Expt 3

David Daniels Div A – C

2023300038

In this experiment, a rolling hash has been used. It is basically like a sliding window of size of pattern to check. If the output of hash funtion ~~and that~~ of window and pattern is same, then there might be a hit. We must then ~~manually~~ ~~che~~ check all characters to verify. It achieves $O(n+m)$ time complexity in best/average funtion if the hash funtion is good, but $O(nm)$ if hash funtion is too simple. in rolling hash, we remove contribution of outgoing char and add that of incoming character.

This is much better than precomputing the entire hash and storing in array.

To implement this, we must first convert ~~on~~ pattern and first m characters of text into numeric hash value using base d (ASCII 256) and ~~pr~~ large number q. Each character gives it's positional weight to hash.

The rolling hash step is,

txt val = (d × (txt_val − text [i] * hash)
+ text [i + m]) % q

if txt_val = ~~ts~~ pat_val → do brute force checking

The Graph of ~~the~~ Time Execution shows ~~a~~ a few large spikes in the data. This may be due to the expected string being in the end of string or many spurious matches to check. Overall as size of pattern to check increases, the time required increases.