000
5	28.250000
6	1199.380000(or around 20minutes)

2. The Symbol Table solution first divides the input table into 2 halves and stores the power set of all the possible sums that can be computed in 2 different Vectors (firstHalf and secondHalf) that hold KeyValue objects. KeyValue has a char[] key and a Key object, which is the corresponding value. The secondHalf vector is then sorted using the standard std::sort() function available in the <algorithm> library. The firstHalf vector is then traversed sequentially, subtracting each possibility from the encrypted password that is input and the difference is searched for in the secondHalf vector using binary search.

I chose to create a custom class because it was easier to implement the binary search and sorting algorithms. Also, vector seemed to be the most apt as it was the quickest to implement and I could dynamically grow or shrink it. It is most similar to a Binary Search Symbol Table.

Complexity:

Computing all possible sums for each half: $2^{N/2}$ where N = B * C, where B is the number of bits per character and C is the length of the password.

Sorting the vector: NlogN (assumed)

Searching: $2^{N/2} * \log(2^{N/2}) = (N/2)*2^{N/2}$

Mihir Jham
CS251
Project 3 Report
04/14/2014

Total complexity: $2(2^{N/2}) + NlogN + (N/2)*2^{N/2}$

The following tests were run on data.cs.purdue.edu, and all values are in seconds.

Char	Symbol
5	1.100000
6	23.570000
8	2098.380000(around 35 minutes)
10	Took very long(got disconnected from data)

3. The idea I used to generate the unique tables was that fact that I needed to end up with a linearly independent set. I managed to do that using the key class that was provided. Depending on the length of the password, I start of with a key which ends with a ‘b’ and the rest are all ‘a’. As there are B*C values that are needed, I add the key to itself B*C times and I end up with a linearly independent table. I then shuffle this to help eradicate backtracking. The reasons why multiple passwords encrypt to the same value is because the tables provided were not linearly independent and at least one of the values can be formed using a combination of the other table rows.