**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech. Sem V

**Course: Design and Analysis of Algorithms**

**List of Experiments**

w.e.f. 1st Jul 2020

**Faculty:** Prof. Abhay Kolhe

LAB Manual

**Experiment No.10**

|  |  |
| --- | --- |
| Roll No : B032 | Name: Naman Garg |
| Class : B.Tech CS-B | Batch : B3 |
| Date of Experiment: 13.10.20 | Date of Submission:14.10.20 |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

# Name: Naman Garg

# Roll No: B032

# AIM: Implementation of String Matching Algorithm - Rabin Karp algorithm

from math import \*

s=input("Enter string : ").upper()                                                                          #s,pat => String and the pattern to match

pat=input("Enter pattern : ").upper()

prime=int(input("Enter prime number : "))                                                                   #prime => Prime number of the hashing function

hash\_pat=int(sum([(ord(ch)-64)\*pow(prime,i) for i,ch in enumerate(pat)]))                                   #hash\_pat => Hash value of pattern

print(f"Hash value of pattern : {hash\_pat}")

match\_index=[]                                                                                      #match\_index => Stores the index where match is found

pat\_len=len(pat)

s\_len=len(s)

hash\_s=0

for i in range(0,s\_len-pat\_len+1):

    comp=1

    if(i==0):

        hash\_s = int(sum([(ord(ch) - 64) \* pow(prime, i) for i, ch in enumerate(s[i:i+pat\_len])]))          # First time assigning hash value

        print(f"Current substring : {s[i:i + pat\_len]} => Hash value of substring = {hash\_s} => ", end="")

    else:

        hash\_s = int((hash\_s-(ord(s[i-1])-64))/prime) +int((ord(s[i+pat\_len-1])-64)\*pow(prime,pat\_len-1))   # Rolling hash function

        print(f"Current substring : {s[i:i + pat\_len]} => Hash value of substring = {hash\_s} => ", end="")

    if(hash\_s==hash\_pat):

        print("Hash values matched! => Character comparison => ",end="")                       # If hash values match then check for character comparisons

        flag=0

        for j in range(i,i+pat\_len):

            comp+=1

            if s[j]!=pat[j-i]:

               flag=1

               break

        if(flag==0):

            print(f"Match => Comparisons : {comp}")

            match\_index.append(f"Index {i} till {i + pat\_len - 1}")

        else:

            print(f"Not a match => Comparisons : {comp}")

    else:

        print(f"Not a match => Comparisons : {comp}")

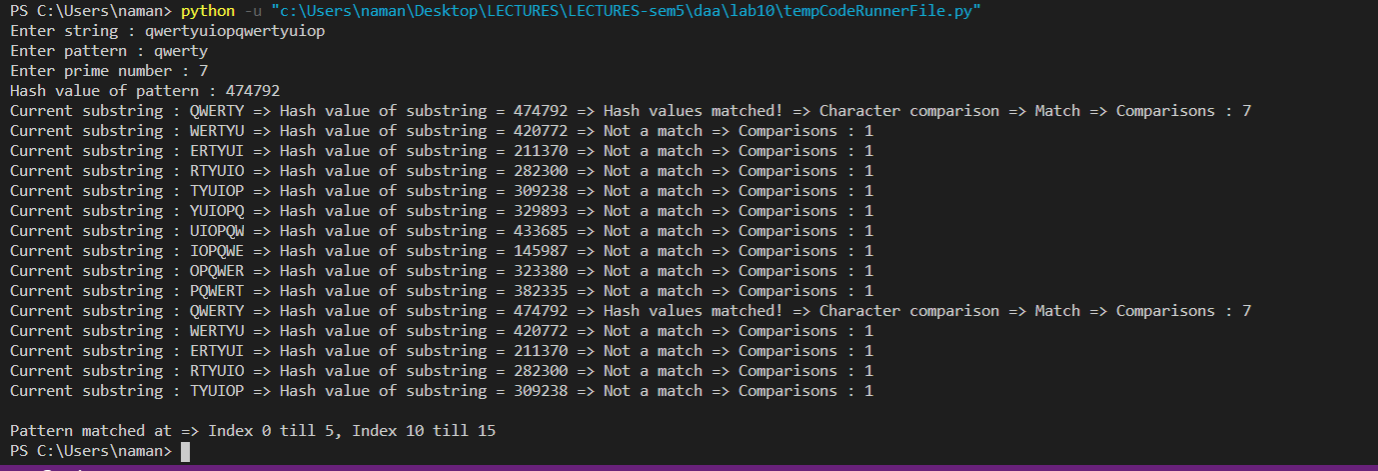
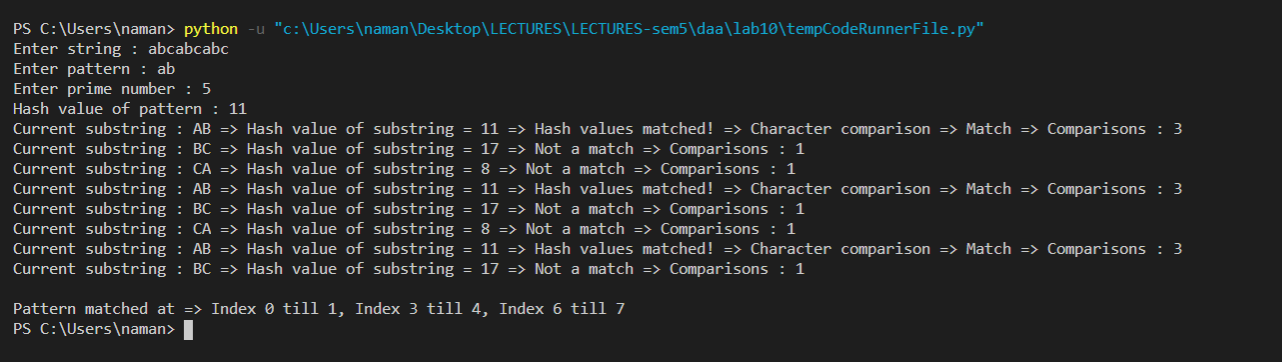
if(match\_index):

    print(f"\nPattern matched at => {', '.join(match\_index)}")

else:

    print("The pattern doesnt exist in the string")

B.2 Input and Output:

****

**B.3 Observations and learning:**

Successfully implemented Rabin Karp algorithm for string matching. String matching algorithms were designed to be an improvement over the naïve approach where character matching is done for every sliding window.

In rabin karp , the preprocessing is done through matching hash values.

Since every character’s hash value in that window is multiplied with different powers of the prime number and then added, the spurious hits are reduced drastically. Calculations for hash values are reduced by the rolling hash functions where we perform operations on the old and the new character to get the hash value of the next sliding window.

After hash values are matched, characters are matched for confirmation.

**B.4 Conclusion:**

I learnt and successfully implemented Rabin Karp algorithm for pattern matching. I understood how it is better than the naïve approach ie by reducing the character comparisons through a hashing function expressed in powers of a prime number. The calculations are further reduced by the rolling hash function where the new hash value is calculated by taking into account only 2 characters ie new and old.

**B.5 Question of Curiosity**

Q.1 Identify & discuss the real life applications of string matching algorithm.

Applications of String Matching Algorithms:

* Plagiarism Detection: The documents to be compared are decomposed into string tokens and compared using string matching algorithms. Thus, these algorithms are used to detect similarities between them and declare if the work is plagiarized or original.
* Bioinformatics and DNA Sequencing: Bioinformatics involves applying information technology and computer science to problems involving genetic sequences to find DNA patterns. String matching algorithms and DNA analysis are both collectively used for finding the occurrence of the pattern set.
* Digital Forensics: String matching algorithms are used to locate specific text strings of interest in the digital forensic text, which are useful for the investigation.
* Spelling Checker: Trie is built based on a predefined set of patterns. Then, this trie is used for string matching. The text is taken as input, and if any such pattern occurs, it is shown by reaching the acceptance state.
* Spam filters: Spam filters use string matching to discard the spam. For example, to categorize an email as spam or not, suspected spam keywords are searched in the content of the email by string matching algorithms. Hence, the content is classified as spam or not.

Q.2 Compare the performance of Rabin-Karp algorithm with naive string-matching algorithm.

The Naive String Matching algorithm slides the pattern one by one. After each slide, it one by one checks characters at the current shift and if all characters match then prints the match.

Like the Naive Algorithm, Rabin-Karp algorithm also slides the pattern one by one. But unlike the Naive algorithm, Rabin Karp algorithm matches the hash value of the pattern with the hash value of current substring of text, and if the hash values match then only it starts matching individual characters. So Rabin Karp algorithm needs to calculate hash values for following strings.

1) Pattern itself.

2) All the substrings of the text of length m.

The average and best-case running time of the Rabin-Karp algorithm is O(n+m), but its worst-case time is O(nm). Worst case of Rabin-Karp algorithm occurs when all characters of pattern and text are same as the hash values of all the substrings of txt match with hash value of pattern. For example pattern = “AAA” and txt = “AAAAAAA”.

In comparison, The time complexity of Naïve Pattern Search method is always O(m\*n). The m is the size of pattern and n is the size of the main string.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*