Penerapan Algoritma String Matching Knuth-Morris-Pratt Untuk Mencari Modifikasi Pada Kode

Afifah Fathimah Qur'ani (13519183)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519183@std.stei.itb.ac.id

***Abstract*—Algoritma *string matching* atau pencocokan string adalah salah satu algoritma yang penting dalam pengolahan data, dimana fungsi utamanya untuk membandingkan dua buah informasi dan mencari kemunculan suatu pola tertentu pada teks informasi. Algoritma Knuth-Morris-Pratt adalah algoritma pencocokan string yang efektif untuk suatu pola yang spesifik, berbeda dengan Regular Expression yang dapat menerima pola non-spesifik. Oleh karena itu, penyelesaian masalah pada makalah ini akan Menggunakan algoritma Knuth-Morris-Pratt.**

Keywords—Pencocokan String; String Matching; Algoritma Knuth-Morris-Pratt; Modifikasi;

I. PENDAHULUAN

Dalam keseharian, ada banyak sekali kasus-kasus dimana diperlukannya pembandingan antara dua buah informasi. Terutama pada bidang pemrograman, merupakan hal yang sangat penting untuk mengetahui perubahan atau modifikasi yang terjadi pada sebuah kode didalam program.

Pentingnya mengetahui modifikasi kode ini akan sangat dirasakan ketika melakukan kolaborasi dengan orang lain dalam pembuatan program. Selain itu, modifikasi kode ini juga perlu diketahui untuk membuat sistem untuk *version control*.

Pada proses pengecekan modifikasi ini, algoritma yang digunakan adalah algoritma Knuth-Morris-Pratt.

II. DASAR TEORI

A. Pencocokan String atau String Matching

Pencocokan string adalah proses mencari kemunculan antara suatu *pattern* atau pola dengan string lain yang merupakan bagian dari teks yang lebih besar. String matching dapat membandingkan teks yang terdiri dari berbagai macam karakter, baik alfabet A-B, *binary*, alfabet DNA (seperti A, C, G, T), dan karakter-karakter unicode lain.

Dalam pengaplikasiannya, proses pencocokan string ini bergantung sekali dengan *encoding*, tergantung kepada algoritma yang digunakan.

Beberapa batasan atau *constraint* yang biasa digunakan adalah

- Pola akan dinilai ditemukan ketika cocok dengan teks yang berbentuk kata utuh, bukan sub-kata
- Mengabaikan penulisan huruf kapital
- Mengabaikan *multiple whitespace*

Dan batasan-batasan lain untuk memastikan proses pencocokan string berjalan lancar tanpa munculnya *error*.

B. Algoritma Knuth-Morris-Pratt

Perbedaan algoritma KMP dari *naive string matching* adalah pergeseran ketika ditemukan ketidakcocokan. Pada *naive string matching*, pergeseran atau *shifting* hanya dilakukan satu langkah ke kanan, dan pencocokan akan diulang dari awal pola. Sedangkan pada KMP, dapat dilakukan pergeseran beberapa karakter sekaligus sehingga dapat mengurangi pembandingan yang tidak diperlukan dan akan mempercepat kerja program.

KMP dapat bergeser secara efektif dikarenakan dilakukannya preprocessing pada pola sehingga dapat diketahui sebuah komponen *Longest Proper Prefix Which Is Suffix* (LPS), yang akan dibuat berdasarkan pola yang dicari sebelum dijalankannya program utama. LPS merupakan array dari sebuah string pada sebuah posisi tertentu yang merupakan *proper prefix* (artinya karakter terakhir dalam string tidak termasuk prefiks) yang juga merupakan sufiks. LPS ini dimanfaatkan untuk pergeseran terbesar yang dapat dilakukan untuk menghindari pembandingan yang tidak diperlukan.

Pada preprocessing dimanfaatkan sebuah fungsi pinggiran atau *KMP Border Function*,

$$b(k) = \text{size of largest prefix of } P[0..k] \text{ that is also a suffix of } P[1..k] \quad (1)$$

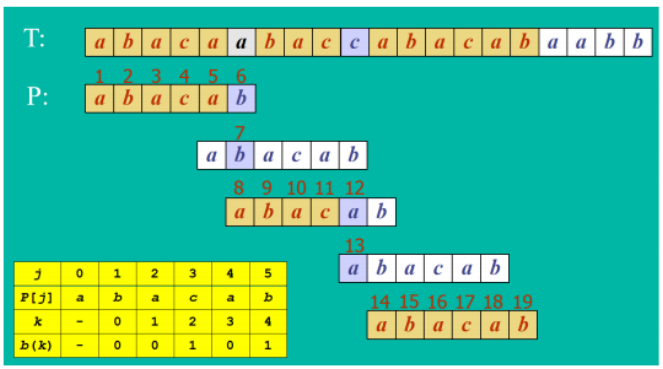
dan berikut ini contoh fungsi pinggiran untuk sebuah string "abaaba"

j	0	1	2	3	4	5
P[j]	*a*	*b*	*a*	*a*	*b*	*a*
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

Gambar 1. Border function untuk pola "abaaba"

Pertama-tama diberikan indeks dari 0 hingga k untuk setiap karakter pada string “abaaba” yang disimpan pada baris j. Kemudian dihasilkan baris k yang merupakan hasil pengurangan 1 satuan pada setiap nilai di baris j. Barulah dilakukan penghitungan fungsi pinggiran untuk setiap nilai di baris k. Pada karakter dengan indeks j bernilai 0 tentunya akan menghasilkan nilai b(k) yang tidak bisa dihitung. Begitu juga dengan indeks k bernilai 0 dan 1, akan menghasilkan b(k) bernilai 0 karena belum memenuhi definisi dari fungsi pinggiran b(k). Sedangkan contohnya pada karakter indeks k = 4 akan menghasilkan P[0..k] = abaab dan P[1..k] = baab dan ditemukan LPS dari pola “abaaba” adalah 2.

Berikut ini adalah visualisasi dari kerja algoritma KMP untuk mencari pola “abacab” dalam string panjang yang tertera



Gambar 2. Visualisasi KMP untuk pola “abacab”

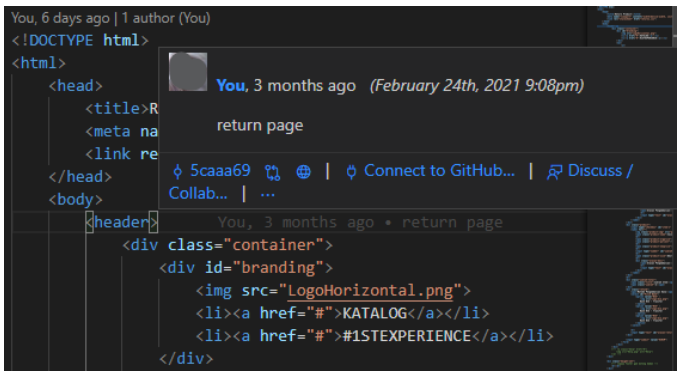
jumlah perbandingan karakter yang dilakukan pada pencocokan string diatas adalah 19 kali, terbukti jauh lebih efektif dari *naive string matching*.

C. *Mencari Modifikasi Pada Kode*

Modifikasi atau perubahan pada kode terjadi ketika programmer mengubah isi kode tersebut, baik pengurangan/penghapusan kode maupun penulisan kode baru. Pada sistem git, terdapat fitur history pada setiap file yang diunggah sehingga dapat dilihat perubahan-perubahan isi file pada setiap unggahan yang dilakukan. Anda juga dapat mengecek perubahan file-file ini melalui ekstensi yang tersedia di editor seperti GitLens pada Visual Studio Code.

Gambar 3. History suatu file pada Git

Fitur untuk melihat perubahan ini sangat berguna untuk *debugging* ketika terjadi kesalahan dalam program. Apabila kode dibuat oleh beberapa programmer, fitur ini juga berfungsi untuk mengecek pekerjaan programmer lain, kemudian menentukan apakah program dapat di merge atau tidak. Dan pastinya sangat berguna untuk fungsi utama git, yaitu sistem untuk *version control*.



Gambar 4. GitLens pada Visual Studio Code

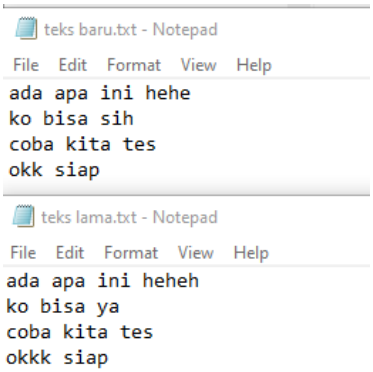
III. Algoritma KMP Untuk Mencari Modifikasi Kode

Proses pencocokan string pada permasalahan di makalah ini akan dilakukan antara dua buah teks yang dimisalkan versi kode terbaru dan versi kode lama. Kode solusi dibuat dalam bahasa Python. Pertama-tama akan dilakukan pembacaan file kode.

```
def read_txt(filename):
    hasil = []

    try:
        txtFile = open(filename, "r")
        lines = txtFile.readlines()
        txtFile.close()

        for line in lines:
            hasil.append(line)

    except FileNotFoundError:
        print("[ERROR] : File tidak ditemukan!")

    return hasil
```

Gambar 5. Fungsi untuk membaca file kode

Misalnya pengecekan string akan menggunakan dua teks berikut, dimisalkan berisi baris-baris kode dari suatu program



Kemudian proses pencocokan string ini akan dijalankan berdasarkan baris pada file kode, sehingga file yang dibaca akan dipisahkan menjadi baris-baris terpisah. Hal ini dilakukan agar kode dapat berjalan lebih efektif. Berikut ini implementasi kode KMP dalam bahasa Python

```python
def KMPSearch(pattern, txt):
    M = len(pattern)
    N = len(txt)

    lps = [0]*M # longest prefix suffix
    j = 0 # index pattern[]

    maks = 0
    count = 0

    # Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pattern, M, lps)

    i = 0 # index txt[]
    while i < N:
        if pattern[j] == txt[i]:
            i += 1
            j += 1
            count += 1
            maks = max(count, maks)

        if j == M:
            j = lps[j-1]
            count = j

            return (i-j)
        elif i < N and pattern[j] != txt[i]:
            if j != 0:
                j = lps[j-1]
                count = j
            else:
                i += 1
                count = 0

    return None
```

Gambar 6. Implementasi Algoritma KMP

Seperti yang sudah dijelaskan sebelumnya, algoritma KMP menggunakan komponen LPS. Oleh karena itu implementasi KMP diatas akan memanggil fungsi computeLPSArray yang akan mengkalkulasi LPS dengan memanfaatkan fungsi pinggiran seperti yang telah dijelaskan sebelumnya.

```python
def computeLPSArray(pattern, M, lps):
    len = 0 # panjang prefix suffix terbesar sebelumnya

    lps[0] = 0 # inisialisasi lps[0] selalu 0
    i = 1

    # the loop hitung lps[i] dari i = 1 ke M-1
    while i < M :
        if pattern[i] == pattern[len] :
            len += 1
            lps[i] = len
            i += 1
        else :
            if len != 0:
                len = lps[len-1]
            else :
                lps[i] = 0
                i += 1
```

Gambar 7. Implementasi Fungsi Pinggiran Untuk Mencari LPS

Fungsi KMPSearch diatas akan mengembalikan indeks terakhir karakter dari pola yang ditemukan pada string. Misalkan pada string “Siapa pembuat algoritma KMP?”, jika digunakan pola “Siapa” program akan mengembalikan nilai 5 dan jika digunakan pola “pembuat” maka program akan mengembalikan nilai 18.

Kemudian algoritma KMP tadi akan dihubungkan dengan teks kode yang sudah dibaca program diawal tadi, menggunakan kode dibawah ini

```python
print(bcolors.HEADER + "teks lama" + bcolors.ENDC)
j = 0
noline = False
for i in range (len(content1)):
    status = []
    if (noline):
        i -= 1

    for word in content1[i]:
        if (KMPSearch(word, test1[i]) == KMPSearch(word, test2[j])
        and KMPSearch(word, test2[j]) != None) :
            print(word, end=" ")
            status.append(True)
        else:
            status.append(False)
            if (len(status) == len(content1[i]) and cekStatus(status)) :
                j += 1
                noline = True
                break
            else :
                print(bcolors.WARNING + word + bcolors.ENDC, end=" ")

        if (len(splitWord(test1[i])) > 1 and len(splitWord(test2[j])) > 1):
            test1[i] = splitWord(test1[i])[1]
            test2[j] = splitWord(test2[j])[1]

    print("")
    if not(noline):
        j += 1

print("")
```

Gambar 7. Menghubungkan algoritma KMP ke teks kode

Kode diatas ini akan dihubungkan baik dengan kode versi awal maupun dengan kode versi terbaru, sehingga kedua kode akan saling dibandingkan. Potongan gambar 7 diatas merupakan kode untuk membandingkan kode versi lama dengan kode versi terbaru.

Sehingga ketika dijalankan, kode akan menghasilkan output berikut

```
PS D:\task\itb\semester 4\Strategi Algoritma\makalah> python -u "d:\
 4\Strategi Algoritma\makalah\Main.py"
teks lama
ada apa ini heheh
ko bisa ya
coba kita tes
okkk siap

teks baru
ada apa ini hehe
ko bisa sih
coba kita tes
okk siap
```

Gambar 8. Hasil output program

Dapat dilihat perbedaan antar kedua file kode akan di *highlight* sehingga memudahkan programmer untuk membandingkan kedua file.

IV. KESIMPULAN

Pada era dimana programming semakin banyak dikuasai dan dilakukan, semakin tinggi pula kebutuhan untuk membandingkan antara kode-kode program yang dibuat. Kebutuhan tersebut didasar berbagai tujuan, mulai dari menghubungkan kerja dari berbagai programmer untuk menghasilkan kode yang berjalan baik, hingga adanya keperluan untuk merilis dan mengontrol versi-versi dalam program.

Algoritma Knuth-Morris-Pratt merupakan salah satu pilihan algoritma yang bisa digunakan dalam pencocokan string. Algoritma ini dinilai jauh lebih efektif daripada *naive string matching* karena adanya pergeseran terbesar menggunakan komponen LPS.

Pada makalah ini dilakukan perbandingan antara dua versi kode, dan algoritma KMP digunakan untuk mencari perbedaan diantara kedua versi tersebut. Hasil keluarannya adalah isi masing-masing kode tersebut, tetapi pada perbedaan yang ditemukan akan dicetak dengan warna kuning sebagai penanda.

Kelebihan dari algoritma KMP ini adalah arah pemrosesannya, dimana tidak akan terjadi pergeseran mundur sehingga algoritma ini cocok untuk memproses file yang sangat besar, sebagaimana sifat-sifat program yang biasanya terdiri dari sangat panjang kode.

UCAPAN TERIMA KASIH

Penulis bersyukur kepada Allah swt. atas segala rahmat dan berkahnya sehingga penulis dapat menyelesaikan malah ini. Selain itu penulis juga mengucapkan terima kasih kepada

1) Orang tua penulis yang senantiasa mendukung dan memberi bimbingan

2) Seluruh dosen dan tim pengajar serta asisten mata kuliah IF2211 Strategi Algoritma tahun akademik 2020/2021

3) Teman-teman penulis yang selalu saling membantu dan menyemangati

REFERENCES

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/ diakses pada 11 Mei 2021.

[2] https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/ diakses pada 11 Mei 2021.

[3] Lefarov, Max. "Pattern Search with the Knuth-Morris-Pratt (KMP) Algorithm". Dari https://towardsdatascience.com/pattern-search-with-the-knuth-morris-pratt-kmp-algorithm-8562407dba5b diakses pada 11 Mei 2021.

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, 11 Mei 2021

Ttd

Afifah Fathimah Qur'ani 13519183