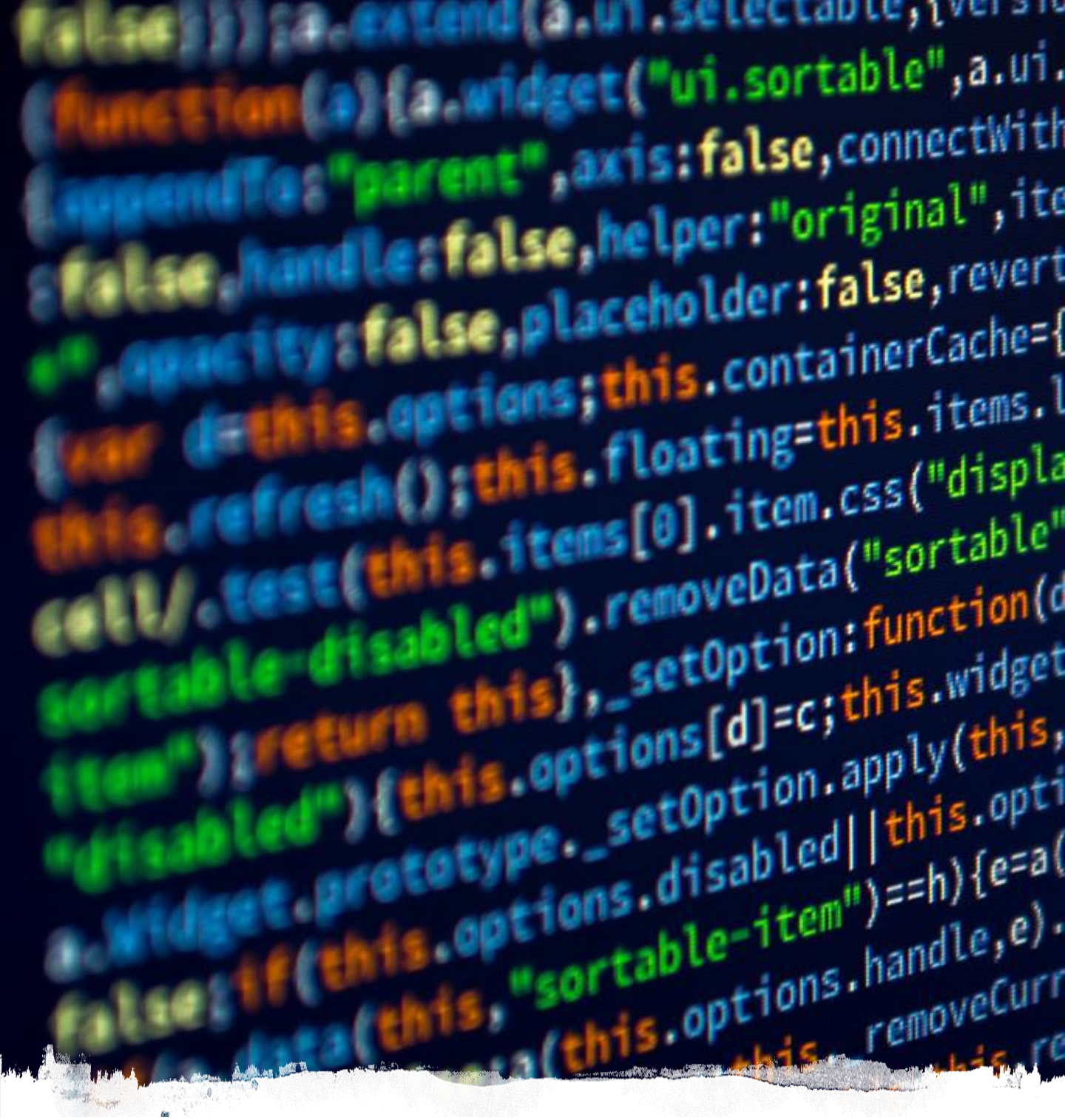


String Matching App

BS – AI – J

Maaz Ali Nadeem – i20-0452
Syed Aman Shah – i20-1835
Ahmed Raza Toor – i20-0936

Work division

Maaz Ali Nadeem
i200452 – BS AI J
1. Brute Force
2. Question 2

Syed Aman Shah
i201835 – BS AI - J
3. Rabin Karp Method
4. Question 3

Ahmed Raza Toor
i200936 – BS AI - J
5. KMP Method
6. Question 4

Language Used

Python 3.10.4

Tools Used

Tkinter – for GUI

How to run code files

- Download the **String Matching.ipynb** on a local machine.
- Open the **.ipynb** file on either ***Jupyter Lab***, ***Jupyter Notebook***, ***Google Colab***, or ***VSCode***.
- Simply Run all cells
- Choose your algorithm
- Enter the search word
- Check the relevant boxes
- Click on the Search button
- An output screen displays the file names

Useful links and references

- https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/
- https://www.geeksforgeeks.org/python-program-for-kmp-algorithm-for-pattern-searching-2/
- https://www.scaler.com/topics/data-structures/kmp-algorithm/
- http://www.csl.mtu.edu/cs4321/www/Lectures/Lecture%205%20-%20Brute%20Force%20Sorting%20and%20String%20Matching.htm
- https://www.youtube.com/watch?v=-Q4lm8eYulw&list=PLu0W_9lII9ajLcqRcj4PoEihkukF_OTzA&ab_channel=CodeWithHarry
- https://www.youtube.com/watch?v=YXPyB4XeYLA&ab_channel=freeCodeCamp.org

Comparison of Algorithms

Brute Force Algorithm

```
Algorithm BruteForceStringMatch(T[0...n-1], P[0...m-1])
    for i ← 0 to n-m do
        j ← 0
        while j < m and P[j] = T[i+j] do
            j++
        if j = m then return i
    return -1
```

The time complexity of brute force is **O(n*m)** . This means searching a string of "n" characters in a string of "m" characters using brute force, it would take us n * m tries.

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m-1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                    // preprocessing
 7      p = (dp + P[i]) mod q
 8      t_0 = (dt_0 + T[i]) mod q
 9  for s = 0 to n - m                // matching
10      if p == t_s
11          if P[1..m] == T[s+1..s+m]
12              print "Pattern occurs with shift" s
13      if s < n - m
14          t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) mod q
```

The time complexity of the searching phase of the Karp-Rabin algorithm is **O(mn)** (when searching for a^m in a^n for instance). Its expected number of text character comparisons is **O(n+m).**

```
KMP Matcher
n ← length[S]
m ← length[p]
Π← Compute-Prefix-Function(p)
q ← 0
for i ← 1 to n
do while  q > 0 and p[q+1] != S[i]
do  q ←Π[q]
if p[q+1] = S[i]
then q ← q + 1
if q = m
then print "Pattern occurs with shift" i – m
q ←Π[ q]
```

Since the two portions of the algorithm have, respectively, complexities of **O(k)** and **O(n)**, the complexity of the overall algorithm is **O(n + k).**

These complexities are the same, no matter how many repetitive patterns are in W or S.

Limitations

Brute Force:

Brute-force algorithm needs backup for every mismatch. Slow as it checks all possible combinations.

Rabin Karp:

Just by applying the hash function based on the table created , then sometimes even though Hash Code matched of both pattern and Text but the characters of the Pattern don't matche to the Text

KMP Algorithm:

KMP doesn't work well when the size of the alphabets increases

Pros

Brute Force:

The brute force approach is a guaranteed way to find the correct solution by listing all the possible candidate solutions for the problem. It is a generic method and not limited to any specific domain of problems.

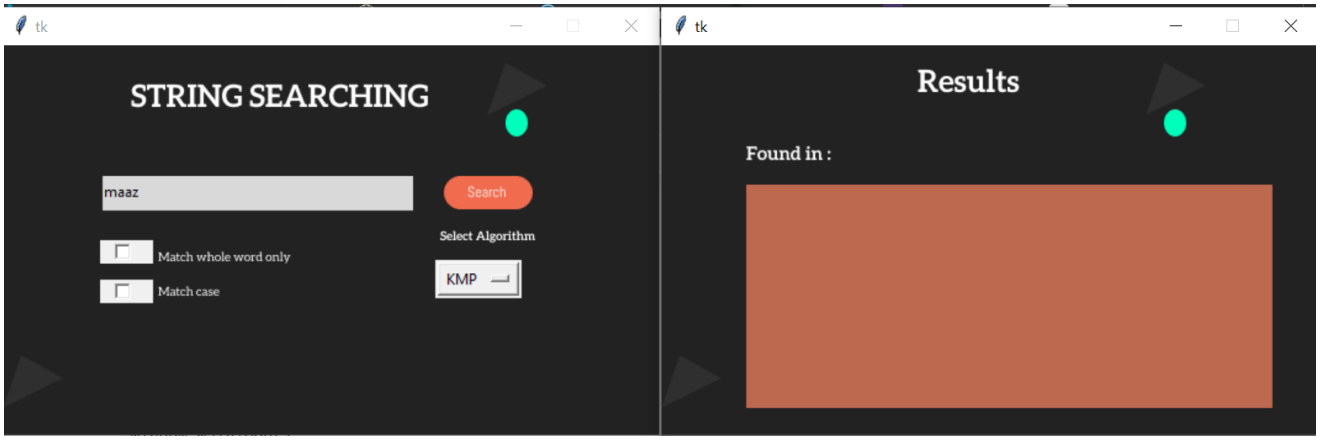
Rabin Karp:

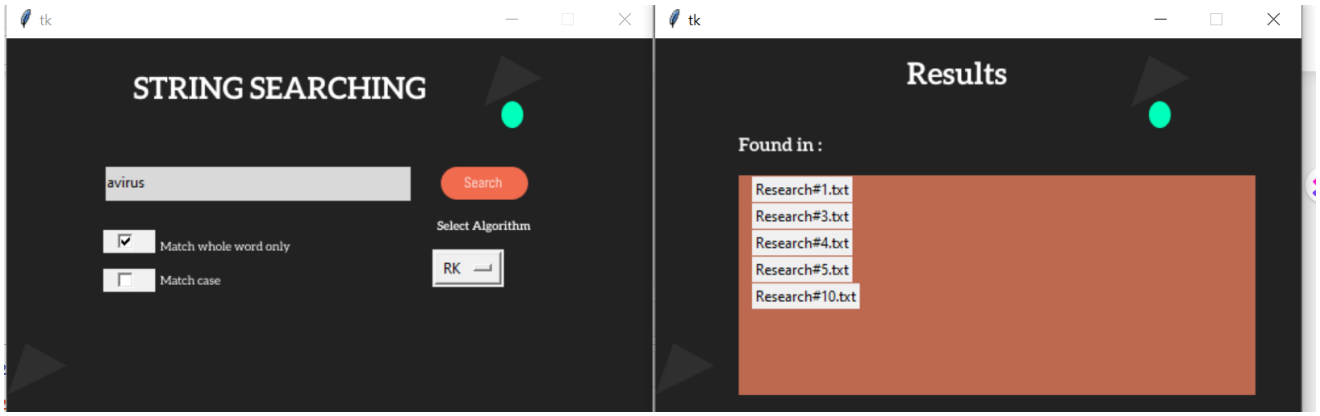
This technique results in only one comparison per text sub-sequence and brute force is only required when the hash values match.

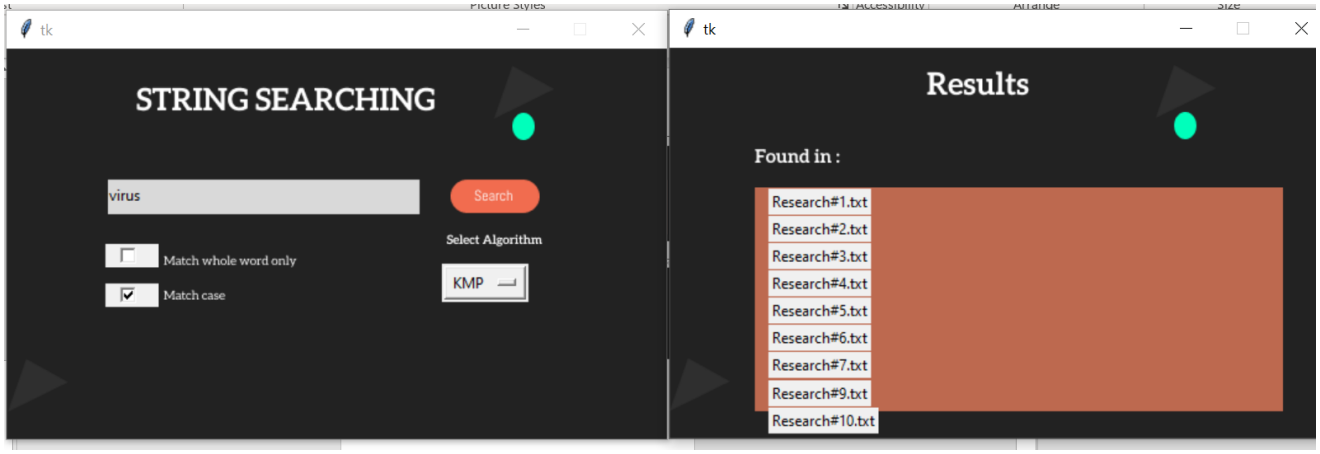
KMP Algorithm:

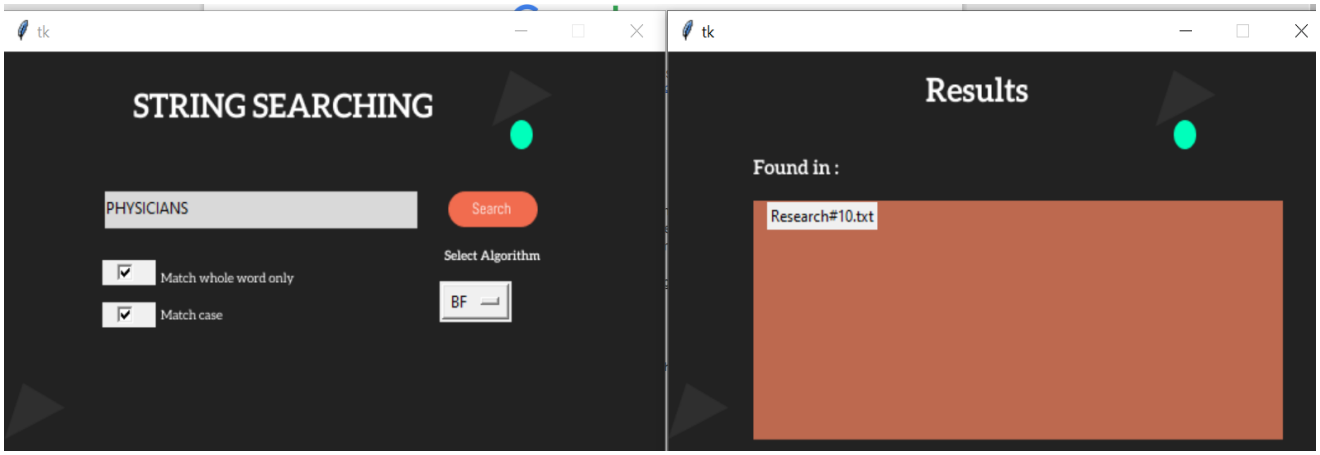
KMP has the nice advantage that it is guaranteed worst-case efficient. The preprocessing time is always O(n), and the searching time is always O(m). There are no worst-case inputs, no probability of getting unlucky, etc. In cases where you are searching for very long strings (large n) inside of really huge strings (large m), this may be highly desirable compared to other algorithms like the naive one (which can take time $\Theta(mn)$ in bad cases), Rabin-Karp (pathological inputs can take time $\Theta(mn)$)

Outputs


tk
STRING SEARCHING
maaz
Search
Match whole word only
Match case
Select Algorithm
KMP
tk
Results
Found in :


tk
STRING SEARCHING
avirus
Search
Match whole word only
Match case
Select Algorithm
RK
tk
Results
Found in :
Research#1.txt
Research#3.txt
Research#4.txt
Research#5.txt
Research#10.txt


tk
STRING SEARCHING
virus
Search
Match whole word only
Match case
Select Algorithm
KMP
tk
Results
Found in :
Research#1.txt
Research#2.txt
Research#3.txt
Research#4.txt
Research#5.txt
Research#6.txt
Research#7.txt
Research#9.txt
Research#10.txt


tk
STRING SEARCHING
PHYSICIANS
Search
Match whole word only
Match case
Select Algorithm
BF
tk
Results
Found in :
Research#10.txt