Penggunaan String Matching dalam Pencarian Normalitas suatu Bilangan

Pendekatan Algoritma Brute Force, Knuth-Morris-Pratt ,dan Boyer-Moore

Pradipta Rafa Mahesa - 13522162
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail 13522162@std.stei.itb.ac.id

***Abstract*— Normalitas bilangan adalah konsep fundamental dalam matematika yang menunjukkan bahwa setiap digit dalam representasi bilangan muncul dengan frekuensi yang sama. Verifikasi normalitas suatu bilangan melibatkan analisis distribusi digit-digitnya. Makalah ini mengkaji metode untuk memeriksa normalitas bilangan menggunakan tiga algoritma pencocokan string: Brute Force, Knuth-Morris-Pratt (KMP), dan Boyer-Moore. Algoritma Brute Force memeriksa setiap kemungkinan posisi pola dalam teks, namun memiliki kompleksitas waktu tinggi yang membuatnya tidak efisien untuk dataset besar. Algoritma KMP meningkatkan efisiensi dengan menggunakan tabel lompatan untuk mengurangi jumlah perbandingan yang diperlukan, sehingga memberikan kompleksitas waktu yang lebih rendah. Algoritma Boyer-Moore menggunakan heuristik pergeseran buruk-karakter dan good-suffix, yang memungkinkan pola melompati bagian besar dari teks, menjadikannya sangat efisien untuk pencarian dalam teks besar. Makalah ini membandingkan efektivitas ketiga algoritma tersebut dalam konteks memeriksa normalitas bilangan. Hasil penelitian menunjukkan bahwa algoritma Boyer-Moore, dengan kemampuan lompatan yang signifikan, menawarkan kinerja terbaik dalam mengidentifikasi pola digit dalam representasi bilangan. Algoritma KMP juga menunjukkan efisiensi yang baik, terutama untuk pola dengan banyak pengulangan. Sementara itu, algoritma Brute Force, meskipun kurang efisien, memberikan pemahaman dasar yang penting tentang pencocokan string.**

Penelitian ini memberikan wawasan mendalam tentang penggunaan algoritma pencocokan string dalam analisis normalitas bilangan dan berkontribusi pada pemahaman yang lebih luas dalam bidang ini.

Keywords—angka normal; pencocokan string; algoritma brute force; algoritma Knuth-Morris-Pratt; algoritma Boyer-Moore

I. PENDAHULUAN

Bilangan normal adalah konsep penting dalam matematika dan ilmu komputer yang berhubungan dengan distribusi digit dalam representasi desimal atau basis lainnya dari suatu bilangan. Bilangan dikatakan normal jika setiap digit dalam representasinya muncul dengan frekuensi yang sama, dan semua kemungkinan urutan digit dengan panjang tertentu juga muncul dengan frekuensi yang sama. Memastikan apakah suatu bilangan adalah normal atau tidak memiliki implikasi signifikan dalam berbagai bidang, termasuk teori bilangan, kriptografi, dan analisis data.

Pada makalah ini, saya akan membahas metode untuk memeriksa normalitas suatu bilangan menggunakan algoritma pencocokan string. Algoritma pencocokan string adalah alat yang kuat dalam pemrosesan teks dan pencarian pola dalam data. Kami akan membandingkan tiga algoritma pencocokan string utama: Brute Force, Knuth-Morris-Pratt (KMP), dan Boyer-Moore, dalam konteks memeriksa normalitas bilangan.

Algoritma Brute Force adalah metode dasar untuk pencocokan string yang melibatkan pengecekan setiap kemungkinan posisi pola dalam teks. Meskipun sederhana dan mudah diimplementasikan, algoritma ini memiliki kompleksitas waktu yang tinggi, yaitu O(mxn)

n adalah panjang teks. Hal ini membuatnya tidak efisien untuk data besar, namun tetap menjadi titik awal yang baik untuk memahami konsep pencocokan string dan sebagai pembanding untuk algoritma lainnya.

Algoritma KMP meningkatkan efisiensi pencocokan string dengan memanfaatkan informasi sebelumnya dari pola untuk menghindari perbandingan yang berlebihan. Dengan menggunakan tabel lompatan, KMP mengurangi kompleksitas waktu menjadi

O(m+n). Algoritma ini sangat berguna ketika pola memiliki banyak pengulangan, sehingga dapat meminimalkan jumlah perbandingan yang diperlukan.

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan string yang paling efisien, terutama untuk pola panjang dalam teks besar. Algoritma ini menggunakan dua heuristik utama: pergeseran buruk-karakter dan pergeseran good-suffix, yang memungkinkan pola untuk melompati bagian besar dari teks. Dengan pendekatan ini, Boyer-Moore mencapai kompleksitas waktu rata-rata yang jauh lebih baik dibandingkan dengan metode lainnya.

Tujuan dari makalah ini adalah untuk mengeksplorasi dan membandingkan efektivitas ketiga algoritma ini dalam memeriksa normalitas bilangan melalui pencocokan string. Kami akan membahas implementasi masing-masing algoritma, menguji performa mereka pada dataset yang beragam, dan

menganalisis hasilnya untuk menentukan metode yang paling efisien dan akurat dalam konteks ini.

Makalah ini disusun sebagai berikut: Bagian pertama akan membahas teori dasar dan definisi bilangan normal. Bagian kedua akan menguraikan algoritma pencocokan string yang digunakan. Bagian ketiga akan menjelaskan metode eksperimen dan dataset yang digunakan. Bagian keempat akan mempresentasikan hasil dan analisis, diikuti dengan diskusi mengenai temuan utama. Bagian terakhir akan memberikan kesimpulan dan saran untuk penelitian lebih lanjut.

Melalui makalah ini, saya berharap dapat memberikan wawasan yang mendalam tentang penggunaan algoritma pencocokan string dalam memeriksa normalitas bilangan, serta kontribusi signifikan terhadap penelitian di bidang ini.

II. Dasar Teori

A. *Pencocokan String*

Pencocokan string adalah subjek yang sangat penting dalam domain pemrosesan teks yang lebih luas. Algoritma pencocokan string adalah komponen dasar yang digunakan dalam implementasi perangkat lunak praktis yang ada di sebagian besar sistem operasi. Selain itu, mereka menekankan metode pemrograman yang berfungsi sebagai paradigma di bidang ilmu komputer lainnya (desain sistem atau perangkat lunak). Akhirnya, mereka juga memainkan peran penting dalam ilmu komputer teoretis dengan menyediakan masalah yang menantang.

Meskipun data dihafal dengan berbagai cara, teks tetap menjadi bentuk utama untuk bertukar informasi. Ini sangat jelas dalam sastra atau linguistik di mana data terdiri dari korpus besar dan kamus. Hal ini juga berlaku dalam ilmu komputer di mana sejumlah besar data disimpan dalam file linier. Dan ini juga terjadi, misalnya, dalam biologi molekuler karena molekul biologis sering kali dapat diaproksimasi sebagai urutan nukleotida atau asam amino. Selain itu, jumlah data yang tersedia di bidang ini cenderung berlipat ganda setiap delapan belas bulan. Inilah sebabnya mengapa algoritma harus efisien meskipun kecepatan dan kapasitas penyimpanan komputer meningkat secara teratur.

Pencocokan string terdiri dari menemukan satu, atau lebih umum, semua kemunculan suatu string (lebih umum disebut pola) dalam sebuah teks. Semua algoritma dalam buku ini menghasilkan semua kemunculan pola dalam teks. Pola tersebut dilambangkan dengan a = x[0..m-1]; panjangnya sama dengan m. Teks tersebut dilambangkan dengan y = y[0..n-1]; panjangnya sama dengan n. Kedua string dibangun di atas satu set karakter terbatas yang disebut alfabet yang dilambangkan dengan σ.

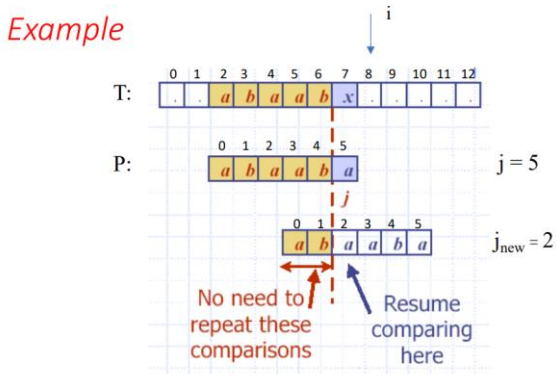
B. *Algoritma Brute-Force*

Brute force adalah sebuah pendekatan langsung untuk memecahkan suatu masalah, yang didasarkan pada *problem statement* dan definisi konsep yang dilibatkan. Algoritma brute force memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang *straight-forward.*

Di dalam pencocokan string, Algoritma brute force terdiri dari memeriksa, di semua posisi dalam teks antara 0 dan n - m apakah terdapat kemunculan pola di sana atau tidak. Kemudian, setelah setiap percobaan, algoritma menggeser pola tepat satu posisi ke kanan. Algoritma brute force tidak memerlukan fase prapemrosesan, dan memerlukan ruang ekstra konstan selain pola dan teks. Selama fase pencarian, perbandingan karakter teks dapat dilakukan dalam urutan apa pun. Kompleksitas waktu dari fase pencarian ini adalah O(m×n). Jumlah ekspektasi perbandingan karakter teks adalah 2n.

C. *Algoritma Knuth-Morris-Pratt*

Algoritma Knuth-Morris-Pratt atau disingkat sebagai KMP adalah sebuah algoritma pencocokan string yang berurutan dari kir ke kanan seperti Brute Force tetapi menggeserkan pattern yang dicari secara lebih pintar dan efisien dengan memanfaatkan Tabel Prefix terbesar yang juga merupakan suffix dari suatu substring.



Gambar 1. Ilustrasi KMP

Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Pergeseran dari algoritma KMP dihitung menggunakan Border Function yang menghitung Prefix terbesar yang juga merupakan suffix dari suatu substring pada saat tertemunya sebuah mismatch dalam pencocokan string.

- Contoh lain: P = `abababababca`
  j = `0123456789`

(k = j-1)

j	0	1	2	3	4	5	6	7	8	9
P [j]	**a**	**b**	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
k	0	1	2	3	4	5	6	7	8	
b[k]	0	0	1	2	3	4	5	6	0	

Gambar 2. Ilustrasi Fungsi Border KMP

Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Pada Proses pencocokannya, algoritma KMP mirip seperti brute force tetapi saat tertemu mismatch pergeseran pattern akan berjumlah sebanyak yang terdapat dalam table diatas.

Gambar 3. Ilustrasi Proses KMP

Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

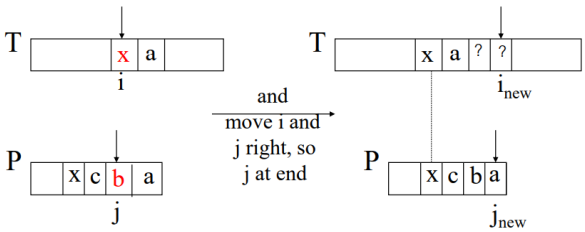
Kompleksitas waktu dari algoritma KMP adalah O(m+n) yang mana O(m) adalah bagian menghitung border function dan O(n) adalah bagian pencarian stringnya.

D. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah sebuah algoritma pencocokan string yang berdasarkan atas 2 teknik, yaitu *looking-glass* , dan *character-jump.*

Looking-glass adalah sebuah teknik yang mencari P di dalam T dengan bergerak mundur dalam P dimulai dari ujung akhir P. Sementara itu character-jump adalah sebuah teknik saat terjadinya ketidakcocokkan. Terdapat 3 kasus character-jump yang dicoba secara berurutan.

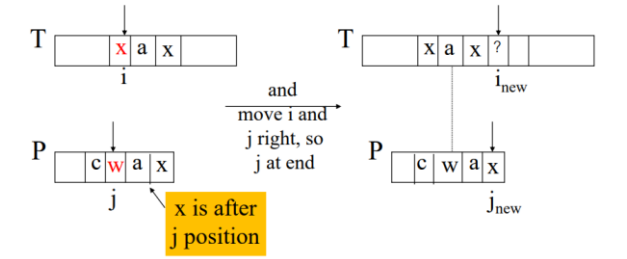
1). Kalau terdapat x dalam P , maka cobalah untuk geser ke kanan sampai kemunculan terakhir dari x.



Gambar 4. Ilustrasi Kasus 1 character-jump

Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

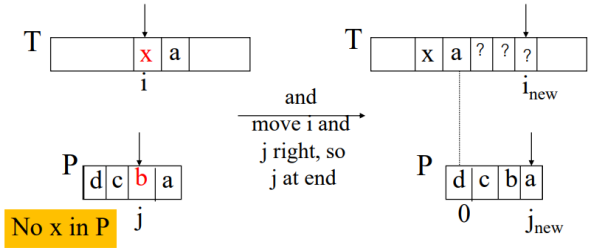
2). Kalau terdapat x dalam P, tetapi geser kanan ke kemunculan terakhir x tidak dapat dilakukan , maka geser ke kanan sebanyak 1 karakter



Gambar 5. Ilustrasi Kasus 2 character-jump

Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

3). Kalau kasus 1 & 2 tidak teraplikasi maka geser ke kanan hingga P[0] itu dicocokkan dengan T[i+1]

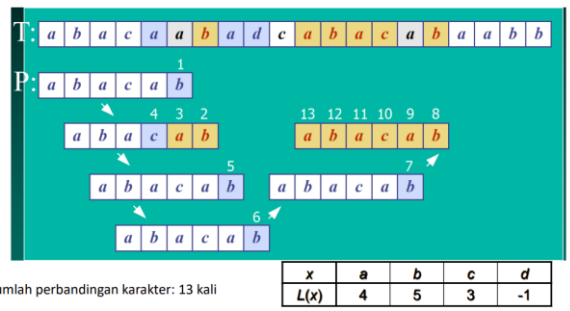


Gambar 6. Ilustrasi Kasus 3 character-jump

Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Kemunculan terakhir dari x yang disebutkan di atas akan dihitung oleh suatu fungsi bernama *Last Occurrence Function* pada saat P pertama kali diinput. Kemunculan terakhir seperti namanya adalah indeks tertinggi dari kemunculan suatu karakter dengan indeks tersebut sama dengan -1 jika karakter tersebut tidak muncul.

Algoritma Boyer-Moore bekerja dengan menerapkan kedua teknik di atas dan ditambah dengan fungsi kemunculan terakhir.



x	a	b	c	d
L(x)	4	5	3	-1

Gambar 7. Ilustrasi Proses algoritma Boyer-Moore

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Kompleksitas kasus terburuk dari Boyer-Moore adalah O(nm+A). Boyer-Moore akan cepat apabila alphabet (A) besar dan lambat apabila A kecil.

E. Angka Normal

Dalam matematika, suatu bilangan riil dikatakan sederhana normal dalam basis bilangan bulat b jika urutan tak hingga dari digit-digitnya tersebar merata dalam arti bahwa setiap nilai digit memiliki kepadatan alami yang sama yaitu 1/b. Sebuah bilangan dikatakan normal dalam basis b jika, untuk setiap bilangan bulat positif n, semua kemungkinan string sepanjang n digit memiliki kepadatan b pangkat negative n

Secara intuitif, sebuah bilangan yang sederhana normal berarti tidak ada digit yang muncul lebih sering daripada yang lain. Jika sebuah bilangan adalah normal, tidak ada kombinasi digit dengan panjang tertentu yang muncul lebih sering daripada kombinasi lainnya dengan panjang yang sama. Bilangan normal dapat dianggap sebagai urutan tak hingga dari lemparan koin (basis biner) atau lemparan dadu (basis 6). Meskipun akan ada urutan seperti 10, 100, atau lebih ekor berturut-turut (biner) atau angka lima berturut-turut (basis 6), atau bahkan 10, 100, atau lebih pengulangan urutan seperti ekor-kepala (dua lemparan koin berturut-turut) atau 6-1 (dua lemparan dadu berturut-turut), akan ada jumlah yang sama dari urutan lain dengan panjang yang sama. Tidak ada digit atau urutan yang "diunggulkan".

Suatu bilangan dikatakan normal (kadang-kadang disebut normal absolut) jika ia normal dalam semua basis bilangan bulat yang lebih besar dari atau sama dengan 2.

Meskipun dapat diberikan bukti umum bahwa hampir semua bilangan riil adalah normal (artinya himpunan bilangan non-normal memiliki ukuran Lebesgue nol), bukti ini tidak konstruktif, dan hanya beberapa bilangan tertentu yang telah terbukti normal. Misalnya, setiap konstanta Chaitin adalah normal (dan tidak dapat dihitung). Dipercaya secara luas bahwa bilangan (yang dapat dihitung) akar dari 2, pi , e adalah normal.

III. IMPLEMENTASI DAN PENGUJIAN

A. Algoritma Brute Force

Algoritma Brute Force cukup sederhana, hanya nge-loop pada setiap indeks dari teks sampai indeks sama dengan Panjang teks dikurangi Panjang pattern.

```python
def bruteforceStringMatcher(text : str , pattern : str):
    start_time = time.time()
    count = 0
    result = []
    for i in range(len(text)):
        for j in range(len(pattern)):
            if text[i+j] != pattern[j]:
                break
            if j == len(pattern)-1:
                count+=1
                result.append(i)
        if (i+len(pattern) >= len(text)):
            break

    return result,count,time.time()-start_time
```

B. Algoritma Knuth-Morris-Pratt

Implementasi algoritma Knuth-Morris-Pratt terbagi menjadi 2 bagian yaitu fungsi perhitungan Border Function dan Fungsi algoritma utama.

```python
def compute_lps(pattern):
    """
    Compute the Longest Prefix Suffix (LPS) array for the pattern.
    The LPS array is used to skip characters while matching.
    """
    length = 0  # length of the previous longest prefix suffix
    lps = [0] * len(pattern)
    i = 1

    while i < len(pattern):
        if pattern[i] == pattern[length]:
            length += 1
            lps[i] = length
            i += 1
        else:
            if length != 0:
                length = lps[length - 1]
            else:
                lps[i] = 0
                i += 1
    return lps
```

Fungsi di atas adalah fungsi untuk mencari Prefix terbesar yang juga merupakan suffix dari suatu substring. Fungsi tersebut berfungsi sebagai border function dari fungsi KMP di bawah ini.

```python
def kmp_search(text, pattern):
    """
    Perform KMP search of the pattern in the given text.
    """
    start_time = time.time()
    m = len(pattern)
    n = len(text)

    # Preprocess the pattern to get the LPS array
    lps = compute_lps(pattern)

    i = 0  # index for text
    j = 0  # index for pattern
    count = 0
    result = []

    while i <= n-m:
        if pattern[j] == text[i]:
            i += 1
            j += 1

        if j == m:
            result.append(i - j)
            count += 1
            j = lps[j - 1]
        elif i < n and pattern[j] != text[i]:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
    return result,count,time.time()-start_time
```

Di atas adalah fungsi algoritma KMP yang memanfaatkan fungsi compute_lps sebagai Border Function.

C. Algortima Boyer-Moore

```python
def lastOccurenceMap(text : str):
    LOM = {}
    for i in range (len(text)):
        LOM[text[i]] = i

    return LOM
```

Algoritma diatas adalah fungsi untuk menentukan tabel dari kemunculan terakhir dari suatu karakter. Karena keterbatasannya waktu saya hanya memasukkan karakter yang terdapat dalam Pattern input , maka tidak ada karakter yang indeks kemunculannya -1.

```python
def boyerMoore(text :str , pattern : str):
    start_time = time.time()
    m = len(pattern)
    n = len(text)
    count = 0
    last_occurrence = lastOccurenceMap(pattern)
    s = 0  # s is the shift of the pattern with respect to the text
    result = []

    while s <= n - m:
        j = m - 1

        # Keep reducing index j of pattern while characters of pattern and text are matching
        while j >= 0 and pattern[j] == text[s + j]:
            j -= 1
        # If the pattern is present at the current shift, then index j will become -1 after the loop
        if j < 0:
            # result.append(s)
            count +=1
            if (s+m >= len(text)):
                break
            elif (text[s + m]) not in last_occurrence:
                s += 1
            else:
                s += (m-last_occurrence[(text[s + m])] if s + m < n else 1)
        else:
            if (s+j >= len(text)):break
            if (text[s + j]) not in last_occurrence:
                s += 1
            else:
                s += max(1, j - last_occurrence[(text[s + j])])

    return result,count,time.time()-start_time
```

Di atas adalah kode algoritma proses utama dari Boyer-Moore. Fungsi di atas akan mengulang loop selama indeks dari pengecekan (s) masih kurang dari sama dengan panjang teks dikurangi panjang pattern yang dicari.

D. Tipe Input

Terdapat 3 jenis input yang disediakan oleh program ini yaitu input manual angkanya, bilangan Cahmpernowne , dan PI. Untuk opsi Champernowne dan Pi pengguna dapat memilih berapa jumlah digit yang terdapat di dalam bilangan tersebut.

```python
def champernowne_constant(n):
    """
    Generate the first n digits of Champernowne's constant in base 10.

    :param n: Number of digits to generate.
    :return: A string representing the first n digits of Champernowne's constant.
    """
    constant = ''
    # i = 1
    # while len(constant) < n:
    #     constant += str(i)
    #     i += 1
    for i in range(n):
        constant += str(i)
    return constant
```

Berikut adalah fungsi untuk menghasilkan bilangan Champernowne. Untuk bilangan Pi program memanfaatkan modul mpmath dan mengambilnya dari modul tersebut.

E. Main

Berikut adalah main program, tempat digabungnya dan dijalankannya fungsi-fungsi lain di atas.

```python
from GeneratePattern import generatePattern as gp
import StringMatcher as sm
import mpmath
import Champernowne as cp

def main():

    print("Pilih masukkan angka :")
    print("1). Input Manual")
    print("2). Bilangan Champernowne")
    print("3). Pi")
    txt_opt = int(input("Pilih salah satu opsi (1/2/3) : "))
    text = ""
    match(txt_opt):
        case 1:
            text = str(input("Masukkan angka : "))
        case 2:
            n = int(input("Banyak digit Champernowne : "))
            text = cp.champernowne_constant(n)
        case 3:
            n = int(input("Banyak digit Pi (belakang koma) : "))
            mpmath.mp.dps = n+2  # Decimal places
            pi_value = str(mpmath.mp.pi)
            text = pi_value[2:n+2]

    # print(text)
    digit = int(input("Masukkan banyak digit dalam suatu sekuens
: ")) # 1 -> 0..9 , 2 -> 00..99, 3 -> 000..999
    pattern = gp(digit)
    freq = [0 for i in range (len(pattern))]
    totaltime = 0
    print("Pilihan Algoritma Pencarian")
    print("1). Brute Force")
    print("2). Knuth Morris Pratt")
    print("3). Boyer-Moore")

    algorithm = int(input("Pilih algoritma (1/2/3) : "))

    for i in range (len(pattern)):
        result,count,time =
searchAlgorithm(algorithm,text,pattern[i])
        freq[i] = count
        totaltime += time
        print(f"pattern : {pattern[i]}, freq : {freq[i]} freq(%)
: {freq[i]/len(text)*100}%")
    print(f"total number : {len(text)}")
    print(f"total time : {totaltime*1000}ms")



def searchAlgorithm(algorithm : int, text,pattern):
    match algorithm:
        case 1:
            return sm.bruteforceStringMatcher(text,pattern)
        case 2:
            return sm.kmp_search(text,pattern)
        case 3:
            return sm.boyerMoore(text,pattern)

main()
```

Program akan menerima input berupa opsi masukkan lalu menerima input digit yaitu berupa jumlah angka dalam sekuens pattern pencarian (jika 1 pattern adalah 0..9, jika 2 pattern adalah 00,01,02,…99 dst).

Total time yang ditampilkan pada akhir program adalah hanya waktu dari pencarian string.

F. Hasil Pengujian

1. Pengujian dengan Bilangan Champernowne (1000 angka)

a. Brute Force

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 2
Banyak digit Champernowne : 1000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 1
pattern : 0, freq : 190 freq(%) : 6.5743944636678195%
pattern : 1, freq : 300 freq(%) : 10.380622837370241%
pattern : 2, freq : 300 freq(%) : 10.380622837370241%
pattern : 3, freq : 300 freq(%) : 10.380622837370241%
pattern : 4, freq : 300 freq(%) : 10.380622837370241%
pattern : 5, freq : 300 freq(%) : 10.380622837370241%
pattern : 6, freq : 300 freq(%) : 10.380622837370241%
pattern : 7, freq : 300 freq(%) : 10.380622837370241%
pattern : 8, freq : 300 freq(%) : 10.380622837370241%
pattern : 9, freq : 300 freq(%) : 10.380622837370241%
total number : 2890
total time : 5.806922912597656ms
```

b. KMP

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 2
Banyak digit Champernowne : 1000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 2
pattern : 0, freq : 190 freq(%) : 6.5743944636678195%
pattern : 1, freq : 300 freq(%) : 10.380622837370241%
pattern : 2, freq : 300 freq(%) : 10.380622837370241%
pattern : 3, freq : 300 freq(%) : 10.380622837370241%
pattern : 4, freq : 300 freq(%) : 10.380622837370241%
pattern : 5, freq : 300 freq(%) : 10.380622837370241%
pattern : 6, freq : 300 freq(%) : 10.380622837370241%
pattern : 7, freq : 300 freq(%) : 10.380622837370241%
pattern : 8, freq : 300 freq(%) : 10.380622837370241%
pattern : 9, freq : 300 freq(%) : 10.380622837370241%
total number : 2890
total time : 4.517078399658203ms
```

c. Boyer-Moore

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 2
Banyak digit Champernowne : 1000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 3
pattern : 0, freq : 190 freq(%) : 6.5743944636678195%
pattern : 1, freq : 300 freq(%) : 10.380622837370241%
pattern : 2, freq : 300 freq(%) : 10.380622837370241%
pattern : 3, freq : 300 freq(%) : 10.380622837370241%
pattern : 4, freq : 300 freq(%) : 10.380622837370241%
pattern : 5, freq : 300 freq(%) : 10.380622837370241%
pattern : 6, freq : 300 freq(%) : 10.380622837370241%
pattern : 7, freq : 300 freq(%) : 10.380622837370241%
pattern : 8, freq : 300 freq(%) : 10.380622837370241%
pattern : 9, freq : 300 freq(%) : 10.380622837370241%
total number : 2890
total time : 2.971649169921875ms
```

2. Pengujian dengan Bilangan Pi (1000 angka)
   a. Brute Force

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 3
Banyak digit Pi (belakang koma) : 1000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 1
pattern : 0, freq : 93 freq(%) : 9.3%
pattern : 1, freq : 116 freq(%) : 11.600000000000001%
pattern : 2, freq : 103 freq(%) : 10.299999999999999%
pattern : 3, freq : 102 freq(%) : 10.2%
pattern : 4, freq : 93 freq(%) : 9.3%
pattern : 5, freq : 97 freq(%) : 9.700000000000001%
pattern : 6, freq : 94 freq(%) : 9.4%
pattern : 7, freq : 95 freq(%) : 9.5%
pattern : 8, freq : 101 freq(%) : 10.100000000000001%
pattern : 9, freq : 106 freq(%) : 10.6%
total number : 1000
total time : 1.6360282897949219ms
```

   b. KMP

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 3
Banyak digit Pi (belakang koma) : 1000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 2
pattern : 0, freq : 93 freq(%) : 9.3%
pattern : 1, freq : 116 freq(%) : 11.600000000000001%
pattern : 2, freq : 103 freq(%) : 10.299999999999999%
pattern : 3, freq : 102 freq(%) : 10.2%
pattern : 4, freq : 93 freq(%) : 9.3%
pattern : 5, freq : 97 freq(%) : 9.700000000000001%
pattern : 6, freq : 94 freq(%) : 9.4%
pattern : 7, freq : 95 freq(%) : 9.5%
pattern : 8, freq : 101 freq(%) : 10.100000000000001%
pattern : 9, freq : 106 freq(%) : 10.6%
total number : 1000
total time : 0.9214878082275391ms
```

   c. Boyer-Moore

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 3
Banyak digit Pi (belakang koma) : 1000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 3
pattern : 0, freq : 93 freq(%) : 9.3%
pattern : 1, freq : 116 freq(%) : 11.600000000000001%
pattern : 2, freq : 103 freq(%) : 10.299999999999999%
pattern : 3, freq : 102 freq(%) : 10.2%
pattern : 4, freq : 93 freq(%) : 9.3%
pattern : 5, freq : 97 freq(%) : 9.700000000000001%
pattern : 6, freq : 94 freq(%) : 9.4%
pattern : 7, freq : 95 freq(%) : 9.5%
pattern : 8, freq : 101 freq(%) : 10.100000000000001%
pattern : 9, freq : 106 freq(%) : 10.6%
total number : 1000
total time : 2.506732940673828ms
```

3. Pengujian dengan Bilangan Champernowne (1000000 angka)
   a. BruteForce

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 2
Banyak digit Champernowne : 1000000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 1
pattern : 0, freq : 488890 freq(%) : 8.30190409398036%
pattern : 1, freq : 600000 freq(%) : 10.18867732289107%
pattern : 2, freq : 600000 freq(%) : 10.18867732289107%
pattern : 3, freq : 600000 freq(%) : 10.18867732289107%
pattern : 4, freq : 600000 freq(%) : 10.18867732289107%
pattern : 5, freq : 600000 freq(%) : 10.18867732289107%
pattern : 6, freq : 600000 freq(%) : 10.18867732289107%
pattern : 7, freq : 600000 freq(%) : 10.18867732289107%
pattern : 8, freq : 600000 freq(%) : 10.18867732289107%
pattern : 9, freq : 600000 freq(%) : 10.18867732289107%
total number : 5888890
total time : 8206.226348876953ms
```

b. KMP

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 2
Banyak digit Champernowne : 1000000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 2
pattern : 0, freq : 488890 freq(%) : 8.30190409398036%
pattern : 1, freq : 600000 freq(%) : 10.18867732289107%
pattern : 2, freq : 600000 freq(%) : 10.18867732289107%
pattern : 3, freq : 600000 freq(%) : 10.18867732289107%
pattern : 4, freq : 600000 freq(%) : 10.18867732289107%
pattern : 5, freq : 600000 freq(%) : 10.18867732289107%
pattern : 6, freq : 600000 freq(%) : 10.18867732289107%
pattern : 7, freq : 600000 freq(%) : 10.18867732289107%
pattern : 8, freq : 600000 freq(%) : 10.18867732289107%
pattern : 9, freq : 600000 freq(%) : 10.18867732289107%
total number : 5888890
total time : 6295.513868331909ms
```

c. Boyer-Moore

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 2
Banyak digit Champernowne : 1000000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 3
pattern : 0, freq : 488890 freq(%) : 8.30190409398036%
pattern : 1, freq : 600000 freq(%) : 10.18867732289107%
pattern : 2, freq : 600000 freq(%) : 10.18867732289107%
pattern : 3, freq : 600000 freq(%) : 10.18867732289107%
pattern : 4, freq : 600000 freq(%) : 10.18867732289107%
pattern : 5, freq : 600000 freq(%) : 10.18867732289107%
pattern : 6, freq : 600000 freq(%) : 10.18867732289107%
pattern : 7, freq : 600000 freq(%) : 10.18867732289107%
pattern : 8, freq : 600000 freq(%) : 10.18867732289107%
pattern : 9, freq : 600000 freq(%) : 10.18867732289107%
total number : 5888890
total time : 9590.156555175781ms
```

4. Pengujian dengan bilangan Pi (1000000 angka)

a. Brute Force

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 3
Banyak digit Pi (belakang koma) : 1000000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 1
pattern : 0, freq : 99959 freq(%) : 9.9959%
pattern : 1, freq : 99758 freq(%) : 9.9758%
pattern : 2, freq : 100026 freq(%) : 10.002600000000001%
pattern : 3, freq : 100229 freq(%) : 10.0229%
pattern : 4, freq : 100230 freq(%) : 10.023%
pattern : 5, freq : 100359 freq(%) : 10.0359%
pattern : 6, freq : 99548 freq(%) : 9.9548%
pattern : 7, freq : 99800 freq(%) : 9.98%
pattern : 8, freq : 99985 freq(%) : 9.9985%
pattern : 9, freq : 100106 freq(%) : 10.0106%
total number : 1000000
total time : 1869.9674606323242ms
```

b. KMP

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 3
Banyak digit Pi (belakang koma) : 1000000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 2
pattern : 0, freq : 99959 freq(%) : 9.9959%
pattern : 1, freq : 99758 freq(%) : 9.9758%
pattern : 2, freq : 100026 freq(%) : 10.002600000000001%
pattern : 3, freq : 100229 freq(%) : 10.0229%
pattern : 4, freq : 100230 freq(%) : 10.023%
pattern : 5, freq : 100359 freq(%) : 10.0359%
pattern : 6, freq : 99548 freq(%) : 9.9548%
pattern : 7, freq : 99800 freq(%) : 9.98%
pattern : 8, freq : 99985 freq(%) : 9.9985%
pattern : 9, freq : 100106 freq(%) : 10.0106%
total number : 1000000
total time : 987.6470565795898ms
```

c. Boyer-Moore

```
Pilih masukkan angka :
1). Input Manual
2). Bilangan Champernowne
3). Pi
Pilih salah satu opsi (1/2/3) : 3
Banyak digit Pi (belakang koma) : 1000000
Masukkan banyak digit dalam suatu sekuens : 1
Pilihan Algoritma Pencarian
1). Brute Force
2). Knuth Morris Pratt
3). Boyer-Moore
Pilih algoritma (1/2/3) : 3
pattern : 0, freq : 99959 freq(%) : 9.9959%
pattern : 1, freq : 99758 freq(%) : 9.9758%
pattern : 2, freq : 100026 freq(%) : 10.002600000000001%
pattern : 3, freq : 100229 freq(%) : 10.0229%
pattern : 4, freq : 100230 freq(%) : 10.023%
pattern : 5, freq : 100359 freq(%) : 10.0359%
pattern : 6, freq : 99548 freq(%) : 9.9548%
pattern : 7, freq : 99800 freq(%) : 9.98%
pattern : 8, freq : 99985 freq(%) : 9.9985%
pattern : 9, freq : 100106 freq(%) : 10.0106%
total number : 1000000
total time : 1708.3849906921387ms
```

IV. Analisis dan Kesimpulan

A. *Analisis Hasil Uji*

Setelah melakukan pengujian terhadap algoritma brute Force, Boyer-Moore dan Knuth-Morris-Pratt. Didapatkan table hasil sebagai berikut:

TABLE I. Tabel Hasil Uji dengan bilangan Champernowne

Bilangan	Table Column Head		
	Algoritma	***Banyak Angka***	***Waktu(ms)***
Champernowne	Brute Force	1000	5.8
Champernowne	KMP	1000	4.5
Champernowne	Boyer-Moore	1000	2.9
Champernowne	Brute Force	1000000	8206
Champernowne	KMP	1000000	6295
Champernowne	Boyer-Moore	1000000	9590

TABLE II. Tabel Hasil Uji dengan bilangan Pi

Bilangan	Table Column Head		
	Algoritma	***Banyak Angka***	***Waktu(ms)***
Pi	Brute Force	1000	1.63
Pi	KMP	1000	0.9
Pi	Boyer-Moore	1000	2.5
Pi	Brute Force	1000000	1869
Pi	KMP	1000000	987
Pi	Boyer-Moore	1000000	1708

Dari Tabel hasil uji tersebut dapat dilihat bahwa ketiga algoritma sudah cukup efisien dalam pencocokannya. Hasil juga sudah cukup akurat dengan yang diteorikan bahwa kedua Bilangan Champernowne dan pi adalah bilangan normal dikarenakan frequensinya yang mendekati 10% untuk setiap bilangan. Algoritma KMP terlihat yang paling konsisten dibandingkan yang lain dalam kecepatannya hanya kalah akan Boyer-Moore saat tes Champernowne 1000 angka .Namun pada jumlah angka yang jauh lebih tinggi (1000000) terlihat bahwa kecepatan dari program masih kurang.

B. *Kesimpulan*

Dari penelitian ini, penulis dapat menyimpulkan bahwa algoritma KMP adalah algoritma yang paling baik untuk dipakai dalam pengecekkan angka normal. Namun sebenarnya ketiga algoritma masih belum optimal dan pastinya belum cukup untuk membuktikan ke normalan bilangan Pi pada keseluruhannya.

LINK KODE PROGRAM

Repository:
https://github.com/Rapa285/AngkaNormalChecker

UCAPAN TERIMA KASIH

Syukur Alhamdulillah penulis sampaikan kepada Tuhan Yang Maha Esa atas berkat rahmat dan kasih karunia-Nya, sehingga penulis dapat menyelesaikan makalah yang berjudul "Penggunaan String Matching dalam Pencarian Normalitas suatu Bilangan" dengan cukup baik dan tepat waktu. Penulis juga ingin mengucapkan terima kasih kepada keluarga yang telah memberikan dukungan sehingga makalah IF2211 Strategi Algoritma – Sem. II Tahun 2023/2024 ini dapat diselesaikan.

Ucapan terima kasih juga ditujukan kepada para dosen pengampu mata kuliah IF2211, terutama kepada Ir. Rila Mandala, M.Eng., Ph.D., dan Monterico Adrian, S.T., M.T., yang telah membimbing dan memberikan sumber belajar yang berharga. Penulis berharap bahwa makalah ini dapat bermanfaat sebagai referensi bagi para pelajar yang tertarik dengan keilmuan terkait atau sebagai referensi di masa mendatang.

REFERENCES

[1] Munir, Rinaldi, "Pencocokan String (String/Pattern Matching) Bahan Kuliah IF2211 Strategi Algoritma". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020 2021/Pencocokan-string-2021.pdf (diakses pada 11 Juni 2024).

[2] Charras, Cristian ; Lecroq, Thierry , "Handbook of Exact String Matching Algorithms". https://www.researchgate.net/publication/220693416_Handbook_of_Exact_String_Matching_Algorithms (diakses pada 12 Juni 2024)

[3] Becher, Veronica; Figueira , Santiago, "An example of a computable absolutely normal number". https://www-2.dc.uba.ar/staff/becher/papers/becherTCS2002.pdf (diakses pada 12 Juni 2024)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Pradipta Rafa Mahesa 13522162