Penerapan Algoritma *String* Matching, BFS, dan Regex

Mencari Judul Menggunakan Kata Kunci Masukan dari Hasil Web Scraping Judul Makalah

Andhika Tantyo Anugrah - 13522094
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: andhikatantyoanugerah@gmail.com

***Abstract*—Pada mata kuliah IF2211 Strategi Algoritma, mahasiswa diwajibkan menulis sebuah makalah ilmiah dengan memilih salah satu topik strategi algoritma. Salah satu tujuan penulisan makalah adalah untuk melakukan eksplorasi terhadap isu, metode, dan masalah yang dipelajari dalam pengembangan serta menyebarkan aplikasi yang mendukung teknologi informasi. Karena semua makalah yang dibuat oleh mahasiswa dimuat di dalam *website* dosen, maka semua orang bisa mengaksesnya dan judul makalah antara 1 makalah dengan makalah lain pun tidak boleh sama. Oleh karena itu dibuatlah program menggunakan algoritma BFS, *string matching, dan regex* untuk mencari judul dengan kata kunci tertentu untuk menjawab pertanyaan apakah kata kunci tersebut pernah diusulkan sebagai judul, jika iya, apakah judul yang ingin digunakan sama dengan judul yang sudah pernah diusulkan oleh angkatan-angkatan sebelumnya.**

Kata kunci—makalah; judul; BFS; string matching, regex

I. PENDAHULUAN

Menurut KBBI VI Daring, makalah memiliki 2 arti, yang pertama adalah tulisan resmi tentang suatu pokok yang dimaksudkan untuk dibacakan di muka umum dalam suatu persidangan dan sering disusun untuk diterbitkan, sedangkan yang kedua adalah karya tulis pelajar atau mahasiswa sebagai laporan hasil pelaksanaan tugas sekolah atau perguruan tinggi. Sebelum menulis makalah, topik bahasan perlu ditentukan terlebih dahulu. Topik yang ingin digunakan tentu tidak bisa sama dengan makalah-makalah yang sudah ada sebelumnya, kecuali ingin menganalisis topik itu dari sudut pandang yang berbeda. Tetapi pada umumnya, judul dari satu makalah dengan makalah yang lain berbeda.

Lingkup makalah IF2211 Strategi Algoritma yang dibuat oleh mahasiswa biasanya berhubungan dengan materi yang telah diajarkan di kelas, tetapi tidak menutup kemungkinan untuk menggunakan algoritma lain yang tidak diajarkan di kelas. Setelah diamati sampel dari beberapa judul makalah yang digunakan oleh angkatan-angkatan sebelumnya, dapat dilihat judul yang digunakan memiliki pola yang mirip, yaitu mencangkup nama algoritma yang dibahas oleh penulis makalah. Pola ini bisa dijadikan kandidat kata kunci bagi calon penulis makalah dengan algoritma yang sama untuk memeriksa apakah topik yang berkaitan dengan algoritma tersebut telah diambil oleh penulis-penulis makalah terdahulu.

Makalah Mahasiswa Tahun 2023

1. Analisis Perbandingan Efisiensi Pencarian NIM Mahasiswa Teknik Informatika ITB dengan Metode Binary Search dan Interpolation Search
   Oleh: M.Farrel Danendra Rachim - 13521048
2. Pemecahan Sandi Caesar Dengan Bantuan Algoritma String Matching dan Brute-Force
   Oleh: Naufal Syifa Firdaus - 13521050
3. Perbandingan Algoritma Greedy dan Branch and Bound dalam Penjadwalan Job Freelance yang Memaksimalkan Keuntungan
   Oleh: Melvin Kent Jonathan -13521052
4. Penerapan Dynamic Programming dalam Menentukan Rute Chey-Yo Terpendek pada Robot ABU Robocon 2023
   Oleh: Muhammad Fadhil Amri - 13521066
5. Analisis Perbandingan Waktu Eksekusi de Bruijn Sequence Constructions dengan Greedy Algorithm dan Euler Cycle
   Oleh: Christian Albert Hasiholan - 13521078
6. Application of Branch and Bound Algorithm to Solve Rider-Driver Batched Matching Problem
   Oleh: Puti Nabilla Aidira - 13521088
7. Pengembangan Algoritma Best First Search dalam Pencarian Solusi Game Tree
   Oleh: Tobias Natalio Siampar - 13521090
8. Penerapan Program Dinamis dalam Penyelesaian Masalah Ransum Optimal untuk Hewan Ternak
   Oleh: Maggie Zeta Rosida Simangunsong - 13521117
9. Branch and Bound Algorithm in Decision Making for a Connect Four Game
   Oleh:
10. Rivalitas Algoritme BFS dan DFS dalam Menentukan Arah Tur-Produk Sekuensial untuk Pengguna Baru
    Oleh: Ryan Samuel Chandra - 13521140
11. Perbandingan Efisiensi Algoritma Path Finding untuk Simulasi Permainan Labirin dalam Game Engine
    Oleh: Johanes Lee - 13521148
12. Penerapan Algoritma DFS (Depth-First Search) dalam Pencarian Struktur Dunia Minecraft Menggunakan Custom Mod Minecraft Forge
    Oleh: Brigita Tri Carolina - 13521156
13. Aplikasi Algoritma KMP(Knutt-Morris-Pratt) dalam Pengecekan Plagiarisme
    Oleh: Sulthan Dzaky Alfaro - 13521159
14. Penerapan Flood-Fill Concept dengan Memanfaatkan Algoritma BFS dan DFS dalam Aplikasi Mewarnai
    Oleh: Mohammad Rifqi Farhansyah - 13521166
15. Optimasi Performa Word Search Game Melalui Pemanfaatan Algoritma BFS dan Bruteforce
    Oleh: Satria Octavianus Nababan - 13521168
16. A Case Study in Optimization of Brute-Force Algorithm : Finding Longest Substring With Unique Characters
    Oleh: Fatih Nararya Rashadyfa Ilhamsyah - 13521060
17. A Comparative Study of Backtracking and Bruteforce Algorithms for Kakuro Puzzle Solving
    Oleh: Tabitha Permalla - 13521111
18. A* Algorithm for Solving Sokobond Puzzles
    Oleh: Addin Munawwar Yusuf - 13521085
19. Algoritma Greedy pada Pemilihan Target Serangan dalam Permainan Mobile Legends
    Oleh: Ilham Akbar - 13521068
20. An Analysis of a Branch and Bound Approach with Apriori Algorithm in Finding Frequent Item Sets for an E-Commerce Recommender System
    Oleh: Enrique Alifio Ditya - 13521142

Gambar 1. Daftar judul makalah IF2211 Strategi Algoritma yang dibuat pada tahun 2023, perhatikan dari 20 sampel yang diambil, semuanya selalu mencantumkan algoritma yang digunakan pada judul makalah

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Makalah2023.htm

Traversal graf merupakan salah satu algoritma pengunjungan simpul-simpul di dalam graf dengan cara yang sistematik. Algoritma traversal graf dibagi menjadi dua, yaitu pencarian melebar (*breadth first search*) atau BFS dan pencarian mendalam (*depth first search*) atau DFS. Dalam proses pencarian solusi dengan traversal graf, terdapat dua pendekatan yaitu pendekatan dengan graf statis dan pendekatan dengan graf dinamis. Graf statis terbentuk sebelum proses

pencarian dilakukan, sedangkan graf dinamis dibangun selama pencarian solusi.

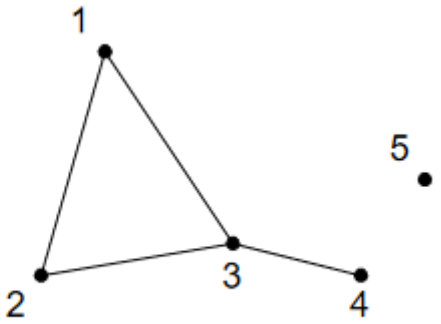
Algoritma BFS adalah salah satu algoritma traversal graf dengan model pencarian melebar untuk mengunjungi simpul-simpul di dalam graf dengan cara yang sistematik. Algoritma BFS (dan DFS) cocok digunakan untuk *web spider*, yaitu kakas yang digunakan untuk mengunjungi halaman-halaman web secara periodik. Di sini, halaman web dimodelkan sebagai graf berarah dengan halaman web dinyatakan oleh simpul dan link ke halaman web menyatakan sisi. Teknik menjelajahi web secara BFS dipilih karena setelah penulis mengamati situs web milik Pak Rinaldi Munir, terlihat bahwa BFS tidak perlu menelusuri semua situs sampai habis, hanya perlu sampai pada depth tertentu untuk menemukan halaman yang berisi kumpulan-kumpulan judul makalah beserta link menuju file PDF-nya. Jadi, implementasi BFS yang digunakan untuk makalah ini akan dibatasi dengan kedalaman tertentu.

Algoritma pencocokan *string* adalah algoritma yang digunakan untuk mencari kecocokan pola teks tertentu dalam suatu teks. Aplikasi algoritma ini dapat digunakan dalam menemukan kemiripan antara judul makalah baru dengan yang sudah ada sebelumnya dan mendeteksi tindakan plagiarisme. Dalam konteks ini, algoritma *string* matching seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) sering kali menjadi pilihan utama karena performanya yang lebih baik dibandingkan algoritma *brute force*.

II. Landasan Teori

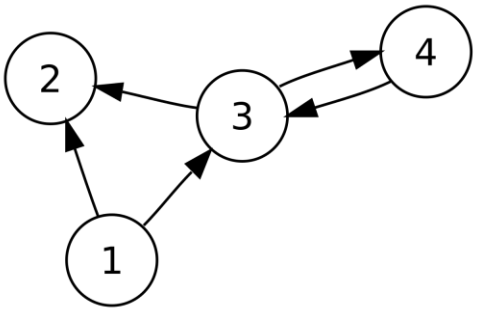
A. *Graf*

Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Sebuah graf didefinisikan sebagai sebuah pasangan himpunan simpul-simpul dan sisi yang menghubungkan simpul. Simpul pada graf dapat memiliki arah dan bobot (*directed graph*) ataupun tidak (*undirected graph* atau *digraph*). Graf dapat direpresentasikan sebagai matriks ketetanggaan (*adjacency matrix*), matriks bersisian (*incidency matrix*) dan senarai ketetanggaan (*adjacency list*).



Gambar 2. Contoh graf tak berarah

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/20-Graf-Bagian2-2023.pdf



Gambar 3. Contoh graf berarah

Sumber: https://commons.wikimedia.org/wiki/File:Directed_graph_no_background.svg

B. *String*

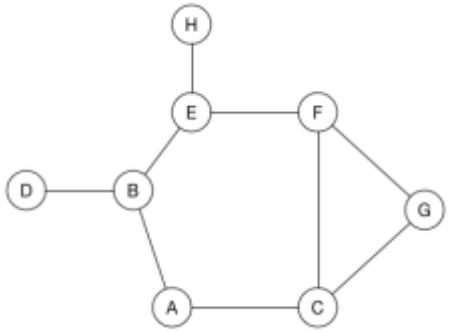
Struktur data *string* adalah sebuah sekuens dari karakter-karakter berupa huruf, angka, tanda baca, dan karakter-karakter valid lainnya. *String* biasanya dipandang sebagai sebuah larik dari karakter-karakter. *String* dapat mengandung huruf, angka, maupun simbol-simbol lainnya. Isi dari *string* diapit dengan tanda petik untuk menyatakan suatu deretan karakter sebagai *string*. Implementasi struktur data *string* yang berbeda-beda tiap bahasa pemrograman membuat sifat *string* itu sendiri juga berbeda. Pada bahasa tertentu, *string* dipandang sebagai variabel yang bisa berubah atau biasa disebut sebagai *mutable string*, pada bahasa lain, *string* dipandang sebagai *literal constant* yang tidak bisa diubah-ubah atau biasa disebut sebagai *immutable string*. Untuk mengetahui akhir dari sebuah *string*, untuk pemrograman seperti C, menggunakan *termination character* "\0" dengan semua bit bernilai 0. *Termination character* ini juga dapat digunakan untuk menentukan panjang *string* secara implisit, yaitu dengan menghitung semua karakter yang ada pada *string* sampai "\0" ditemukan.

Karakter ke-n dari *string* pada umumnya dapat diakses dengan menggunakan notasi indeks yang biasanya digunakan untuk mengakses elemen ke-n dari larik biasa. Notasi ini dimulai dari nama variabel *string* tersebut, misannya s, lalu diikuti dengan kurung siku pembuka, sebuah angka, lalu ditutup dengan kurung siku penutup. Contohnya s[0], yang berarti karakter ke-0 pada *string* s. Notasi ini sering digunakan karena pencarian *string* akan memanfaatkan posisi suatu karakter pada teks. Sementara itu T[0..i] merupakan sub*string* dari *string* T, yaitu *string* yang berisi karakter pada T dari indeks 0 hingga indeks i.

C. *Algoritma BFS*

Algoritma BFS adalah algoritma traversal graf untuk node yang memenuhi properti tertentu. Algoritma ini termasuk dalam kategori algoritma pencarian buta (*uninformed/blind search*) karena teknik pencariannya yang mirip dengan algoritma *bruteforce*. Pencarian BFS dimulai dengan mengunjungi simpul v, kemudian simpul-simpul yang bertetangga dengan simpul v, baru selanjutnya mengunjungi simpul-simpul yang bertetangga dengan simpul-simpul yang

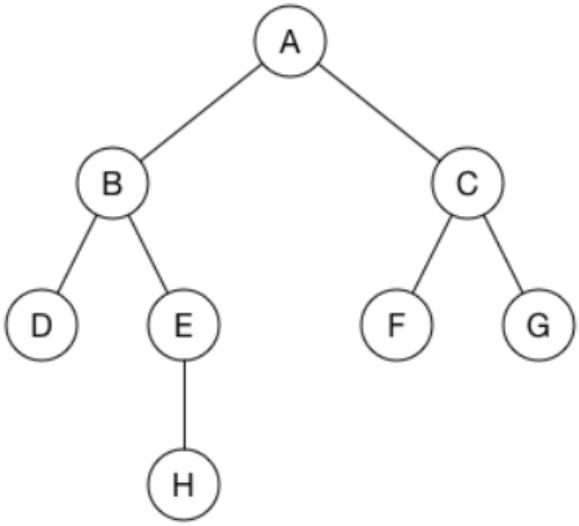
sudah dikunjungi sebelumnya. Misalnya untuk graf di bawah ini:



Gambar 4. Contoh Graf

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf

Setelah dilakukan pencarian BFS akan dihasilkan pohon BFS:



Gambar 5. Hasil BFS

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf

Ketika melakukan pencarian solusi dengan BFS, akan dibentuk pohon dinamis dengan representasi:

1. Pohon ruang status (*state space tree*)
2. Simpul (*problem state*), terdiri atas akar yang merepresentasikan *initial state* dan daun yang merepresentasikan *goal state*
3. Cabang, merepresentasikan langkah penyelesaian persoalan
4. Ruang status (*state space*), merepresentasikan himpunan semua simpul, dan
5. Ruang solusi, merepresentasikan himpunan status solusi

Pembangkitan status baru dilakukan dengan cara mengaplikasikan operator kepada status pada suatu jalur. Jalur dari simpul akar ke daun berisi rangkaian operator yang mengarah pada solusi persoalan. Proses pencarian yang baik dapat dipertimbangkan dengan melihat aspek-aspek berikut:

a. *Completeness*, artinya jaminan apakah semua solusi ditemukan jika memang ada;
b. *Optimality*, artinya jaminan bahwa solusi yang didapatkan optimal;
c. *Time complexity*, artinya seberapa lama waktu yang diperlukan untuk mendapatkan solusi; serta
d. *Space complexity*, artinya seberapa besar memori yang dibutuhkan untuk mendapatkan solusi.

Algoritma BFS memiliki kompleksitas ruang bd dengan b yaitu jumlah maksimal cabang dan d merupakan kedalaman yang dicari. Jika dibandingkan dengan algoritma traversal graf lain, seperti DFS, algoritma BFS memiliki kompleksitas ruang yang lebih buruk dibandingkan DFS yang kompleksitas ruangnya adalah bd. Berikut *psuedocode* algoritma BFS secara umum:

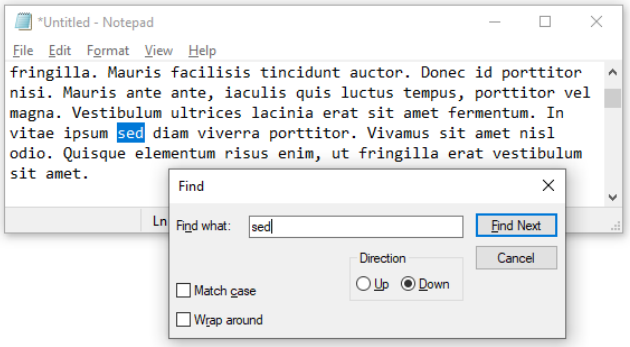
```
ALGORITMA BFS(G)


q = createEmptyQueue()
Proses(v)
visited[v] ← true
q.add(v)
while not q.empty() do
    q.dequeue(v)
    for each simpul w yang bertetangga dengan simpul v do
        if not visited[w] then
                Proses(w)
                q.add(w)
                visited[w] ← true
```

D. Algoritma String Matching

Algoritma *String Matching* atau pencocokan *string* adalah sebuah algoritma yang digunakan untuk mencari kemunculan suatu *string* atau pola tertentu dalam sebuah teks dalam kemunculan pertama. Pola ini harapannya adalah sub*string* dari teks yang ingin dicari (berarti panjang pola ≤ panjang teks).

String matching digunakan untuk pencarian di dalam editor teks (Ctrl+F pada editor teks), *web search engine* (perbandingan kata kunci dengan teks yang ada pada suatu situs web), analisis citra (sidik jari, *face recognition*, *retina*

identification), *bioinformatics* (pencocokan rantai asam amino pada rantai DNA, RNA, dan lain-lain), dan lain-lain. Dalam konsep *string*, dikenal konsep *prefix* dan *suffix*.



Gambar 6. Contoh *string matching* untuk pencarian di dalam editor teks

Sumber: Dokumentasi penulis

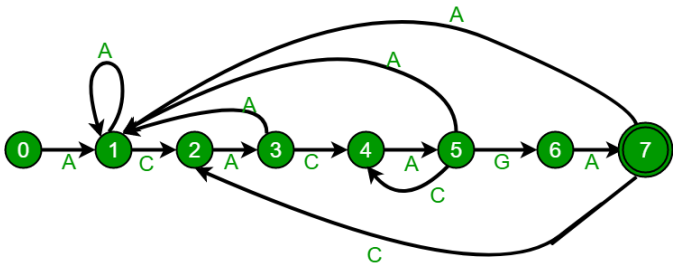
Misal ada *string* S = "*string*", *prefix* dari S adalah sub*string* dengan definisi: karakter ke-0 sampai karakter ke-k, dengan ketentuan karakter ke-k bukan karakter terakhir dari *string* S. Jadi *prefix* adalah sebuah karakter atau sebuah sub*string* dari S yang terletak pada bagian depan dari S. *Suffix* dari S adalah sub*string* dengan definisi: karakter ke-0 sampai karakter terakhir. Jadi *suffix* adalah sebuah karakter atau sebuah sub*string* dari S yang terletak pada bagian belakang dari S. *Prefix* dan *suffix* dapat berupa *string* itu sendiri.

Prefix = ["s", "st", "str", "stri", "strin", "*string*"]

Suffix = ["*string*", "tring", "ring", "ing", "ng", "g"]

Ada banyak algoritma yang dapat melakukan *string matching*, bisa menggunakan *finite automata* maupun algoritma. Pembahasan kali ini difokuskan pada algoritma yang diajarkan di kelas yaitu:

- The Brute Force Algorithm
- The KMP Algorithm
- The Boyer-Moore Algorithm

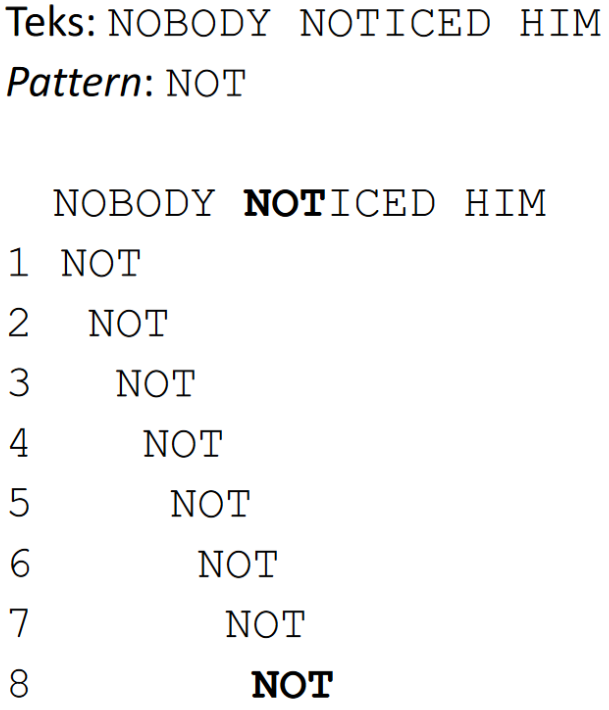


Gambar 7. *String matching* dengan *finite automata*

Sumber: https://www.geeksforgeeks.org/finite-automata-algorithm-for-pattern-searching/

E. Algoritma Brute Force pada String Matching

Algoritma ini biasa disebut sebagai algoritma naif. Algoritma ini memiliki kompleksitas waktu rata-rata O(m + n) dengan n sebagai panjang teks dan m sebagai panjang pola yang ingin dicari. Algoritma ini bekerja dengan memeriksa karakter pertama pola dengan karakter pada teks pada posisi tertentu, dimulai dari posisi ke-0, jika sama, indeks pada pola akan bergeser, jika berbeda, indeks pada teks yang akan bergeser. Pencarian selesai ketika ditemukan kecocokan atau sampai akhir teks tidak ditemukan kecocokan sama sekali.



Gambar 8. Contoh *string matching* dengan algoritma *brute force*

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

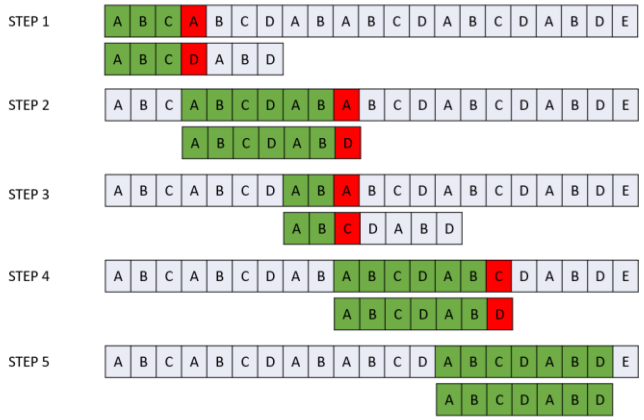
Kasus terbaiknya adalah ketika karakter pertama pola tidak pernah sama dengan karakter teks yang dicocokkan sehingga kompleksitas waktunya O(n). Kasus terburuknya adalah ketika karakter terakhir pola tidak sama dengan karakter teks yang dicocokkan sehingga kompleksitas waktunya O(mn).

F. Algoritma KMP pada String Matching

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma *string matching* yang ditemukan oleh James H. Morris, lalu secara independen ditemukan oleh Donald Knuth dari teori automata. Algoritma KMP mencari pola dalam teks dengan urutan kiri ke kanan (sama seperti algoritma *brute force*). Dibandingkan dengan *brute force*, yang menggeser pola satu kali ke kiri setiap kali karakter pola dengan karakter teks

berbeda, algoritma KMP menggeser pola dengan lebih pintar daripada algoritma *brute force*.

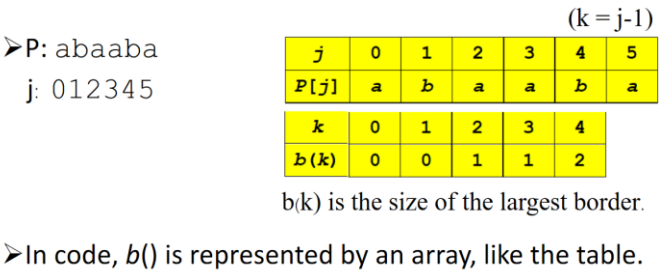
Jika ditemukan perbedaan antara karakter pola dengan karakter teks, misal pada karakter pola P[j], pola bisa digeser sebanyak *prefix* terbesar dari P[0..j-1] yang merupakan *suffix* dari P[1..j-1].



Gambar 9. Pergeseran pencarian teks pada algoritma KMP

Sumber: https://www.vonhumboldt.edu.pe/*string*-matching-kmp-knuth-morris-pratt-algorithmby-amit-C18176891

Agar algoritma ini dapat berjalan seperti gambar di atas, dibutuhkan fungsi pinggiran KMP (*KMP Border Function*) untuk mem-*preprocess* pola untuk mencari kecocokan *prefix-prefix* dari pola dengan pola itu sendiri. Fungsi pinggiran b(k) didefinisikan sebagai *prefix* dengan ukuran terbesar dari P[0..k] yang juga merupakan *suffix* dari P[1.k]. Fungsi ini juga memiliki nama lain, yaitu *failure function*.



Gambar 10. *Failure array* yang merepresentasikan *border function* untuk pola abaaba

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

Fungsi pinggiran (*border function*) memiliki kompleksitas waktu O(m), sedangkan waktu pencariannya sendiri memiliki kompleksitas waktu O(n), sehingga total kompleksitas waktunya adalah O(m + n). Algoritma ini tentu memiliki kompleksitas yang jauh lebih cepat dibandingkan algoritma *brute force*.

Kelebihan KMP adalah algoritma ini tidak perlu menggeser teks ke belakan sehingga algoritma ini bagus untuk memproses *file* besar yang dibaca dari perangkat eksternal atau melalui aliran jaringan. Kelemahan dari algoritma ini adalah jika alfabet dari teks bertambah (misal: perubahan dari alfabet menjadi alfanumerik), probabilitas untuk menemukan ketidakcocokan bertambah, KMP memiliki performa yang lebih baik jika ketidakcocokan ditemukan di belakang.

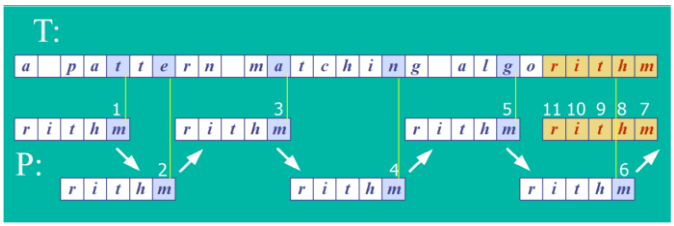
G. Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) adalah algoritma *string matching* yang ditemukan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Algoritma ini memanfaatkan dua teknik yaitu:

- *The looking-glass technique* → cari pola dalam teks dimulai dari kanan lalu ke kiri.
- *The character-jump technique* → ketika karakter ke-i dari pola tidak sama dengan karakter ke-j dari teks, ada 3 kemungkinan.

Ketiga kemungkinan tersebut dapat berupa:

1. Kasus 1: Karakter x ada pada pola, coba geser pola ke kanan untuk mencocokkan karakter x milik teks dengan karakter x milik pola.
2. Kasus 2: Karakter x ada pada pola, tetapi menggeser pola ke kanan tidak memungkinkan, maka geser pola ke kanan sebanyak 1 karakter
3. Kasus 3: Karakter x tidak ada pada pola, geser pola agar indeks ke-0 pola berada pada sebelah kanan karakter x milik teks.

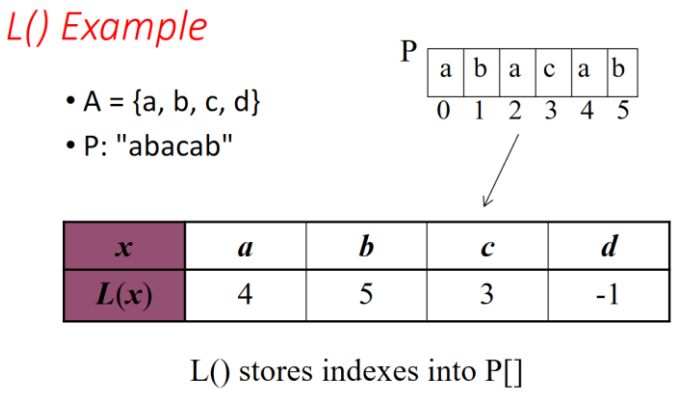


Gambar 11. Contoh kasus pencocokan *string* dengan algoritma BM

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

Algoritma BM mem-*preprocess* pola dan alfabet untuk membangun fungsi kemunculan terakhir (*last occurence function*). Fungsi ini menyimpan semua huruf dalam alfabet menjadi *integer*. Fungsi ini didefinisikan sebagai: (x dalam alfabet)

1. Indeks terbesar i agar P[i] == x, atau
2. -1 jika tidak ada indeks



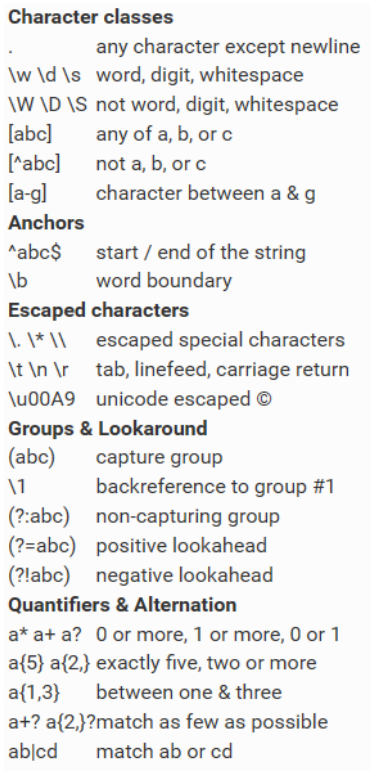
Gambar 12. Contoh fungsi kemunculan terakhir pada algoritma BM

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

Algoritma ini memiliki kompleksitas waktu terburuk O(mn + A). Tetapi, BM cepat ketika alfabetnya besar, dan lambat ketika alfabetnya kecil. BM sangat cepat ketika menelusuri teks berbahasa Inggris.

H. *Regular Expressions*

Regular Expression atau biasa disebut regex adalah sebuah *string* yang mendefinisikan sebuah pola pencarian sehingga memudahkan pencocokan, pencarian, dan manipulasi teks. Berikut notasi regex:



Gambar 13. *Cheat sheet* untuk regex

Sumber: https://www.regexpal.com

III. Implementasi Program

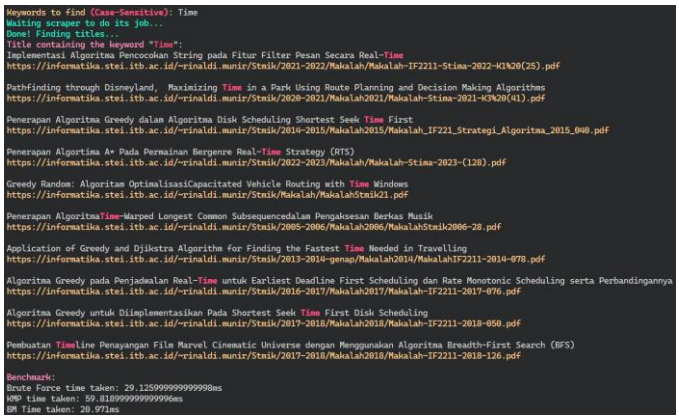
Program pencarian judul menggunakan *string matching*, BFS, dan regex diimplementasikan menggunakan kakas VS Code dan bahasa pemrograman Python. Program dipisah menjadi beberapa modul untuk meningkatkan modularitas sekaligus mempermudah proses *debugging*. Yang akan di-*benchmark* di sini adalah performa antara ketiga algoritma *string matching* yang sudah diajarkan di kelas. Ketiga algoritma akan dibandingkan dan dilampirkan hasil ujinya untuk mengetahui algoritma mana yang lebih cocok untuk digunakan pada topik pencarian judul ini. Implementasi kode sumber dari program yang telah penulis buat dapat diakses dan diunduh pada pranala yang telah dilampirkan pada Bab V: Lampiran.

A. *Metodologi Pengujian dan Cara Kerja Program*

Pengujian dilakukan dengan memberikan masukan berupa kata kunci yang ingin dicari, selanjutnya, program akan dengan sendirinya melakukan *web scraping* ke pranala https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm, yang akan menjadi akar dari struktur data pohon dinamis yang dibuat oleh algoritma BFS dalam bentuk *queue*. Algoritma BFS ini akan mengembalikan sebuah *set* yang berisi pranala yang mengandung kata dengan aturan regex: /.*[Mm]akalah.*\.htm$/g. *Set* ini akan diberikan kepada fungsi *find_titles*. Fungsi ini akan mengekstrak elemen html <a>, mengambil teks anotasi sebagai judul, lalu mengambil *href* nya sebagai URL yang nantinya akan menjadi luaran dari program ini. Pada akhirnya, program akan mencetak judul makalah yang mengandung kata kunci beserta pranala yang menuju ke *file* PDF makalah dengan judul tersebut.

B. *Hasil Pengujian*

Setelah menguji ketiga algoritma *string matching* (*brute force*, KMP, dan BM) dengan 10 buah *testcase* yang terdiri dari 10 kata kunci, didapatkan hasil seperti berikut:



Gambar 14. Output salah satu hasil keluaran program yang dibuat

Sumber: Dokumentasi penulis

Tabel 1. Tabel hasil rata-rata durasi ketiga algoritma untuk memproses kumpulan judul dan pola yang sama

Nama Algoritma	Rata-rata Durasi yang

	Dibutuhkan (ms)
Brute Force	27.6771
Knuth-Morris-Pratt (KMP)	58.6911
Boyer-Moore (BM)	18.3972

IV. Kesimpulan

Telah diimplementasikan program pencarian judul menggunakan kata kunci masukan sebagai pola dan hasil web scraping sebagai teks. Algoritma traversal graf BFS digunakan untuk menelusuri situs web, algoritma *string matching* digunakan untuk membandingkan pola masukan dengan teks judul, dan *regex* digunakan untuk menyaring judul yang telah dikumpulkan oleh *web scrapper* tadi.

Menurut hasil pengujian pada tabel 1, algoritma yang paling cepat adalah Boyer-Moore, diikuti oleh *brute force*, dan yang paling lambat adalah Knuth-Morris-Pratt. BM memiliki performa yang paling cepat karena algoritma ini memiliki keunggulan ketika menemukan perbedaan antara pola dengan teks, lompatan yang dihasilkan oleh algoritma ini membuatnya lebih cepat untuk menentukan bahwa judul tidak relevan dengan kata kunci yang diberikan. KMP memiliki waktu yang paling lambat karena alfabet yang digunakan banyak (tidak hanya alfanumerik tetapi juga simbol) sehingga lebih sering menemukan ketidakcocokan di awal dan akhirnya menjadi lebih lambat dibandingkan *brute force*.

Hasil yang diperoleh melalui pembuatan makalah ini menunjukkan kegunaan algoritma-algoritma di atas yang bisa digunakan tidak hanya oleh diri sendiri, tetapi juga angkatan-angkatan berikutnya untuk mempermudah penentuan judul makalah mereka di masa yang akan datang.

V. Lampiran

Pranala Repositori GitHub:

https://github.com/CrystalNoob/Makalah_13522094

Pranala Hasil Pengujian Versi Lengkap:

https://docs.google.com/spreadsheets/d/1rsW5g08W3Uw7h6kHnlVdQAAlWrsW24NU/edit?usp=sharing&ouid=114940513674561767498&rtpof=true&sd=true

VI. Ucapan Terima Kasih

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena makalah sebagai tugas akhir mata kuliah IF2211 Strategi Algoritma berjudul “Penerapan Algoritma *String* Matching, BFS, dan Regex: Mencari Judul Menggunakan Kata Kunci Masukan dari Hasil Web Scraping Judul Makalah” bisa diselesaikan tepat waktu. Makalah IF2211 Strategi Algoritma – Sem. II Tahun 2023/2024 ini tidak mungkin terselesaikan dengan baik tanpa adanya dukungan dari berbagai pihak. Maka di kesempatan ini, izinkan penulis mengucapkan banyak terima kasih kepada

1) Dr. Nur Ulfa Maulidevi sebagai dosen pengajar mata kuliah IF2211 Strategi Algoritma yang sudah mengajar dengan baik.
2) Dr. Rinaldi Munir sebagai pembuat materi yang banyak membantu penulis dalam menyelesaikan makalah ini.

Penulis harap makalah yang sudah disusun bisa bermanfaat bagi banyak orang.

References

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/20-Graf-Bagian2-2023.pdf, diakses pada 11 Juni 2024 pukul 16.42 UTC+7

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf, diakses pada 12 Juni 2024 pukul 11.46 UTC+7

[3] https://www.geeksforgeeks.org/finite-automata-algorithm-for-pattern-searching/, diakses pada 12 Juni 2024 pukul 16.25 UTC+7

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-*string*-2021.pdf, diakses pada 12 Juni 2024 pukul 17.55 UTC+7

[5] https://www.regexpal.com, diakses pada 12 Juni 2024 pukul 20.23 UTC+7

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

Andhika Tantyo Anugrah 13522094