Optimalisasi Penyimpanan dan Pencarian Kata pada Kamus Bahasa Indonesia Menggunakan Fungsi Hash Berbasis Teori Bilangan

Alfian Hanif Fitria Yustanto - 13523073[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]alfian.yustanto@gmail.com, 13523073@std.stei.itb.ac.id

***Abstract*— Kamus Bahasa Indonesia memiliki ribuan kata yang perlu dikelola dengan baik, terutama dalam era digital saat ini, di mana kata-kata tersebut biasanya disimpan dalam dataset. Tantangan utama yang muncul adalah efisiensi dalam penyimpanan dan pencarian data, terutama jika data tidak disusun secara optimal. Salah satu solusi efektif adalah menggunakan metode hash, yang memanfaatkan fungsi hash berbasis teori bilangan, seperti aritmetika modular dan bilangan prima. Konsep teori bilangan ini membantu mendistribusikan data secara merata ke dalam dataset, meminimalkan tabrakan, dan meningkatkan efisiensi pencarian. Dengan teknik chaining sebagai penanganan tabrakan, metode ini mampu mengoptimalkan memori sekaligus mempercepat waktu pencarian data.**

***Keywords*—bilangan prima, fungsi hash, optimalisasi, kamus bahasa indonesia**

I. PENDAHULUAN

Kamus Bahasa Indonesia merupakan salah satu sumber referensi utama yang menyimpan ribuan kata dalam bahasa Indonesia beserta maknanya. Dalam bentuk buku, pencarian kata di dalam kamus memerlukan waktu yang cukup lama, terutama jika dilakukan secara manual. Proses ini dimulai dengan mencari huruf pertama dari kata yang dicari, kemudian melakukan pengecekan terhadap setiap kata pada subbagian huruf tersebut hingga kata yang dicari ditemukan pada kamus. Proses seperti ini tidak efisien, terutama jika jumlah kata yang harus dicari semakin besar.

Dalam era teknologi digital, kata-kata dalam kamus Bahasa Indonesia kini dapat disimpan dalam dataset atau sistem penyimpanan lainnya. Penyimpanan dalam dataset memungkinkan akses yang lebih cepat dan terorganisir. Salah satu cara sederhana yang dapat diterapkan adalah dengan menyimpan seluruh kata secara terurut berdasarkan alfabet, seperti dalam kamus cetak. Pencarian dilakukan dengan iterasi atau membandingkan kata yang dicari satu per satu dalam dataset tersebut. Meskipun lebih terstruktur, cara ini tetap memerlukan waktu yang cukup besar, terutama jika data dalam dataset sangat banyak.

Untuk mengatasi masalah tersebut, optimalisasi dalam pencarian kata dapat menjadi solusi. Salah satu solusi efektif yang dapat diterapkan adalah memanfaatkan fungsi hash dalam penyimpanan dan pencarian kata. Fungsi hash memungkinkan setiap kata dikonversi menjadi indeks numerik yang unik, yang kemudian digunakan untuk menyimpan data di tabel hash. Metode ini tidak hanya mengurangi jumlah iterasi dalam proses pencarian, tetapi juga mempercepat akses data secara signifikan dibandingkan metode konvensional. Dengan memanfaatkan teori bilangan seperti aritmetika modular, fungsi hash mampu mendistribusikan data secara merata, sehingga mengurangi kemungkinan terjadinya tabrakan data (*collision*) dan meningkatkan efisiensi secara keseluruhan.

Melalui pengimplementasian fungsi hash, pencarian data pada kamus Bahasa Indonesia dapat dilakukan dengan waktu yang mendekati konstan, menjadikannya metode yang lebih unggul dibandingkan metode konvensional. Optimisasi ini menjadi relevan dalam konteks pengelolaan data besar, di mana efisiensi dan kecepatan menjadi kunci utama keberhasilan sistem.

II. TEORI DASAR

A. Teori Bilangan

Teori bilangan merupakan salah satu cabang matematika yang dikhusukan untuk mempelajari bilangan bulat (*integer*) atau fungsi-fungsi yang memiliki bilangan bulat. Bilangan bulat merupakan bilangan yang tidak memiliki pecahan desimal. Bilangan riil merupakan bilangan yang berbanding terbalik dengan bilangan bulat yaitu seluruh bilangan baik yang memiliki pecahan desimal maupun tidak dan baik bilangan positif maupun negatif. [1]

$$Bilangan\ Bulat\ =\ 1, 2, 30, \ldots$$
$$Bilangan\ riil\ =\ 1, 2.123\ , -0.123123, \ldots$$

Bilangan bulat memiliki beberapa sifat salah satunya adalah sifat pembagian. Pada bilangan bulat suatu bilangan bulat a dikatakan habis membagi b jika terdapat suat bilangan bulat c sehingga $ac\ =\ b$ atau dapat ditulis $a \mid b$. Contoh,

$$20 / 4\ =\ 5$$

sehinnga,

$$4 \mid 20$$

Teorema euclidean menyatakan bahwa jika terdapat suatu bilangan bulat m dan n , $n\ >\ 0$. Maka hasil dari m dibagi n

adalah q (*quotient*) dan sisanya r (*remainder*) sehingga

$$m = nq + r$$

dengan $0 \leq r < n$

Operator modulo merupakan suatu operator dalam teori bilangan yang akan memberikan sisa pembagian $r\,(remainder)$. Misalkan terdapat dua bilangan a dan m , maka

$$a \bmod m = r$$

sedemikian sehingga,

$$a = mq + r$$

Variabel m biasa disebut modulus atau modulo dan hasil aritmatika modulo bernilai diantara $0 \leq r \leq m - 1$. Salah satu aplikasi dari konsep teori bilangan adalah fungsi hash. [1]

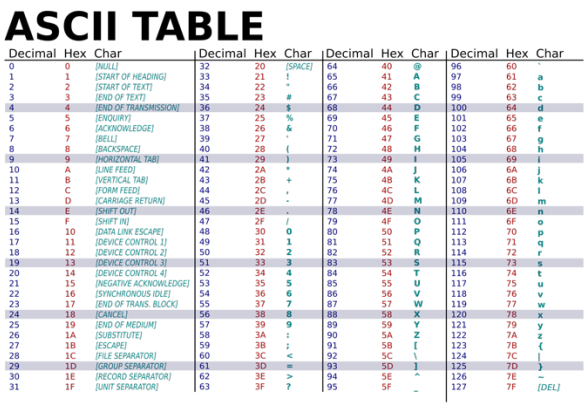
B. Fungsi Hash

Fungsi hash merupakan salah satu dari aplikasi teori bilangan. Fungsi ini berfungsi untuk mengatur pengalamatan memori untuk tujuan pengaksesan yang lebih cepat. Fungsi hash memiliki bentuk,

$$h(K) = K \bmod m$$

$m = jumlah\ lokasi\ memori\ yang\ tersedia$
$K = Kunci\ integer$
$h(K) = lokasi\ memori\ untuk\ data\ dengan\ kunci\ K$

Untuk kepentingan optimalisasi penyimpanan dan pencarian kata dalam Kamus Besar Bahasa Indonesia, fungsi hash digunakan untuk menyimpan dan mengakses kata di dalam sebuah array. Agar setiap kata dapat direpresentasikan sebagai numerik, maka setiap karakter pada kata akan di konversikan menjadi kode ASCII dan dijumlahkan.

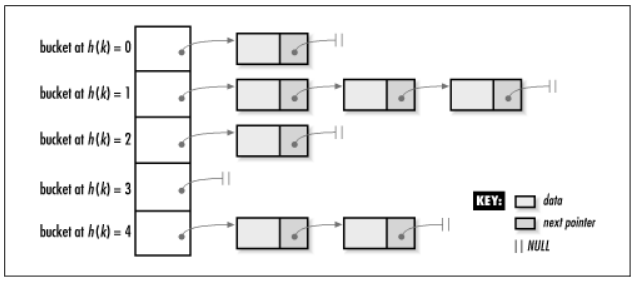


Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Gambar 2.2.1 Tabel ASCII
Sumber :
https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/1200px-ASCII-Table-wide.svg.png

Setelah kata direpresentasikan dalam bilangan numerik , hasil konversi tersebut akan dilakukan operasi mod untuk mendapatkan index lokasi pada array. Dalam proses pembuatan key , dapat terbentuk key yang sama untuk dua kata yang berbeda. Kejadian ini biasa disebut *collision*.

Untuk menangani *collision*, dapat dilakukan dengan beberapa metode salah satunya adalah *hash chaining*. Metode ini memungkinkan beberapa kata yang memiliki key sama tetap dapat disimpan pada dataset. Metode ini bekerja dengan cara menyediakan suatu list baru pada index dengan key sama yang berfungsi sebagai tempat penyimpanan kata.

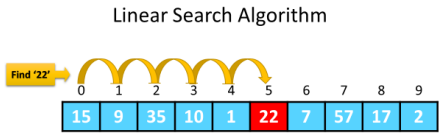


Gambar 2.2.2 Visualisasi hash chaining
Sumber :
https://www.oreilly.com/api/v2/epubs/1565924533/files/httpatomoreillycomsourceoreillyimages1778578.png

C. Pencarian Kata

Setelah melakukan menyimpan kata di dalam dataset, diperlukan suatu metode untuk mencari kata pada elemen-elemen dataset tersebut. Metode yang dapat digunakan adalah *search algorithm.* Salah satu *search algorithm* yang paling sederhana adalah *linear search.*

Linear search merupakan metode pencarian paling sederhana. Proses pencarian dilakukan dengan mengunjungi setiap elemen pada dataset secara sekuensial dan melakukan perbandingan dengan elemen yang dicari dimulai dari elemen pertama hingga elemen terakhir. [2]



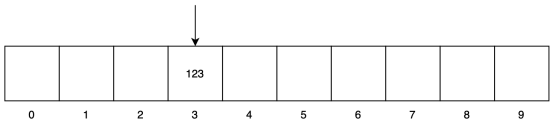
Gambar 2.3.1 Ilustrasi *Linear search*
Sumber :
https://miro.medium.com/v2/resize:fit:800/1*eTQoIHGdG58sy-iMwcp97w.png

Pada kasus optimalisasi penyimpanan menggunakan fungsi hash, metode pencarian kata tidak sepenuhnya menggunakan *linear search.* Proses pencarian kata dilakukan dengan membangun *hash key* suatu elemen yang dicari, kemudian

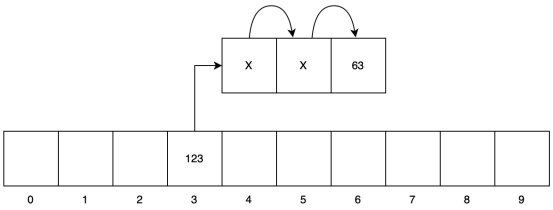
mencari lokasi index yang bersesuaian dengan *hash key* yang telah dibuat. *Linear search* digunakan untuk mencari elemen yang memiliki nilah *hash key* serupa dengan elemen lain.

Misalkan terdapat sebuah array dengan kapasitas 10 dan sebuah elemen dengan nilai 123. Saat proses penyimpanan akan elemen tersebut akan dilakukan porses modulo untuk menentukan *hash key* yang akan menjadi lokasi index pada array. Untuk mencari apakan elemen dengan nilai 123 terdapat pada array dilakukan cara serupa seperti saat penyimpanan, yaitu membuat *hash key* dan mencari keberadaan elemen pada array dengan index *hash key*.

$$123 \; mod \; 10 \; = \; 3$$
$$63 \; mod \; 10 \; = \; 3$$



Gambar 2.3.2 Penyimpanan dan pencarian elemen bernilai 123 berdasarkan *hash key*
Sumber : Dokumen penulis



Gambar 2.3.3 Penyimpanan dan pencarian elemen bernilai 63 berdasarkan *hash key*
Sumber : Dokumen penulis

D. Kompleksitas

Sebuah algoritma yang baik tidak hanya dilihat dari kebenaran algoritma tersebut dalam menyelesaikan suatu persoaalan. Algoritma yang baik juga harus efisien. Efisiensi algoritma dapat diukur melalui dua hal yaitu waktu (*time*) yang diperlukan untuk menyelesaikan masalah dan ruan (*space*) yang diperlukan untuk menjalankan algoritma tersebut. Waktu dan ruang yang diperlukan suatu algoritma bergantung pada ukuran masukan (n) , yang menyatakan ukuran data yang akan diproses.[3]

Besaran yang dipakai untuk menerangkan model abstrak pengukuran waktu atau ruang disebut kompleksitas algoritma. Terdapat dua jenis kompleksitas algoritma yaitu kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu suatu algoritma dihitung melalui jumlah tahapan komputasi yang dilakukan algoritma tersebut. Kompleksitas ruang diukur melalui besaran memori yang digunakan untuk struktur data pada algoritma tersebut.[3]

Kompleksitas waktu , T(n) , dibagi menjadi tiga macam yaitu kompleksitas waktu untuk waktu terburuk, kompleksitas waktu untuk kasus terbaik , dan kompleksitas waktu untuk kasus rata-rata. Untuk *linear search,* kasus terbaik terjadi ketika elemen yang dicari berada pada index pertama sehingga hanya terjadi satu kali proses perbandingan. Kasus terburuk *lienar search* terjadi ketika elemen yang dicari tidak dapat ditemukan pada array sehingga proses perbandingan dilakukan untuk setiap elemen pada array berukuran n. Kasus rata-rata terjadi ketika elemen-x ditemukan pada index-j , maka proses perbandingan akan dilakukan sebanyak j kali. Jika mengimplematasikan fungsi hash pada penyimpanan dan pencarian elemen array, maka jumlah perbandingan rata-rata yang diperlukan akan berkurang. Hal ini dikarenakan fungsi hash tidak perlu melakukan pengecekan elemen array dari elemen pertama hingga elemen terakhir.[3]

III. IMPLEMENTASI

Program ini dibuat menggunakan bahasa Python, yang merupakan salah satu bahasa pemrograman yang umum digunakan dalam pengolahan data secara matematis. Python dipilih karena dianggap sesuai untuk mengimplementasikan fungsi hash untuk optimalisasi penyimpanan dan pencarian kata dalam bahasa indonesia. Program ini terdiri dari dua *file* utama, yaitu main.py dan data.txt.

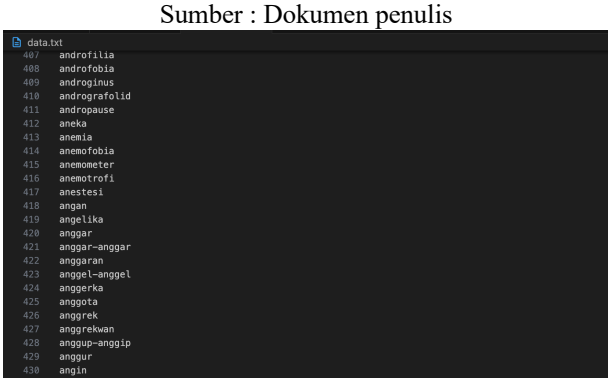
File main.py berisi kumpulan fungsi dan prosedur yang diperlukan untuk melakukan penyimpanan dan pencarian kata *file* ini juga berisi simulasi program koreksi kata dalam bentuk *command line interface* (CLI). *File* data.txt berisi kumpulan kata dalam Kamus Besar Bahasa Indonesia yang akan disimpan.

Pada implementasi ini akan dilakukan dua proses penyimpanan dan pencarian. Metode pertama penyimpanan dan pencarian dengan hash. Metode ke dua penyimpanan dan pencarian secara sekuensial (*linear search*). Implementasi menggunakan dua metode untuk menunjukkan perbandingan antara proses hasd dan sekuensial.

Proses penyimpanan kata dilakukan secara terstruktur. Langkah pertama adalah membaca data dari *file* data.txt. Pembacaan dilakukan dengan sederhana, yaitu membaca setiap baris dalam *file* data.txt dan menyimpannya ke dalam sebuah array. Pada tahap ini kata disimpan secara sekuensial tanpa menggunakan fungsi hash.

```python
def readFile(file_path):
    try:
        with open(file_path, 'r') as file:
            words = file.read().splitlines()
        return words
    except FileNotFoundError:
        print("\033[1;31mFile tidak ditemukan.\033[0m")
        return []
```

Gambar 3.1 Fungsi untuk menyimpan kata pada
Sumber : Dokumen penulis



Gambar 3.2 Contoh sebagian kata pada *file* data.txt
Sumber : Dokumen penulis

Setelah dataset berupa array kata terbentu, setiap kata akan diproses untuk didapatkan *hash key.* Fungsi hash() akan mengolah kata dengan cara mengubah setiap karakter menjadi kode numerik ASCII dengan fungsi ord() serta melakukan sumasi untuk setiap karakter pada suatu kata. Kemudian nilai dari proses tersebut dilakukan operasi modulo terhadap besar dataset penyimpanan.

```
def hash(word, datasetSize):
    ascii_sum = sum(ord(char) for char in word)
    return ascii_sum % datasetSize
```

Gambar 3.3 Fungsi untuk mendapatkan *hash key*
Sumber : Dokumen penulis

Dalam pembentukan *hash key* sangat mungkin terbentuk beberapa *key* yang bernilai sama untuk kata yang berbeda. Sebagai contoh, kata "kasur" dan "rusak" memiliki nilai ord() yang sama yaitu 550. Hal tersebut akan menyebabkan *collission* saat program berusaha untuk menyimpan ke dalam dataset. Untuk mengatasi hal ini, digunakan metode *hash chaining* yang akan membuat array baru pada index yang akan diisi kata.

```
def addHashChaining(hashDataset, word, datasetSize):
    index = hash(word, datasetSize)

    if hashDataset[index] == []:
        hashDataset[index] = [word]
    else:
        hashDataset[index].append(word)
```

Gambar 3.4 Fungsi untuk membuat *hash chaining*
Sumber : Dokumen penulis

Fungsi addHashChaining() menerima masukan berupa hashDataset yaitu sebuah variabel untuk menyimpan dataset dengan metode hash, word yang merupakan kata yang akan dimasukan kedalam hashDataset, dan datasetSize yaitu besar ukuran hashDataset. Setelah fungsi selesai dibuat, diperlukan fungsi untuk melakukan proses terhadap seluruh dataset. Fungsi gabungan ini dibuat dengan nama createHashDataset().

```
def createHashDataset(words, datasetSize):
    hashDataset = [[] for _ in range(datasetSize)]
    collisions = 0

    for word in words:
        index = hash(word, datasetSize)
        if hashDataset[index]:
            collisions += 1
        addHashChaining(hashDataset, word, datasetSize)

    return hashDataset, collisions
```

Gambar 3.5 Fungsi untuk membuat hash dataset
Sumber : Dokumen penulis

Fungsi createHashDataset akan melakukan proses penyimpanan kata dengan metode *hash chaining.* Fungsi ini akan mengembalikan dua variabel yaitu hashDataset, sebuah dataset berisi kata yang terletak pada posisi *hash key,* dan collisions yaitu jumlah *collisions* yang terjadi saat pembentukan hashDataset. Setelah penyimpanan selesai dilakukan, diperlukan fungsi untuk melakukan pencarian kata. Pada implementasi ini akan dilakukan dua jenis pencarian yaitu pencarian dengan metode hash pada hashDataset dan pencarian secara sekuensial pada dataset biasa.

```
def searchHash(hashDataset, target,datasetSize):
    index = hash(target, datasetSize)
    comparisons = 0

    if hashDataset[index] != []:
        for word in hashDataset[index]:
            comparisons += 1
            if word == target:
                return True, comparisons
    return False, comparisons
```

Gambar 3.6 Fungsi *search* menggunakan *hash key*
Sumber : Dokumen penulis

Fungsi searchHash melakukan menerima tiga parameter masukkan yaitu hashDataset, target, dan datasetSize. Fungsi ini akan mencari target pada index di hashDataset sesuai *hash key* yang didapat dengan fungsi hash. Kemudian fungsi akan mencari secara traversal untuk setiap kata pada *chain* yang terbentuk. Fungsi mengembalikan boolean dan jumlah perbandingan yang terjadi selama fungsi dijalankan.

```
def linearSearch(words, target):
    comparisons = 0
    for word in words:
        comparisons += 1
        if word == target:
            return True, comparisons
    return False, comparisons
```

Gambar 3.7 Fungsi *search* menggunakan *hash key*
Sumber : Dokumen penulis

Fungsi linearSearch melakukan pencarian kata secara sekuensial pada dataset. Fungsi ini menerima dua parameter yaitu words yang berupada dataset pada data.txt dan target yang merupakan kata yang ingin dicari pada dataset. Fungsi mengembalikan boolean dan jumlah perbandingan yang terjadi selama proses pencarian sekuensial.

```
datasetSize = 14557
filePath = "data.txt"
words = readFile(filePath)
if words:
    print("\n\033[1;32mProses dataset dengan " + str(len(words)) + " kata...\033[0m\n")
    print("\033[1;33mHash Chaining:\033[0m")
    hashDataset, collisions = createHashDataset(words, datasetSize)
    print(f"\033[1;32mTotal Collisions : {collisions}\033[0m\n")

    while True:⋯

else:
    print("\033[1;31mTidak ada kata yang dibaca dari file.\033[0m")
```

Gambar 3.8 Bagian pertama program utama
Sumber : Dokumen penulis

Pada bagian program utama, hal pertama yang dilakukan adalah menyiapkan dataset. Dataset awal yang disiapkan berasal dari data.txt dan disimpan pada variabel words. Selanjutnya dilakukan proses pembuatan hashDataset dan program menampilkan jumlah kata yang berhasi disimpan serta *collisions* yang terjadi saat pembuatan hashDataset.

```
while True:
    print("\n\033[1;34mMasukkan Kata yang ingin dicari (atau '\033[1;31mexit\033[1;34m' untuk keluar): \033[0m")
    word_to_search = input().strip()
    if word_to_search.lower() == "exit":
        print("\033[1;31mProgram selesai.\033[0m")
        break

    if not word_to_search:
        print("\033[1;31mInput tidak boleh kosong.\033[0m")
        continue

    startTime = time.time()
    found, comparisons = searchHash(hashDataset, word_to_search,datasetSize)
    elapsedTime = time.time() - startTime
    simple_search_time = elapsedTime * 1000

    startTime = time.time()
    found_conv, comparisons_conv = linearSearch(words, word_to_search)
    elapsedTime = time.time() - startTime
    linearSearch_time = elapsedTime * 1000

    print(f"\n\033[1;33mHasil Pencarian Kata: {word_to_search}\033[0m")
    print("+----+-------------------+-----------------+--------------------+--------------------+")
    print("| No |  Fungsi           | Ketemu          | Perbandingan (n)   | Waktu (ms)         |")
    print("+----+-------------------+-----------------+--------------------+--------------------+")
    print(f"| 1  | Hash              | {'\033[1;32m✓\033[0m' if found else '\033[1;31m✗\033[0m'}              | {comparisons:<17} | {simple_search_time:.6f}          |")
    print(f"| 2  | Linear Search     | {'\033[1;32m✓\033[0m' if found else '\033[1;31m✗\033[0m'}              | {comparisons_conv:<17} | {linearSearch_time:.6f}          |")
    print("+----+-------------------+-----------------+--------------------+--------------------+")
```

Gambar 3.9 Bagian kedua program utama
Sumber : Dokumen penulis

Setelah dataset berhasil disiapkan, program akan meminta masukkan dari pengguna berupa kata yang ingin dicari. Program akan terus meminta masukkan kepada pengguna sampai pengguna memasukkan kata "*exit*" . Setelah pengguna memasukkan kata yang ingin dicari, program akan mengeksekusi fungsi searchHash dan linearSearch. Program akan menampilkan informasi berupa status keberadaan kata pada dataset, jumlah perbandingan yang diperlukan untuk masing-masing metode, dan waktu yang diperlukan untuk masing-masing metode. Program menampilkan data tersebut dalam bentuk tabel sehingga memudahkan pengguna untuk melihat perbedaannya.

IV. Hasil dan Pembahasan

Hasil pengujian diperoleh dari serangkaian pengujian kasus yang mencakup berbagai skenario yang mungkin terjadi dalam proses pencarian kata. Proses pengujian dilakukan untuk menguji keakuratan dan efisiensi pencarian kata dalam berbagai kondisi. Pada sistem pencarian kata seperti yang dirancang, pengguna memiliki kemampuan untuk mencari sebuah kata di dalam dataset. Pengguna cukup memasukkan kata yang ingin dicari dan sistem akan memberikan hasil pencarian beserta keterangan tambahan terkait kompleksitas untuk metode hash dan *linear*.

Pengujian dilakukan dengan mengisi file data.txt yang akan digunakan sebagai referensi. Untuk pengujian ini, diambil dataset berupa 14557 kata bahasa indonesia yang didapat dari https://github.com/damzaky/kumpulan-kata-bahasa-indonesia-KBBI . Dataset tersebut digunakan sebagai referensi utama dalam pengujian.

```
data.txt
baja
bajak
bajakan
bajang
bajang-bajang
bajing
baju
bajul
bak
baka
bakal
bakar
bakas
bakat
bakau
bakaya
bakda
bakdiah
bake
baki
bakit
bako
bakteremia
bakteriofobia
bakteriolisis
bakteriolitik
bakterisid
bakti
baku
bakul
bakung
```

Gambar 4.1 data.txt
Sumber : Dokumen penulis

Program dimulai dengan membaca *file* data.txt yang berisi kumpulan kata yang terdapat pada KBBI. Kemudian program menyimpan kata-kata tersebut di dalam dataset. Kata-kata dimasukkan secara sekuensial sesuai dengan urutan kemunculan pada data.txt. Proses selanjutnya program akan menyusun dataset ke dua yaitu hashDataset. Dataset ini berisi kata-kata yang sama pada dataset pertama namun posisi penyimpanan kata-kata disesuaikan dengan *hash key* yang terbentuk. Program menampilkan banyaknya kata yang berhasil dimasukkan beserta jumlah *collisions* yang terjadi saat pembentukan hashDataset.

```
Proses dataset dengan 14557 kata...

Hash Chaining:
Total Collisions : 13665
```

Gambar 4.2 Proses pembentukan dataset dari data.txt
Sumber : Dokumen penulis

Tahap pengujian selanjutnya adalah pengujian program melalui masukan pengguna. Untuk pengujian pertama akan dimasukkan kata "romawi" , kata ini tersedia pada data.txt sehingga program diharapkan dapat menemukan kata "romawi" pada dataset.

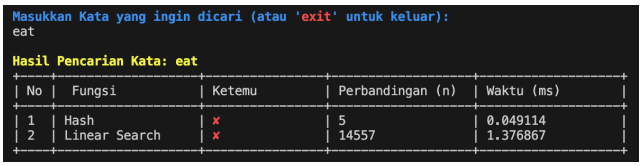
```
Masukkan Kata yang ingin dicari (atau 'exit' untuk keluar):
romawi

Hasil Pencarian Kata: romawi
+----+-------------------+-----------------+--------------------+--------------------+
| No |  Fungsi           | Ketemu          | Perbandingan (n)   | Waktu (ms)         |
+----+-------------------+-----------------+--------------------+--------------------+
| 1  | Hash              | ✓               | 11                 | 0.299215           |
| 2  | Linear Search     | ✓               | 12120              | 1.357079           |
+----+-------------------+-----------------+--------------------+--------------------+
```

Gambar 4.2 Pengujian pencarian kata romawi
Sumber : Dokumen penulis

Sesuai harapan, program menemukan kata "romawi" terdapat perbedaan yang sangat signifikan antara jumlah perbandingan pada fungsi hash dan *linear search.* Pada fungsi hash , hanya dibtuhkan 11 perbandingan dan waktu yang dibutuhkan hanya 0.299215 ms. Hasil ini sangat berbeda jauh dengan fungsi *linear*

search yang membutuhkan 12120 perbandingan dan 1.357079 ms.

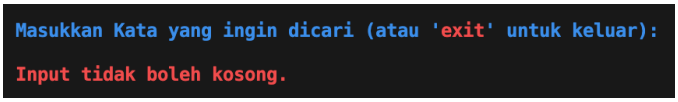
Pengujian selanjutnya dilakukan untuk menguji program saat menangani kata yang tidak tersedia pada dataset. Program akan diberi masukan berupa kata yang bukan bahasa indonesia yaitu "eat". Program diharapkan tidak menemukan kata ini karena tidak terdapat pada dataset.



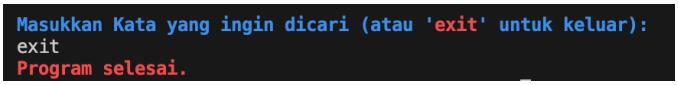
Gambar 4.3 Pengujian pencarian kata eat
Sumber : Dokumen penulis

Hasil program menunjukkan bahwa kedua fungsi tersebut tidak menemukan kata "eat" pada dataset. Namun, terjadi perbedaan signifikan pada waktu dan jumlah perbandingan yang terjadi. Jumlah perbandingan yang terjadi pada fungsi hash hanya terjadi sebanyak lima kali, sedangkan pada *linear search* terjadi perbandingan sebanyak 14557 kali. *Linear Search* membandingkan seluruh data yang tersedia pada dataset mulai dari index awal hingga akhir. Perbedaan waktu juga terjadi diantara ke dua metode, fungsi hash memerlukan 0.049114 ms sedangkan *linear search* memerlukan 1.376867 ms untuk menyelesaikan pencarian.

Pengujian terakhir dilakukan dengan tidak memasukkan kata apapun pada program dan memasukkan kata "exit". Tujuan dari pengujian ini adalah memastikan program dapat mengatasi masukan kosong dan program dapat berhenti dengan baik.



Gambar 4.4 Pengujian masukan kosong pada program
Sumber : Dokumen penulis



Gambar 4.5 Pengujian masukan exit untuk keluar dari program
Sumber : Dokumen penulis

V. KESIMPULAN DAN SARAN

Kesimpulan dari makalah ini adalah bahwa fungsi hash dapat dimanfaatkan untuk mengoptimalkan proses penyimpanan dan pencarian data dalam bahasa Indonesia. Dengan memanfaatkan teori bilangan, terutama konsep modulo, fungsi hash sederhana dapat digunakan untuk menyimpan dan mencari kata pada dataset secara efektif.

Meskipun program ini mampu memberikan hasil pencarian yang cukup akurat, masih terdapat beberapa kekurangan yang perlu diperbaiki. Salah satu masalah yang muncul adalah terjadinya banyak tabrakan (*collision*) saat membentuk dataset dengan hash. Masalah ini dapat diatasi dengan menggunakan pengali berupa bilangan prima dalam pembentukan kunci hash serta memperbesar ruang penyimpanan dataset yang tersedia.

VI. UCAPAN TERIMAKASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya yang melimpah dalam proses penulisan makalah ini sehingga dapat diselesaikan tepat waktu. Penulis juga mengucapkan terima kasih kepada Bapak Dr. Rinaldi Munir, sebagai dosen mata kuliah Matematik Diskrit K01. Penulis juga berterima kasih kepada orang tua dan keluarga yang selalu memberikan dukungan moril dan spiritual, serta menjadi sumber inspirasi dan motivasi selama proses penulisan makalah. Semoga makalah ini dapat memberikan manfaat bagi pembaca dan menjadi kontribusi yang berarti dalam bidang terkait.

VII. LAMPIRAN

- https://github.com/AlfianHanifFY/AplikasiFungsiHash-Matdis.git (*link* GitHub program optimalisasi penyimpanan dan pencarian kata)
- https://github.com/damzaky/kumpulan-kata-bahasa-indonesia-KBBI (*link* github dataset pengujian penyimpanan dan pecarian kata)

REFERENSI

[1] Munir, R., "Teori Bilangan (Bagian 1)," Bahan kuliah IF1220 Matematika Diskrit,[Online].Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/15-Teori-Bilangan-Bagian1-2024.pdf. [Diakses pada 5 Januari 2025].

[2] Jacob, A. E. dan Ashodariya, N., "Hybrid Search Algorithm: Combined Linear and Binary Search Algorithm," [Online]. Tersedia: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8389704. [Diakses pada 6 Januari 2025].

[3] Munir, R., "Kompleksitas Algoritma (Bagian 1)," Bahan kuliah IF1220 Matematika Diskrit, [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/25-Kompleksitas-Algoritma-Bagian1-2024.pdf. [Diakses pada 6 Januari 2025].

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, 6 Januari 2025

Alfian Hanif Fitria Yustanto 13523073