DAY 17:

ASSIGNMENT 2:

Task 4: Rabin-Karp Substring Search

Implement the Rabin-Karp algorithm for substring search using a rolling hash. Discuss the impact of hash collisions on the algorithm's performance and how to handle them.

ANSWER:

```
import java.util.ArrayList;
import java.util.List;


public class RabinKarpAlgorithm {


   public static void main(String[] args) {
      String text = "ABABDABACDABABCABAB";
      String pattern = "ABAB";
      searchPattern(text, pattern);
   }


   public static void searchPattern(String text, String pattern) {
      int textLength = text.length();
      int patternLength = pattern.length();
      long patternHash = calculateHash(pattern);
      long textHash = calculateHash(text.substring(0, patternLength));


      for (int i = 0; i <= textLength - patternLength; i++) {
         if (textHash == patternHash && text.substring(i, i + patternLength).equals(pattern)) {
            System.out.println("Pattern found at index " + i);
         }
         if (i < textLength - patternLength) {
            textHash = recalculateHash(textHash, text.charAt(i), text.charAt(i + patternLength),
patternLength);
```

```
            }
        }
    }

    public static long calculateHash(String str) {
        final int prime = 31;
        long hash = 0;
        for (int i = 0; i < str.length(); i++) {
            hash = hash * prime + str.charAt(i);
        }
        return hash;
    }

    public static long recalculateHash(long oldHash, char oldChar, char newChar, int patternLength) {
        final int prime = 31;
        long newHash = oldHash - oldChar * (long)Math.pow(prime, patternLength - 1);
        newHash = newHash * prime + newChar;
        return newHash;
    }
}
```

Explanation of the algorithm:

Hashing: This implementation uses polynomial rolling hashing to compute hash values for both the pattern and substrings of the text.

Rolling Hash: As it iterates through the text, it recalculates the hash value of the next substring using a rolling hash function. This ensures that the hash value can be updated efficiently without recomputing the entire hash.

Comparison: It compares the hash values of the pattern and the current substring. If they match, it verifies the equality of the substrings character by character.

Impact of Hash Collisions:

Performance: Hash collisions can impact the performance of the Rabin-Karp algorithm, especially if they occur frequently. Collisions may lead to false positives, requiring additional character comparisons to confirm the match, which can increase the runtime of the algorithm.

Handling Collisions: Proper handling of hash collisions involves performing additional character comparisons whenever hash values match. These comparisons ensure that the pattern matches the substring correctly, mitigating the impact of collisions on the algorithm's accuracy.

Hash Function Selection: Choosing a good hash function is crucial to minimize the likelihood of collisions. A well-designed hash function distributes hash values evenly across different substrings, reducing the chances of collisions and improving the algorithm's performance.