|  |  |
| --- | --- |
|  | DEPARTMENT OF ARTIFICIAL INTELLIGNECE & DATA SCIENCE |

|  |  |
| --- | --- |
| Subject: Analysis of Algorithm | Course Code: CSC402 |
| Semester: 4 | Course: AI & DS |
| Laboratory No: 205 | Name of Subject Teacher: Pramod Bhavarthe |
| Name of Student: Sahil Shaikh | Roll Id: VU2S2223012 |

|  |  |
| --- | --- |
| Title of Practical | N-queen problem. |

**Theory –**

The [Naive String Matching](https://www.geeksforgeeks.org/searching-for-patterns-set-1-naive-pattern-searching/) algorithm slides the pattern one by one. After each slide, one by one checks characters at the current shift, and if all characters match then print the match

*Like the Naive Algorithm, the Rabin-Karp algorithm also slides the pattern one by one. But unlike the Naive algorithm, the Rabin Karp algorithm matches the hash value of the pattern with the hash value of the 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 the following strings.*

*Pattern itself*

*All the substrings of the text of length m*

Since we need to efficiently calculate hash values for all the substrings of size m of text, we must have a hash function that has the **following property**:

* Hash at the next shift must be efficiently computable from the current hash value and next character in text or we can say hash(txt[s+1 .. s+m]) must be efficiently computable from hash(txt[s .. s+m-1]) and txt[s+m] i.e., hash(txt[s+1 .. s+m]) = rehash(txt[s+m], hash(txt[s .. s+m-1])) and
* Rehash must be O(1) operation.

**Program –**

# Following program is the python implementation of

# Rabin Karp Algorithm given in CLRS book

# d is the number of characters in the input alphabet

d = 256

# pat -> pattern

# txt -> text

# q -> A prime number

def search(pat, txt, q):

M = len(pat)

N = len(txt)

i = 0

j = 0

p = 0 # hash value for pattern

t = 0 # hash value for txt

h = 1

# The value of h would be "pow(d, M-1)%q"

for i in range(M-1):

h = (h\*d) % q

# Calculate the hash value of pattern and first window

# of text

for i in range(M):

p = (d\*p + ord(pat[i])) % q

t = (d\*t + ord(txt[i])) % q

# Slide the pattern over text one by one

for i in range(N-M+1):

# Check the hash values of current window of text and

# pattern if the hash values match then only check

# for characters one by one

if p == t:

# Check for characters one by one

for j in range(M):

if txt[i+j] != pat[j]:

break

else:

j += 1

# if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]

if j == M:

print("Pattern found at index " + str(i))

# Calculate hash value for next window of text: Remove

# leading digit, add trailing digit

if i < N-M:

t = (d\*(t-ord(txt[i])\*h) + ord(txt[i+M])) % q

# We might get negative values of t, converting it to

# positive

if t < 0:

t = t+q

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

txt = "GEEKS FOR GEEKS"

pat = "GEEK"

# A prime number

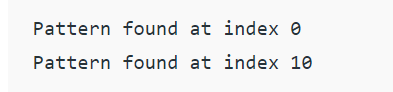
q = 101

# Function Call

search(pat, txt, q)

# This code is contributed by Bhavya Jain

**Output –**

****

**Conclusion –**

**Therefore, we have successfully understood and Implemented N-queen problem.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Grade and Dated Signature of Teacher** | **Total (10)** | **Remark** | **Dated signature of teacher** |
|  |  |  |