PATTERN MATCHING ALGORITHMS

Pattern matching in Computer Science is the checking and locating of specific sequences of data of some pattern among raw data or a sequence of tokens. Unlike pattern recognition, the match has to be exact in the case of pattern matching. There are several algorithms for pattern matching and the algorithms considered for this project are explained below.

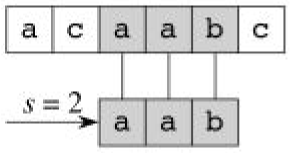
NAIVE ALGORITHM(BRUTE-FORCE ALGORITHM)

The brute-force pattern matching algorithm compares the pattern P with the text T for each possible shift of P relative to T until either,

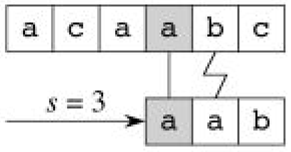
a) a match is found, or
b) all placements of the pattern have been tried

The obvious approach is to start at the first character of both strings, P[0] and T[0], and then step through T and P together, checking to see whether the characters match at P[i] and T[i].
If the match fails on some character, start this process over at P[0] and T[1], and so on, but quitting after start point T[n−m]. The brute force method does not use information about what has been matched and does not use information about recurrences within the pattern itself. Consideration of these factors leads to improvements. This led to the failure of this algorithm. This algorithm may occur in images and DNA Sequences.

For example, when we matched P = aab at s = 2, we found that T[4] = b:



Then it is not possible for a shift of s = 3 (or s = 4 if T were longer) to be valid, because these shifts juxtapose P[1] = a (and P[0] = a if applicable) against T[4] = b:



DATA STRUCTURE

We have used the List data structure in Python.

RUNTIME

The worst-case running time of this algorithm is O(nm), where n=Length of the text and m=Length of the pattern.

This can be explained with the help of a text and a pattern, along with the execution.

EXAMPLE

Text	**AAAAAAAAAAH**
Pattern	**AAAAH**

EXECUTION

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
i	A	A	A	A	A	A	A	A	A	A	H
0	A	A	A	A	H						
1		A	A	A	A	H					
2			A	A	A	A	H				
3				A	A	A	A	H			
4					A	A	A	A	H		
5						A	A	A	A	H	
6							A	A	A	A	H

BOYER - MOORE ALGORITHM

The Boyer-Moore algorithm is considered the most efficient string-matching algorithm in usual applications, for example, in-text editors and commands substitutions. The reason is that it works the fastest when the alphabet is moderately sized and the pattern is relatively long. This algorithm is a combination of two approaches, namely

a) Bad Character Heuristic
b) Good Suffix Heuristic

Both of the above heuristics can also be used independently to search a pattern in a text. The Boyer Moore algorithm does preprocessing for the same reason. It processes the pattern and creates different arrays for both heuristics. At every step, it slides the pattern by the max of the slides suggested by the two heuristics. So it uses the best of the two heuristics at every step. Boyer Moore algorithm starts matching from the last character of the pattern. The gist is that *the Bad Character indicates how much to shift based on the text's character, causing a mismatch and the Good Suffix indicates how much to shift based on matched part (suffix) of the pattern*.

BAD CHARACTER HEURISTIC(d_1)

The idea of bad character heuristic is simple. The character of the text which doesn't match with the current character of the pattern is called the Bad Character. Upon mismatch, we shift the pattern until,
a) The mismatch becomes a match
b) Pattern P moves past the mismatched character

Bad-symbol shift is computed by $t_1(c) - k$, where $t_1(c)$ is the entry in the precomputed table and k is the number of matched characters. Also, $d_1 = \max\{ t_1(c) - k, 1\}$

GOOD CHARACTER HEURISTIC(d_2)

Just like bad character heuristic, a table is generated for this as well. Let t be a substring of text T which is matched with a substring of pattern P. Now we shift pattern until,
a) Another occurrence of t in P matched with t in T.
b) A prefix of P, which matches with a suffix of t
c) P moves past t
d_2 is the distance between matched suffix of size k and its rightmost occurrence in the pattern that is not preceded by the same character as the suffix and suff(k) is the ending portion of the pattern (P) as its suffix of size k.

What is to be done if there is no other occurrence of suff(k) not preceded by the same character as in its rightmost occurrence?

a) Most cases - shift the pattern by its entire length m
b) Unfortunately, shifting the pattern by its entire length when there is no other occurrence of suff(k) not preceded by the same character as in its rightmost occurrence is not always correct.

Hence, the rule is, if there is no such occurrence, match the longest part of the k-character suffix with the corresponding prefix. After matching successfully $0 < k < m$ characters, the algorithm shifts the pattern right by,
$d = d_1$, if $k=0$
$d = \max\{d_1, d_2\}$, if $k>0$

DATA STRUCTURE

We have used the List data structure in Python.

RUNTIME

The algorithm has a worst-case running time of O(n+m), only if the pattern does not appear in the text. When the pattern does occur in the text, the running time of the original algorithm is O(nm) in the worst case, where n=Length of the text, m=Length of the pattern.

This can be explained with an example and also by building the Bad Shift and Good Suffix Tables.

EXAMPLE

Text	**BESS_KNEW_ABOUT_BAOBABS**
Pattern	**BAOBAB**

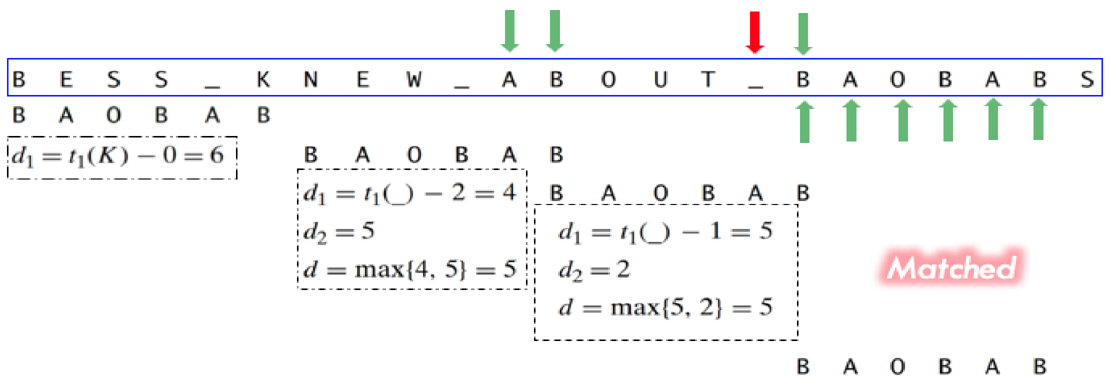
BAD-SYMBOL SHIFT TABLE

c	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	*****
$t_1(c)$	1	2	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6	6	6	6

GOOD SUFFIX TABLE

k	PATTERN	d_2
1	BAO**B**AB	2
2	**B**AOBAB	5
3	**B**AOBAB	5
4	**B**AOBAB	5
5	**B**AOBAB	5

EXECUTION



BOYER - MOORE HORSPOOL ALGORITHM

The Horspool algorithm checks first the text character aligned with the last pattern character. If it doesn't match, move (shift) the pattern forward until there is a match. A simplified version of the Boyer Moore algorithm, where it preprocesses the pattern to generate a shift table that determines how much to shift the pattern when a mismatch occurs and always makes a shift based on the text's character c aligned with the last character in the pattern according to the shift table's entry for c.
Shift sizes can be precomputed by the formula by scanning the pattern before the search begins and stored in a table called shift table.
t(c) = the pattern's length m, if c is not among the first m-1 characters of the pattern
t(c) = the distance from the rightmost c among the first m-1 characters of the pattern to its last character

DATA STRUCTURE

We have used the List data structure in Python.

RUNTIME

The pre-processing time is O(S+m), which is the time taken to construct the Shift table. The best-case running time is O(n/m) and the worst-case running time is O(nm), where n=Length of the text and m=Length of the pattern.

This can be explained with the help of a text and a pattern, along with the execution and also by building the Shift Table.

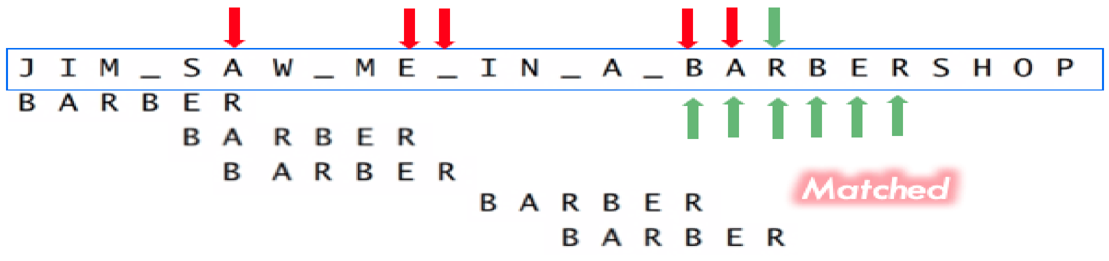
EXAMPLE

Text	**JIM_SAW_ME_IN_A_BARBERSHOP**
Pattern	**BARBER**

SHIFT TABLE

c	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	*****
t(c)	4	2	6	6	1	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6

EXECUTION



COMPARISON OF THE ALGORITHMS WITH EXAMPLES

CASE 1:

TEXT	pandaiswhiteandblack		
PATTERN	black		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	20	5	9

CASE 2:

TEXT	bagbrandiswildcraft		
PATTERN	wild		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	14	5	8

CASE 3:

TEXT	santahasabeard		
PATTERN	beard		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	14	5	9

CASE 4:

TEXT	doormirrorismainstays		
PATTERN	main		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	17	4	7

CASE 5:

TEXT	new_juice_is_not_good		
PATTERN	juice		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	9	5	6

CASE 6:

TEXT	christmas_is_near		
PATTERN	near		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	17	4	8

CASE 7:

TEXT	new_laptop_is_awesome		
PATTERN	awesome		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	22	7	9

CASE 8:

TEXT	cababababcd		
PATTERN	ababc		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	12	5	7

CASE 9:

TEXT	fan_is_unstable		
PATTERN	stable		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	16	6	8

CASE 10:

TEXT	nivedita		
PATTERN	ved		
ALGORITHM	NAIVE/BRUTE FORCE	BOYER MOORE	HORSPOOL
COMPARISONS	5	3	4

SOURCE CODE

```
import time
#BOYERMOORE ALGORITHM
def badCharHeuristic(string, size):
   badChar = [-1]*256
   for i in range(size):
       badChar[ord(string[i])] = i
   return badChar

def BoyerMoore(text, pattern):
   count = 0
   m = len(pattern)
   n = len(text)
```

```
    badChar = badCharHeuristic(pattern, m)
    s = 0
    while(s <= n-m):
        j = m-1
        while j >= 0 and pattern[j] == text[s+j]:
            j -= 1
            count = count+1
        if j < 0:
            print("Pattern found at index: {}".format(s))
            s += (m-badChar[ord(text[s+m])] if s+m < n else 1)
        else:
            s += max(1, j-badChar[ord(text[s+j])])
    print("Total number of comparisons in Boyer Moore Algorithm: "+str(count))

#HORSPOOL ALGORITHM
def shift_table(text, pattern):
    m = len(pattern)
    n = len(text)
    dict = {}
    for i in range(n):
        dict[text[i]] = m
    for j in range(m - 1):
        dict[pattern[j]] = m - 1 - j
    return dict

def Horspool(text, pattern):
    #Shift table
    table1 = shift_table(text, pattern)
    n = len(text)
    m = len(pattern)
    count = 0
    i = m - 1
    while i <= n:
        k = 0
        while k < m and pattern[m - 1 - k] == text[i - k]:
            k = k + 1
            count += 1
        if k == m:
            i = i + k if i<n else 1
```

```
                break
        else:
            i = i + table1[text[i]]
            count += 1
    print('Total number of comparisons in Horspool Algorithm: '+str(count))
    return False

#NAIVE ALGORITHM
def Naive(text, pattern):
    m = len(pattern)
    n = len(text)
    count = 0
    i=0
    for i in range(n - m+1):
        j = 0
        while(j < m):
            if (text[i + j] != pattern[j]):
                count = count + 1
                break
            j += 1
            count = count+1
        if (j == m):
            print("Total number of comparisons in Brute Force Algorithm: "+str(count))
            return count

#MAIN FUNCTION
'''--------------------------EXAMPLE 1--------------------------'''
print("CASE 1")
text1 = "pandaiswhiteandblack"
pattern1 = "black"
print("Text: "+text1)
print("Pattern: "+pattern1)
start1 = time.time()
BoyerMoore(text1,pattern1)
Horspool(text1,pattern1)
Naive(text1,pattern1)
end1 = time.time()
timetotal1 = end1 - start1
print('Total time taken: '+str(timetotal1)+' seconds')
```

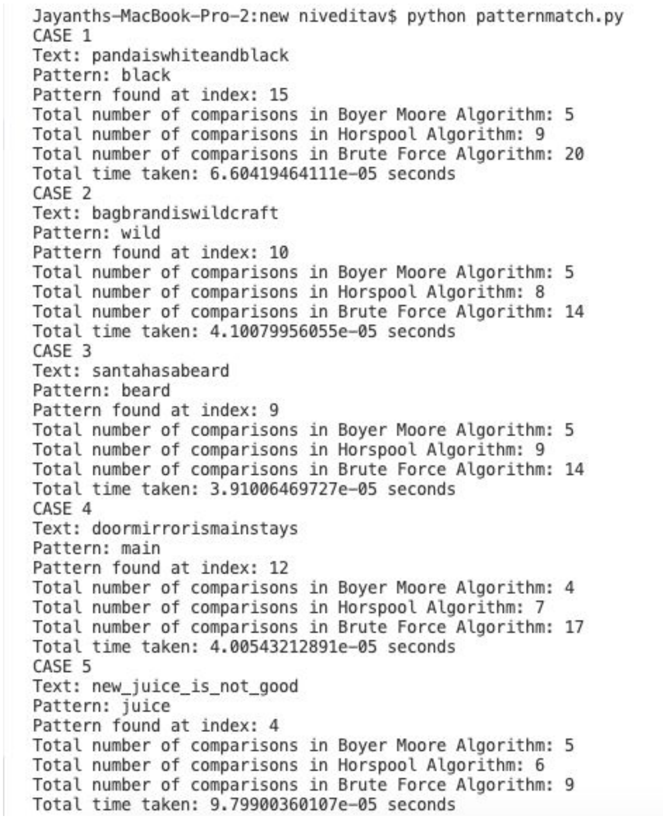
```
'''---------------------------EXAMPLE 2--------------------------'''
print("CASE 2")
text2 = "bagbrandiswildcraft"
pattern2 = "wild"
print("Text: "+text2)
print("Pattern: "+pattern2)
start2 = time.time()
BoyerMoore(text2,pattern2)
Horspool(text2,pattern2)
Naive(text2,pattern2)
end2 = time.time()
timetotal2 = end2 - start2
print('Total time taken: '+str(timetotal2)+' seconds')

'''---------------------------EXAMPLE 3--------------------------'''
print("CASE 3")
text3 = "santahasabeard"
pattern3 = "beard"
print("Text: "+text3)
print("Pattern: "+pattern3)
start3 = time.time()
BoyerMoore(text3,pattern3)
Horspool(text3,pattern3)
Naive(text3,pattern3)
end3 = time.time()
timetotal3 = end3 - start3
print('Total time taken: '+str(timetotal3)+' seconds')

'''---------------------------EXAMPLE 4--------------------------'''
print("CASE 4")
text4 = "doormirrorismainstays"
pattern4 = "main"
print("Text: "+text4)
print("Pattern: "+pattern4)
start4 = time.time()
BoyerMoore(text4,pattern4)
Horspool(text4,pattern4)
Naive(text4,pattern4)
```

```
end4 = time.time()
timetotal4 = end4 - start4
print('Total time taken: '+str(timetotal4)+' seconds')

'''--------------------------EXAMPLE 5--------------------------'''
print("CASE 5")
text5 = "new_juice_is_not_good"
pattern5 = "juice"
print("Text: "+text5)
print("Pattern: "+pattern5)
start5 = time.time()
BoyerMoore(text5,pattern5)
Horspool(text5,pattern5)
Naive(text5,pattern5)
end5 = time.time()
timetotal5 = end5 - start5
print('Total time taken: '+str(timetotal5)+' seconds')

'''--------------------------EXAMPLE 6--------------------------'''
print("CASE 6")
text6 = "christmas_is_near"
pattern6 = "near"
print("Text: "+text6)
print("Pattern: "+pattern6)
start6 = time.time()
BoyerMoore(text6,pattern6)
Horspool(text6,pattern6)
Naive(text6,pattern6)
end6 = time.time()
timetotal6 = end6 - start6
print('Total time taken: '+str(timetotal6)+' seconds')

'''--------------------------EXAMPLE 7--------------------------'''
print("CASE 7")
text7 = "new_laptop_is_awesome"
pattern7 = "awesome"
print("Text: "+text7)
print("Pattern: "+pattern7)
start7 = time.time()
```

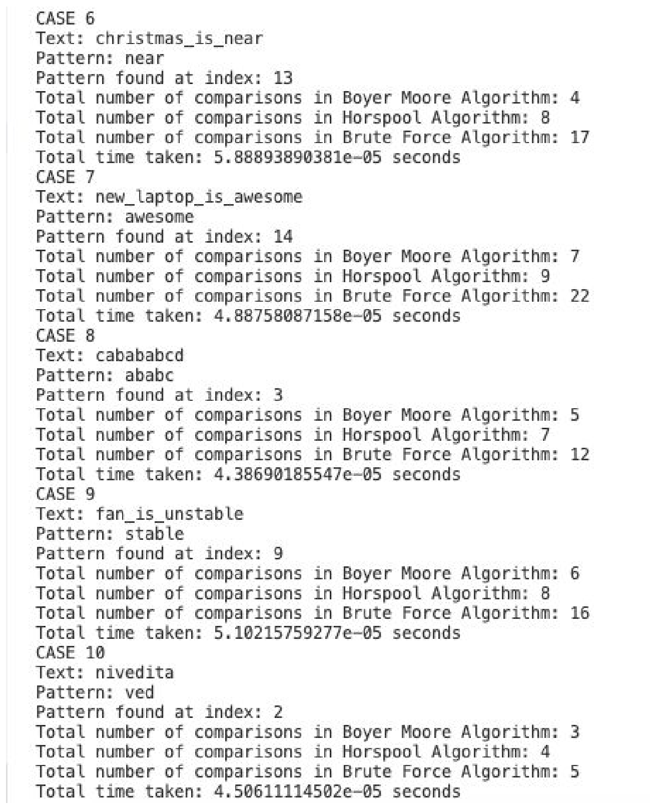
```
BoyerMoore(text7,pattern7)
Horspool(text7,pattern7)
Naive(text7,pattern7)
end7 = time.time()
timetotal7 = end7 - start7
print('Total time taken: '+str(timetotal7)+' seconds')

'''-------------------------EXAMPLE 8-------------------------'''
print("CASE 8")
text8 = "cabababcd"
pattern8 = "ababc"
print("Text: "+text8)
print("Pattern: "+pattern8)
start8 = time.time()
BoyerMoore(text8,pattern8)
Horspool(text8,pattern8)
Naive(text8,pattern8)
end8 = time.time()
timetotal8 = end8 - start8
print('Total time taken: '+str(timetotal8)+' seconds')

'''-------------------------EXAMPLE 9-------------------------'''
print("CASE 9")
text9 = "fan_is_unstable"
pattern9 = "stable"
print("Text: "+text9)
print("Pattern: "+pattern9)
start9 = time.time()
BoyerMoore(text9,pattern9)
Horspool(text9,pattern9)
Naive(text9,pattern9)
end9 = time.time()
timetotal9 = end9 - start9
print('Total time taken: '+str(timetotal9)+' seconds')

'''-------------------------EXAMPLE 10-------------------------'''
print("CASE 10")
text10 = "nivedita"
pattern10 = "ved"
```

```
print("Text: "+text10)
print("Pattern: "+pattern10)
start10 = time.time()
BoyerMoore(text10,pattern10)
Horspool(text10,pattern10)
Naive(text10,pattern10)
end10 = time.time()
timetotal10 = end10 - start10
print('Total time taken: '+str(timetotal10)+' seconds')
```

OUTPUT



```
Jayanths-MacBook-Pro-2:new niveditav$ python patternmatch.py
CASE 1
Text: pandaiswhiteandblack
Pattern: black
Pattern found at index: 15
Total number of comparisons in Boyer Moore Algorithm: 5
Total number of comparisons in Horspool Algorithm: 9
Total number of comparisons in Brute Force Algorithm: 20
Total time taken: 6.60419464111e-05 seconds
CASE 2
Text: bagbrandiswildcraft
Pattern: wild
Pattern found at index: 10
Total number of comparisons in Boyer Moore Algorithm: 5
Total number of comparisons in Horspool Algorithm: 8
Total number of comparisons in Brute Force Algorithm: 14
Total time taken: 4.10079956055e-05 seconds
CASE 3
Text: santahasabeard
Pattern: beard
Pattern found at index: 9
Total number of comparisons in Boyer Moore Algorithm: 5
Total number of comparisons in Horspool Algorithm: 9
Total number of comparisons in Brute Force Algorithm: 14
Total time taken: 3.91006469727e-05 seconds
CASE 4
Text: doormirrorismainstays
Pattern: main
Pattern found at index: 12
Total number of comparisons in Boyer Moore Algorithm: 4
Total number of comparisons in Horspool Algorithm: 7
Total number of comparisons in Brute Force Algorithm: 17
Total time taken: 4.00543212891e-05 seconds
CASE 5
Text: new_juice_is_not_good
Pattern: juice
Pattern found at index: 4
Total number of comparisons in Boyer Moore Algorithm: 5
Total number of comparisons in Horspool Algorithm: 6
Total number of comparisons in Brute Force Algorithm: 9
Total time taken: 9.79900360107e-05 seconds
```



```
CASE 6
Text: christmas_is_near
Pattern: near
Pattern found at index: 13
Total number of comparisons in Boyer Moore Algorithm: 4
Total number of comparisons in Horspool Algorithm: 8
Total number of comparisons in Brute Force Algorithm: 17
Total time taken: 5.88893890381e-05 seconds
CASE 7
Text: new_laptop_is_awesome
Pattern: awesome
Pattern found at index: 14
Total number of comparisons in Boyer Moore Algorithm: 7
Total number of comparisons in Horspool Algorithm: 9
Total number of comparisons in Brute Force Algorithm: 22
Total time taken: 4.88758087158e-05 seconds
CASE 8
Text: cabababcd
Pattern: ababc
Pattern found at index: 3
Total number of comparisons in Boyer Moore Algorithm: 5
Total number of comparisons in Horspool Algorithm: 7
Total number of comparisons in Brute Force Algorithm: 12
Total time taken: 4.38690185547e-05 seconds
CASE 9
Text: fan_is_unstable
Pattern: stable
Pattern found at index: 9
Total number of comparisons in Boyer Moore Algorithm: 6
Total number of comparisons in Horspool Algorithm: 8
Total number of comparisons in Brute Force Algorithm: 16
Total time taken: 5.10215759277e-05 seconds
CASE 10
Text: nivedita
Pattern: ved
Pattern found at index: 2
Total number of comparisons in Boyer Moore Algorithm: 3
Total number of comparisons in Horspool Algorithm: 4
Total number of comparisons in Brute Force Algorithm: 5
Total time taken: 4.50611114502e-05 seconds
```

INSTRUCTIONS TO RUN THE PROGRAM

In the terminal, type python patternmatch.py