

**PERBANDINGAN ALGORITMA RAITA DAN ALGORITMA NOT SO NAÏVE
DALAM PEMBUATAN KAMUS BAHASA
INDONESIA – BAHASA SPANYOL**

SKRIPSI

NURAPRILLIA

171401144



**PROGRAM STUDI S1 ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2024**

PERBANDINGAN ALGORITMA RAITA DAN ALGORITMA NOT SO NAÏVE
DALAM PEMBUATAN KAMUS BAHASA
INDONESIA – BAHASA SPANYOL

SKRIPSI

NURAPRILLIA
171401144



PROGRAM STUDI S1 ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2024

PERSETUJUAN

Judul : PERBANDINGAN ALGORITMA RAITA DAN ALGORITMA
NOT SO NAÏVE DALAM PEMBUATAN KAMUS BAHASA
INDONESIA – BAHASA SPANYOL

Kategori : SKRIPSI

Nama : NURAPRILLIA

Nomor Induk Mahasiswa : 171401144

Program Studi : SARJANA (S-1) ILMU KOMPUTER

Fakultas : ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA

Komisi Pembimbing :

Pembimbing II



Dewi Sartika Br Ginting S.Kom., M.Kom.
NIP. 1990050420196032023

Pembimbing I



Sri Melvani Hardi, S.Kom., M.Kom.
NIP. 198805012015042006

Diketahui / Disetujui oleh
Program Studi S-1 Ilmu Komputer

Ketua,




Dr. Amalia, S.T., M.T.
NIP. 197812212014042001

PERNYATAAN

**PERBANDINGAN ALGORITMA RAITA DAN ALGORITMA NOT SO NAÏVE
DALAM PEMBUATAN KAMUS BAHASA
INDONESIA – BAHASA SPANYOL**

SKRIPSI

Saya mengakui bahwa skripsi ini adalah hasil karya saya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing telah disebutkan sumbernya.

Medan, Januari 2024

Nuraprillia
171401144

UCAPAN TERIMA KASIH

Segala puji dan syukur atas kehadiran Allah SWT yang telah memberikan segala limpahan rahmat, karunia serta hidayah-Nya sehingga penulis mampu menyelesaikan penyusunan skripsi ini sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer.

Ucapan terima kasih penulis berikan kepada pihak yang telah memberikan dukungan dan dorongan kepada penulis, baik secara materil maupun moril dan baik secara langsung maupun tidak langsung. Pada kesempatan ini, penulis ingin mengucapkan banyak terima kasih kepada :

1. Bapak Dr. Muryanto Amin, S.Sos., M.Si. selaku Rektor Universitas Sumatera Utara.
2. Ibu Dr. Maya Silvi Lydia, B.Sc., M.Sc. selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara.
3. Ibu Dr. Amalia, S.T., M.T. selaku Kepala Program Studi S-1 Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara dan selaku Dosen Penguji I yang telah memberikan kritik dan saran dalam penyempurnaan skripsi ini.
4. Ibu Sri Melvani Hardi, S.Kom. M.Kom., selaku Dosen Pembimbing I yang telah banyak memberikan bimbingan, saran, dan dukungan dalam pengerjaan skripsi ini.
5. Ibu Dewi Sartika Br Ginting S.Kom., M.Kom. selaku Dosen Pembimbing II yang telah banyak memberikan bimbingan, saran, dan dukungan dalam pengerjaan skripsi ini.
6. Bapak Ivan Jaya S.Si., M.Kom. selaku Dosen Penguji II yang telah memberikan kritik dan saran dalam penyempurnaan skripsi ini.
7. Seluruh dosen dan staf pegawai Program Studi S-1 Ilmu Komputer Fakultas Ilmu Komputer dan teknologi Informasi Universitas Sumatera Utara yang telah memberikan ilmu pengetahuan yang bermanfaat selama penulis mengikuti proses perkuliahan.
8. Keluarga besar penulis kepada ayahanda alm. Abdul Jalil, ibunda Suryani, kakak Nurliza, SKM., Ernita A.ma., Jufrina, S.Pd dan adik tersayang Nursyifa. Terima kasih untuk doa, nasihat dan kerja keras serta selalu memberikan dukungan semangat kepada penulis dalam menyelesaikan skripsi ini.
9. Teman-teman Kom C 2017 dan seluruh keluarga besar stambuk 2017 Ilmu Komputer Universitas Sumatera Utara yang telah banyak memberi motivasi kepada penulis dalam pengerjaan skripsi ini.

10. Sahabat – sahabat terbaik penulis yaitu SN: 30080926 (Ghina Handayani Siregar S.Kom, Nurul Azizah Daulay S.Kom, Rizky Ayu Azhari) dan Rini Kurniasari S.Si yang telah banyak memberikan doa, dukungan dan semangat selama proses menyelesaikan skripsi ini.
11. Keluarga Asrama Putri USU yaitu Nuraini, Hasanah, Sinta Nurhayati, Yunis Mayasari yang telah mendukung dalam doa maupun semangat dalam penyelesaian skripsi ini.
12. Semua pihak yang terlibat langsung maupun tidak langsung yang tidak dapat ditulis satu persatu.

Medan, Januari 2024

Penulis

ABSTRAK

Bahasa Spanyol merupakan bahasa resmi yang digunakan oleh 18 negara di Amerika dan 1 negara di Afrika. Bahasa ini juga bahasa dengan pengaruh luas di banyak tempat berbeda di seluruh dunia. Menjadi salah satu bahasa yang populer tentunya membuat orang sangat tertarik untuk menguasai bahasa tersebut. Kamus menjadi salah satu media yang bisa digunakan untuk menguasai kosakata bahasa asing. Namun masih banyak kamus bahasa asing dalam bentuk buku dan berukuran tebal, hal tersebut menjadikannya kurang efektif. Diperlukan sebuah media yang lebih praktis dari kamus cetak, seperti sebuah aplikasi kamus yang mudah dibawa kemana saja. Maka dibuatlah kamus bahasa Indonesia – bahasa Spanyol yang diharapkan dapat mempermudah pengguna agar bisa menguasai kosakata yang ada pada bahasa Spanyol. Aplikasi kamus yang dibuat adalah berbasis android untuk mempermudah pengguna tanpa harus membeli kamus bahasa Spanyol. Penelitian ini menggunakan dua algoritma *string matching* untuk diimplementasikan ke dalam sistem yang dibuat yaitu algoritma Raita dan algoritma *Not So Naïve* dengan membandingkan *running time* dan kompleksitas waktu (Θ) pada fase pencarian. Dari hasil dari penelitian ini menunjukkan bahwa Algoritma *Not So Naïve* melakukan pencarian kata lebih cepat dibanding Algoritma Raita. Hal ini ditunjukkan dari hasil rata-rata *running time* yang didapat yaitu, algoritma *Not So Naïve* mempunyai rata-rata *running time* selama 7,4 ms sedangkan algoritma Raita mempunyai rata-rata *running time* selama 10,3 ms. Kompleksitas algoritma *Not So Naïve* adalah $\Theta(n-m)$ dan kompleksitas algoritma Raita adalah $\Theta(mn)$.

Kata Kunci : Kamus Bahasa Spanyol, Algoritma *String Matching*, Algoritma Raita, Algoritma *Not So Naïve*.

ABSTRACT

Spanish is the official language used by 18 countries in America and 1 country in Africa. This language is also a language with wide influence in many different places around the world. Being one of the popular languages certainly makes people very interested in mastering the language. Dictionaries are one of the media that can be used to master foreign language vocabulary. However, there are still many foreign language dictionaries in book form and thick in size, which makes them less effective. We need a medium that is more practical than a printed dictionary, such as a dictionary application that is easy to carry anywhere. So an Indonesian – Spanish dictionary was created which is expected to make it easier for users to master the vocabulary in Spanish. The dictionary application created is Android-based to make things easier for users without having to buy a Spanish dictionary. This research uses two string matching algorithms to be implemented into the system created, namely the Raita algorithm and the Not So Naïve algorithm by comparing the running time and time complexity (Θ) in the search phase. The results of this research show that the Not So Naïve algorithm performs word searches faster than the Raita algorithm. This is shown by the average running time results obtained, namely, the Not So Naïve algorithm has an average running time of 7.4 ms while the Raita algorithm has an average running time of 10.3 ms. The complexity of the Not So Naïve algorithm is $\Theta(n-m)$ and the complexity of the Raita algorithm is $\Theta(mn)$.

Keywords : Spanish Dictionary, String Matching Algorithm, Raita Algorithm, Not So Naïve Algorithm.

DAFTAR ISI

PERSETUJUAN.....	ii
PERNYATAAN	iii
UCAPAN TERIMA KASIH.....	iv
ABSTRAK.....	vi
ABSTRACT.....	vii
DAFTAR ISI.....	viii
DAFTAR GAMBAR	xi
DAFTAR TABEL.....	xii
BAB 1 PEDAHULUAN.....	1
1.1 Latar Belakang Masalah.....	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah.....	2
1.4 Tujuan Penelitian.....	3
1.5 Manfaat Penelitian.....	3
1.6 Metodologi Penelitian	3
1.7 Sistematika Penulisan.....	4
BAB 2 LANDASAN TEORI.....	6
2.1 Algoritma	6
2.2 Algoritma pencocokan string (string matching).....	6
2.3 Algoritma Raita	7
2.4 Algoritma Not So Naïve.....	10
2.5 Kompleksitas Algoritma	15
2.6 Kamus.....	15
2.7 Penelitian Relevan.....	15
BAB 3 ANALISIS DAN PERANCANGAN SISTEM	17
3.1 Analisis Sistem.....	17
3.1.1 Analisis Masalah.....	17
3.1.2 Analisis Kebutuhan.....	19

3.1.3 Analisis Proses.....	20
3.2 Perancangan Sistem.....	20
3.2.1 General Arsitektur Sistem	20
3.2.2 Use Case Diagram	21
3.2.3 Activity Diagram	23
3.2.4 Sequence Diagram	24
3.2.5 <i>Flowchart</i>	24
3.2.6 Flowchart Algoritma Raita	26
3.2.7 Flowchart Algoritma Not So Naïve	27
3.3 Perancangan Antarmuka (<i>Interface</i>)	28
3.3.1 Perancangan halaman splash screen	28
3.3.2 Rancangan Halaman Utama	29
3.3.3 Rancangan <i>Navigation Drawer</i>	29
3.3.4 Rancangan Halaman Tentang.....	30
3.3.5 Rancangan Halaman Bantuan.....	31
BAB 4 IMPLEMENTASI DAN PENGUJIAN.....	33
4.1 Implementasi Sistem	33
4.1.1 Tampilan Halaman <i>Splash Screen</i>	33
4.1.2 Tampilan Halaman Utama	34
4.1.3 Tampilan Menu Navigasi.....	35
4.1.4 Tampilan Halaman Tentang.....	35
4.1.5 Tampilan Halaman Bantuan	36
4.2 Pengujian Sistem	37
4.2.1 Pengujian Pencarian kata pada Kamus Bahasa Indonesia – Bahasa Spanyol dengan algoritma Raita	38
4.2.2 Pengujian Pencarian kata pada Kamus Bahasa Indonesia – Bahasa Spanyol dengan algoritma Not So Naïve	53
4.2.3 Hasil Pengujian	68
4.3 Kompleksitas Algoritma	70
4.3.1 Kompleksitas Algoritma Raita.....	71
4.3.2 Kompleksitas Algoritma Not So Naïve	73

BAB 5 KESIMPULAN DAN SARAN	76
5.1 Kesimpulan.....	76
5.2 Saran.....	76
DAFTAR PUSTAKA	77
LISTING PROGRAM	A

DAFTAR GAMBAR

Gambar 3.1 Diagram Ishikawa	18
Gambar 3.2 General Arsitektur Sistem.....	21
Gambar 3.3 <i>Use Case</i> Diagram	22
Gambar 3.4 <i>Activity</i> Diagram	23
Gambar 3.5 <i>Sequence</i> Diagram.....	24
Gambar 3.6 <i>Flowchart</i> Sistem	25
Gambar 3.7 <i>Flowchart</i> Proses Pencarian Algoritma Raita.....	26
Gambar 3.8 <i>Flowchart</i> Proses Pencarian Algoritma <i>Not So Naïve</i>	27
Gambar 3.9 Rancangan <i>Splash Screen</i>	28
Gambar 3.10 Rancangan Halaman Utama.....	29
Gambar 3.11 Rancangan <i>Navigation Drawer</i>	30
Gambar 3.12 Rancangan Halaman Tentang	31
Gambar 3.13 Rancangan Halaman Bantuan	32
 Gambar 4.1 Halaman <i>Splash Screen</i>	 33
Gambar 4.2 Halaman Utama.....	34
Gambar 4.3 Menu Navigasi	35
Gambar 4.4 Halaman Tentang	36
Gambar 4.5 Halaman Bantuan.....	37
Gambar 4 6 Perbandingan Hasil <i>Running Time</i> Algoritma Raita dan Not So Naïve	69

DAFTAR TABEL

Tabel 2.1 Perhitungan tabel BmBc (a).....	8
Tabel 2.2 Hasil BmBc (a)	9
Tabel 2.3 Proses Pencarian Pada Teks Ke-1.....	9
Tabel 2.4 Proses Pencarian Pada Teks Ke-2.....	9
Tabel 2.5 Proses Pencarian Pada Teks Ke-3.....	10
Tabel 2.6 Proses Pencarian Pada Teks Ke-1.....	11
Tabel 2.7 Proses Pencarian Pada Teks Ke-2.....	12
Tabel 2.8 Proses Pencarian Pada Teks Ke-3.....	12
Tabel 2.9 Proses Pencarian Pada Teks Ke-4.....	12
Tabel 2.10 Proses Pencarian Pada Teks Ke-5.....	13
Tabel 2.11 Proses Pencarian Pada Teks Ke-6.....	13
Tabel 2.12 Proses Pencarian Pada Teks Ke-7.....	13
Tabel 2.13 Proses Pencarian Pada Teks Ke-8.....	14
Tabel 2.14 Proses Pencarian Pada Teks Ke-9.....	14
Tabel 2.15 Proses Pencarian Pada Teks Ke-10.....	14
 Tabel 4.1 Hasil Pencarian Kata Algoritma Raita.....	 38
Tabel 4.2 Hasil Pencarian Kata Algoritma Not So Naive	53
Tabel 4.3 Hasil Pengujian Algoritma Raita dan Algoritma Not So Naive	68
Tabel 4.4 Analisis Kompleksitas pada Fase Preprocessing Algoritma Raita	71
Tabel 4.5 Analisis Kompleksitas pada Fase Pencarian Algoritma Raita.....	71
Tabel 4.6 Kompleksitas Hasil Algoritma Not So Naive.....	73

BAB 1

PEDAHULUAN

1.1 Latar Belakang Masalah

Bahasa Spanyol adalah bahasa pertama untuk sekitar 480 juta orang, dengan jumlah 577 juta orang yang mempelajarinya serta menjadikannya bahasa asli kedua yang paling banyak digunakan setelah Bahasa Mandarin. Bahasa Spanyol menjadi bahasa resmi 18 negara di Amerika dan 1 negara di Afrika (Paloma Cascales, 2019). Bahasa ini juga bahasa dengan pengaruh luas di banyak tempat berbeda di seluruh dunia. Menjadi bahasa yang populer tentunya membuat orang sangat tertarik untuk menguasai bahasa tersebut. Kamus menjadi salah satu media yang bisa digunakan untuk memudahkan manusia agar bisa menguasai kosakata bahasa asing.

Masih banyak kamus bahasa asing dalam bentuk buku dan berukuran tebal, hal tersebut yang menjadikannya kurang efektif. Pemanfaatan media *smartphone* bisa menjadi salah satu solusi yang praktis. *Smartphone* mendukung berbagai macam aplikasi seperti aplikasi kamus. Maka dari itu penulis ingin membuat aplikasi kamus bahasa asing yang dapat memudahkan pengguna. Kamus yang dibuat merupakan kamus bahasa Indonesia – bahasa Spanyol. Pembuatan kamus bahasa dalam bentuk teknologi digital diimplementasikan dengan metode pencocokan *string*. Pencocokan kata atau *string matching* merupakan algoritma yang digunakan untuk mencari sebuah *string* yang terdiri dari beberapa karakter dalam sejumlah besar teks. Beberapa algoritma *string matching* adalah algoritma Knuth- Morris-Pratt, algoritma *Not so Naïve*, algoritma Raita dan algoritma Boyer Moore.

Algoritma Raita merupakan bagian dari algoritma *exact string matching* yaitu pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama. Algoritma Raita membandingkan karakter terakhir dari jendela teks yang bersesuaian dengan *pattern*, jika cocok dilanjutkan ke karakter pertama, jika cocok juga dilanjutkan ke karakter tengah. Jika karakter akhir, pertama dan tengah cocok, selanjutnya algoritma akan membandingkan karakter lain, dimulai dari karakter kedua sampai ke satu karakter sebelum karakter terakhir, dan hal ini akan membuat

karakter tengah dibandingkan lagi. Proses pencarian *string* dilakukan dengan memastikan bahwa setiap karakter dari pola *string* sesuai dengan setiap karakter pada target *string* pada posisi yang sama (Lecroq, 2014).

Algoritma *Not So Naive* merupakan algoritma yang digunakan untuk melakukan proses pencocokan *string*. Algoritma ini merupakan variasi simpel dari algoritma *Naive* namun menjadi lebih efisien untuk beberapa kasus. Seperti algoritma *Naive*, algoritma ini memiliki fase pencarian dengan mengecek teks dan pola dari kiri ke kanan. Namun, Algoritma *Not So Naive* mengidentifikasi terlebih dahulu dua kasus yang dimana di setiap akhir fase pencocokan pergeseran bisa dilakukan sebanyak 2 posisi ke kanan, tidak seperti algoritma *Naive* yang hanya sebanyak 1 posisi (Cantone & Faro, 2004).

Berdasarkan latar belakang yang telah dijelaskan, maka penulis melakukan penelitian dengan judul “Perbandingan Algoritma Raita dan Algoritma *Not So Naïve* dalam pembuatan kamus Bahasa Indonesia – Bahasa Spanyol”.

1.2 Rumusan Masalah

Berdasarkan uraian dari latar belakang diatas, rumusan masalah yang dibahas dalam penelitian ini adalah bagaimana perbandingan algoritma Raita dan *Not So Naïve* dalam pencarian atau pencocokan kata dengan *running time* dan kompleksitas algoritma sebagai parameter pembanding.

1.3 Batasan Masalah

Batasan penelitian ini adalah sebagai berikut:

1. Menggunakan algoritma Raita dan *Not So Naïve* dalam penelitian ini.
2. Aplikasi menggunakan database *NoSQL* untuk penyimpanan kata.
3. Bahasa pemrograman yang digunakan adalah bahasa Java dan Android Studio sebagai compiler.
4. Parameter pembandingan yang diukur adalah *running time (ms)* dan kompleksitas Algoritma.
5. Jumlah kata yang tersedia pada database adalah 1000 kata.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk mengetahui perbandingan kompleksitas algoritma dan mengetahui perbandingan waktu pencocokan *string* antara algoritma Raita dan algoritma *Not So Naïve* pada aplikasi kamus bahasa Indonesia – bahasa Spanyol.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah sebagai berikut :

1. Sebagai pengetahuan dan perbandingan bagi *user* yang ingin menggunakan aplikasi kamus bahasa Indonesia – bahasa Spanyol tersebut, manakah algoritma yang lebih cepat dalam melakukan pencocokan *string* dengan menggunakan algoritma Raita dan algoritma *Not So Naïve*.
2. Untuk mempermudah pengguna dalam mencari kata pada aplikasi kamus bahasa Indonesia – bahasa Spanyol.

1.6 Metodologi Penelitian

1. Studi Literatur

Pada bagian ini penulis mengumpulkan referensi yang relevan dengan penelitian. Referensi yang digunakan dapat berupa buku, jurnal, artikel, makalah baik berupa media cetak maupun media internet yang berhubungan dengan bahasa Spanyol, Algoritma Raita dan Algoritma *Not So Naïve*.

2. Analisis dan Perancangan

Pada langkah ini digunakan untuk mengolah data dari hasil studi literatur yang kemudian dilakukan analisis dan perancangan sistem. Kemudian sistem dirancang dengan membuat gambaran sistem menggunakan *flowchart*, dan *design interface*.

3. Implementasi

Pada tahap ini perancangan diimplementasikan dalam pembuatan suatu aplikasi berbasis android dengan menggunakan Bahasa Pemrograman Java.

4. Pengujian

Pada tahap ini dilakukan pengujian kinerja sistem serta mencari hasil perbandingan dengan dua algoritma pencarian *string* yaitu algoritma Raita dan algoritma *Not So Naïve*.

5. Dokumentasi

Melakukan pembuatan dokumentasi sistem mulai dari tahap awal hingga pengujian sistem, untuk selanjutnya dibuat dalam bentuk laporan penelitian (skripsi).

1.7 Sistematika Penulisan

Agar pembahasan lebih sistematis, maka penulisan ini dibuat dalam lima bab, yaitu :

BAB 1 PENDAHULUAN

Bab ini akan menjelaskan mengenai latar belakang penelitian judul skripsi “Perbandingan Algoritma Raita dan Algoritma *Not So Naïve* dalam pembuatan kamus Bahasa Indonesia – Bahasa Spanyol”. Rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metode penelitian dan sistematika penulisan skripsi.

BAB 2 LANDASAN TEORI

Bab ini membahas secara singkat teori-teori yang berhubungan dengan Algoritma, Algoritma *Not So Naïve*, Algoritma Raita, *String Matching*, Kamus dan Android.

BAB 3 ANALISIS DAN PERANCANGAN

Bab ini berisi uraian tentang analisis mengenai proses kerja algoritma *Not So Naïve* dalam pembuatan kamus Bahasa Indonesia – Bahasa Spanyol. Disertai Pembuatan *flowchart*, *Unified Modeling Language (UML)* dan *Design Interface*.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Bab ini berisi tentang pembuatan sistem yang sesuai dengan analisis dan perancangan, kemudian dilakukan pengujian sistem.

BAB 5 KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dari seluruh uraian bab maupun sub-bab sebelumnya dan saran-saran dari hasil yang diperoleh agar dapat digunakan untuk pengembangan penelitian selanjutnya.

BAB 2

LANDASAN TEORI

2.1 Algoritma

Algoritma adalah suatu serangkaian langkah-langkah yang terstruktur dan spesifik yang diterapkan untuk menyelesaikan suatu masalah atau memecahkan suatu tugas. Algoritma membantu untuk memastikan bahwa proses pemecahan masalah berlangsung secara efisien dan tepat sasaran. Algoritma dapat digambarkan secara visual, seperti dengan menggunakan *flowchart*, atau ditulis dalam bahasa pemrograman seperti Java, Python, atau C++. Algoritma merupakan suatu prosedur langkah demi langkah untuk menyelesaikan suatu masalah (Knuth, 1997). Algoritma adalah metode yang menentukan cara untuk menyelesaikan suatu masalah dengan menggunakan teknik-teknik yang terstruktur dan sistematis (Hopcroft et al., 2001).

2.2 Algoritma pencocokan string (string matching)

Pencocokan string adalah masalah mendasar yang terjadi dalam berbagai aplikasi praktis (Mitani et al., 2016). Algoritma *string matching* digunakan untuk memecahkan masalah ini yaitu dengan cara menemukan suatu kecocokan atau pola dalam suatu *string* (sekumpulan karakter). Proses menemukan semua kemunculan suatu pola dalam teks tertentu (Chen, 2016). Untuk kombinasi pencocokan *pattern*, pencarian maupun pencocokan *string*, mempunyai pola yang lebih rumit seperti ekspresi reguler, grafik maupun array, mempunyai tujuan untuk mendapatkan sifat kombinatorial yang tidak mudah seperti struktur tersebut dan kemudian mengeksploitasi sifat-sifat tersebut menjadi peningkatan kinerja (Singla & Garg, 2012).

Cara yang jelas untuk mencari *pattern* yang cocok dengan teks adalah dengan mencoba mencari di setiap posisi awal dari teks dan mengabaikan pencarian secepat mungkin jika karakter yang salah ditemukan (Knuth et al., 1977). Proses pertama adalah menyelaraskan bagian paling kiri dari *pattern* dengan teks. Kemudian dibandingkan karakter yang sesuai dari teks dan *pattern*. Setelah seluruhnya cocok maupun tidak cocok dari *pattern*, *window* digeser ke kanan sampai posisi $(n-m+1)$ pada teks. Menurut (Singh & Verma, 2011), efisiensi dari algoritma terletak pada dua tahap :

1. Tahap praproses, tahap ini mengumpulkan informasi penuh tentang *pattern* dan menggunakan informasi ini pada tahap pencarian.
2. Tahap pencarian, *pattern* dibandingkan dengan *window* dari kanan ke kiri atau kiri ke kanan sampai kecocokan atau ketidakcocokan terjadi.

2.3 Algoritma Raita

Algoritma Raita merupakan bagian dari algoritma *exact string matching* yaitu pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama. Algoritma Raita membandingkan karakter terakhir dari jendela teks yang bersesuaian dengan *pattern*, jika cocok dilanjutkan ke karakter pertama, jika cocok juga dilanjutkan ke karakter tengah. Jika karakter akhir, pertama dan tengah cocok, selanjutnya algoritma akan membandingkan karakter lain, dimulai dari karakter kedua sampai ke satu karakter sebelum karakter terakhir, dan hal ini akan membuat karakter tengah dibandingkan lagi. Proses pencarian *string* dilakukan dengan memastikan bahwa setiap karakter dari pola *string* sesuai dengan setiap karakter pada target *string* pada posisi yang sama (Lecroq, 2014).

Kebanyakan algoritma pencocokan string terdiri dari fase *preprocessing* dan fase pencarian. Tahap *preprocessing* menganalisis karakter dalam pola untuk menggunakan informasi ini dalam menentukan pergeseran pola dalam kasus ketidakcocokan atau seluruh pencocokan, dengan tujuan mengurangi jumlah perbandingan karakter, sedangkan fase pencarian mendefinisikan urutan perbandingan karakter dalam setiap upaya antara pola dan teks (Klaib & Osborne, 2009).

Langkah-langkah yang dilakukan algoritma ini saat mencocokkan *string* adalah :

1. Algoritma Raita mulai mencocokkan karakter terakhir dari jendela teks yang bersesuaian dengan *pattern*.
2. Jika karakter tidak cocok, maka pola akan bergeser kekanan sebanyak nilai teks.
3. Jika cocok dilanjutkan ke karakter pertama.
4. Jika cocok juga dilanjutkan ke karakter tengah.
5. Jika karakter akhir, pertama dan tengah cocok, selanjutnya algoritma akan membandingkan karakter lain, dimulai dari karakter kedua sampai ke satu karakter sebelum karakter terakhir, dan hal ini akan membuat karakter tengah dibandingkan lagi.

Sebagai contoh perhitungan menggunakan algoritma Raita yaitu susunan teks **MI PADRE TOMA CAFE** dan pola yang ingin dicari adalah **TOMA**.

Maka diketahui pula perhitungannya sebagai berikut :

$T = \text{MI PADRE TOMA CAFE}$

$m = \text{TOMA}$

diketahui bahwa :

$m = \text{Panjang pola}$

$T = \text{Teks yang akan dicari}$

Maka :

$m = 4$

dibuatlah tabel BmBc untuk melakukan perhitungan dengan persamaan sebagai berikut:

$m - 2 \dots\dots\dots(1)$

$4 - 2 = 2$

Mencari nilai BmBc (a)

$m - 1 - i \dots\dots\dots(2)$

sebagai pencarian nilai karakter pada tabel BmBc

Tabel 2.1 Perhitungan tabel BmBc (a)

I	0	1	2	3
A	T	O	M	A
BmBc(a)	3	2	1	4

Perhitungan mencari nilai BmBc (a) dengan menggunakan rumus persamaan

$m - 1 - i \dots\dots\dots(2)$

$4 - 1 - 0 = 3$ maka nilai diletakkan pada indeks ke-0 dengan karakter T

$4 - 1 - 1 = 2$ maka nilai diletakkan pada indeks ke-1 dengan karakter O

$4 - 1 - 2 = 1$ maka nilai diletakkan pada indeks ke-2 dengan karakter M

Nilai A adalah 4 sesuai dengan panjang pola, karena abjad yang tidak ada pada tabel

maka diinisialisasikan dengan tanda (*) kemudian nilainya sesuai dengan panjang

pola. Jadi, untuk perhitungan algoritma raita sesuai tabel BmBc adalah sebagai berikut :

Tabel 2.2 Hasil BmBc (a)

A	T	O	M	A	*
BmBc(a)	3	2	1	4	4

Perhitungan mencari nilai hasil BmBc (a) dengan menggunakan rumus persamaan

$$m - 2 \dots (1)$$

$$3 - 2 = 1$$

Maka, dapatlah hasil tabel BmBc (a) menjadi 3 karakter.

T = MI PADRE TOMA CAFE

m = TOMA

Proses Pencarian Algoritma Raita :

1. Tahap pertama, yaitu mencocokkan pola akhir pada teks. Jika tidak cocok maka pola akan bergeser kekanan sebanyak nilai teks.

Tabel 2.3 Proses Pencarian Pada Teks Ke-1

Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola	T	O	M	A											

Pada proses dinyatakan terjadi ketidakcocokan pada teks, maka pada pola terakhir teks terdapat huruf terakhir yaitu A. Pada tabel hasil BmBc bahwa A bernilai 4 karakter.

2. Tahap kedua, yaitu pola akan bergeser sebanyak 4 karakter (sesuai dengan A huruf terakhirnya).

Tabel 2.4 Proses Pencarian Pada Teks Ke-2

Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola					T	O	M	A							

Pada pola terakhir terdapat huruf terakhir yaitu T. Pada tabel hasil BmBc bahwa T bernilai 3 karakter.

3. Tahap ketiga, yaitu jika pergeseran pola terdapat ketidakcocokan, maka pola akan terus bergeser sebanyak 3 karakter (sesuai dengan T huruf terakhirnya).

Tabel 2.5 Proses Pencarian Pada Teks Ke-3

Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola								T	O	M	A				

Pada percobaan terakhir, dengan teks MI PADRE TOMA CAFÉ dimana pola adalah TOMA telah ditemukan dengan Algoritma Raita akan terus bergerak kekanan hingga karakter teks berakhir. Namun, karena pada kondisi tersebut pola dan teks sudah ditemukan, maka proses pencarian telah berakhir.

2.4 Algoritma Not So Naïve

Algoritma *Not So Naïve* pertama kali dipublikasikan oleh Christophe Hancart tahun 1992. Algoritma *Not So Naïve* merupakan variasi turunan dari algoritma *Naïve* atau yang sering disebut algoritma *Brute Force*. Cara kerja algoritma ini adalah dengan memiliki fase pencarian mengecek teks dan pola dari kiri ke kanan. Lalu, algoritma *Not So Naïve* akan mengidentifikasi terlebih dahulu dua kasus yang dimana di setiap akhir fase pencocokan pergeseran dapat dilakukan sebanyak 2 posisi ke kanan, tidak seperti algoritma *Naïve* yang dimana pergeseran tetapkan sebanyak 1 posisi ke kanan.

Kita asumsikan bahwa $P[0] \neq P[1]$. Jika $P[0] = T[s]$ dan $P[1] = T[s+1]$, maka diakhir fase pencocokan pergeseran s bisa dilakukan sebanyak 2 posisi. Karena $P[0] \neq P[1] = T[s+1]$. Dan jika $P[0] = P[1]$. Jika $P[0] = T[s]$ tapi $P[1] \neq T[s+1]$. Maka sekali lagi pergeseran s dapat dilakukan sebanyak 2 posisi dimana P adalah *Pattern*, T adalah *Teks* dan s adalah nilai posisi (Cantone & Faro, 2004).

Saat fase pencarian dari algoritma *Not So Naïve* perbandingan karakter dilakukan dengan posisi pola mengikuti urutan 1, 2, ..., $m-2$, $m-1$, 0 dimana m adalah panjang *pattern*. Di setiap percobaan dimana “jendela” diposisikan di teks faktor $y[i..j+m-1]$. jika $x[0] = x[1]$ dan $x[1] y[$

$j+1]$ atau jika $x[0] \neq x[1]$ dan $x[1] = y[j+1]$ polanya akan digeser sebanyak 2 posisi di setiap akhir percobaan dan sebanyak 1 posisi jika kondisi di atas tidak terpenuhi, dimana y adalah teks dan x adalah *pattern* (Alapati & Mannava, 2011).

Langkah-langkah yang dilakukan algoritma ini saat mencocokkan *string* adalah :

1. Algoritma *No So Naïve* mulai mencocokkan karakter dari kiri ke kanan.
2. Algoritma ini akan mencocokkan karakter per karakter pola dengan karakter teks yang bersesuaian, sampai salah satu kondisi berikut terpenuhi :
 - a. Jika perbandingan karakter pertama cocok, namun perbandingan kedua tidak cocok, maka posisi pola akan digeser sebanyak 2 posisi sesuai dengan nilai variabel *ell*.
 - b. Jika perbandingan karakter pertama tidak cocok, maka akan dilakukan percobaan dengan pergeseran 1 posisi sesuai dengan variabel *k*.
 - c. Semua karakter di pola cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian terus menggeser pola dari kiri ke kanan, dan mengulangi langkah ke-2 sampai sisa teks lebih kecil dari pada pola.

Berikut diberikan contoh untuk menunjukkan proses pencarian Algoritma *Not So Naive* dimana karakter urutan 0 dan karakter urutan 1 pada *pattern* tidak mengalami kesamaan ($x[0] \neq x[1]$) maka nilai variabel *k* akan diinisialisasi dengan nilai 1 dan nilai variabel *ell* akan diinisialisasi dengan nilai 2 dimana kedua variabel tersebut akan digunakan untuk nilai pergeseran pada proses pencocokan.

Sebagai contoh perhitungan menggunakan Algoritma *Not So Naive* yaitu susunan teks **MI PADRE TOMA CAFE** dan pola yang ingin dicari adalah **TOMA**.

Teks = MI PADRE TOMA CAFE

Pola = TOMA

Tabel 2.6 Proses Pencarian Pada Teks Ke-1

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola	T	O	M	A											

Pada Tabel 2.6 perbandingan karakter pertama ($x[1] \neq y[j+1]$) mengalami ketidakcocokan. Maka akan dilakukan percobaan kedua dengan posisi pola digeser sebanyak 1 posisi sesuai dengan nilai variabel k.

Tabel 2.7 Proses Pencarian Pada Teks Ke-2

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola		T	O	M	A										

Pada Tabel 2.7 perbandingan karakter pertama ($x[1] \neq y[j+1]$) belum mengalami kecocokan karakter. Maka akan dilakukan percobaan selanjutnya dengan posisi pola digeser sebanyak 1 posisi sesuai dengan nilai variabel k.

Tabel 2.8 Proses Pencarian Pada Teks Ke-3

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola			T	O	M	A									

Pada Tabel 2.8 perbandingan karakter pertama ($x[1] \neq y[j+1]$) belum mengalami kecocokan karakter. Maka akan dilakukan percobaan selanjutnya dengan posisi pola digeser sebanyak 1 posisi sesuai dengan nilai variabel k.

Tabel 2.9 Proses Pencarian Pada Teks Ke-4

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola				T	O	M	A								

Pada Tabel 2.9 perbandingan karakter pertama ($x[1] \neq y[j+1]$) belum mengalami kecocokan karakter. Maka akan dilakukan percobaan selanjutnya dengan posisi pola digeser sebanyak 1 posisi sesuai dengan nilai variabel k.

Tabel 2.10 Proses Pencarian Pada Teks Ke-5

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola					T	O	M	A							

Pada Tabel 2.10 perbandingan karakter pertama ($x[1] \neq y[j+1]$) belum mengalami kecocokan karakter. Maka akan dilakukan percobaan selanjutnya dengan posisi pola digeser sebanyak 1 posisi sesuai dengan nilai variabel k.

Tabel 2.11 Proses Pencarian Pada Teks Ke-6

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola						T	O	M	A						

Pada Tabel 2.11 perbandingan karakter pertama ($x[1] \neq y[j+1]$) belum mengalami kecocokan karakter. Maka akan dilakukan percobaan selanjutnya dengan posisi pola digeser sebanyak 1 posisi sesuai dengan nilai variabel k.

Tabel 2.12 Proses Pencarian Pada Teks Ke-7

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola							T	O	M	A					

Pada Tabel 2.12 perbandingan karakter pertama ($x[1] \neq y[j+1]$) belum mengalami kecocokan karakter. Maka akan dilakukan percobaan selanjutnya dengan posisi pola digeser sebanyak 1 posisi sesuai dengan nilai variabel k.

Tabel 2.13 Proses Pencarian Pada Teks Ke-8

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola								T	O	M	A				

Pada Tabel 2.13 Perbandingan karakter mengalami kecocokan ($x[1] == y[j+1]$) dimulai dari perbandingan karakter T, O, M dan A semua mengalami kecocokan, oleh sebab itu teks akan dikeluarkan. Namun algoritma *Not So Naive* belum berhenti dan akan melakukan percobaan sampai sisa teks lebih kecil dari pola. Pada percobaan selanjutnya, posisi pola akan digeser sebanyak 2 posisi sesuai dengan nilai variabel *ell*.

Tabel 2.14 Proses Pencarian Pada Teks Ke-9

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola										T	O	M	A		

Pada Tabel 2.14 perbandingan karakter pertama ($x[1] != y[j+1]$) tidak mengalami kecocokan, namun pada karakter kedua mengalami kecocokan. Maka pada hal ini saat perbandingan karakter utama tidak terjadi kecocokan, maka posisi pola akan digeser sebanyak 1 posisi sesuai dengan nilai variabel *k*.

Tabel 2.15 Proses Pencarian Pada Teks Ke-10

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teks	M	I	P	A	D	R	E	T	O	M	A	C	A	F	E
Pola											T	O	M	A	

Pada Tabel 2.15 perbandingan karakter pertama ($x[1] != y[j+1]$) tidak mengalami kecocokan, namun pada karakter kedua mengalami kecocokan. Maka pada hal ini saat perbandingan karakter utama tidak terjadi kecocokan, maka posisi pola akan digeser sebanyak 1 posisi sesuai dengan nilai variabel *k*. Namun karena sisa teks lebih kecil dari pada pola, maka pencarian berhenti di tahap percobaan ini.

2.5 Kompleksitas Algoritma

Kompleksitas algoritma adalah ukuran atau metrik yang mengukur performa dan efisiensi suatu algoritma dalam memproses data. Ini membantu untuk menentukan berapa lama waktu yang dibutuhkan oleh algoritma untuk menyelesaikan masalah dan berapa banyak sumber daya yang dibutuhkan. Algoritma yang bagus adalah algoritma yang efisien. Algoritma yang efisien dapat menyelesaikan suatu permasalahan dalam waktu yang singkat dan memiliki kompleksitas yang rendah.

2.6 Kamus

Menurut Kamus Besar Bahasa Indonesia, pengertian kamus adalah buku acuan yang memuat kata dan ungkapan yang biasanya disusun berdasarkan abjad berikut keterangan dan maknanya, pemakaiannya atau terjemahannya. Kamus juga dapat digunakan sebagai buku rujukan yang menerangkan makna kata-kata yang berfungsi untuk membantu seseorang mengenal perkataan baru atau asing bagi pembaca. Kamus dapat diterbitkan dalam bentuk buku cetak atau digital, dan bisa berisi kata-kata dalam bahasa lokal atau bahasa asing.

2.7 Penelitian Relevan

Berikut ini beberapa penelitian yang terkait dengan algoritma Raita dan algoritma *Not So Naive* :

1. Penelitian yang dilakukan oleh (Suwitri, 2017) dalam skripsi yang berjudul “Perbandingan Algoritma Raita Dan Algoritma *Quick Search* Pada Aplikasi Kamus Bahasa Indonesia – Bahasa Prancis”. Hasil penelitian yang didapat bahwa *running time* dari algoritma Raita lebih cepat dibandingkan algoritma *Quick Search* serta kompleksitas algoritma Raita lebih kecil dari algoritma *Quick Search*. Rata-rata total running time untuk algoritma Raita adalah 0,231 ms dan algoritma *Quick Search* adalah 6,536 ms.
2. Penelitian yang dilakukan oleh (Fajar, 2018) dalam skripsi yang berjudul “Implementasi Dan Perbandingan Algoritma Reverse Colussi Dan Algoritma Raita Pada Aplikasi Pencarian Lirik Lagu Bahasa Indonesia Dengan *Speech Recognition* Berbasis Android”. Hasil penelitian yang didapat bahwa untuk 10 kali percobaan dengan rentang panjang *pattern* sebesar 5-263 karakter, algoritma Reverse Colussi memiliki rata-rata waktu pencarian sebesar 954,4 ms dan algoritma Raita sebesar 265,8 ms.

3. Penelitian yang dilakukan oleh (Faathir, 2018) dalam skripsi yang berjudul “Perbandingan Algoritma Horspool Dan *Not So Naive* Dalam Pembuatan Kamus Bahasa Indonesia – Bahasa Aceh Berbasis Android” menyimpulkan bahwa algoritma Horspool lebih lambat dibandingkan algoritma *Not So Naive*. Hal ini ditunjukkan dari *running time* yang didapat yaitu, algoritma Horspool mempunyai rata-rata *running time* selama 26,2 ms, sedangkan algoritma *Not So Naive* mempunyai rata-rata *running time* selama 18,9 ms.
4. Penelitian yang dilakukan oleh (Situmorang, 2019) dalam skripsi yang berjudul “Perbandingan Algoritma *Not So Naive* Dan Algoritma Zhu-Takaoka Pada Aplikasi Kamus Farmakologi Berbasis Android” mendapatkan hasil penelitian bahwa *running time* algoritma *Not So Naive* lebih cepat dibandingkan algoritma Zhu-Takaoka. Rata-rata *running time* untuk algoritma *Not So Naive* adalah 11,6 ms dan algoritma Zhu-Takaoka adalah 48,6 ms.

BAB 3

ANALISIS DAN PERANCANGAN SISTEM

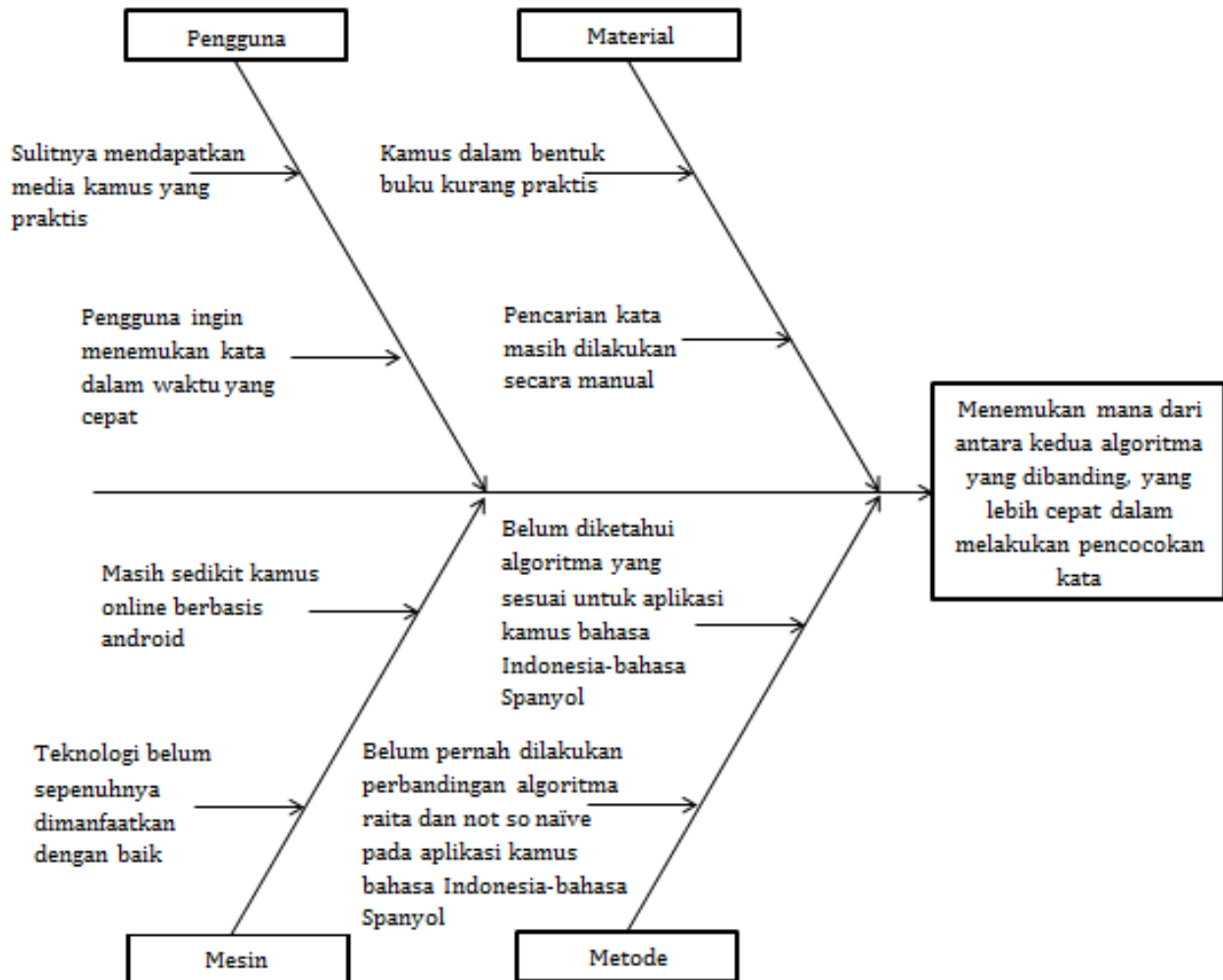
3.1 Analisis Sistem

Analisis sistem adalah proses evaluasi, pemahaman serta perencanaan suatu sistem untuk meningkatkan efisiensi, kinerja dan tujuan sistem tersebut. Analisis sistem penting dalam pengembangan dan pengelolaan sistem komputer, bisnis dan organisasi lainnya untuk memastikan bahwa mereka beroperasi secara efisien dan efektif sesuai dengan tujuan yang ditetapkan. Analisis sistem terdiri dari tiga fase untuk mendeskripsikan pengembangan sistem yaitu, analisis masalah, analisis kebutuhan dan analisis proses.

3.1.1 Analisis Masalah

Analisis masalah merupakan proses sistematis yang dilakukan untuk memahami, mengidentifikasi dan menganalisis masalah atau tantangan yang dihadapi dalam suatu konteks. Seperti yang telah dipaparkan pada latar belakang tugas akhir ini, permasalahan yang akan diselesaikan dengan menggunakan sistem ini adalah bagaimana membuat kamus bahasa Indonesia – bahasa Spanyol menggunakan algoritma raita dan algoritma *not so naïve*. Tujuan dari analisis masalah adalah untuk mendapatkan pemahaman yang mendalam tentang masalah tersebut sehingga solusi yang efektif dapat dirancang dan diimplementasikan.

Untuk membantu mengidentifikasi masalah tersebut cara populer yang sering digunakan adalah diagram Ishikawa (*fishbone diagram*). Untuk pertama kalinya pada tahun 1968 diagram Ishikawa ini dikenalkan oleh Kaoru Ishikawa. Diagram ishikawa adalah diagram yang menggambarkan secara detail semua penyebab yang berhubungan dengan suatu permasalahan. Diagram Ishikawa berbentuk seperti ikan yang strukturnya terdiri dari kepala ikan (*fish's head*) dan tulang-tulang ikan (*fish's bones*). Nama atau judul dari masalah yang diidentifikasi terletak pada bagian kepala ikan, sedangkan tulang-tulang ikan menggambarkan penyebab-penyebab masalah tersebut (Whitten et al., 2004).



Gambar 3.1 Diagram Ishikawa

Pada gambar 3.1 di atas terdapat empat kategori yang menjadi penyebab masalah dalam pencarian kata menggunakan metode pencocokan *string*. Keempat kategori tersebut adalah pengguna (*user*), material, mesin, dan metode yang direpresentasikan di dalam bagian sirip. Setiap detail dari empat kategori masalah tersebut direpresentasikan di dalam bagian duri dengan masing-masing penjelasannya. Keseluruh detail tersebut mengarah ke bagian kepala yang merupakan masalah sekaligus menjadi topik utama dalam analisis masalah.

3.1.2 Analisis Kebutuhan

Analisis kebutuhan terbagi atas dua bagian, yaitu kebutuhan fungsional dan kebutuhan nonfungsional. Kebutuhan fungsional mendeskripsikan aktivitas yang disediakan suatu sistem. Sedangkan kebutuhan nonfungsional mendeskripsikan fitur, karakteristik dan batasan lainnya (Whitten, 2004).

3.1.2.1 Analisis Kebutuhan Fungsional

Berikut ini beberapa hal yang menjadi kebutuhan fungsional pada Aplikasi kamus bahasa Indonesia – bahasa Spanyol :

1. Sistem dapat melakukan pencocokan *string* dari kata yang dimasukkan oleh pengguna.
2. Sistem dapat melakukan pencarian kata dengan menggunakan algoritma Raita dan algoritma *Not So Naïve*.
3. Sistem dapat menampilkan hasil dari pencarian dan menampilkannya pada antarmuka sistem.
4. Sistem dapat menghitung waktu pada saat proses pencarian kata dengan satuan (*ms*).

3.1.2.2 Analisis Kebutuhan Nonfungsional

Berikut ada beberapa hal yang menjadi kebutuhan nonfungsional pada aplikasi kamus bahasa Indonesia – bahasa Spanyol :

1. Performa
Sistem yang dibangun dapat menunjukkan hasil pencarian kata yang diinputkan oleh pengguna.
2. Mudah digunakan
Sistem yang dibangun mudah dipahami dan digunakan (*user friendly*), artinya sistem mempunyai (*interface*) yang sederhana dan mudah dimengerti sehingga pengguna tertarik untuk menggunakan sistem dan mudah memahami setiap desain sistem.
3. Hemat biaya
Sistem yang dibangun hemat biaya karena aplikasi dapat di *share* secara gratis dengan *bluetooth* dan ukuran *file* aplikasi tidak terlalu besar.
4. Panduan
Sistem yang dibangun memiliki panduan pengguna.

5. Kontrol

Sistem yang dibangun akan menampilkan pesan peringatan ‘kata tidak ditemukan’ ketika kata yang dicari tidak terdapat di dalam database.

3.1.3 Analisis Proses

Sistem yang akan dibangun menggunakan algoritma Raita dan *Not So Naïve* untuk proses pencocokan *string*. Sistem akan melakukan pencocokan *string* berdasarkan kata yang dimasukkan oleh pengguna (*pattern*) dan kata yang ada di dalam database. Kemudian, kata di dalam database yang mengandung *pattern* yang dimasukkan oleh *user* akan ditampilkan di sistem.

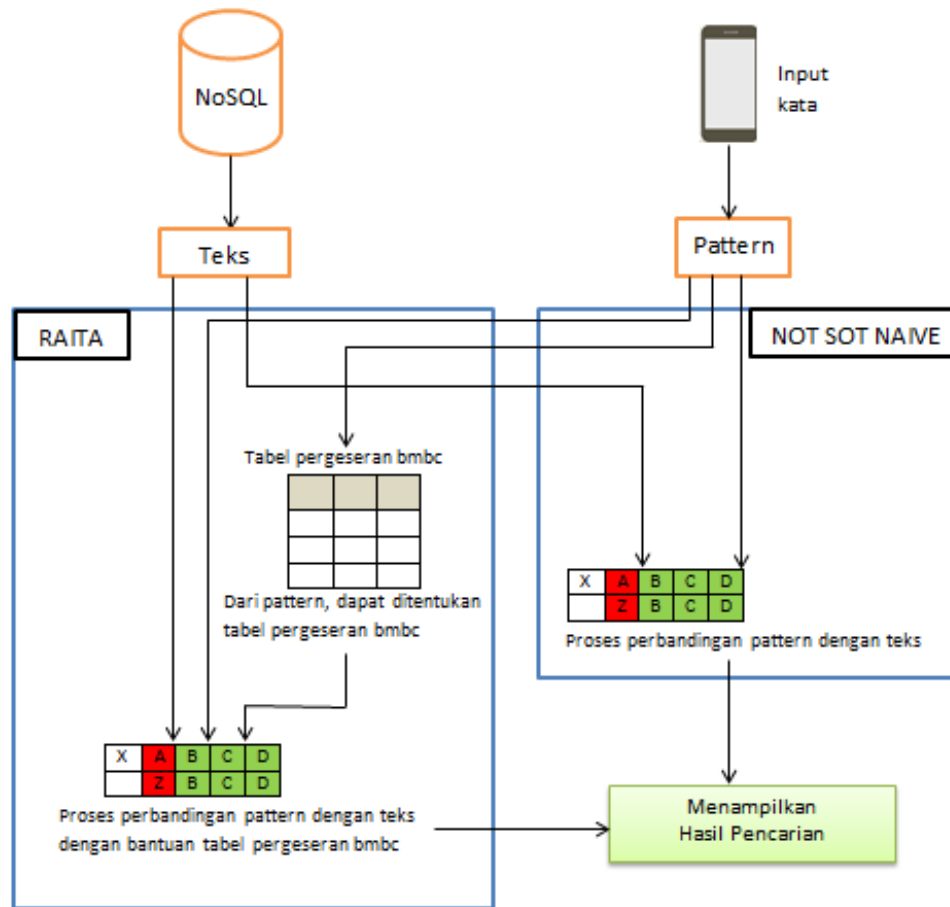
3.2 Perancangan Sistem

Perancangan sistem merupakan cara untuk mendapatkan gambaran dengan jelas tentang apa yang akan dikerjakan pada analisis sistem dan dilanjutkan dengan memikirkan bagaimana membuat sistem tersebut. Tujuan dari tahap perancangan sistem ini adalah untuk menjelaskan, mempermudah dan mengevaluasi implementasi sistem yang akan dibangun.

Pada penelitian ini, penulis menggunakan UML (*Unified Modeling Language*) sebagai bahasa pemodelan untuk merancang sistem kamus bahasa Indonesia – bahasa Spanyol dengan metode pencocokan string. Adapun UML yang akan digunakan yaitu general arsitektur, *use case* diagram, *activity* diagram, *sequence* diagram, dan *flowchart*.

3.2.1 General Arsitektur Sistem

General arsitektur sistem adalah gambaran sistem secara keseluruhan yang meliputi proses *input* data, proses sistem berlangsung, dan *output* data yang dikeluarkan dari sistem. Ada beberapa hal yang dapat dilakukan pengguna kepada sistem, yaitu seperti yang ditunjukkan pada **Gambar 3.2** berikut.



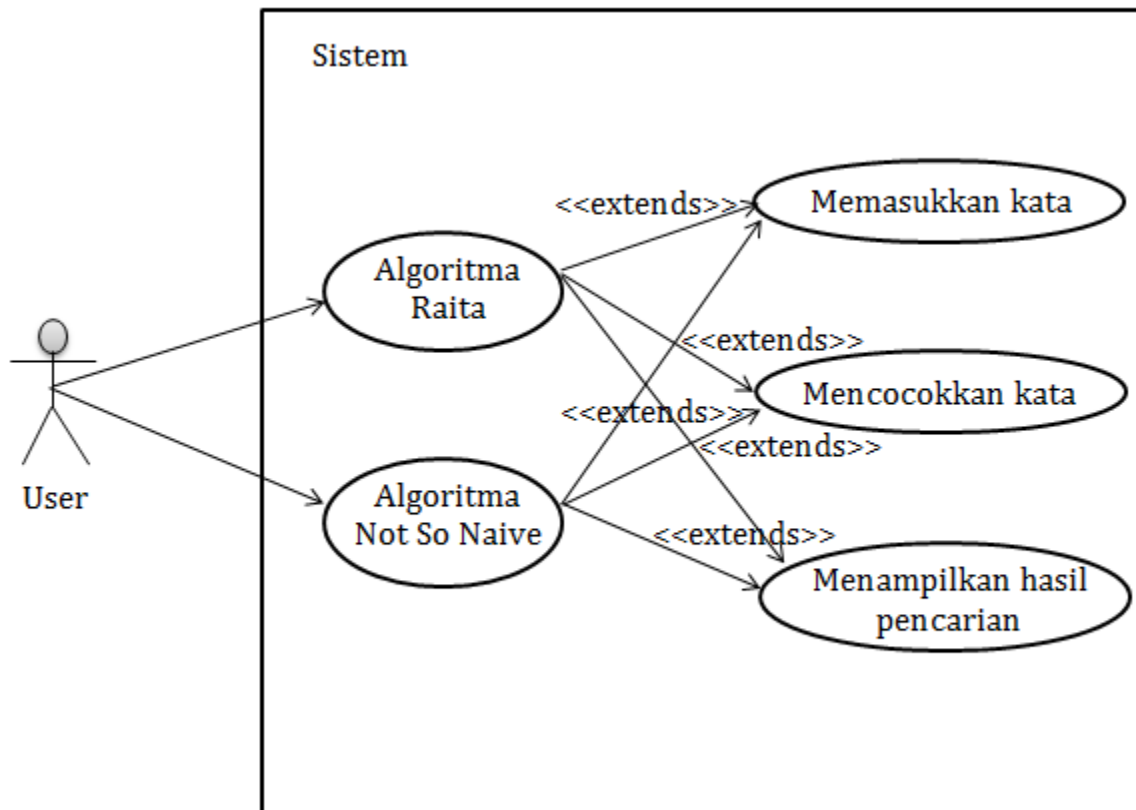
Gambar 3.2 General Arsitektur Sistem

Pada gambar diatas dijelaskan ketika user sudah memasukkan kata (*pattern*), *user* akan memilih untuk memakai algoritma Raita atau *Not So Naïve*. Setelah itu sistem akan mencocokkan kata dengan teks yang terdapat didalam database. Setelah melakukan pencocokan, maka sistem akan mengeluarkan hasil dari pencocokan yang sudah dilakukan.

3.2.2 Use Case Diagram

Use case diagram merupakan sebuah diagram yang dapat mempresentasikan interaksi yang terjadi antara *user* dengan sistem. Diagram *use case* ini mendeskripsikan siapa saja yang menggunakan sistem dan bagaimana cara mereka berinteraksi dengan sistem. Sebuah *use case*

digambarkan sebagai elips horizontal dalam suatu diagram *use case* (Haviluddin, 2011). *Use case* diagram dari sistem yang akan dibangun dapat ditunjukkan pada **Gambar 3.3**

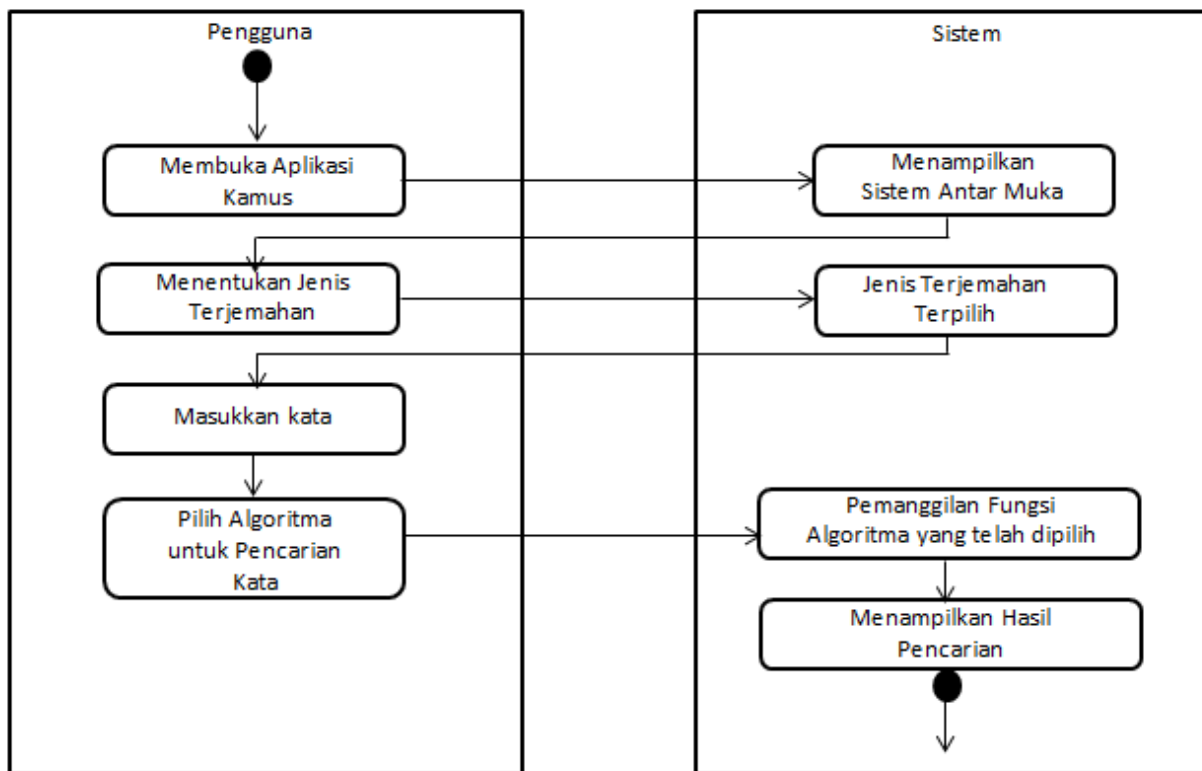


Gambar 3.3 *Use Case* Diagram

Diagram *use case* pada gambar 3.3 seorang *Actor* berinteraksi dengan dua *use case* dan tiga *use case* yang merupakan *extend* dari *use case* Raita dan *Not So Naïve*. Panah *extend* pada *use case* Raita dan *Not So Naïve* menjelaskan lebih detail perilaku yang dapat dilakukan pada *use case* tersebut.

3.2.3 Activity Diagram

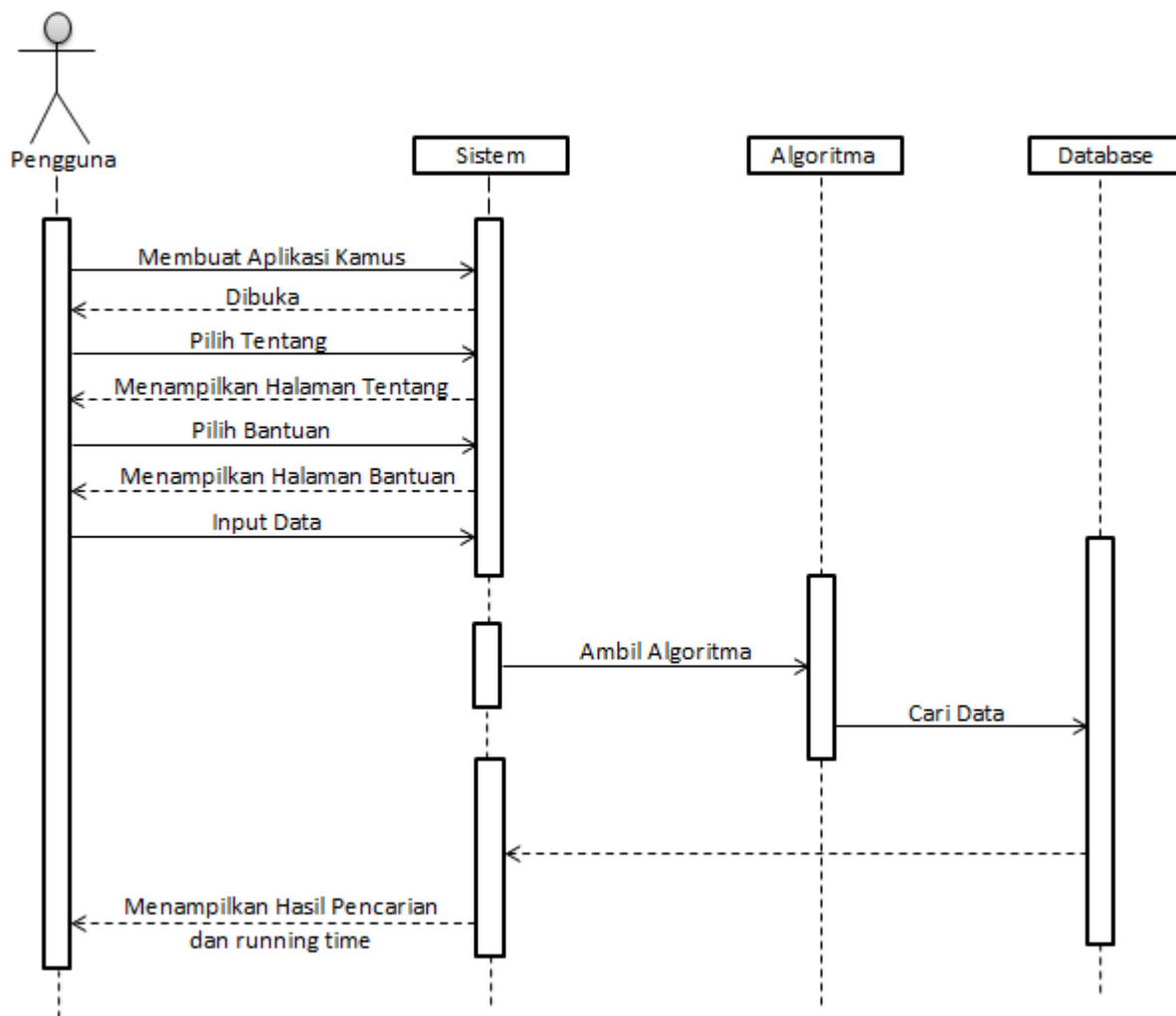
Activity diagram menggambarkan aliran kerja (*workflow*) atau aktivitas dari sebuah sistem atau proses bisnis. Aktivitas suatu sistem serta proses timbal balik antar sistem tersebut digambarkan dalam *activity diagram* (Setiady & Yulistia, 2016). Pada diagram aktivitas, aktivitas atau tugas direpresentasikan sebagai simpul (*node*) yang dihubungkan oleh panah yang menunjukkan aliran kontrol antara aktivitas tersebut. Ini membantu dalam memodelkan urutan langkah – langkah yang harus diambil dalam suatu proses, termasuk pengambilan keputusan dan pengulangan tugas tertentu.



Gambar 3.4 *Activity Diagram*

3.2.4 Sequence Diagram

Sequence diagram membantu dalam memahami dan merancang komunikasi antar objek-objek dalam sistem dengan jelas. Diagram ini memberikan gambaran visual tentang urutan pesan atau panggilan yang dikirim antara objek-objek selama eksekusi skenario tertentu. *Sequence* diagram dari sistem aplikasi kamus bahasa Indonesia – Spanyol yang akan dibangun dapat ditunjukkan pada Gambar 3.5.

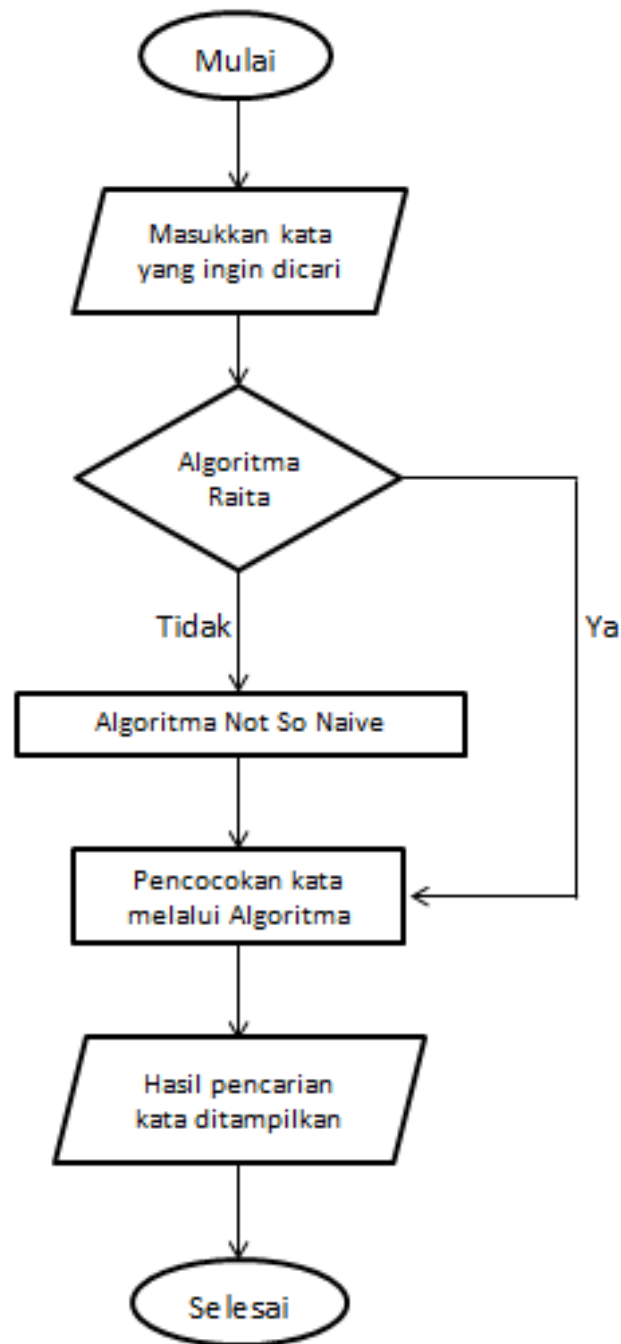


Gambar 3.5 *Sequence* Diagram

3.2.5 Flowchart

Flowchart merupakan alat visual yang digunakan untuk merepresentasikan urutan langkah-langkah atau proses dalam bentuk diagram. *Flowchart* membantu dalam menjelaskan suatu

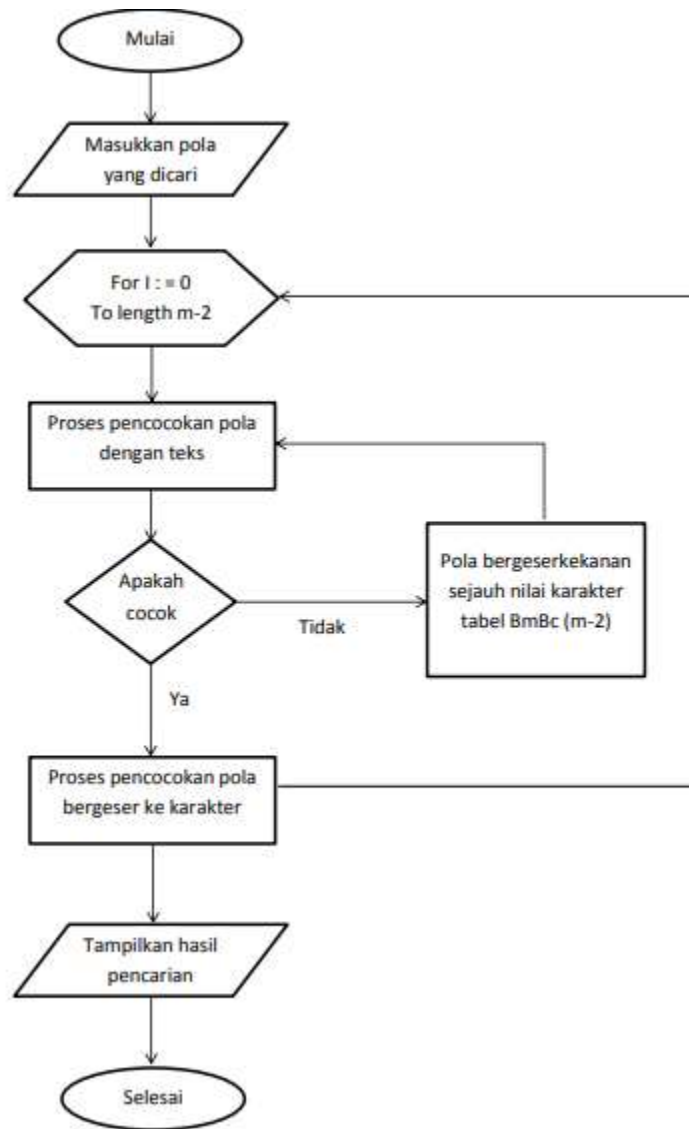
proses dengan cara yang mudah. *Flowchart* divisualisasi dengan simbol, setiap simbol menggambarkan proses tertentu. Sedangkan antara proses digambarkan dengan garis penghubung. Untuk *flowchart* sistem ini akan ditunjukkan pada Gambar 3.6 berikut ini.



Gambar 3.6 *Flowchart* Sistem

3.2.6 Flowchart Algoritma Raita

Untuk *flowchart* algoritma Raita dapat dilihat pada Gambar 3.7 di bawah ini.

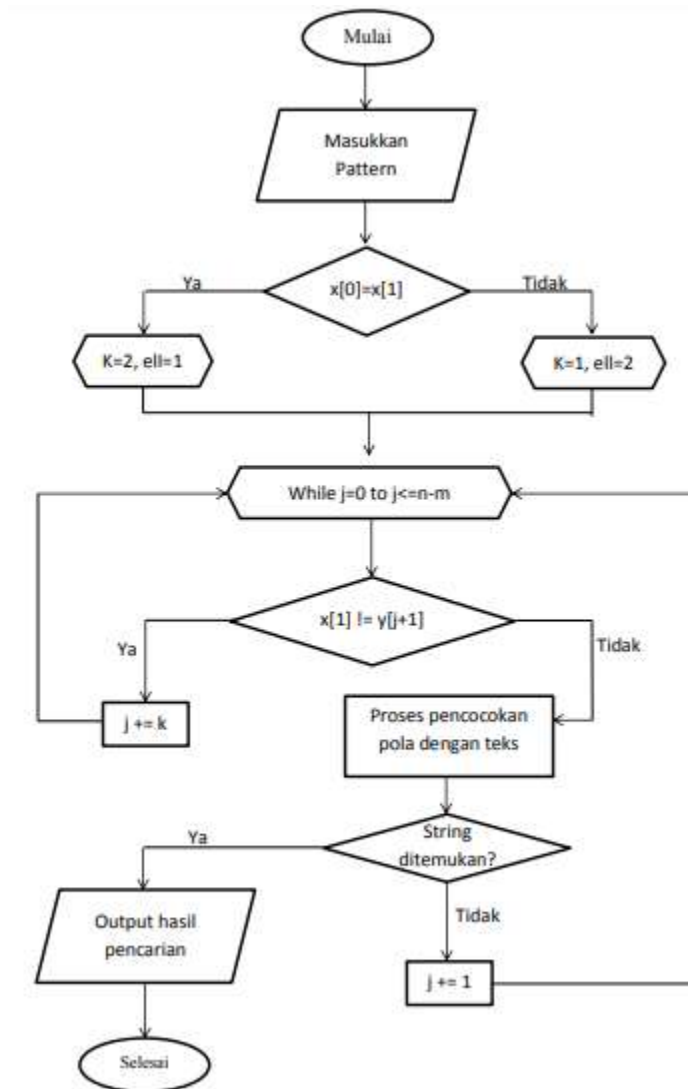


Gambar 3.7 *Flowchart* Proses Pencarian Algoritma Raita

Gambar 3.7 menggambarkan alur proses pencarian pada algoritma Raita, dimana proses awal dengan dimulai kemudian masukkan pola yang tersedia, setelah itu pola akan dinilai sesuai pada tabel BmBc m-2, kemudian pola akan bergeser kekanan dengan melihat karakter yang paling kanan jendela pola tersebut, jika sudah ditemukan maka ditampillah hasil *running time* tersebut.

3.2.7 Flowchart Algoritma Not So Naïve

Untuk *flowchart* algoritma *Not So Naïve* dapat dilihat pada Gambar 3.8 di bawah ini.



Gambar 3.8 *Flowchart* Proses Pencarian Algoritma *Not So Naïve*

Pada gambar 3.8 menggambarkan alur pada proses pencarian algoritma *Not So Naïve*, dimana proses awal yang dilakukan adalah melihat apakah karakter urutan 0 dan 1 pada pola berupa karakter yang sama atau tidak. Jika sama, maka variabel k akan diberi nilai 2 dan variabel ell diberi nilai 1 (nilai k digunakan sebagai nilai pergeseran jika saat fase pencocokan karakter di urutan 1 mengalami ketidakcocokan dan nilai ell digunakan sebagai nilai pergeseran jika saat fase pencocokan karakter di urutan 1 mengalami kecocokan namun di urutan selanjutnya mengalami ketidakcocokan). Lalu dilanjutkan ke fase pencocokan dimana variabel x adalah

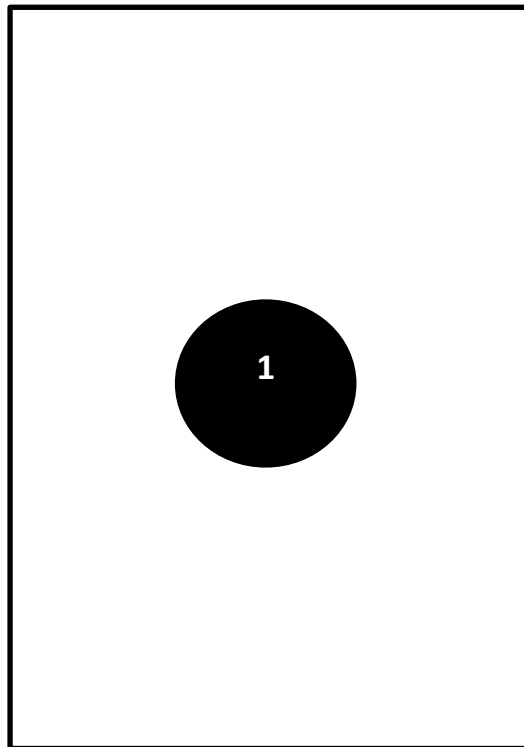
panjang teks, variabel y adalah panjang *pattern*, variabel j adalah nilai untuk perulangan pencocokan, variabel m untuk panjang pola, dan n untuk panjang teks, dari kiri ke kanan sampai string ditemukan atau posisi pola bergeser sampai penghujung teks.

3.3 Perancangan Antarmuka (*Interface*)

Perancangan *interface* merupakan suatu bagian yang sangat penting dalam sebuah perancangan sistem. Sebuah *interface* harus memperhatikan keadaan pengguna (*user*) sehingga sistem yang dibangun dapat memberikan kenyamanan dan kemudahan untuk digunakan oleh *user*.

3.3.1 Perancangan halaman splash screen

Halaman *splash screen* adalah halaman yang pertama kali di tampilkan ketika seseorang membuka sebuah aplikasi. Seperti yang ditunjukkan pada gambar berikut.



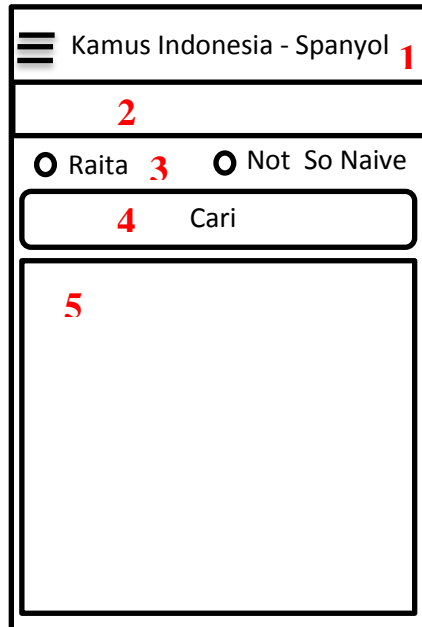
Gambar 3.9 Rancangan *Splash Screen*

Keterangan :

1. Merupakan gambar yang menampilkan icon aplikasi kamus.

3.3.2 Rancangan Halaman Utama

Halaman utama adalah halaman yang muncul setelah *splash screen*. Halaman utama terdiri dari *Navigation Drawer*, *Edit Text*, *Button* dan *List View*. Rancangan halaman beranda dapat dilihat pada gambar 3.10.



Gambar 3.10 Rancangan Halaman Utama

Keterangan :

1. *Toolbar* menunjukkan judul dari aplikasi yang dibangun
2. *Text Box* untuk menampung kata yang akan dicari
3. *Radio Button* untuk memilih algoritma yang akan digunakan
4. *Button* untuk memulai pencarian
5. *List View* untuk menampilkan hasil pencarian dari kata yang di cari

3.3.3 Rancangan *Navigation Drawer*

Navigation Drawer terdiri dari *Header*, Halaman utama, Tentang dan Bantuan. Rancangan *Navigation Drawer* dapat dilihat pada Gambar 3.11.



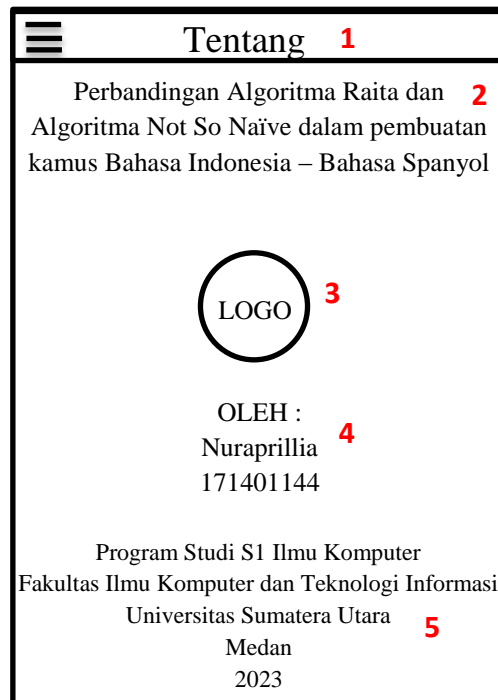
Gambar 3.11 Rancangan *Navigation Drawer*

Keterangan :

1. *Header* berisi icon aplikasi kamus.
2. Menu mode terjemahan Indonesia – Spanyol merupakan halaman utama aplikasi.
3. Menu mode terjemahan Spanyol – Indonesia merupakan pilihan untuk *user* apabila ingin mengubah mode terjemahan.
4. Menu Tentang berisi judul aplikasi dan pembuat aplikasi.
5. Menu Bantuan berisikan panduan *user* untuk menggunakan aplikasi.

3.3.4 Rancangan Halaman Tentang

Pada halaman tentang memuat judul penelitian dan identitas singkat dari penulis. Rancangan halaman tentang dapat dilihat pada gambar 3.12.



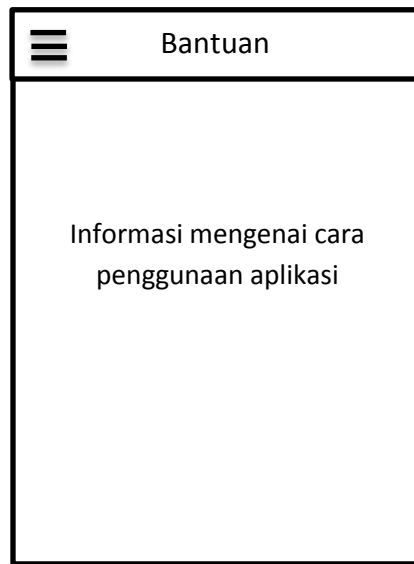
Gambar 3.12 Rancangan Halaman Tentang

Keterangan :

1. *Action Bar* berisikan judul dari halaman aplikasi.
2. *TextView* untuk menampilkan judul dari penelitian.
3. *ImageView* untuk menampilkan logo Universitas.
4. *TextView* untuk menampilkan identitas singkat penulis.
5. *TextView* untuk menampilkan Program Studi, Fakultas, Universitas, kota dan tahun.

3.3.5 Rancangan Halaman Bantuan

Halaman bantuan berisikan panduan untuk menggunakan aplikasi yang akan dibangun. Tampilan untuk halaman bantuan dapat dilihat pada Gambar 3.13.



Gambar 3.13 Rancangan Halaman Bantuan

Keterangan :

1. *Action Bar* berisikan judul dari halaman aplikasi.
2. *TextView* untuk menampilkan poin – poin bantuan.

BAB 4

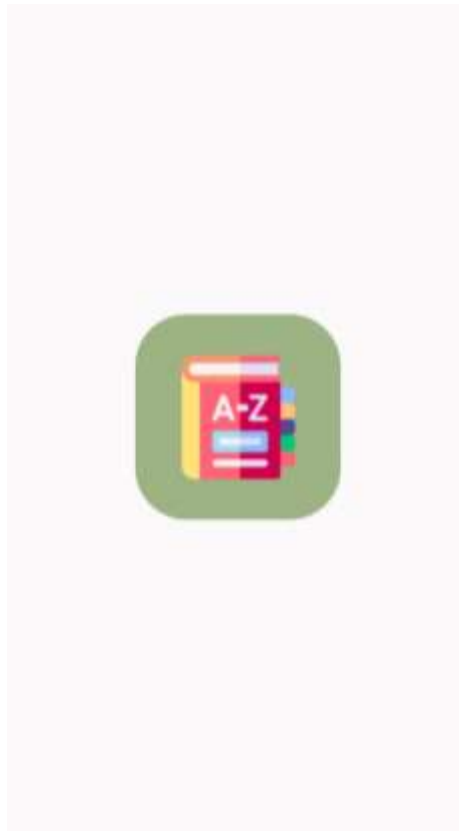
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Sistem

Implementasi sistem merupakan langkah lanjutan dari tahapan analisis dan perancangan sistem yang dirangkum di bab tiga. Pada tahap ini akan diimplementasikan segala hal yang telah dibahas pada tahapan analisis dan perancangan ke dalam bahasa pemrograman *Java* dan menggunakan *Software Android Studio*.

4.1.1 Tampilan Halaman *Splash Screen*

Halaman *Splash Screen* adalah tampilan awal sementara untuk menampilkan logo aplikasi. Tampilan Halaman *Splash Screen* dapat dilihat pada Gambar 4.1.



Gambar 4.1 Halaman *Splash Screen*

4.1.2 Tampilan Halaman Utama

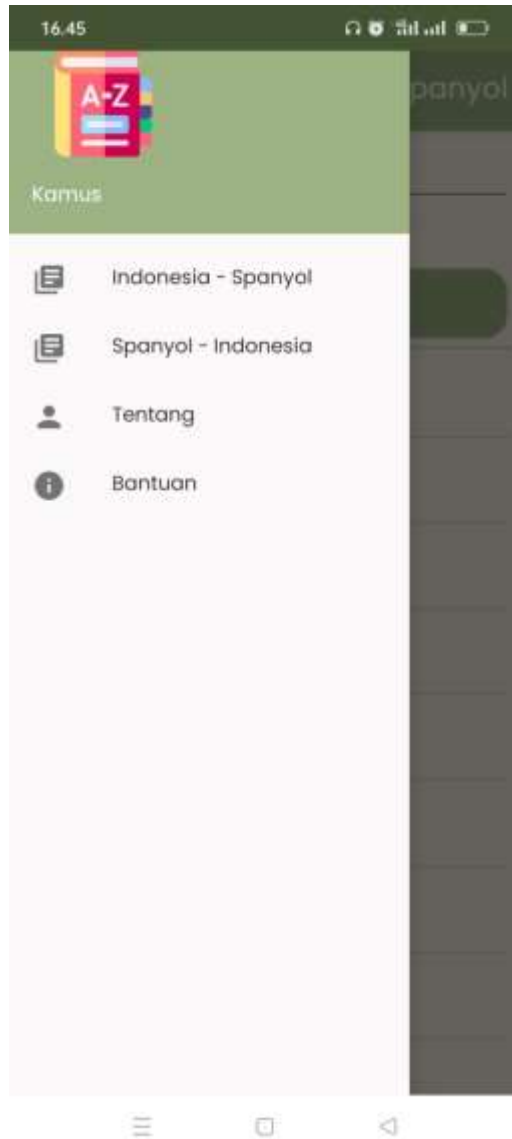
Pada menu utama terdapat menu navigasi yang berfungsi untuk menampilkan beberapa menu yang telah disediakan. Halaman utama berfungsi untuk melakukan pencarian kata pada kamus sesuai dengan algoritma yang dipilih. Tampilan halaman utama dapat dilihat pada Gambar 4.2.



Gambar 4.2 Halaman Utama

4.1.3 Tampilan Menu Navigasi

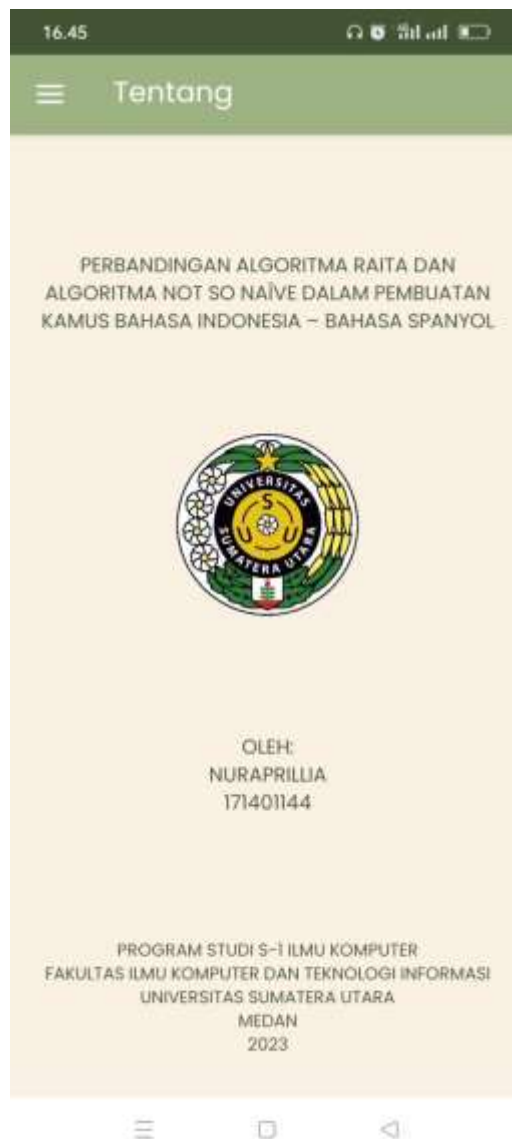
Pada menu navigasi terdapat menu pilihan yaitu mode terjemahan Indonesia – Spanyol, Spanyol – Indonesia, Tentang dan Bantuan. Menu navigasi akan muncul ketika *icon* navigasi pada menu utama dipilih. Tampilan Menu Navigasi dapat dilihat pada Gambar 4.3.



Gambar 4.3 Menu Navigasi

4.1.4 Tampilan Halaman Tentang

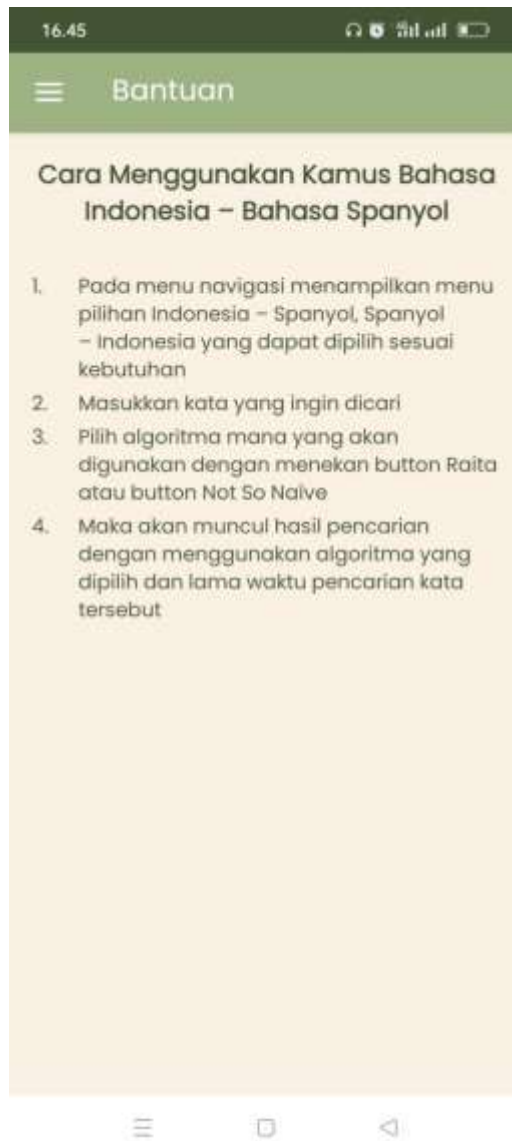
Pada halaman tentang memuat informasi singkat tentang penulis yang dapat dilihat pada Gambar 4.4



Gambar 4.4 Halaman Tentang

4.1.5 Tampilan Halaman Bantuan

Pada halaman bantuan memuat langkah – langkah penggunaan aplikasi yang dapat dilihat pada Gambar 4.5.



Gambar 4.5 Halaman Bantuan

4.2 Pengujian Sistem



Pengujian sistem dilakukan untuk memastikan bahwa sistem yang telah dibangun berjalan dengan baik serta sesuai dengan analisis dan perancangan sistem yang telah dibuat sebelumnya.



4.2.1 Pengujian Pencarian kata pada Kamus Bahasa Indonesia – Bahasa Spanyol dengan algoritma Raita



Pada **Tabel 4.1** dapat dilihat pencarian kata pada Kamus Bahasa Indonesia – Bahasa Spanyol menggunakan Algoritma Raita.



Tabel 4.1 Hasil Pencarian Kata Algoritma Raita



Pola	Hasil Pencocokan	Tampilan Hasil Pencocokan	Running Time
ab	cocok		89,9 ms
ada	cocok		98 ms



es	cocok	 <p>06.22 @</p> <p>Kamus Indonesia - Spanyol</p> <p>es</p> <p><input checked="" type="radio"/> Raita <input type="radio"/> Not So Naive</p> <p>CARI</p> <p>menyelesaikan acabar de agar sesuai caber kesopanan cortesia kesepuluh</p> <p>Running Time: 94.489846 ms OK</p>	94,4 ms
hujan	cocok	 <p>06.22 @</p> <p>Kamus Indonesia - Spanyol</p> <p>hujan</p> <p><input checked="" type="radio"/> Raita <input type="radio"/> Not So Naive</p> <p>CARI</p> <p>jas hujan impermeable hujan lluvia</p> <p>Running Time: 104.424077 ms OK</p>	104,4 ms



laku	cocok		99,2 ms
makan	cocok		109 ms



milik	cocok	 <p>06.23 06.23</p> <p>Kamus Indonesia - Spanyol</p> <p>milik</p> <p><input checked="" type="radio"/> Raita <input type="radio"/> Not So Naive</p> <p>CARI</p> <p>milikku mio pemilik propietario memiliki tener milikmu</p> <p>Running Time: 105.643385 ms OK</p>	105,6 ms
mungkin	cocok	 <p>06.23 06.23</p> <p>Kamus Indonesia - Spanyol</p> <p>mungkin</p> <p><input checked="" type="radio"/> Raita <input type="radio"/> Not So Naive</p> <p>CARI</p> <p>mungkin posible</p> <p>Running Time: 113.319 ms OK</p>	113,3 ms



nulis	cocok		104,7 ms
pergi	cocok		103,2 ms



mengeluh	cocok		82,9 ms
kehilangan	cocok		81 ms



memblokir	cocok		81,9 ms
pelayaran	cocok		75,8 ms


pemandangan	cocok		81,9 ms
kentang goreng	cocok		90,9 ms



Musim gugur	cocok		81,1 ms
Lebih sedikit	cocok		85,3 ms



Jendela toko	cocok	 <p>The screenshot shows the app interface with the input 'jendela toko' and the selected option 'Raita'. The output shows 'jendela toko' translated to 'escaparate'. The running time at the bottom is 77.904616 ms.</p>	77,9 ms
Bermain catur	cocok	 <p>The screenshot shows the app interface with the input 'bermain catur' and the selected option 'Raita'. The output shows 'bermain catur' translated to 'jugar a las damas'. The running time at the bottom is 78.533924 ms.</p>	78,5 ms

Di dalam	cocok		50,9 ms
Dokter gigi	cocok		61,0 ms

Terlalu banyak	cocok	 <p>The screenshot shows the app interface with the Indonesian word 'terlalu banyak' entered. The Spanish translation 'demasiado' is displayed below. The 'Raita' option is selected, and the 'CARI' button is visible. The running time at the bottom is 82.242847 ms.</p>	82,2 ms
Sama sekali	cocok	 <p>The screenshot shows the app interface with the Indonesian word 'sama sekali' entered. The Spanish translation 'completamente' is displayed below. The 'Raita' option is selected, and the 'CARI' button is visible. The running time at the bottom is 57.124846 ms.</p>	57,1 ms

Kasih sayang	cocok		35,5 ms
Menjadi bosan	cocok		28,5 ms

Selamat tinggal	cocok	 <p>The screenshot shows the app interface with the input 'selamat tinggal'. The selected option is 'Raita'. The results list shows 'adios' and 'despedir'. The running time at the bottom is 29.996923 ms.</p>	29,9 ms
mendemonstrasikan	cocok	 <p>The screenshot shows the app interface with the input 'mendemonstrasikan'. The selected option is 'Raita'. The results list shows 'demostrar'. The running time at the bottom is 28.836231 ms.</p>	28,8 ms



Pada layanan anda	cocok		29,2 ms
Untuk mengatakan selamat tinggal	cocok		35,4 ms


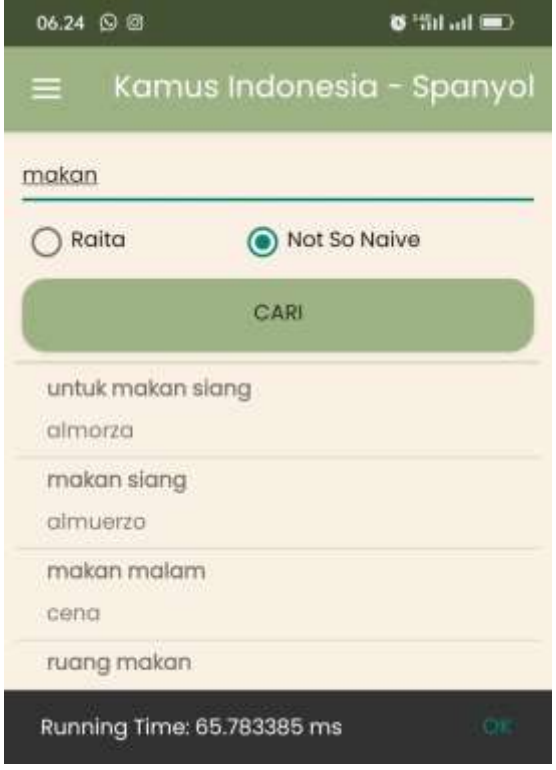
4.2.2 Pengujian Pencarian kata pada Kamus Bahasa Indonesia – Bahasa Spanyol dengan algoritma Not So Naive



Pada **Tabel 4.2** dapat dilihat pencarian kata pada Kamus Bahasa Indonesia – Bahasa Spanyol menggunakan Algoritma Not So Naive.



Tabel 4.2 Hasil Pencarian Kata Algoritma Not So Naive



Pola	Hasil Pencocokan	Tampilan Hasil Pencocokan	Running Time
ab	Cocok		62,8 ms
ada	Cocok		56,2 ms



es	Cocok	 <p>06.24 06.24 06.24</p> <p>Kamus Indonesia - Spanyol</p> <p>es</p> <p><input type="radio"/> Raita <input checked="" type="radio"/> Not So Naive</p> <p>CARI</p> <p>menyelesaikan acabar de agar sesuai caber kesopanan cortesia kesepuluh</p> <p>Running Time: 53.239692 ms</p>	53,2 ms
hujan	Cocok	 <p>06.24 06.24 06.24</p> <p>Kamus Indonesia - Spanyol</p> <p>hujan</p> <p><input type="radio"/> Raita <input checked="" type="radio"/> Not So Naive</p> <p>CARI</p> <p>jas hujan impermeable hujan lluvia</p> <p>Running Time: 60.716308 ms</p>	60,7 ms


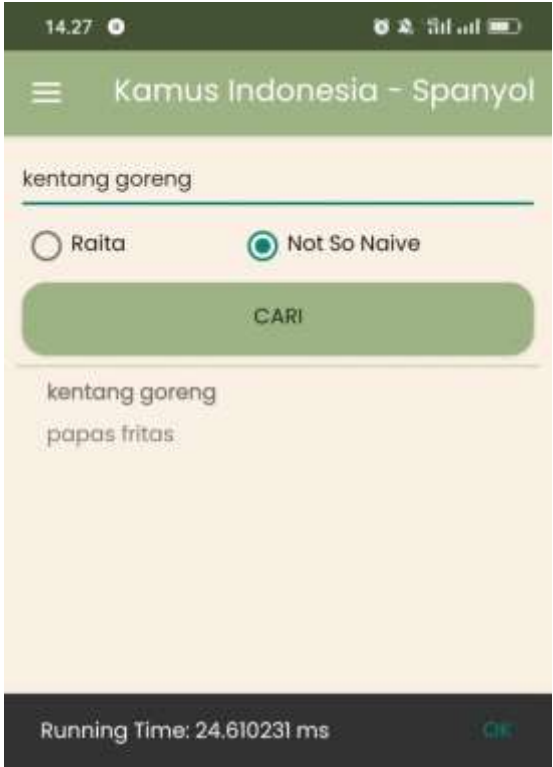
laku	Cocok	 <p>06.24 06.24 06.24</p> <p>Kamus Indonesia - Spanyol</p> <p>laku</p> <p><input type="radio"/> Raita <input checked="" type="radio"/> Not So Naive</p> <p>CARI</p> <p>perilaku comportamiento</p> <p>melakukan kesalahan equivocarse</p> <p>Running Time: 63.842077 ms OK</p>	63,8 ms
makan	Cocok	 <p>06.24 06.24 06.24</p> <p>Kamus Indonesia - Spanyol</p> <p>makan</p> <p><input type="radio"/> Raita <input checked="" type="radio"/> Not So Naive</p> <p>CARI</p> <p>untuk makan siang almorzo</p> <p>makan siang almuerzo</p> <p>makan malam cena</p> <p>ruang makan</p> <p>Running Time: 65.783385 ms OK</p>	65,7 ms

milik	Cocok	 <p>06.24 06.24 06.24</p> <p>Kamus Indonesia - Spanyol</p> <p>milik</p> <p><input type="radio"/> Raita <input checked="" type="radio"/> Not So Naive</p> <p>CARI</p> <p>milikku</p> <p>mio</p> <p>pemilik</p> <p>propietario</p> <p>memiliki</p> <p>tener</p> <p>milikmu</p> <p>Running Time: 61.772307 ms</p>	61,7 ms
mungkin	Cocok	 <p>06.24 06.24 06.24</p> <p>Kamus Indonesia - Spanyol</p> <p>mungkin</p> <p><input type="radio"/> Raita <input checked="" type="radio"/> Not So Naive</p> <p>CARI</p> <p>mungkin</p> <p>posible</p> <p>Running Time: 66.474846 ms</p>	66,4 ms



nulis	Cocok		60,9 ms
pergi	Cocok		66 ms



mengeluh	cocok		26,9 ms
kehilangan	cocok		25,4 ms



memblokir	cocok	 <p>14.26</p> <p>Kamus Indonesia - Spanyol</p> <p>memblokir</p> <p><input type="radio"/> Raita <input checked="" type="radio"/> Not So Naive</p> <p>CARI</p> <p>memblokir</p> <p>cuadra</p> <p>Running Time: 26.159615 ms</p>	26,1 ms
pelayaran	cocok	 <p>14.27</p> <p>Kamus Indonesia - Spanyol</p> <p>pelayaran</p> <p><input type="radio"/> Raita <input checked="" type="radio"/> Not So Naive</p> <p>CARI</p> <p>pelayaran</p> <p>crucero</p> <p>Running Time: 26.091077 ms</p>	26 ms

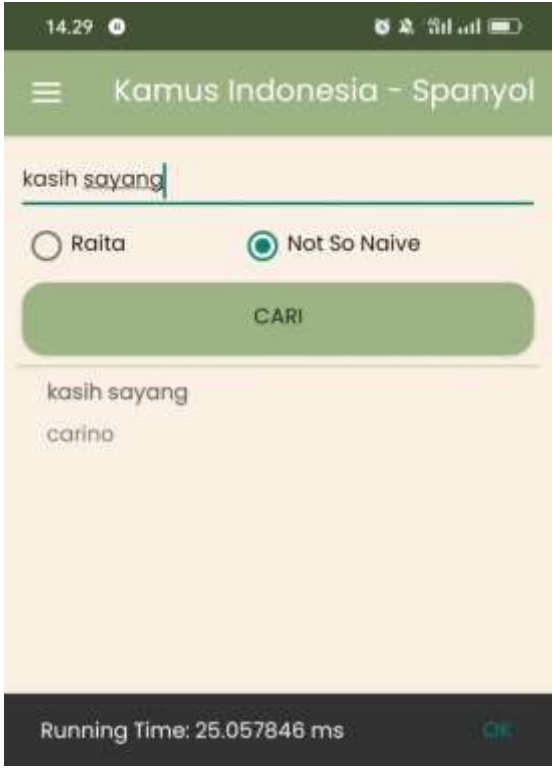

pemandangan	cocok		25,1 ms
Kentang goreng	cocok		24,6 ms



Musim gugur	cocok		24,3 ms
Lebih sedikit	cocok		24,9 ms



Jendela toko	cocok	 <p>The screenshot shows the app interface with the input 'jendela toko'. The selected option is 'Not So Naive', and the result shown is 'escaparate'. The running time at the bottom is 25.334077 ms.</p>	25,3 ms
Bermain catur	cocok	 <p>The screenshot shows the app interface with the input 'bermain catur'. The selected option is 'Not So Naive', and the result shown is 'jugar a las damas'. The running time at the bottom is 25.247923 ms.</p>	25,2 ms

Di dalam	cocok		37,1 ms
Dokter gigi	cocok		23,8 ms

Terlalu banyak	cocok		24,6 ms
Sama sekali	cocok		24,9 ms

Kasih sayang	cocok		25 ms
Menjadi bosan	cocok		24,8 ms

Selamat tinggal	cocok	 <p>The screenshot shows the app interface with the input 'selamat tinggal'. The selected option is 'Not So Naive', and the result is 'adios'. The running time is 24.908 ms.</p>	24,9 ms
mendemonstrasikan	cocok	 <p>The screenshot shows the app interface with the input 'mendemonstrasikan'. The selected option is 'Not So Naive', and the result is 'demostrar'. The running time is 25.104846 ms.</p>	25,1 ms

Pada layanan anda	cocok		24,9 ms
Untuk mengatakan selamat tinggal	cocok		26 ms

4.2.3 Hasil Pengujian

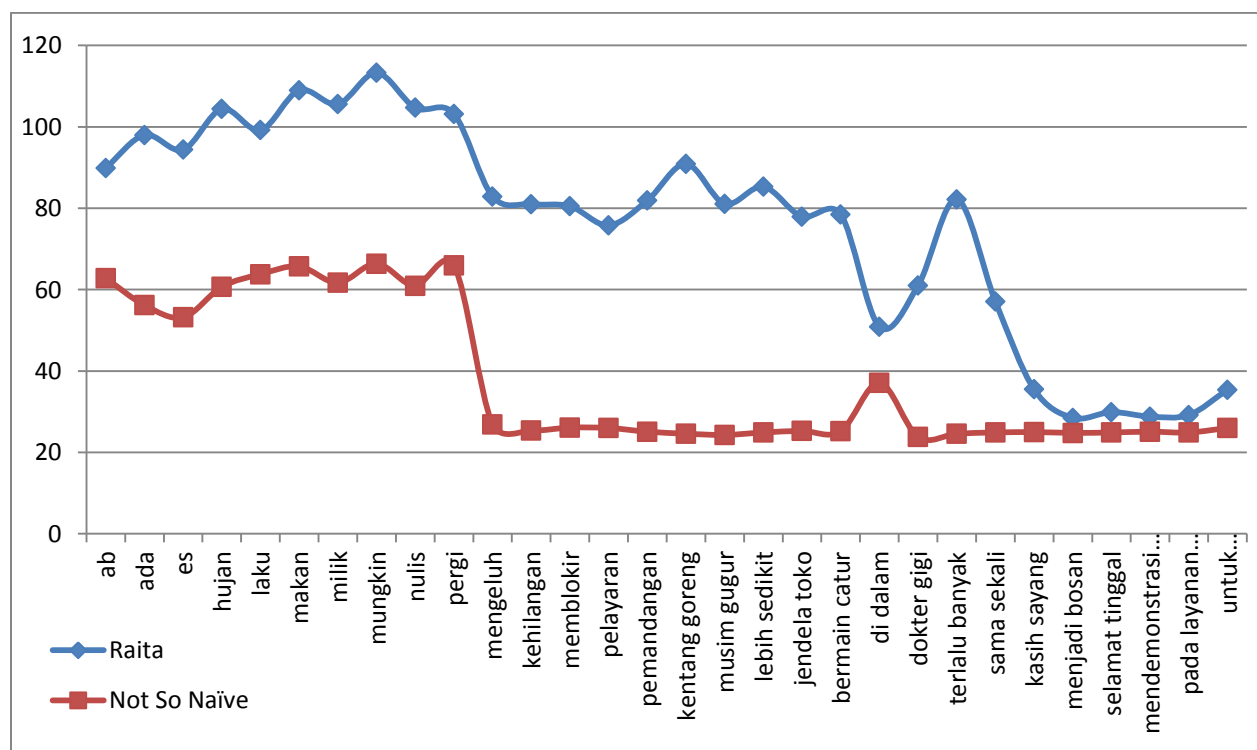
Hasil pengujian dari penelitian ini adalah *Running Time* dari pencarian kata dan jumlah kata yang ditemukan pada algoritma Raita dan algoritma *Not So Naïve* yang dilakukan terhadap *string* yang berbeda dengan 10 kali percobaan dan menghasilkan nilai rata-rata dari setiap *string* tersebut.

Tabel 4.3 Hasil Pengujian Algoritma Raita dan Algoritma Not So Naïve

No	String	Running Time Raita (ms)	Running Time Not So Naïve (ms)
1	ab	89,9	62,8
2	ada	98	56,2
3	es	94,4	53,2
4	hujan	104,4	60,7
5	laku	99,2	63,8
6	makan	109	65,7
7	milik	105,6	61,7
8	mungkin	113,3	66,4
9	nulis	104,7	60,9
10	pergi	103,2	66
11	mengeluh	82,9	26,9
12	kehilangan	81	25,4
13	memblokir	80,5	26,1
14	pelayaran	75,8	26
15	pemandangan	81,9	25,1
16	kentang goreng	90,9	24,6
17	musim gugur	81,1	24,3
18	lebih sedikit	85,3	24,9
19	jendela toko	77,9	25,3
20	bermain catur	78,5	25,2

21	di dalam	50,9	37,1
22	dokter gigi	61	23,8
23	terlalu banyak	82,2	24,6
24	sama sekali	57,1	24,9
25	kasih sayang	35,5	25
26	menjadi bosan	28,5	24,8
27	selamat tinggal	29,9	24,9
28	mendemonstrasikan	28,8	25,1
29	pada layanan anda	29,2	24,9
30	untuk mengucapkan selamat tinggal	35,4	26
Total		2276	1132,3
Rata-rata		75,8	37,7

Setelah mendapatkan hasil pengujian dari tabel di atas maka dibuat grafik perbandingan hasil pengujian dari algoritma tersebut. Grafik dapat dilihat pada **Gambar 4.6** berikut.



Gambar 4.6 Perbandingan Hasil *Running Time* Algoritma Raita dan Not So Naïve

Grafik pada gambar 4.6 menunjukkan perbandingan hasil *running time* algoritma Raita dan *Not So Naïve* dalam pencarian kata bahasa Indonesia – bahasa Spanyol. Dari hasil *running time* yang didapat pada proses pencarian kata dapat disimpulkan bahwa algoritma Raita mendapatkan hasil *running time* yang lebih tinggi dibandingkan dengan algoritma *Not So Naïve*. Hal ini menunjukkan bahwa algoritma *Not So Naïve* bekerja lebih cepat untuk proses pencocokan dan pencarian kata dibandingkan dengan algoritma Raita.

4.3 Kompleksitas Algoritma

Pada penelitian ini kompleksitas algoritma dilakukan dengan mencari *big theta*, *big O* dan *big omega* dari *pseudocode* yang ada. *Pseudocode* ini merupakan bagian dari program yang telah dirancang sesuai dengan algoritma Raita dan algoritma *Not So Naïve*. Hasil perhitungan dari kompleksitas dua algoritma tersebut dengan menggunakan notasi *big theta*, *big O* dan *big omega* disajikan tabel berikut :

4.3.1 Kompleksitas Algoritma Raita

Tabel 4.4 Analisis Kompleksitas pada Fase Preprocessing Algoritma Raita

Kode Program	C	#	C#
<code>int m = pattern.length();</code>	C1	1	C1
<code>// preprocessing phase</code>			
<code>int bmBc[] = new int[256];</code>	C1	1	C1
<code>for(int i = 0; i < 256; i++)</code>	C2	256	256C2
<code>bmBc[i] = m; // *</code>	C1	256	256C1
<code>for(int i = 0; i < m - 1; i++)</code>	C2	m-1	(m-1)C2
<code>bmBc[pattern.charAt(i)] = m - i - 1;</code>	C1	m-1	(m-1)C1

$$\begin{aligned}
 T(m) &= 258C1 + (m-1)C1 + 256C2 + (m-1)C2 \\
 &= 258C1 + 256C2 + (1C1 + 1C2) m \\
 &\text{Diasumsikan nilai C adalah konstan (diabaikan)} \\
 &= 514 + 2m \\
 &= \Theta(m)
 \end{aligned}$$

Tabel 4.5 Analisis Kompleksitas pada Fase Pencarian Algoritma Raita

Kode Program	C	#	C#
<code>// raita</code>			
<code>if(m == 0 m > text.length())</code>	C1	1	C1
<code>return false;</code>	C2	1	C2
<code>else if(m == 1) {</code>	C1	n	nC1
<code>if(text.indexOf(pattern) > 0)</code>	C1	1	C1
<code>return true;</code>	C2	n	nC2
<code>else</code>			
<code>return false;</code>	C2	1	C2
<code>}</code>			
<code>char firstChar = pattern.charAt(0);</code>	C3	1	C3
<code>char middleChar = pattern.charAt(m / 2);</code>	C3	1	C3
<code>char lastChar = pattern.charAt(m - 1);</code>	C3	1	C3
<code>// searching</code>			
<code>int j = 0;</code>	C4	1	C4
<code>while(j <= n - m) {</code>	C5	n	nC5
<code>char checkFirstChar = text.charAt(j);</code>	C3	1	C3
<code>char checkMiddleChar = text.charAt(j + m / 2);</code>	C3	1	C3
<code>char checkLastChar = text.charAt(j + m - 1);</code>	C3	1	C3

if(firstChar == checkFirstChar && middleChar == checkMiddleChar && lastChar == checkLastChar && pattern.equals(text.substring(j, j + m)))	C1	n*m	mnC1
return true;	C2	n*m	mnC2
j += bmBc[checkLastChar];	C2	n	nC2
}			
return false;	C2	1	C2
}			

1. *Big O* (O)

$$\begin{aligned}
 T(n) &= 2C1 + nC1 + 3C2 + 2C2 + 6C3 + C4 + nC5 + mnC1 + mnC2 \\
 &= 2C1 + 3C2 + 2C2 + 6C3 + 1C4 + (1C1 + 1C5)n + (1C1 + 1C2)mn \\
 &\text{Diasumsikan nilai C adalah konstan (diabaikan)} \\
 &= 14 + 2n + 2mn \\
 &= O(mn)
 \end{aligned}$$

2. *Big theta* (Θ)

$$\begin{aligned}
 T(n) &= 2C1 + nC1 + 3C2 + 2C2 + 6C3 + C4 + nC5 + mnC1 + mnC2 \\
 &= 2C1 + 3C2 + 2C2 + 6C3 + 1C4 + (1C1 + 1C5)n + (1C1 + 1C2)mn \\
 &\text{Diasumsikan nilai C adalah konstan (diabaikan)} \\
 &= 14 + 2n + 2mn \\
 &= \Theta(mn)
 \end{aligned}$$

3. *Big Omega* (Ω)

Pada kode program yang di sajikan dalam tabel 4.5 tidak ada perulangan dua kali, semuanya hanya satu kali perulangan. Maka, hasil yang di dapatkan adalah $\Omega(1)$.

4.3.2 Kompleksitas Algoritma Not So Naïve

Tabel 4.6 Kompleksitas Hasil Algoritma Not So Naïve

Kode Program	C	#	C#
<code>pattern = pattern.toLowerCase(Locale.<i>ROOT</i>);</code>	C1	1	C1
<code>text = text.toLowerCase(Locale.<i>ROOT</i>);</code>	C2	1	C2
<code>int n = text.length();</code>	C3	1	C3
<code>int m = pattern.length();</code>	C3	1	C3
<code>if(m == 0 m > text.length())</code>	C4	1	C4
<code> return false;</code>	C5	1	C5
<code>else if(m == 1) {</code>	C4	1	C4
<code> if(text.indexOf(pattern) > 0)</code>	C4	1	C4
<code> return true;</code>	C5	1	C5
<code> else</code>	C6	1	C6
<code> return false;</code>	C5	1	C5
<code>}</code>			
<code>int j, k, ell;</code>	C3	1	C3
<code>// preprocessing phase</code>			
<code>if(pattern.charAt(0) == pattern.charAt(1)) {</code>	C4	1	C4
<code> k = 2;</code>	C7	1	C7
<code> ell = 1;</code>	C8	1	C8
<code>} else {</code>	C6	1	C6
<code> k = 1;</code>	C7	1	C7
<code> ell = 2;}</code>	C8	1	C8
<code>// searching</code>			
<code>j = 0;</code>	C9	1	C9
<code>while(j <= n - m) {</code>	C10	n-m	C10(n-m)
<code> if(pattern.charAt(1) != text.charAt(j + 1))</code>	C4	n-m	C4(n-m)
<code> j += k;</code>	C9	n-m	C9(n-m)
<code> else {</code>	C6	n-m	C6(n-m)
<code> if(pattern.substring(2, 2+m-2).equals(text.substring(j + 2, j + 2 + m - 2)) && pattern.charAt(0) == text.charAt(j))</code>	C4	n-m	C4(n-m)
<code> return true;</code>	C5	n-m	C5(n-m)
<code> j += ell;</code>	C9	n-m	C9(n-m)
<code> }</code>			
<code>}</code>			
<code>return false;</code>	C5	1	C5
<code>}</code>			
<code>}</code>			

Keterangan :

Kolom C : Variabel untuk menghitung seberapa banyak processor melakukan komputasi.

Kolom # : Sebagai variabel untuk menghitung berapa kali komputasi dikerjakan.

Kolom C# : Menunjukkan hasil perkalian dari C dan #.

Dari perhitungan *running time* pada tabel 4.6 kemudian didapat kompleksitas algoritma Not So Naïve :

1. Big O (O)

$$\begin{aligned}
 T(n) &= C1 + C2 + 3C3 + 4C4 + C4(n-m) + C4(n-m) + 4C5 + C5(n-m) + 2C6 + C6(n-m) + \\
 &\quad 2C7 + 2C8 + C9 + C9(n-m) + C9(n-m) + C10(n-m) \\
 &= (C1 + C2 + 3C3 + 4C4 + 4C5 + 2C6 + 2C7 + 2C8 + C9) + (2C4(n-m) + C5(n-m) + \\
 &\quad C6(n-m) + 2C9(n-m) + C10(n-m)) \\
 &\text{Diasumsikan nilai C konstan (nilai C diabaikan)} \\
 &= n - m \\
 &= O(n - m)
 \end{aligned}$$

2. Big Theta (Θ)

$$\begin{aligned}
 T(n) &= C1 + C2 + 3C3 + 4C4 + C4(n-m) + C4(n-m) + 4C5 + C5(n-m) + 2C6 + C6(n-m) + \\
 &\quad 2C7 + 2C8 + C9 + C9(n-m) + C9(n-m) + C10(n-m) \\
 &= (C1 + C2 + 3C3 + 4C4 + 4C5 + 2C6 + 2C7 + 2C8 + C9) + (2C4(n-m) + C5(n-m) + \\
 &\quad C6(n-m) + 2C9(n-m) + C10(n-m)) \\
 &\text{Diasumsikan nilai C konstan (nilai C diabaikan)} \\
 &= n - m \\
 &= \Theta(n - m)
 \end{aligned}$$

3. Big Omega (Ω)

Pada kode program yang di sajikan dalam tabel 4.6 tidak ada perulangan dua kali, semuanya hanya satu kali perulangan. Maka, hasil yang di dapatkan adalah $\Omega(1)$

Tabel 4.4 , **tabel 4.5** dan **tabel 4.6** adalah tabel kompleksitas algoritma *Raita* dan algoritma *Not So Naïve*, dimana proses pencarian kompleksitasnya menggunakan C sebagai konstanta, $\#$ sebagai ukuran masukan, dan $C\#$ (C kali $\#$) adalah untuk mencari *Theoretical Running Time* ($T(n)$) atau kompleksitas waktu, sehingga dapat dijumlahkan hasil dari perkalian C kali $\#$.

Berdasarkan pengujian yang dilakukan, ditemukan bahwa rata-rata hasil *running time* dan kompleksitas algoritma *Raita* sebesar 10,35 ms dan $\Theta(mn)$, $O(mn)$, $\Omega(1)$. Sedangkan hasil *running time* dan kompleksitas algoritma *Not So Naïve* sebesar 7,48 ms dan $\Theta(n-m)$, $O(m-n)$, $\Omega(1)$. Hasil pengujian *running time* dan kompleksitas algoritma *Raita* yang didapatkan lebih besar dibandingkan dengan algoritma *Not So Naïve*. Jadi dapat disimpulkan bahwa *running time* dan kompleksitas algoritma *Not So Naïve* lebih cepat daripada algoritma *Raita*.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan analisis, perancangan, dan pengujian dari penelitian, maka didapat beberapa kesimpulan sebagai berikut :

1. Algoritma *Raita* dan Algoritma *Not So Naïve* dapat diimplementasikan pada pembuatan aplikasi kamus bahasa Indonesia – bahasa Spanyol.
2. Algoritma *Not So Naïve* melakukan pencocokan kata lebih cepat apabila dibandingkan dengan algoritma *Raita*. Dalam pengujian 10 pola, rata-rata *running time* algoritma *Not So Naïve* adalah 37,7 ms sedangkan algoritma *Raita* adalah 75,8 ms.
3. Untuk nilai *running time* sangat dipengaruhi oleh jumlah huruf. Bila semakin banyak jumlah data satu huruf dalam database maka *running time* dalam proses pencarian kata lebih besar, sedangkan jika satu huruf jarang ditemukan dalam database maka *running time* dalam proses pencarian lebih kecil.
4. Hasil kompleksitas algoritma *Raita* adalah $\Theta(mn)$, $O(mn)$, $\Omega(1)$ sedangkan hasil kompleksitas algoritma *Not So Naïve* adalah $\Theta(n-m)$, $O(n-m)$, $\Omega(1)$.

5.2 Saran

Adapun saran-saran yang diperlukan untuk penelitian maupun pengembangan berikutnya adalah:

1. Pada penelitian ini hanya menggunakan dua algoritma *string matching*, disarankan agar mengimplementasikan algoritma-algoritma *string matching* lainnya.
2. Pada penelitian ini tidak ada menu pengolahan data, diharapkan aplikasi kamus bahasa Indonesia – bahasa Spanyol ditambah menu pengolahan data seperti penambahan data, edit data dan hapus data agar data dalam kamus semakin banyak untuk pencarian kata.
3. Penulis hanya menyediakan 1000 kata dalam penelitian ini, diharapkan kedepannya ditambah lebih banyak lagi kata untuk data kamus.
4. Pada penelitian ini belum diterapkan *voice recognition*, disarankan untuk pengembangan kedepannya menerapkan *voice recognition* sehingga sistem mengenal inputan suara *user*.

DAFTAR PUSTAKA

- Alapati, K. P., & Mannava. (2011). An GUI Based Implementation of Pattern Matching Algorithms. *International Journal Of System And Technologies*, 4(2):207-215.
- Cantone, D., & Faro, S. (2004). Searching for a substring with constant extra-space complexity. *Proc. of Third International Conference on Fun with Algorithms*, 118–131.
- Chen, C. H. (2016). Handbook Of Pattern Recognition And Computer Vision: 5th Edition. In *Handbook of Pattern Recognition and Computer Vision (5th Edition)*. World Scientific Publishing Company, Incorporated.
- Faathir, M. W. (2018). *Perbandingan Algoritma Hosrpool Dan Not So Naive Dalam Pembuatan Kamus Bahasa Indonesia – Bahasa Aceh Berbasis Android*. Universitas Sumatera Utara.
- Fajar, R. (2018). Implementasi Dan Perbandingan Algoritma Reverse Colussi Dan Algoritma Raita Pada Aplikasi Pencarian Lirik Lagu Bahasa Indonesia Dengan Speech Recognition Berbasis Android. In *Skripsi*. Universitas Sumatera Utara.
- Haviluddin. (2011). Memahami Penggunaan UML (Unified Modelling Language). *Memahami Penggunaan UML (Unified Modelling Language)*, 6(1), 1–15.
<https://informatikamulawarman.files.wordpress.com/2011/10/01-jurnal-informatika-mulawarman-feb-2011.pdf>
- Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). *Introduction to Automata Theory, Languages, and Computation*.
- Klaib, A. ., & Osborne, H. (2009). RSMA Matching Algorithm for Searching Biological Sequences. *Journal International Conference on Future Computer and Communication*, 195–199.
- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms (Art of Computer Programming)*.
- Knuth, D. E., Morris (Jr), J. H., & Pratt, V. R. (1977). Fastest pattern matching in strings. *Journal of Algorithms*, Vol.6, 323–350.
- Lecroq, T. (2014). *Handbook of Exact String-Matching Algorithms Christian Charras* (Issue May).
- Mitani, Y., Ino, F., & Hagihara, K. (2016). Parallelizing exact and approximate string matching via inclusive scan on a GPU. *IEEE Transactions on Parallel and Distributed Systems*.

- Paloma Cascales, M. (2019). *Bahasa Spanyol Bahasa Sahabatku*. Badan Pengembangan Bahasa dan Perbukuan Kementerian Pendidikan dan Kebudayaan Republik Indonesia.
- Setiady, H., & Yulistia. (2016). Sistem Informasi Pemesanan Dan Penjualan Berbasis Web Pada Dewi Florist. *Sistem Informasi*, 1–7. <http://eprints.mdp.ac.id/829/>
- Singh, R., & Verma, H. . (2011). A fast string matching algorithm. *International Journal of Computer Technology and Applications*, Vol.2, No. 6: 1877-1883.
- Singla, N., & Garg, D. (2012). *String Matching Algorithms and their Applicability in various Applications*. 6, 218–222.
- Situmorang, E. C. (2019). *Perbandingan Algoritma Not So Naive Dan Algoritma Zhu-Takaoka Pada Aplikasi Kamus Farmakologi Berbasis Android*. Universitas Sumatera Utara.
- Suwitri. (2017). Perbandingan Algoritma Raita Dan Algoritma Quick Search Pada Aplikasi Kamus Bahasa Indonesia – Bahasa Prancis. In *Skripsi*. Universitas Sumatera Utara.
- Whitten, J. ., Bentley, L. ., & Dittman, K. . (2004). Metode Desain & Analisis Sistem. *Terjemahan. TIM Penerjemah ANDI. ANDI : Yogyakarta*.

LISTING PROGRAM

```

public static boolean raita(String pattern, String text) {
    pattern = pattern.toLowerCase(Locale.ROOT);
    text = text.toLowerCase(Locale.ROOT);
    int n = text.length();
    int m = pattern.length();
    // preprocessing phase
    int bmBc[] = new int[256];
    for(int i = 0; i < 256; i++)
        bmBc[i] = m; // *

    for(int i = 0; i < m - 1; i++)
        bmBc[pattern.charAt(i)] = m - i - 1;

    // raita
    if(m == 0 || m > text.length())
        return false;
    else if(m == 1) {
        if(text.indexOf(pattern) > 0)
            return true;
        else
            return false;
    }

    char firstChar = pattern.charAt(0);
    char middleChar = pattern.charAt(m / 2);
    char lastChar = pattern.charAt(m - 1);

    // searching
    int j = 0;
    while(j <= n - m) {
        char checkFirstChar = text.charAt(j);
        char checkMiddleChar = text.charAt(j + m / 2);
        char checkLastChar = text.charAt(j + m - 1);
        if(firstChar == checkFirstChar &&
           middleChar == checkMiddleChar &&
           lastChar == checkLastChar &&
           pattern.equals(text.substring(j, j + m)))
            return true;

        j += bmBc[checkLastChar];
    }
    return false;
}

public static boolean not_so_naive(String pattern, String text) {
    pattern = pattern.toLowerCase(Locale.ROOT);
    text = text.toLowerCase(Locale.ROOT);
    int n = text.length();
    int m = pattern.length();
    if(m == 0 || m > text.length())
        return false;
    else if(m == 1) {

```

```

        if(text.indexOf(pattern) > 0)
            return true;
        else
            return false;
    }

    int j, k, ell;
    // preprocessing phase
    if(pattern.charAt(0) == pattern.charAt(1)) {
        k = 2;
        ell = 1;
    } else {
        k = 1;
        ell = 2;
    }

    // searching
    j = 0;
    while(j <= n - m) {
        if(pattern.charAt(1) != text.charAt(j + 1))
            j += k;
        else {
            if(pattern.substring(2, 2 + m - 2).equals(text.substring(j +
2, j + 2 + m - 2)) && pattern.charAt(0) == text.charAt(j))
                return true;
            j += ell;
        }
    }
    return false;
}
}

```