Assignment 04
Algorithm Implementation

Submitted To:

Sir Shoaib Saleem

Submitted By:

Ismail Ramzan 20i-0941

Muhammad Usman Shahid 20i-1797

Musaab Imran 20i-1794

Contents

1. Work Division:

All the 3 members were given 3 algorithms to implement.

Member 01: Q-1 KMP and Question (03 + 04)

Roll No: 20i-0941

Section: T

Name: Ismail Ramzan

Member 02: Q-1 Rabin Karp and Question 02

Roll No: 20i-1797

Section: T

Name: Muhammad Usman Shahid

Member 03: Q-1 Brute Force

Roll No: 20i-1794

Section: T

Name: Musaab Imran

2. Implementation Information:

Which language was used?

Python was used for the implementation.

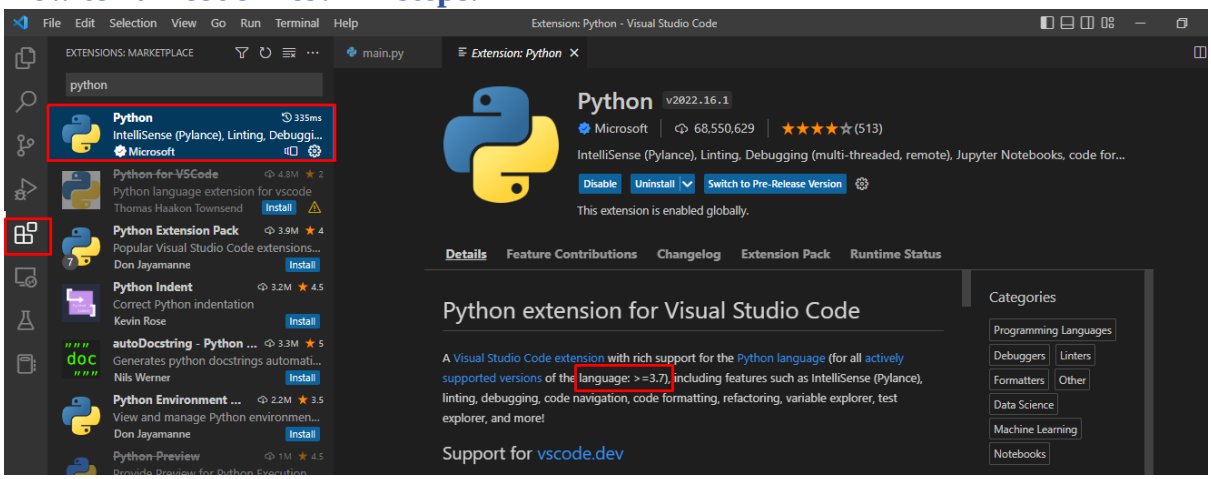
Which tools were used?

We used the **Tkinter** library to develop the front end. As python offers many tools for GUI development but TKinter is the most used method.

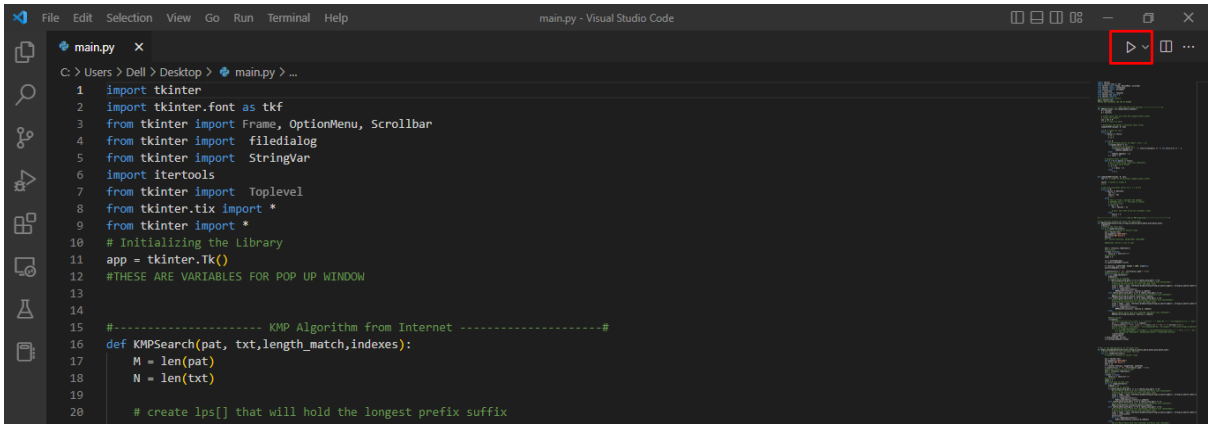
Visual Studio Code and GitHub were used for the assignment.

Algorithm Implementation

How to run code files? All steps.



- For the project you must have python version.

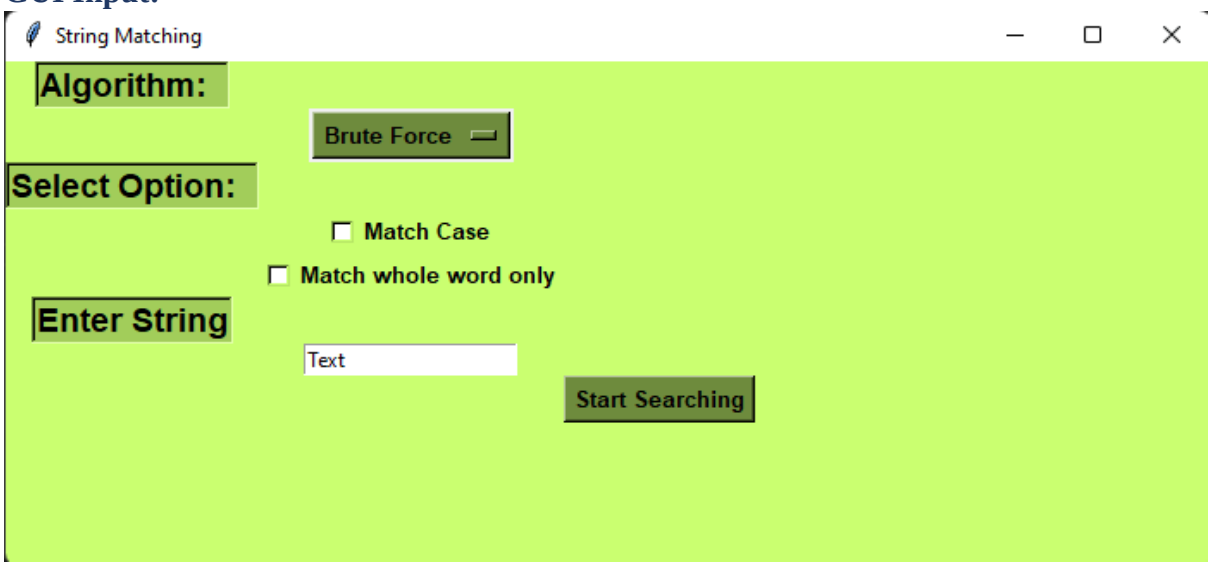


- The play button is used to run the project. After this you can see the output.

All the input and output screenshots.

The below screenshots show the input and output of each case for string matching.

GUI Input:



- The main page of the string matching.

Algorithm Implementation

```
lef KMPImplementation(files,string_to_search,match_whole_word,match_case):
    indexes=[]
    # Reading the Files Data
    for i in range(len(files)):
        # Create an instance of Tkinter frame
        win = tkinter.Tk()
        win.geometry("1000x1000")
        win.title("KMP Result")
        text=[]
        T = tkinter.Text(win, height=500, width=500)
        T.insert(str(i) + ".0", str(files[i].name + "\n"))
        #Readlines returns a list of text
        text = (files[i].readlines())
        #print(text)
        if(text != None):
            text[-1] = text[-1]+"\n"
        #print(text)
        index = 1
        #Applying Algo on each Line
        for j in range(len(text)):
            indexes=[]
            # Conditions to Consider
            if (match_whole_word.get() == 1) & (match_case.get() == 0):
                #print("Match whole word only [Checked] and Match case [Unchecked]")
                # Getting the possiblites of lower and Upper Cases
                lists = (map(''.join, itertools.product(*zip(string_to_search.upper(), string_to_search.lower()))))
                lists = (list(lists))
                for l in range(len((lists))):
                    KMPSearch((lists[l]), text[j],1,indexes)
            elif (match_whole_word.get() == 0) & (match_case.get() == 1):
                #print("Match whole word only [Unchecked] and Match case [Checked]")
                KMPSearch(string_to_search, text[j],0, indexes)
            elif (match_whole_word.get() == 0) & (match_case.get() == 0):
                #print("Match whole word only [Unchecked] and Match case [Unchecked]")
                # Getting the possiblites of lower and Upper Cases
                lists = (map(''.join, itertools.product(*zip(string_to_search.upper(), string_to_search.lower()))))
                lists = (list(lists))
                for l in range(len((lists))):
                    KMPSearch((lists[l]), text[j],0, indexes)
            else:
                #print("Match whole word only [Checked] and Match case [checked]")
                KMPSearch(string_to_search, text[j],1, indexes)
```

- KMP backend implementation

```
def Rabin_CarpImplementation(files,string_to_search,match_whole_word,match_case):
    # Driver program to test the above function
    for i in range(len(files)):
        # Create an instance of Tkinter frame

        win = tkinter.Tk()
        win.geometry("1000x1000")
        win.title("KMP Result")
        text = []
        T = tkinter.Text(win, height=105, width=80)
        T.insert(str(i) + ".0", str(files[i].name + "\n"))
        #Readlines returns a list of text
        text = (files[i].readlines())
        #print(text)
        if(text != None):
            text[-1] = text[-1]+"\n"
        #print(text)
        index = 1
        #Applying Algo on each Line
        for j in range(len(text)):
            indexes = []
            # Conditions to Consider
            if (match_whole_word.get() == 1) & (match_case.get() == 0):
                #print("Match whole word only [Checked] and Match case [Unchecked]")
                # Getting the possiblites of lower and Upper Cases
                lists = (map(''.join, itertools.product(*zip(string_to_search.upper(), string_to_search.lower()))))
                lists = (list(lists))
                for l in range(len((lists))):
                    Rabin_Carp(text[j],lists[l],1,indexes)
            elif (match_whole_word.get() == 0) & (match_case.get() == 1):
                #print("Match whole word only [Unchecked] and Match case [Checked]")
                Rabin_Carp(text[j],string_to_search,0,indexes)
            elif (match_whole_word.get() == 0) & (match_case.get() == 0):
                #print("Match whole word only [Unchecked] and Match case [Unchecked]")
                # Getting the possiblites of lower and Upper Cases
                lists = (map(''.join, itertools.product(*zip(string_to_search.upper(), string_to_search.lower()))))
                lists = (list(lists))
                print(lists)
                for l in range(len((lists))):
                    Rabin_Carp(text[j],lists[l],0,indexes)
            else:
                #print("Match whole word only [Checked] and Match case [checked]")
                Rabin_Carp(text[j],string_to_search,1,indexes)
```

- Rabin Karp backend implementation.

Algorithm Implementation

```
def Brute_ForceImplementation(files,string_to_search,match_whole_word,match_case):
    for i in range(len(files)):
        # Create an instance of Tkinter frame
        win = tkinter.Tk()
        win.geometry("1000x1000")
        win.title("KMP Result")
        text = []
        T = tkinter.Text(win, height=500, width=500)
        T.insert(str(i) + ".0", str(files[i].name + "\n"))
        #Readlines returns a list of text
        text = (files[i].readlines())
        #print(text)
        if(text != None):
            text[-1] = text[-1]+"\n"
        #print(text)
        index = 1
        #Applying Algo on each Line
        for j in range(len(text)):
            indexes = []
            # Conditions to Consider
            if (match_whole_word.get() == 1) & (match_case.get() == 0):
                #print("Match whole word only [Checked] and Match case [Unchecked]")
                # Getting the possiblites of lower and Upper Cases
                lists = (map(''.join, itertools.product(*zip(string_to_search.upper(), string_to_search.lower()))))
                lists = (list(lists))
                for l in range(len((lists))):
                    Brute_Force(lists[l], text[j],1,indexes)
            elif (match_whole_word.get() == 0) & (match_case.get() == 1):
                #print("Match whole word only [Unchecked] and Match case [Checked]")
                Brute_Force(string_to_search,text[j],0,indexes)
            elif (match_whole_word.get() == 0) & (match_case.get() == 0):
                #print("Match whole word only [Unchecked] and Match case [Unchecked]")
                # Getting the possiblites of lower and Upper Cases
                lists = (map(''.join, itertools.product(*zip(string_to_search.upper(), string_to_search.lower()))))
                lists = (list(lists))
                print(lists)
                for l in range(len((lists))):
                    Brute_Force(lists[l], text[j],0,indexes)
            else:
                # print("Match whole word only [Checked] and Match case [checked]")
                Brute_Force(string_to_search, text[j],1,indexes)
```

- Brute force backend implementation.

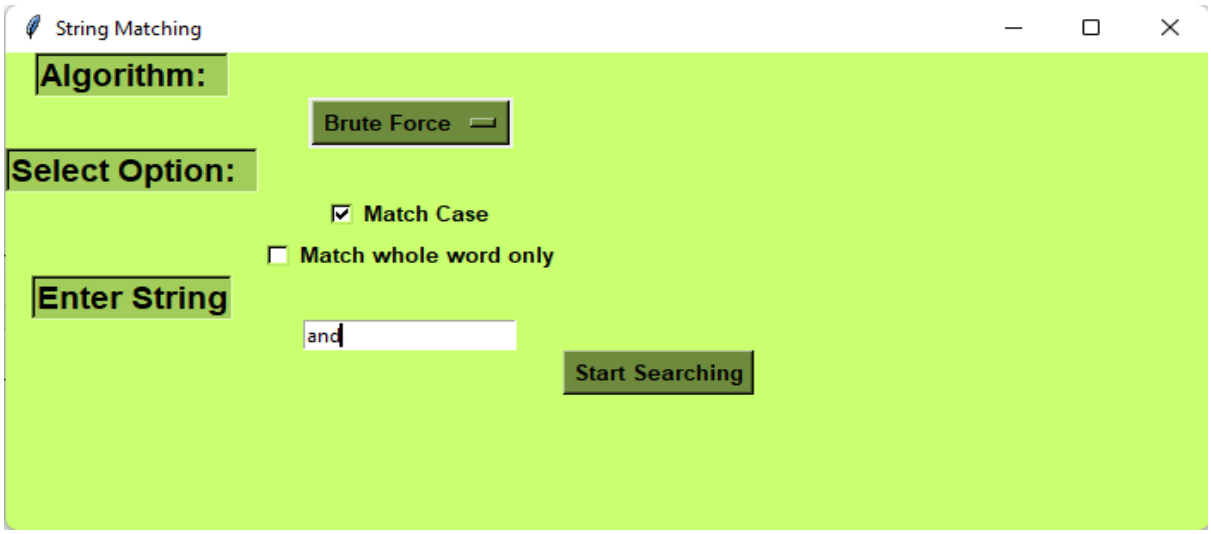
Output:

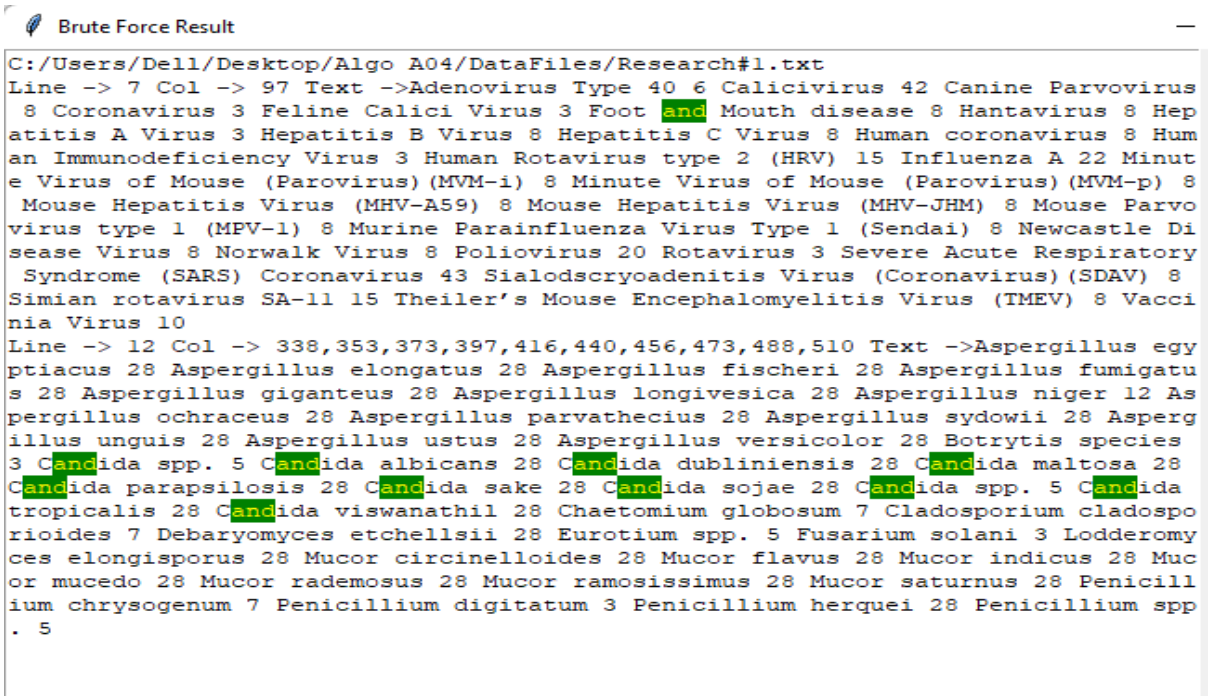
```
C:/Users/Dell/Desktop/Algo A04/DataFiles/Research#8.txt
Line -> 4 Col -> 399
Line -> 6 Col -> 171,234,604,633
Line -> 7 Col -> 135,265,462,566,654
Line -> 8 Col -> 68,514
Line -> 9 Col -> 117,291,315,526
Line -> 10 Col -> 73,202,317,581,622,705
Line -> 12 Col -> 99,177,473
Line -> 17 Col -> 108
Line -> 24 Col -> 50,119,159,477,709,831,1041
Line -> 25 Col -> 55,59,232,457,597,806,1005,1064
Line -> 26 Col -> 175,387,484,536,646,873,922,1071,1357
Line -> 27 Col -> 486,684,738,1046,1052
Line -> 28 Col -> 94,319,957,1538,1562,1617
Line -> 31 Col -> 32,1031
Line -> 32 Col -> 635,737,760,1106,1151,1220,1255
Line -> 33 Col -> 23,306,324,328,577,654,828,940,984,1271,1449
Line -> 35 Col -> 313,626,630
Line -> 36 Col -> 97,232,274,289,403,466,596,687,700,872,915,1035,1047,1141,1188,1292
Line -> 37 Col -> 255,444,458,475,578,754,786,860,873,886,893,1047,1626,1630,1681,1794,1819,2030,2070,2499,2503,2507,2511,2519
```

- The sample output.

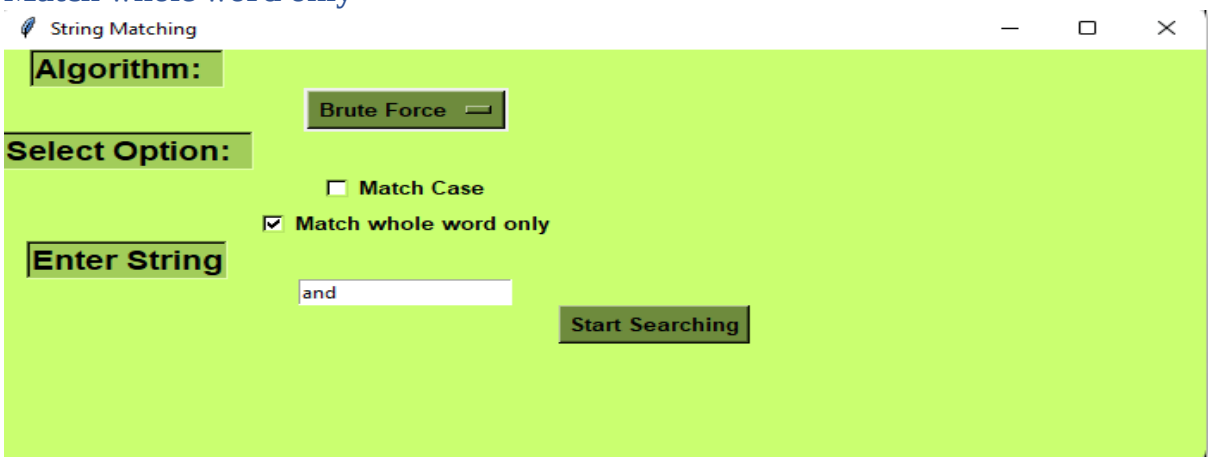
3. Brute Force

Match Case

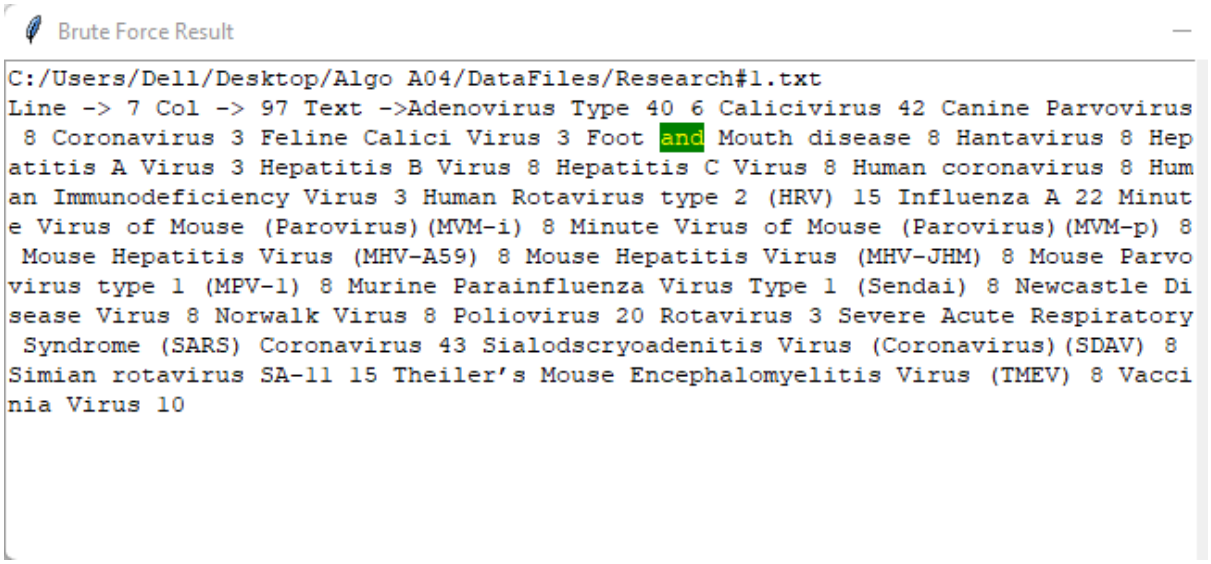




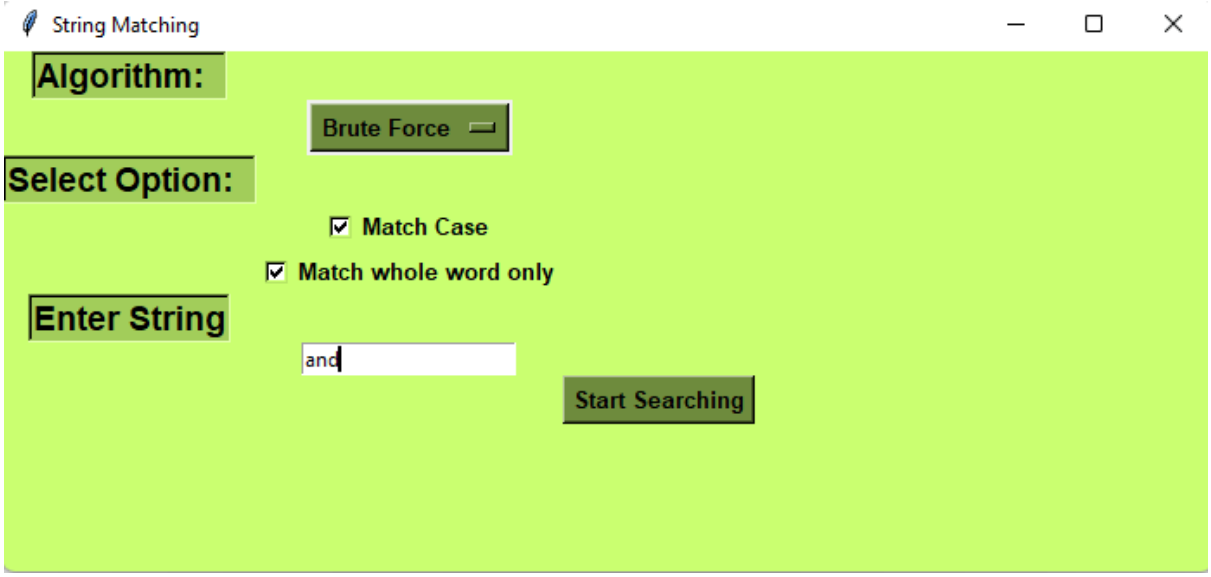
Match whole word only

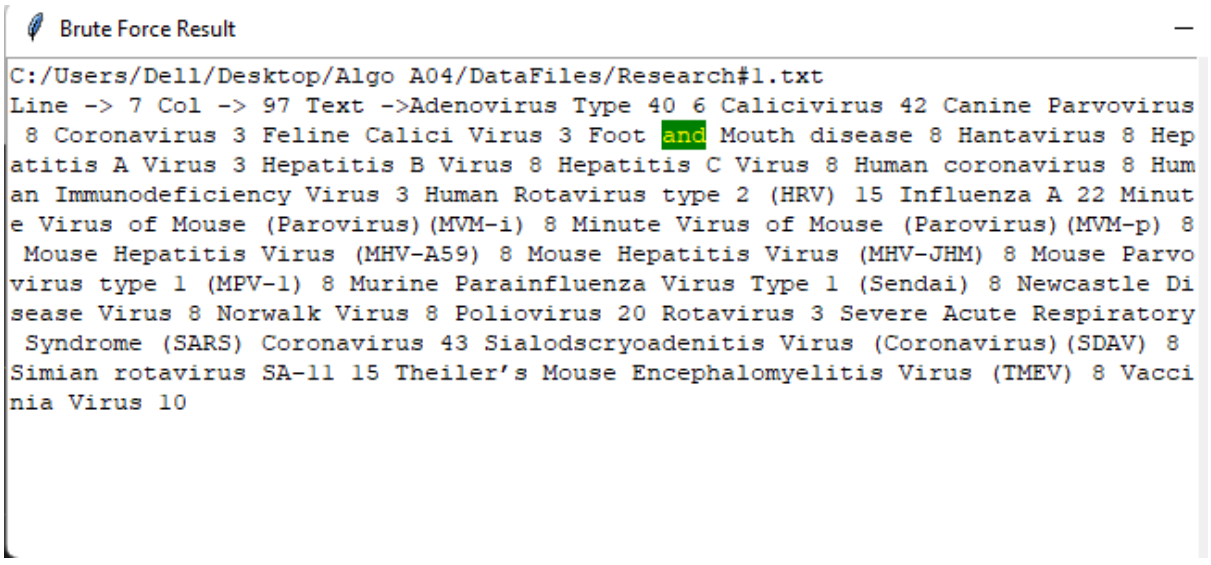


Algorithm Implementation



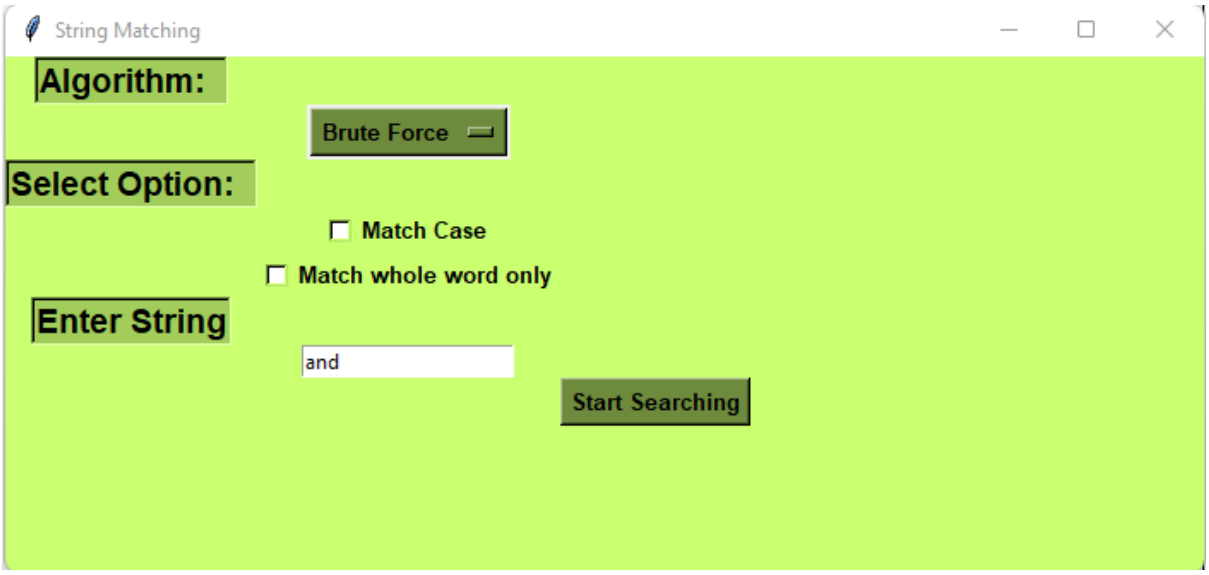
Match Case & Match whole word only

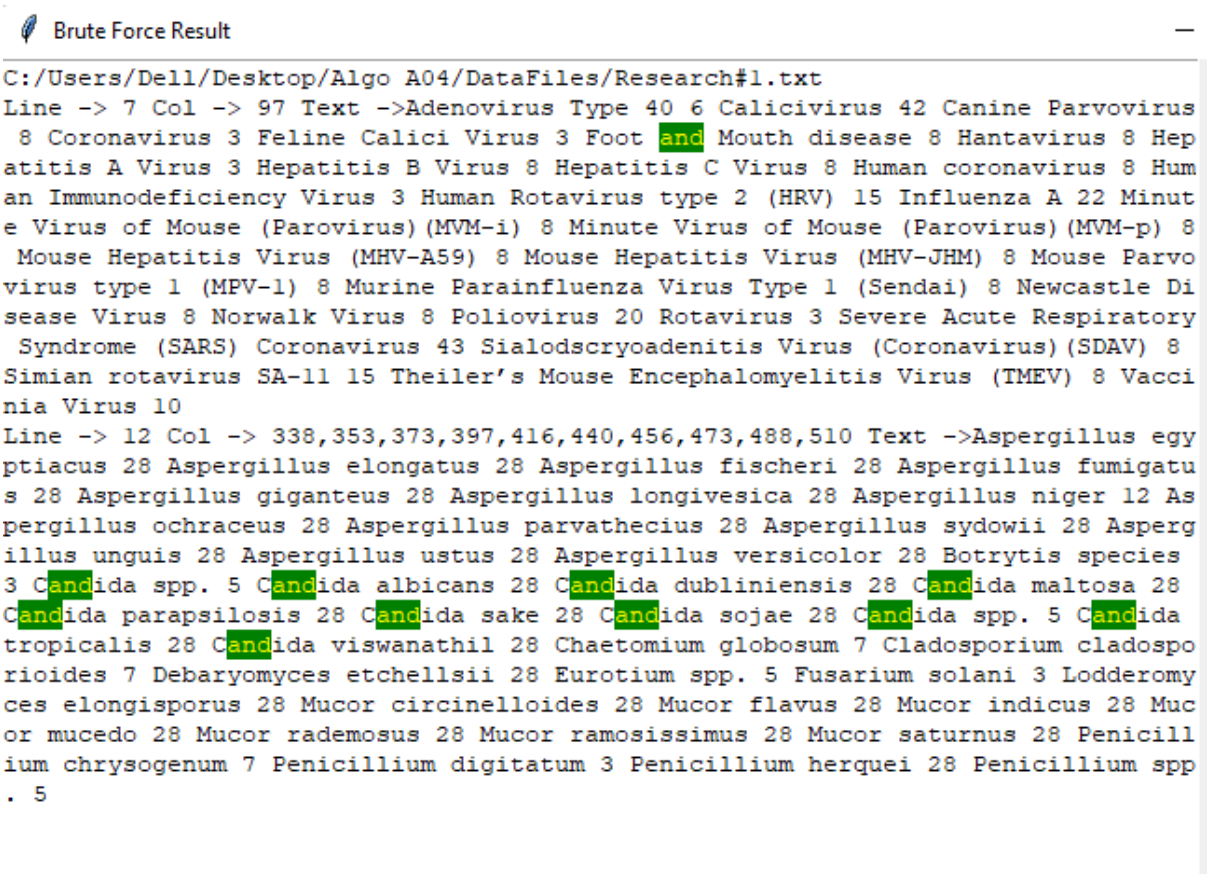




Algorithm Implementation

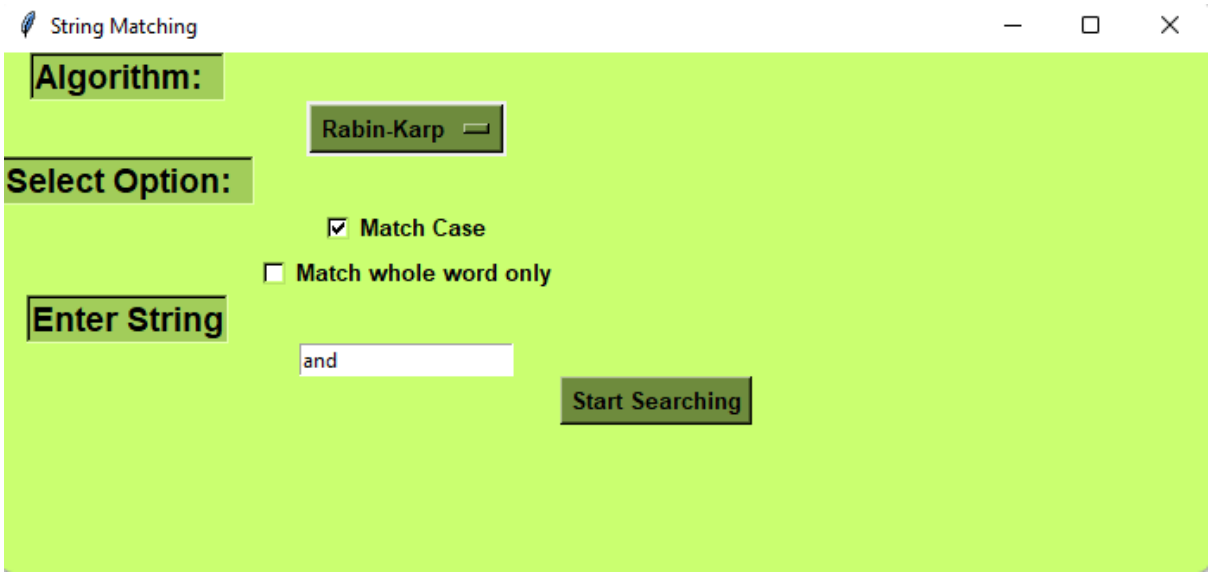
Both not checked





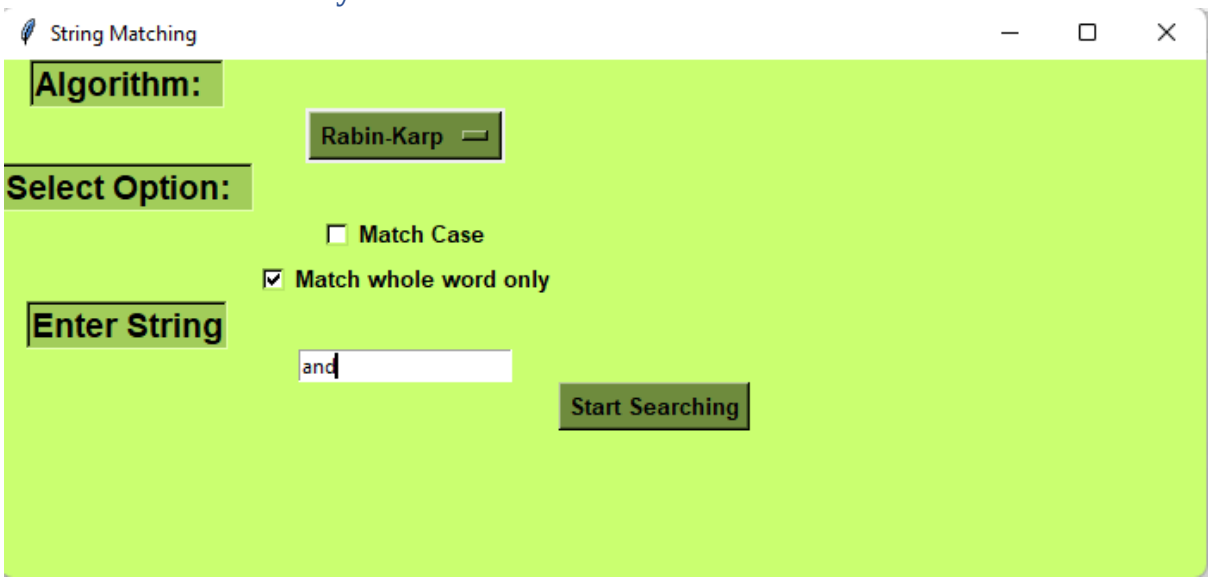
4. Rabin Karp

Match Case




```
C:/Users/Dell/Desktop/Algo A04/DataFiles/Research#1.txt
Line -> 7 Col -> 97 Text ->Adenovirus Type 40 6 Calicivirus 42 Canine Parvovirus
 8 Coronavirus 3 Feline Calici Virus 3 Foot and Mouth disease 8 Hantavirus 8 Hep
atitis A Virus 3 Hepatitis B Virus 8 Hepatitis C Virus 8 Human coronavirus 8 Hum
an Immunodeficiency Virus 3 Human Rotavirus type 2 (HRV) 15 Influenza A 22 Minut
e Virus of Mouse (Parovirus)(MVM-i) 8 Minute Virus of Mouse (Parovirus)(MVM-p) 8
 Mouse Hepatitis Virus (MHV-A59) 8 Mouse Hepatitis Virus (MHV-JHM) 8 Mouse Parvo
virus type 1 (MPV-1) 8 Murine Parainfluenza Virus Type 1 (Sendai) 8 Newcastle Di
sease Virus 8 Norwalk Virus 8 Poliovirus 20 Rotavirus 3 Severe Acute Respiratory
 Syndrome (SARS) Coronavirus 43 Sialodscryoadenitis Virus (Coronavirus)(SDAV) 8
Simian rotavirus SA-11 15 Theiler's Mouse Encephalomyelitis Virus (TMEV) 8 Vacci
nia Virus 10
Line -> 12 Col -> 338,353,373,397,416,440,456,473,488,510 Text ->Aspergillus egy
ptiacus 28 Aspergillus elongatus 28 Aspergillus fischeri 28 Aspergillus fumigatu
s 28 Aspergillus giganteus 28 Aspergillus longivesica 28 Aspergillus niger 12 As
pergillus ochraceus 28 Aspergillus parvathecius 28 Aspergillus sydowii 28 Asperg
illus unguis 28 Aspergillus ustus 28 Aspergillus versicolor 28 Botrytis species
3 Candida spp. 5 Candida albicans 28 Candida dubliniensis 28 Candida maltosa 28
Candida parapsilosis 28 Candida sake 28 Candida sojae 28 Candida spp. 5 Candida
tropicalis 28 Candida viswanathil 28 Chaetomium globosum 7 Cladosporium cladospo
rioides 7 Debaryomyces etchellsii 28 Eurotium spp. 5 Fusarium solani 3 Lodderomy
ces elongisporus 28 Mucor circinelloides 28 Mucor flavus 28 Mucor indicus 28 Muc
or mucedo 28 Mucor rademosus 28 Mucor ramosissimus 28 Mucor saturnus 28 Penicill
ium chrysogenum 7 Penicillium digitatum 3 Penicillium herquei 28 Penicillium spp
. 5
```

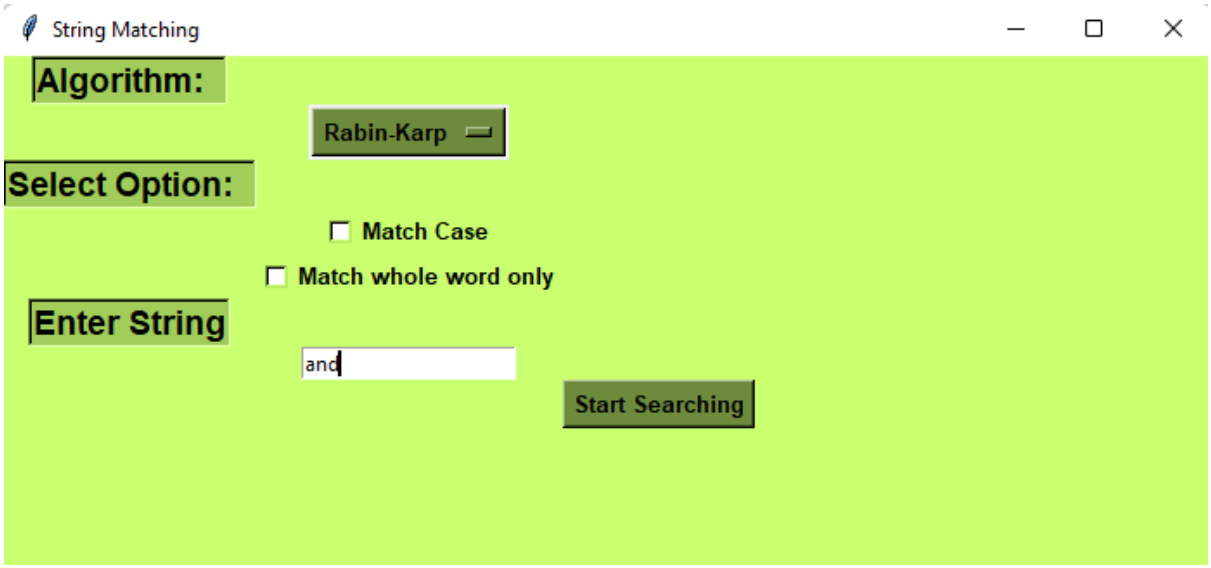
Match whole word only



Algorithm Implementation

```
C:/Users/Dell/Desktop/Algo A04/DataFiles/Research#1.txt
Line -> 7 Col -> 97 Text ->Adenovirus Type 40 6 Calicivirus 42 Canine Parvovirus
 8 Coronavirus 3 Feline Calici Virus 3 Foot and Mouth disease 8 Hantavirus 8 Hep
atitis A Virus 3 Hepatitis B Virus 8 Hepatitis C Virus 8 Human coronavirus 8 Hum
an Immunodeficiency Virus 3 Human Rotavirus type 2 (HRV) 15 Influenza A 22 Minut
e Virus of Mouse (Parovirus)(MVM-i) 8 Minute Virus of Mouse (Parovirus)(MVM-p) 8
 Mouse Hepatitis Virus (MHV-A59) 8 Mouse Hepatitis Virus (MHV-JHM) 8 Mouse Parvo
virus type 1 (MPV-1) 8 Murine Parainfluenza Virus Type 1 (Sendai) 8 Newcastle Di
sease Virus 8 Norwalk Virus 8 Poliovirus 20 Rotavirus 3 Severe Acute Respiratory
 Syndrome (SARS) Coronavirus 43 Sialodscryoadenitis Virus (Coronavirus)(SDAV) 8
Simian rotavirus SA-11 15 Theiler's Mouse Encephalomyelitis Virus (TMEV) 8 Vacci
nia Virus 10
```

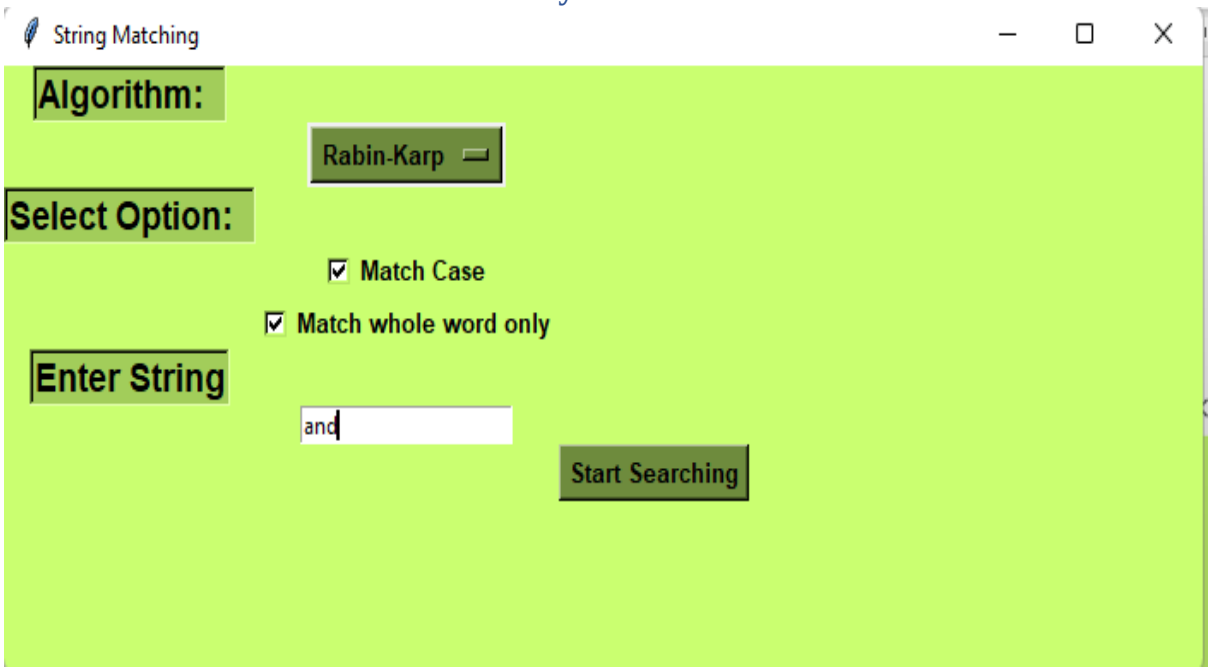
Both not checked




```
C:/Users/Dell/Desktop/Algo A04/DataFiles/Research#1.txt
Line -> 7 Col -> 97 Text ->Adenovirus Type 40 6 Calicivirus 42 Canine Parvovirus
 8 Coronavirus 3 Feline Calici Virus 3 Foot and Mouth disease 8 Hantavirus 8 Hep
atitis A Virus 3 Hepatitis B Virus 8 Hepatitis C Virus 8 Human coronavirus 8 Hum
an Immunodeficiency Virus 3 Human Rotavirus type 2 (HRV) 15 Influenza A 22 Minut
e Virus of Mouse (Parovirus)(MVM-i) 8 Minute Virus of Mouse (Parovirus)(MVM-p) 8
 Mouse Hepatitis Virus (MHV-A59) 8 Mouse Hepatitis Virus (MHV-JHM) 8 Mouse Parvo
virus type 1 (MPV-1) 8 Murine Parainfluenza Virus Type 1 (Sendai) 8 Newcastle Di
sease Virus 8 Norwalk Virus 8 Poliovirus 20 Rotavirus 3 Severe Acute Respiratory
 Syndrome (SARS) Coronavirus 43 Sialodscryoadenitis Virus (Coronavirus)(SDAV) 8
Simian rotavirus SA-11 15 Theiler's Mouse Encephalomyelitis Virus (TMEV) 8 Vacci
nia Virus 10
Line -> 12 Col -> 338,353,373,397,416,440,456,473,488,510 Text ->Aspergillus egy
ptiacus 28 Aspergillus elongatus 28 Aspergillus fischeri 28 Aspergillus fumigatu
s 28 Aspergillus giganteus 28 Aspergillus longivesica 28 Aspergillus niger 12 As
pergillus ochraceus 28 Aspergillus parvathecius 28 Aspergillus sydowii 28 Asperg
illus unguis 28 Aspergillus ustus 28 Aspergillus versicolor 28 Botrytis species
3 Candida spp. 5 Candida albicans 28 Candida dubliniensis 28 Candida maltosa 28
Candida parapsilosis 28 Candida sake 28 Candida sojae 28 Candida spp. 5 Candida
tropicalis 28 Candida viswanathil 28 Chaetomium globosum 7 Cladosporium cladospo
rioides 7 Debaryomyces etchellsii 28 Eurotium spp. 5 Fusarium solani 3 Lodderomy
ces elongisporus 28 Mucor circinelloides 28 Mucor flavus 28 Mucor indicus 28 Muc
or mucedo 28 Mucor rademosus 28 Mucor ramosissimus 28 Mucor saturnus 28 Penicill
ium chrysogenum 7 Penicillium digitatum 3 Penicillium herquei 28 Penicillium spp
. 5
```

Algorithm Implementation

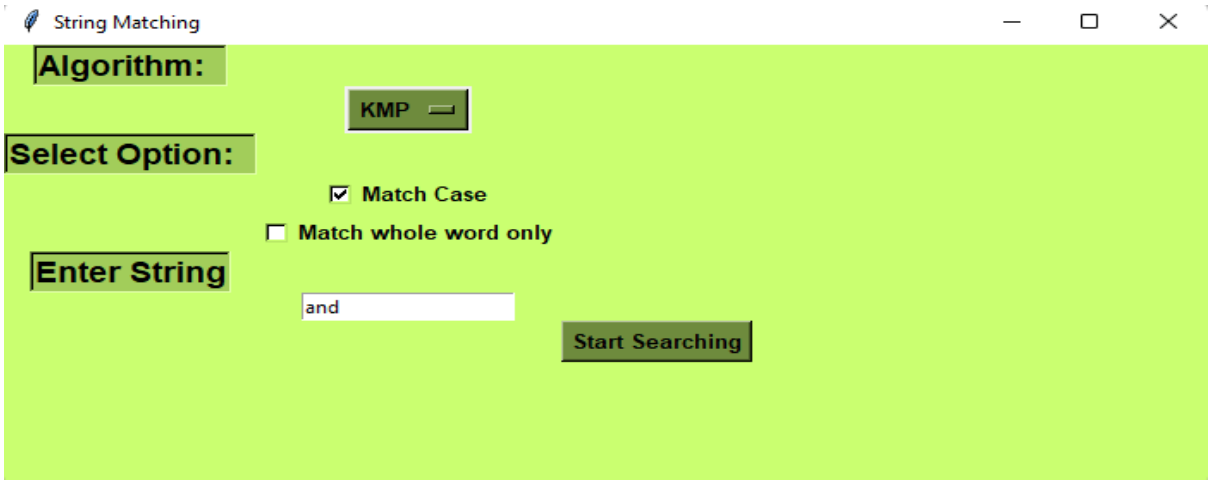
Match Case & Match whole word only

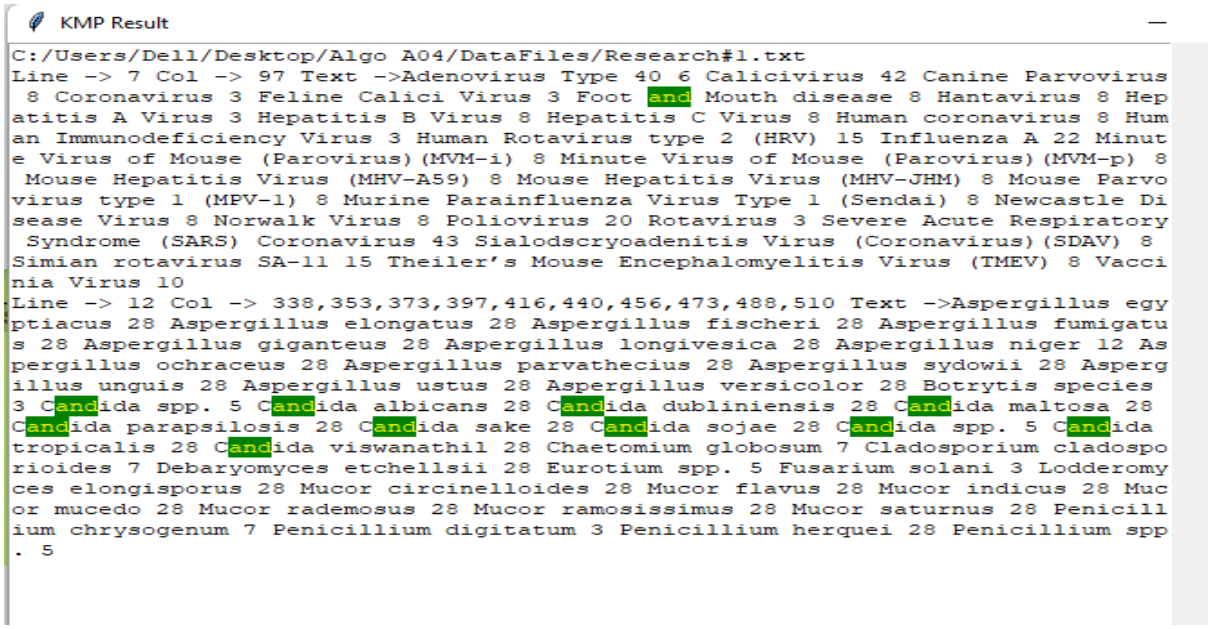



```
C:/Users/Dell/Desktop/Algo A04/DataFiles/Research#1.txt
Line -> 7 Col -> 97 Text ->Adenovirus Type 40 6 Calicivirus 42 Canine Parvovirus
 8 Coronavirus 3 Feline Calici Virus 3 Foot and Mouth disease 8 Hantavirus 8 Hep
atitis A Virus 3 Hepatitis B Virus 8 Hepatitis C Virus 8 Human coronavirus 8 Hum
an Immunodeficiency Virus 3 Human Rotavirus type 2 (HRV) 15 Influenza A 22 Minut
e Virus of Mouse (Parovirus)(MVM-i) 8 Minute Virus of Mouse (Parovirus)(MVM-p) 8
 Mouse Hepatitis Virus (MHV-A59) 8 Mouse Hepatitis Virus (MHV-JHM) 8 Mouse Parvo
virus type 1 (MPV-1) 8 Murine Parainfluenza Virus Type 1 (Sendai) 8 Newcastle Di
sease Virus 8 Norwalk Virus 8 Poliovirus 20 Rotavirus 3 Severe Acute Respiratory
 Syndrome (SARS) Coronavirus 43 Sialodscryoadenitis Virus (Coronavirus)(SDAV) 8
Simian rotavirus SA-11 15 Theiler's Mouse Encephalomyelitis Virus (TMEV) 8 Vacci
nia Virus 10
```

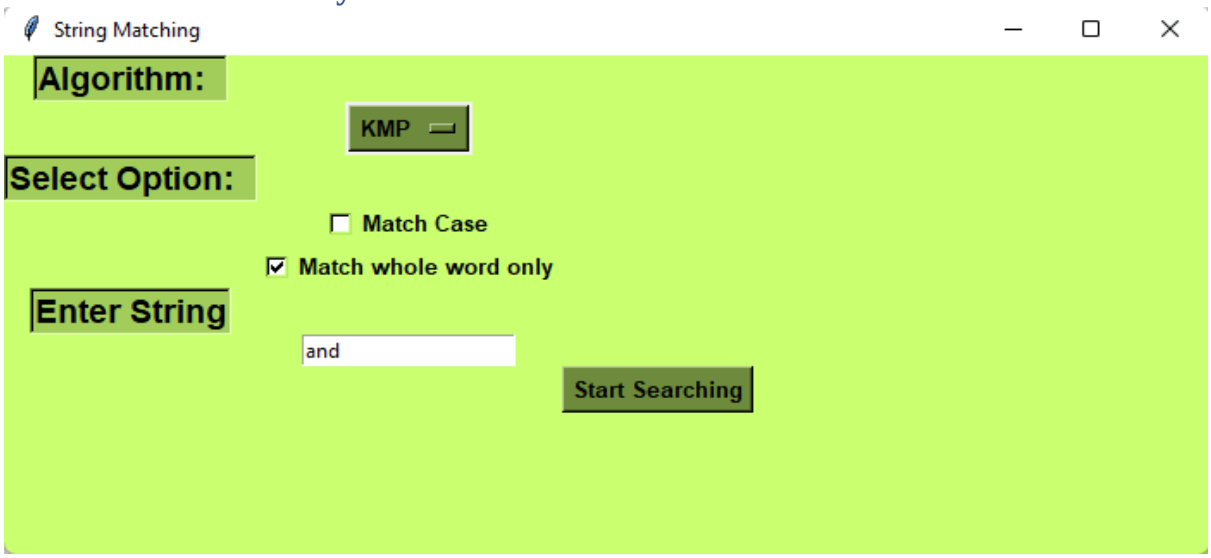
5. KMP

Match Case

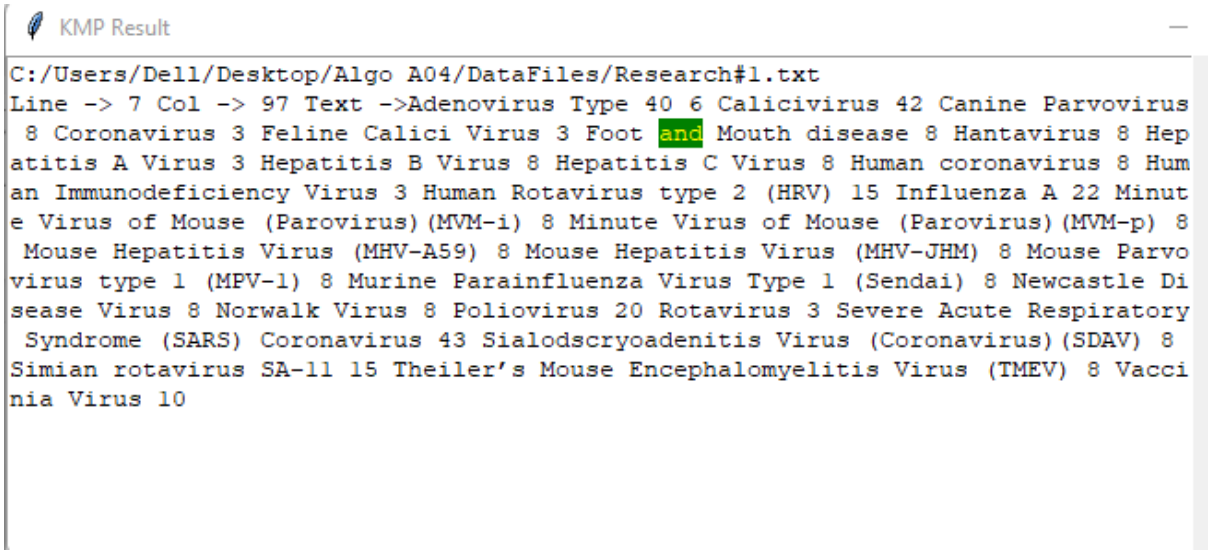




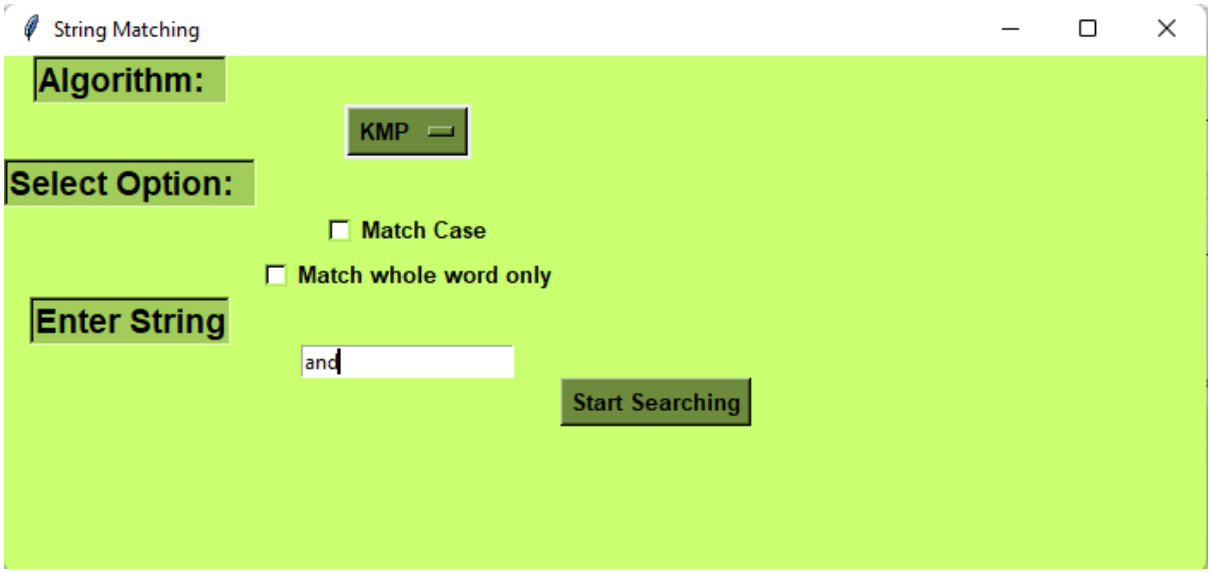
Match whole word only

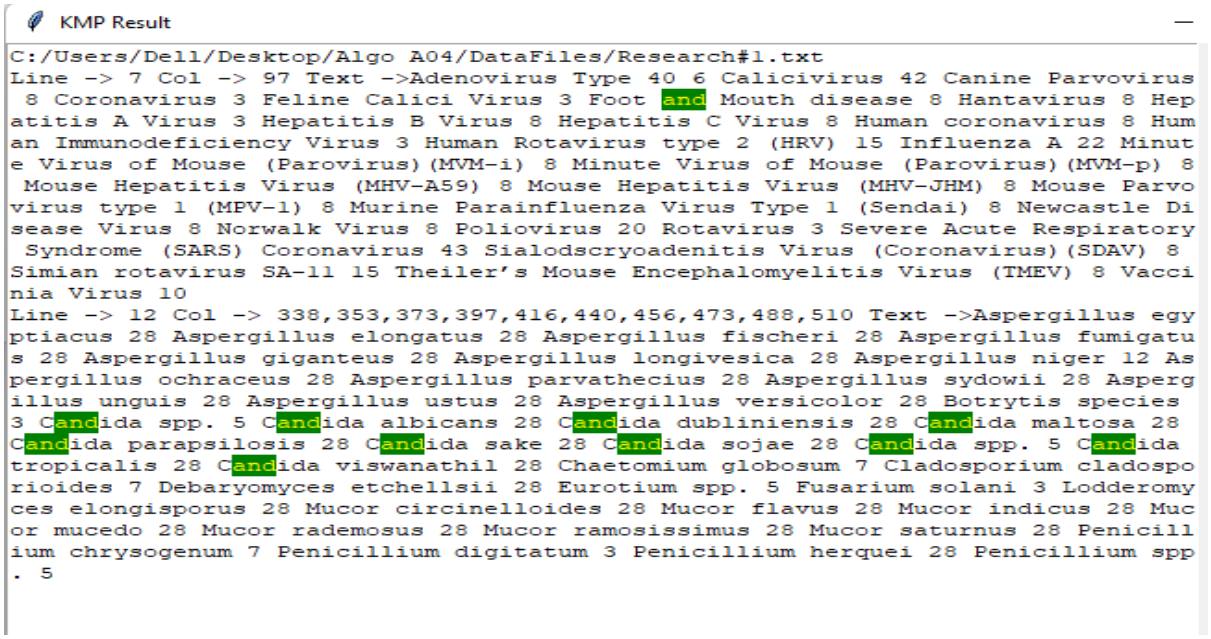


Algorithm Implementation



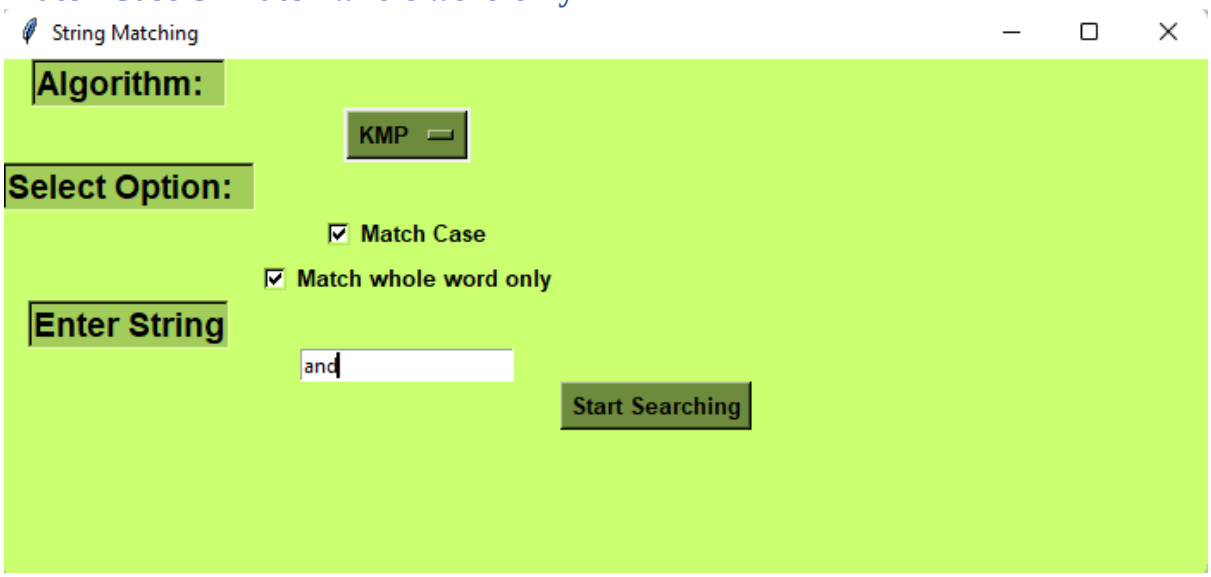
Both not checked

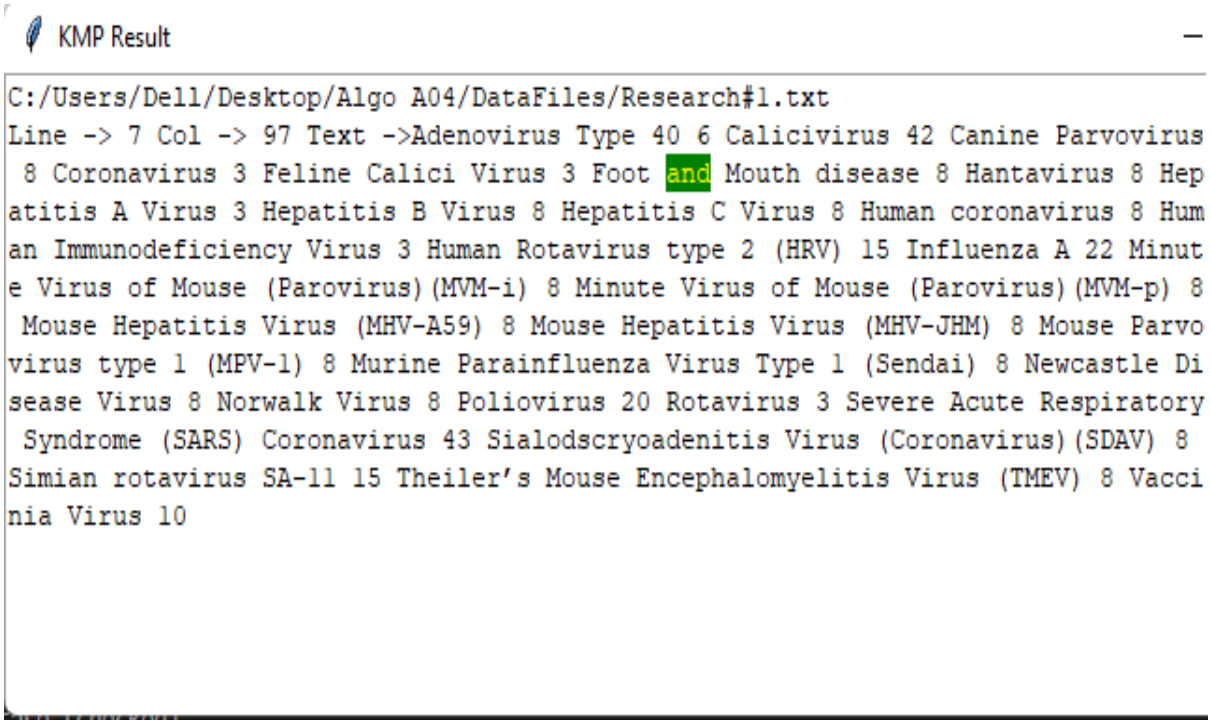




Algorithm Implementation

Match Case & Match whole word only





6. Write a detail comparison on Brute force, RK and KMP algorithm of String Matching.

Naïve Algorithm

- Naive pattern searching is the simplest method among other pattern searching algorithms. It checks for all character of the main string to the pattern.
- Naive algorithm is exact string matching(means finding one or all exact occurrences of a pattern in a text) algorithm.
- This algorithm is helpful for smaller texts. It does not need any pre-processing phases. We can find substring by checking once for the string. It also does not occupy extra space to perform the operation.
- The naive approach tests all the possible placement of Pattern P [1…….m] relative to text T [1……n]. We try shift s = 0, 1…….n-m, successively and for each shift s. Compare T [s+1…….s+m] to P [1……m].It returns all the valid shifts found.

ALGORITHM *BruteForceStringMatch*$(T[0..n-1],\ P[0..m-1])$
```
//Implements brute-force string matching
//Input: An array T[0..n − 1] of n characters representing a text and
//       an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//        matching substring or −1 if the search is unsuccessful
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

Advantages:

- No Pre-processing phase required because the running time of Naive-String-Matcher is equal to its matching time.
- No extra space is needed.
- Also, the comparisons can be done in any order.

Disadvantage:

- Naive method is inefficient because information from a shift is not used again.

KMP Algorithm

Knuth Morris Pratt pattern searching algorithm searches for occurrences of a pattern P within a string S using the key idea that when a mismatch occurs, the pattern P has sufficient information to determine where the next potential match could begin thereby avoiding several unnecessary matchings bringing the time complexity to linear.

```
j = 0;
for (i = 0; i < n; i++)
for (;;) {          // loop until break
    if (T[i] == P[j]) { // matches?
    j++;            // yes, move on to next state
    if (j == m) {    // maybe that was the last state
         found a match;
         j = overlap[j];
    }
    break;
    } else if (j == 0) break;    // no match in state j=0, give up
    else j = overlap[j];     // try shorter partial match
}
```

Advantages:

- The most obvious advantage of KMP Algorithm – data is that it's guaranteed worst-case efficiency as discussed.
- The pre-processing and the always-on time are pre-defined.
- There are no worst-case or accidental inputs.
- Preferable where the search string in a larger space is easier and more efficiently searched due to it being a time linear algorithm.
- The algorithm needs to move backward in the input text. This is particularly favourable in processing large files.

Disadvantage:

- The only disadvantage of the KMP algorithm is that it is very complex to understand.
- One of the glaring disadvantages of KMP Algorithm – data is that it doesn't work well when the size of the alphabets increases. Due to this more and more errors occurs.
- For processing very large files it also requires resources in the form of processors and that could be a problem for smaller organizations to adopt KMP Algorithm – data

Rabin-Karp

Rabin-Karp is another pattern searching algorithm to find the pattern in a more efficient way. It also checks the pattern by moving window one by one, but without checking all characters for all cases, it finds the hash value. When the hash value is matched, then only it tries to check each character. This procedure makes the algorithm more efficient.

```
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
```

Advantages:

- Not faster than brute force matching in theory, but in practice its complexity is O(n+m)
- Good hashing function it can be quite effective and it's easy to implement!
- Multiple patterns matching support
- Good for plagiarism because it can deal with multiple pattern matching!

Disadvantage:

- There are lots of string-matching algorithms that are faster than O(n+m)
- It's practically as slow as brute force matching and it requires additional space

Time Complexity Analysis	Execution	Searching	Concept
Naïve/Brute Force Algorithm	O(mn).	O(mn)	Comparing with all the characters while searching
KMP	O(m+ n).	O(n)	Construction of prefix table and then doing the comparison.
Rabin Karp	O(m+n),	O(mn).	Comparing the pattern with the calculated hash values.

7. Resources and References:

1. https://www.geeksforgeeks.org/python-program-for-rabin-karp-algorithm-for-pattern-searching/
2. https://www.geeksforgeeks.org/python-program-for-kmp-algorithm-for-pattern-searching-2/
3. https://www.declarecode.com/code-solutions/python/brute-force-string-matching-algorithm-in-python
4. https://www.geeksforgeeks.org/python-gui-tkinter/