Algoritma *FJS* Dalam Menemukan Jumlah Kemunculan Pola Khusus di Rantai DNA

Reyvan Rizky Irsandy - 13518136
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518136@std.stei.itb.ac.id

***Abstraksi* — Algoritma pencocokan pola dalam suatu teks seringkali digunakan untuk memproses suatu informasi spesifik seperti Search Engine, Pencarian teks pada teks editor, Analisis Citra, Bioinformatika dan lain-lain. Pencocokan pola sangat berguna di dunia Bioinformatika contohnya seperti menganalisis rantai DNA dari suatu virus, menemukan pola tidak biasa yang terkandung dalam DNA. Dalam makalah ini membahas aplikasi pencocokan string dalam bidang Bioinformatika yaitu dengan menemukan jumlah kemunculan pola spesifik dalam rantai DNA menggunakan algoritma FJS.**

Kata Kunci* — *Pattern Matching, DNA sequence, FJS algorithm, KMP algorithm, BMS algorithm

I. Pendahuluan

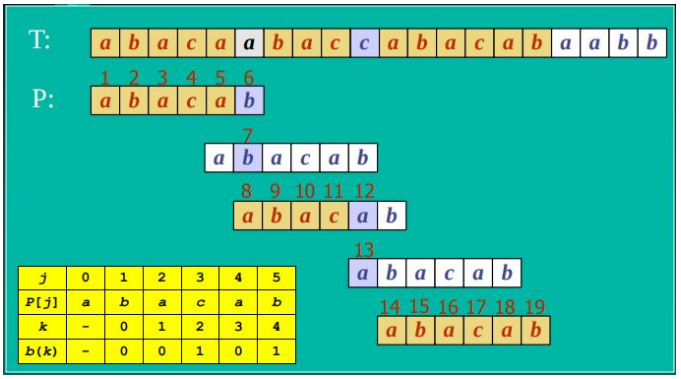
Pencocokan pola merupakan komponen vital pada bidang Bioinformatika sebagai bentuk pengaplikasian teknologi komputer untuk menganalisis data biologis subjek yang diteliti. Pada bidang ini teknologi komputer digunakan untuk mengumpulkan, menganalisis, menyimpan dan menyatukan data biologis suatu subjek yang diteliti, seperti nama dan cara kerjanya Bioinformatika adalah bidang yang menengahi biologi dan informatika. Pencocokan pada DNA adalah problem dimana kita dituntut untuk menemukan suatu subrantai spesifik yang ada pada rantai DNA, dengan menyelesaikan permasalahan ini dapat ditarik suatu informasi mengenai DNA subjek. Permasalahan dalam hal ini adalah waktu, mengingat rantai DNA suatu subjek sangatlah panjang pencocokan pola dapat menjadi operasi yang mahal dalam artian komputasi, maka diperlukanlah algoritma yang efektif dan efisien dalam menemukan pola spesifik ini.

DNA (*Deoxyribonucleic acid*) atau asam nukleat adalah sejenis biomolekul yang menyimpan dan menyandi instruksi-instruksi genetika setiap organisme (termasuk virus). Asam Nukleat biasanya disebut *double helix* karena merupakan molekul beruntai ganda. DNA terdiri dari satuan-satuan molekul yang disebut nukleotida, dan tiap nukleotida terdiri atas salah satu jenis basa nitrogen (Guanin(G), Adenin(A), Timin(T), Sisosin(C)). DNA dapat direpresentasikan dengan *String*. Contohnya : ATGCTGATGCATGATGCATG. Kemunculan pola spesifik pada DNA dapat menghasilkan suatu informasi yang berguna untuk penelitian medis maka algoritma yang efektif dan efisien dalam pencocokan string sangatlah vital untuk mendapatkan data yang dapat dianalisis dari DNA suatu organisme. Contohnya adalah jumlah kemunculan rantai DNA X pada penderita HIV, dimana X adalah suatu ekspresi untuk pola spesifik yang dicari pada DNA yang diteliti.

II. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt atau KMP adalah algoritma yang mencari suatu pola dalam suatu teks panjang dengan urutan pencocokan dari kiri ke kanan. Pencocokan pola ini hampir sama dengan algoritma *Brute-Force* namun perbedaanya adalah pergeserannya, pada algoritma KMP pergeseran dilakukan lebih cerdas dibanding algoritma *Brute-Force*.

Pada *Brute-Force* apabila terjadi *mismatch* pergeseran dilakukan sebanyak satu karakter atau maju satu karakter dan dilakukan terus menerus sampai akhir dari teks. Algoritma seperti ini kurang efisien. Berbeda dari *Brute-Force,* pergeseran pada algoritma KMP dilakukan dengan mengevaluasi suatu fungsi yang menentukan jumlah pergeseran, fungsi ini disebut *Border Function*.



Gambar 2.1, Algoritma KMP

Cara kerja Algoritma KMP dapat dilihat dari gambar terkait, Tabel kuning di pojok bawah adalah tabel evaluasi *Border Function,* terlihat terjadi *mismatch* pada karakter ke-6 pada teks dan karakter ke-6 pada pola, algoritma ini akan mencari indeks pada pola yang akan dicocokan selanjutnya melalui evaluasi *Border Function,* diketahui bahwa nilai border function adalah 1, jadi pencocokan selanjutnya akan dimulai lagi pada karakter ke-6 pada pada teks dan karakter ke-2 atau karakter dengan indeks 1 pola, hal ini akan terus dilakukan sampai algoritma menemukan solusi atau semua karakter pada teks telah selesai dicocokkan

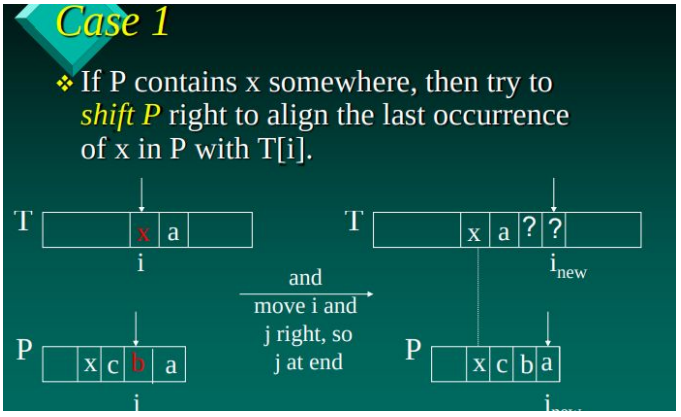
```
# Computing border Funct
def computeBorder(pattern):
    borderFunct = []
    borderFunct.append(0)
    m = len(pattern) # pattern len
    j = 0 # prefx
    i = 1 # suffx
    while(i<m):
        if(pattern[i] == pattern[j]):
            borderFunct.append(j+1)
            i+=1
            j+=1
        elif(j>0):
            j = borderFunct[j-1]
        else:
            borderFunct.append(0)
            i+=1
    return borderFunct
def kmpMatch(text,pattern):
    n = len(text)
    m = len(pattern)
    borderFunct = computeBorder(pattern)
    i = 0
    j = 0
    while(i<n):
        if(pattern[j]==text[i]):
            if(j == m-1):
                return i - m +1
            i+=1
            j+=1
        elif(j>0):
            j = borderFunct[j-1]
        else:
            i+=1
    return -1
```

Gambar 2.2, Implementasi KMP

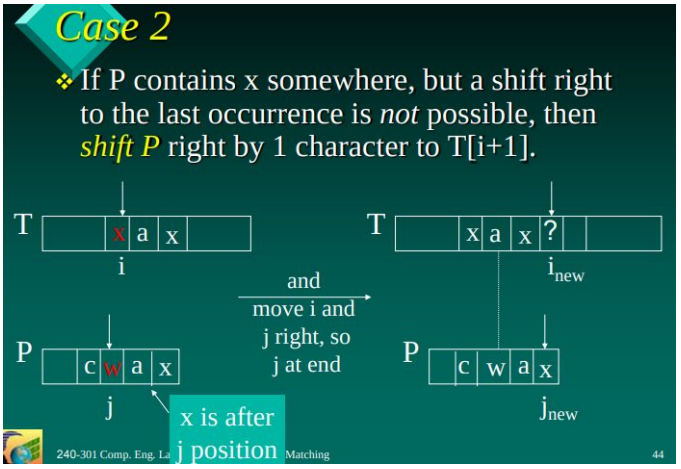
III. Algoritma Boyer-Moore

Algoritma Boyer-Moore atau BM adalah algoritma pencocokan pola pada teks yang didasar oleh dua teknik yaitu *looking-glass technique* dan *character-jump technique*, Algoritma ini lebih cerdas dari algoritma *Brute-Force*, yang membedakan algoritma ini dengan algoritma KMP adalah jika di KMP terdapat *Border Function* di BM terdapat *Last Occurrence Function* yang berguna untuk menentukan state selanjutnya pada prosess pencocokan pola.

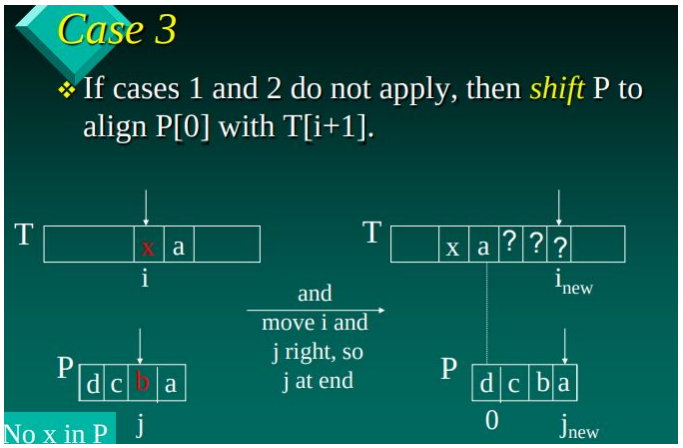
Teknik *Looking-glass* adalah teknik dimana pencocokan pola pada teks seolah-olah dimulai dari belakang atau terbalik, dan Teknik *character-jump* adalah teknik yang mengatasi kemungkinan bila terjadi *mismatch* ada 3 kasus bila terjadi *mismatch*. Dijelaskan pada beberapa gambar berikut :



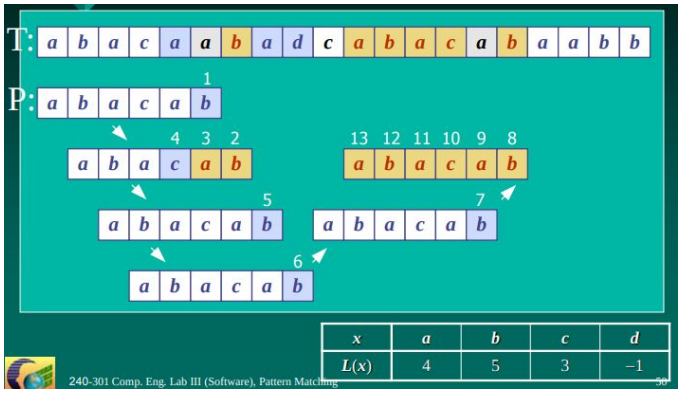
Gambar 3.1, Case 1



Gambar 3.2, Case 2



Gambar 3.3, Case 3



Gambar 3.4, Contoh Algoritma BM

Cara kerja algoritma *Boyer-Moore* direpresentasikan dalam beberapa gambar terkait. Tabel di bagian pojok bawah adalah representasi hasil dari *Last-Occurrence Function*, terlihat bahwa terjadi mismatch saat mencocokkan karakter ke-6 pada teks dan karakter ke-6 pada pola, mismatch ini masuk ke kasus ke 1 dimana proses pencocokkan selanjutnya akan dimulai dengan menggeser pola sejajar dengan kemunculan "a" terakhir dilihat dari tabel pojok bawah yang merupakan representasi *Last-Occurrence Function*. Proses ini dilakukan berulang-ulang sampai ditemukan pola terkait atau sampai teks selesai di cek semua.

```
def lastOccurence(pattern):
    lastOccur = []
    for i in range (128):
        lastOccur.append(-1)
    for i in range (len(pattern)):
        lastOccur[ord(pattern[i])] = i
    return lastOccur

def bmMatch(text,pattern):
    lastOccur = lastOccurence(pattern)
    n = len(text)
    m = len(pattern)
    i = m-1
    if(i > n-1):
        return -1
    j =  m-1
    while(i <= n-1):
        if(pattern[j]==text[i]):
            if(j == 0):
                return i
            i-=1
            j-=1
        else:
            lo = lastOccur[ord(text[i])]
            i = i + m - min(j,1+lo)
            j = m - 1
    return -1
```

Gambar 3.5 , Implementasi Algoritma BM

IV. ALGORITMA FJS

Algoritma FJS adalah algoritma yang diciptakan atau ditemukan oleh Fransitek Franek,Christopher G. Jennings, dan W. F. Smyth pada masa studinya memperoleh gelar Master. Jennings dan timnya memperkenalkan algoritma ini sebagai algoritma general tercepat untuk menemukan semua keberadaan pola tertentu dalam suatu teks.

Algoritma FJS adalah algoritma hybrid dari KMP dan BM, yang artinya algoritma ini telah dibuat sedemikian rupa sehingga menyatukan kedua algoritma KMP dan BM, Kita tahu bahwa KMP menyelesaikan persoalan dimana algoritma naive atau *Brute-Force* mengecek kembali string yang telah dicek benar, dengan KMP kelemahan ini bisa di solve namun ada kelemahan lain yaitu jika charset jumlahnya tinggi maka algoritma ini bisa lama, namun kita tahu bahwa Algoritma Boyer-Moore menyelesaikan typical error ini. Jadi bila kedua algoritma ini digabungkan terciptalah suatu algoritma yang mangkus untuk menemukan pola dari suatu teks. Cara kerja FJS adalah dengan menggabungkan *character-jump technique, looking-glass technique* dengan Algoritma KMP sehingga KMP akan dapat menangani typical error, FJS menggunakan kedua array yang didapat dari *Border Function* dan *Last Occurrence Function* hasilnya adalah 2 tabel yaitu β(Modifikasi *Border Function*) dan Δ(BM array). Pada kasus terburuk algoritma ini pencocokan pola tidak lebih dari m-n+1 , m adalah panjang teks, n adalah panjang pola.

Telah dibuktikan bahwa FJS lebih cepat 5%-10% dari algoritma Boyer-Moore. Kompleksitas algoritma ini adalah O(m+α).

```
def makeBeta(pattern):
    m = len(pattern)
    i = 0
    j = -1
    beta = []
    beta.append(-1)
    while(i<m):
        while((j>-1) and (pattern[i] != pattern[j])):
            j = beta[j]
        i+=1
        j+=1
        if((i<m) and (pattern[i] == pattern[j])):
            beta[i] = beta[j]
        else:
            beta[i] = j
    return beta

def makeDelta(pattern):
    m = len(pattern)
    delta = []
    for i in range(128):
        delta.append(m+1)
    for i in range(m):
        ch = pattern[i]
        slot = ord(ch) & 127
        jump = m-i
        if(jump < delta[slot]):
            delta[slot] = jump
    return delta
```

Gambar 4.1, Implementasi Algoritma untuk mencari tabel Beta dan Delta

```
def FJSsearch(pattern, text):
    n = len(text)
    m = len (pattern)
    if(m!=0 and n>m):
        beta = makeBeta(pattern)
        delta = makeDelta(pattern)
        arrRes = []
        mp = m-1
        np = n-1
        i = j = 0
        ip = i+mp
        while(ip<np):
            if(j<=0):
                while(pattern[mp] != text[ip]):
                    ip += delta[text[ip+1] & 127]
                    if(ip>=np):
                        return -1
                j = 0
                i = ip-mp
                while( (j<mp) and (text[i] == pattern[j])):
                    i+=1
                    j+=1
                if(j==mp):
                    arrRes.append(i-mp)
                    i+=1
                    j+=1
                if(j<=0):
                    i+=1
                else:
                    j = beta[j]
            else:
                while( (j<m) and (text[i] == pattern[j])):
                    i+=1
                    j+=1
                if(j==m):
                    arrRes.append(i-m)
                j = beta[j]
            ip = i + mp - j
        if(ip == np):
            if(j<0):
                j = 0
            i = n - m + j
            while ( (j<m) and (text[i] == pattern[j])):
                i+=1
                j+=1
            if(j == m):
                arrRes.append(n-m)
        return arrRes
```

Gambar 4.2, Implementasi Algoritma FJS

V. Algoritma FJS Untuk Mencari Jumlah Kemunculan Pola Spesifik

Penjelasan mengenai DNA sudah dijelaskan pada bab sebelumnya (Bab I). Dari rantai DNA yang terdapat pada organisme tertentu kita dapat mengambil beberapa informasi penting, misal mendeteksi penyakit tertentu, pola tidak biasa(kelainan) pada DNA, bahkan dapat membantu suatu riset untuk kedepannya, jumlah pola spesifik pada suatu rantai DNA dan masih banyak lagi

Dengan menggunakan algoritma FJS pencarian pola spesifik pada suatu rantai DNA akan lebih cepat mengingat cara kerjanya yang dibuat dari gabungan dua algoritma yang saling melengkapi.

Untuk mempermudah demo kita akan menggunakan Sampel DNA yang mempunyai panjang 26 karakter

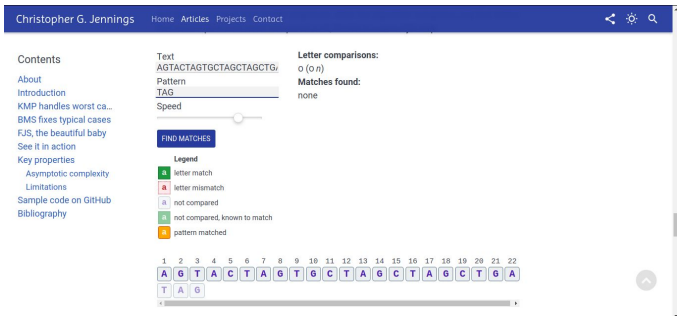
DNA-X : “AGTACTAGTGCTAGCTAGCTGA”

anggap saja pattern yang dicari adalah

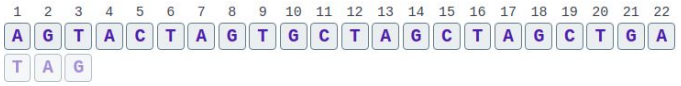
Pattern-Y : “TAG”

untuk mempresentasikan hasil yang didapat dan tahapannya saya menggunakan webapps sang penemu algoritma ini.

https://cgjennings.ca/articles/fjs/#see-it-in-action



Gambar 5.1, Web-apps yang digunakan penulis



Gambar 5.2, Teks yang akan dievaluasi

Jika sekilas dilihat Pola TAG akan ditemukan pada indeks ke 6,12, dan 16 pada teks yang dievaluasi.

Berikut adalah tahapan tahapan yang akan dilakukan oleh algoritma FJS dalam menemukan semua pola yang ada dalam teks.

Gambar 5.3, demo FJS tahap 1

Dapat dilihat bahwa FJS menggunakan *looking-glass technique* namun berbeda sedikit karena setelah mengecek karakter terakhir dia akan mengecek karakter awal pada pola dan disesuaikan di teks, dan terjadi mismatch pada karakter ke - 3 dalam teks, karena FJS juga menggunakan *character-jump technique* dapat dilihat bahwa “T” juga terdapat pada pola, sehingga pada pengecekan selanjutnya “T” di pola akan sejajar dengan “T” di teks.

Gambar 5.4, demo FJS tahap 2

Dapat dilihat bahwa pergeseran sesuai ekspektasi tahap sebelumnya dilakukan, selanjutnya terjadi *mismatch* pada karakter “C”, karena karakter tersebut tidak ada dalam pola, maka pengecekan selanjutnya akan dimulai pada karakter ke-5 (masih menggunakan *character-jump technique)* atau dilakukan pergeseran sebanyak m karakter dimana m adalah panjang pola.

Gambar 5.5, demo FJS tahap 3

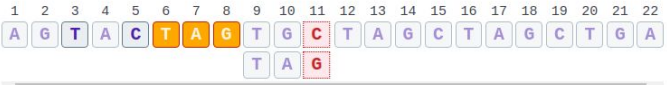
Dapat dilihat pergeseran karena *mismatch* sebelumnya telah dilakukan sesuai ekspektasi, selanjutnya terjadi kecocokan pada karakter yang terakhir di dalam pola, selanjutnya karakter paling awal akan di cek oleh FJS.

Gambar 5.6, demo FJS tahap 4

Dapat dilihat pengecekan terjadi sesuai ekspektasi, dan terjadi kecocokan lagi, selanjutnya karakter akan di cek sesuai dengan algoritma KMP dengan iterasi kedepan.

Gambar 5.7, demo FJS tahap 5

Dapat dilihat bahwa pengecekan terjadi sesuai ekspektasi, dan terjadi kecocokan pertama, setelah itu akan di cek untuk *border function* dengan k = 2, disini border function bernilai 0 jadi pergeseran dilakukan seperti Boyer-Moore dg langsung maju sebanyak m karakter dimana m adalah panjang pola.



Gambar 5.8, demo FJS tahap 6

Dapat dilihat bahwa pergeseran sesuai ekspektasi dan terjadi *mismatch* pada karakter “C”, sama kasusnya pada tahap 2, pergeseran akan dilakukan sebanyak m karakter m adalah panjang pola.

Gambar 5.9, demo FJS tahap 7

Dapat dilihat bahwa pergeseran dilakukan sesuai ekspektasi dan terjadi seperti tahap 3

Gambar 5.10, demo FJS tahap 8

Terjadi kecocokan kedua, selanjutnya terjadi pergeseran seperti yang dilakukan pada pada tahap ke-5

Gambar 5.11, demo FJS tahap 9

Pergeseran terjadi sesuai ekspektasi dan terjadi mismatch pada karakter “A” karena karakter “A” terdapat pada pola, selanjutnya pergeseran dilakukan dengan *character-jump technique* dilakukan dengan mensejajarkan karakter “A” dalam teks dan karakter “A” dalam pola.

Gambar 5.12, demo FJS tahap 10

Pergeseran dilakukan sesuai ekspektasi dan terjadi kecocokan pada karakter terakhir, selanjutnya akan dicek karakter awal sesuai dengan tahap sebelumnya yaitu tahap ke-5

Gambar 5.13, demo FJS tahap 11

Terjadi kecocokan ketiga, selanjutnya terjadi pergeseran seperti yang dilakukan pada tahap ke-5

Gambar 5.14, demo FJS tahap 12

Terjadi pergeseran sesuai ekspektasi, dan karakter terakhir dinyatakan cocok pada pola namun setelah dilanjutkan pengecekannya terjadi *mismatch* pada karakter pertama sehingga akan dilakukan pergeseran maju sekali karena diketahui mismatch pada awal.

Gambar 5.15, demo FJS tahap 13

Terjadi mismatch, hal ini mengakibatkan semua teks sudah di cek, atau pengecekan sudah berakhir karena diketahui karakter terakhir pada teks sudah tidak cocok dengan pola.

didapati hasil akhir adalah terdapat 3 subrantai yang cocok dimulai masing-masing pada karakter ke-6,ke-12 dan ke-16.

Pada demo diatas kita dapat mengerti bahwa FJS memanfaatkan kelebihan algoritma *Knuth-Morris-Pratt* dan melengkapinya dengan algoritma Boyer-Moore, hal ini akan menjadikan algoritma ini sangat efisien dan lebih cepat dari kedua algoritma tersebut secara general.

VI. KESIMPULAN

Algoritma pencocokan pola memiliki banyak sekali aplikasi, salah satunya di bidang Bioinformatika, Dengan algoritma yang efektif dan efisien, pencocokan pola pada kasus tertentu dapat menjadi lebih cepat waktu komputasinya, contohnya adalah pencocokan DNA, DNA terdiri dari rantai yang panjang bahkan sangat panjang sekali, pencocokan secara naive akan membuat waktu komputasi sangatlah lama hal ini adalah hal yang krusial bisa saja ada kasus khusus dimana cek DNA harus dilakukan secara singkat, misal donor suatu organ, apabila pasien telah kritis dan DNA checking berlangsung terlalu lama mungkin saja pasien tidak sempat terselamatkan karena lamanya checking tersebut.

Algoritma FJS sangatlah cocok digunakan pada Bioinformatika dikarenakan sangatlah banyak kasus dimana pencocokan pola dilakukan dengan algoritma ini pencocokan string secara general dapat dilakukan dengan lebih cepat.

Video Link at Youtube

Penulis juga membuat video tentang makalah ini untuk menekankan kejelasan demo algoritma FJS, video dapat diakses pada tautan berikut. https://youtu.be/90kvafv6w-o

Ucapan Terima Kasih

Pertama-tama penulis bersyukur atas nikmat Allah SWT. karena dengan nikmat dan kehendak-Nya penulis dimudahkan dalam menyelesaikan makalah ini dengan baik dan tepat waktu. Penulis ingin mengucapkan terima kasih kepada Ibu Dr. Masayu Leylia Khodra, ST., MT., Ibu Dr. Nur Ulfa Maulidevi, ST., M.Sc., Bapak Dr. Ir. Rinaldi, MT selaku dosen mata kuliah IF2211 Strategi Algoritma atas bimbingannya selama satu semester perkuliahan ini.

Referensi

[1] Munir, Rinaldi, Diktat Kuliah IF2211 Strategi Algoritma. Bandung:Program Studi Teknik Informatika Institut Teknologi

Diakses pada : 2 Mei 2020

[2] The FJS String Matching Algorithm, diakses pada https://cgjennings.ca/articles/fjs/#see-it-in-action, waktu : 2 Mei 2020

[3] KMP Algorithm for Pattern Searching, diakses pada https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/, waktu 2 Mei 2020.

[4] Boyer-Moore Algorithm for Pattern Searching, diakses pada https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/, waktu 2 Mei 2020

[5] Performance Evaluation of Various DNA Pattern Matching Algorithms Using Different Genome Datasets, diakses pada https://www.researchgate.net/publication/331342608_Performance_Evaluation_of_Various_DNA_Pattern_Matching_Algorithms_Using_Different_Genome_Datasets, waktu 2 Mei 2020.

[6] YHow DNA Evidence Works, diakses pada https://science.howstuffworks.com/life/genetic/dna-evidence4.htm, waktu 2 Mei 2020

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, 2 Mei 2020

Reyvan Rizky Irsandy
13518136