CS741 Assignment 2 - Q2 Report

Neel Aryan Gupta 180050067
Pulkit Agrawal 180050081
Tathagat Verma 180050111

NOTE - The code has been written in C++ inside the file q2.cpp. To run this code, O2 flag is recommended for better performance.

```
g++ -O2 q2.cpp -o q2
./q2 < input.txt
```

Time complexity - $O(S * 2^{3S} + T * 2^{2N} * (N / S))$

The first term comes from precalculating all the biases. Explanation for this term has been given in Q1 where S = 8 for AES Sbox.

The second term is the complexity of filling the DP states where we iterate over the number of stages/rounds (= T) and over all pairs of (mask, mask') where each mask lies in [0, 2^N) and N/S is the number of Sboxes in each block, on which we have to iterate to calculate biases using Piling-Up Lemma.

Algorithm - A dynamic programming approach has been used to find the maximum bias subgraph. The DP state is given by dp[i][mask] which gives the maximum bias for the situation - at the input of i'th round, what would be the maximum bias if only the bits set in 'mask' were to be used. dp[i][mask] is updated from dp[i + 1][mask'] where **all** combinations of mask and mask' are used for calculation. Finally, the maximum value of dp[0][i] over all values i in the range(0, 2^N) is the maximum bias we can obtain. This approach is correct because bias of **every** path is considered in this approach.

Intuitively, one can also see this approach as a bottom-up version of BFS where states are given by different masks at each round i.

Note that for computation of encryption involving T rounds, answers (maximum bias) are available for all rounds i (<= T) inside the dp itself.

Max bias for i'th round = maximum value of dp[T - i][j], where j lies in [0, 2^N)

Final result reports the bias having the highest absolute value. The value is shown along with its sign.

Output - The output is exactly as given in the problem statement. For clarity, a visual representation of the subgraph is also shown. For example,

```
P5, P7, K05, K07, K13, K29, C9
Bias = 9/32

**************
0000|0101|0000
----+----+----
0000|0001|0000

0001|0000|0000
----+----+----
0010|0000|0000

0000|0000|0100
----+----+----
0000|0000|0100

0000|0000|0100
**************
```

Here the 1 bits show the subgraph chosen for the maximum bias. The ---+---+--- refers to S-box computations (Eg. S(0101)=0001). The empty line represents the permutation. (Eg P(0000 0001 0000) = 0001 0000 0000)

Claim 2.1 - *"A greedy strategy will always work."*

This claim is **incorrect**. The reason is that we are not considering all possible paths in the graph. It is possible that one path which takes up a higher bias initially, later finds paths with lower biases and hence obtains a lower overall bias. Hence it is necessary to check all paths, which is done in the DP solution.

We have implemented the greedy strategy (q2_greedy.cpp). This input gives different maximum biases in the greedy and DP strategies:

```
4
6
3 2 4 5 1 0
3
3 0 6 1 5 7 4 2
```

DP output:

```
P3, P5, K03, K05, K11, K24, K30,
C0
Bias = 1/8

*******
000|101
---+---
000|010

010|000
---+---
001|000

000|010
---+---
000|001

100|000
---+---
100|000

100|000
*******
```

Greedy output:

```
P2, P5, K02, K05, K10, K12, K13, K14,
K15, K20, K22, K32, C2
Bias = 1/16

*******
001|001
---+---
111|101

101|111
---+---
010|001

101|000
---+---
010|000

001|000
---+---
001|000

001|000
*******
```