Aplikasi Program Dinamis dalam Implementasi Algoritma Levenshtein *Distance* untuk Pembuatan *Difftool* Sederhana

Moch. Sofyan Firdaus - 13521083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521083@std.stei.itb.ac.id

***Abstrak*—*Difftool* merupakan sebuah program utilitas yang digunakan untuk membandingkan dan menganalisis isi dari dua buah berkas. Beberapa algoritma dapat digunakan untuk membuat program ini, salah satunya adalah algoritma Levenshtein *Distance*. Algoritma Levenshtein *Distance* pada dasarnya digunakan untuk menghitung jarak antara dua *string*. Jarak tersebut didefinisikan sebagai banyaknya upaya minimum yang perlu dilakukan terhadap *string* pertama sedemikian sehingga menjadi serupa dengan *string* kedua. Makalah ini akan mencoba menggeneralisasi algoritma Levenshtein *Distance* sehingga dapat digunakan untuk menentukan perbedaan isi dari dua buah berkas. Strategi algoritma program dinamis akan digunakan dalam implementasi algoritma Levenshtein *Distance* agar memiliki kinerja yang optimal.**

Kata kunci—program dinamis; Levenshtein Distance; Difftool; optimasi

I. PENDAHULUAN

Difftool merupakan sebuah program/perangkat lunak utilitas yang kegunaan utamanya adalah untuk membandingkan isi dari dua buah berkas. *Difftool* dapat menunjukkan baris mana yang perlu dihapus, ditambahkan, atau diubah agar isi kedua berkas menjadi serupa. Hal ini mengakibatkan pemrogram dapat memanfaatkan *difftool* untuk membandingkan sebuah berkas sebelum dan sesudah dilakukan perubahan sehingga dapat membantu dalam melacak setiap perubahan berkas selama proses pengembangan perangkata lunak.

Git merupakan contoh perangkat lunak yang memanfaatkan utilitas *difftool* dan bahkan sangat bergantung pada utilitas tersebut untuk menunjang fungsi utama dari Git. Git menggunakan *difftool* untuk melacak setiap perubahan yang dilakukan oleh permrogram pada satu *commit* ke *commit* lainnya. Git juga menggunakan fungsi dari *difftool* untuk melakukan operasi *merge* jika terjadi konflik di dalam *branch*.

Gambar 1.1. Contoh penggunaan *difftool* dalam perangkat lunak Git (*Sumber: Arsip Penulis*)

Ada banyak algoritma yang dapat digunakan untuk membuat sebuah *difftool*, di antaranya adalah algoritma *Lowest Common Subsequence* dan algoritma Levenshtein *Distance*. Algoritma yang akan digunakan dalam makalah ini adalah algoritma Levenshtein Distance.

Algoritma Levenshtein *Distance* awalnya digunakan untuk menghitung jarak dari dua buah *string*. Algoritma Levenshtein *Distance* mencari berapa upaya paling sedikit yang diperlukan untuk mengubah *string* pertama menjadi sama dengan *string* kedua. Aksi yang dapat dilakukan untuk mengubah *string* tersebut adalah penyisipan (*insertion*), penghapusan (*deletion*), dan penggantian (*substitution*).

Meskipun algoritma Levenshtein *Distance* merupakan algoritma yang sederhana dan relatif mudah diimplementasikan dengan metode rekursif, algoritma tersebut dapat berjalan dengan sangat lambat jika tidak dioptimasi. Kinerja yang lambat ini adalah akibat dari mengevaluasi fungsi yang sama berkali-kali meskipun fungsi tersebut sudah pernah dievaluasi. Untuk

itu, perlu sebuah strategi agar algoritma ini berjalan dengan kinerja yang optimal.

Salah satu metode yang dapat digunakan untuk melakukan optimasi terhadap algoritma Levenshtein *Distance* adalah *caching*. Dengan *caching*, semua fungsi yang sudah dievaluasi akan disimpan di dalam memori. Jika suatu saat fungsi yang sama diperlukan, nilai fungsi tersebut hanya perlu diambil dari memori sehingga tidak perlu melakukan evaluasi ulang. Hal yang menarik dalam *caching* adalah bahwa pada dasarnya *caching* hanya pemanfaatan dari sebuah memori yang disimpan dalam sebuah struktur data seperti *array*. Metode ini sangat mirip dengan metode yang bernama program dinamis.

Metode program dinamis memecahkan permasalahan dengan membaginya menjadi tahapan-tahapan kecil. Setiap tahapan tersebut akan menghasilkan nilai yang disimpan dalam sebuah struktur data (memori) yang nantinya akan digunakan pada tahapan selanjutnya. Hal ini memungkinkan kinerja algoritma menjadi lebih optimal karena fungsi yang sama tidak perlu dievaluasi berkali-kali, seperti halnya *caching*. Makalah ini akan menggunakan metode program dinamis untuk mengoptimasi kinerja algoritma Levenshtein *Distance*.

II. Landasan Teori

A. Cara Kerja Difftool

Difftool akan menunjukkan perbedaan dari dua buah berkas. Perbedaan tersebut antara lain mana yang dihapus, ditambahkan, dan mana yang diubah. Terdapat tiga mode yang biasanya dipakai dalam perbandingan, yaitu perbandingan karakter per karakter, perbandingan kata per kata, dan perbandingan baris per baris. Perbandingan karakter per karakter menghasilkan tingkat detail yang paling tinggi, tetapi membutuhkan waktu eksekusi algoritma yang paling lama. Perbandingan kata per kata atau baris per baris akan berjalan lebih cepat. Makalah ini akan menggunakan mode perbandingan baris per baris yang merupakan mode yang cukup sering digunakan untuk membandingkan *source code*.

B. Algoritma Levenshtein Distance

Algoritma Levenshtein *Distance*, juga sering disebut *Edit Distance*, akan menghitung jumlah upaya minimum yang diperlukan untuk mengubah *string* pertama menjadi *string* kedua. Upaya tersebut meliputi penyisipan (*insertion*), penghapusan (*deletion*), dan penggantian (*substitution*). Algoritma ini bisa diimplementasikan secara rekursif atau menggunakan matriks dua dimensi untuk kinerja yang lebih optimal.

Sebagai contoh, misalnya terdapat dua buah *string*, yaitu "taksonomi" dan "klakson". Operasi-operasi yang perlu dilakukan untuk mengubah "taksonomi" menjadi "klakson" di antaranya:

1. Ganti karakter "t" menjadi "k". *String* "taksonomi" berubah menjadi "kaksonomi".
2. Sisipkan karakter "l" di antara karakter "k" dan "a". *String* "kaksonomi" berubah menjadi "klaksonomi".
3. Hapus karakter "o" setelah karakter "n". Kata "klaksonomi" menjadi "klaksonmi".
4. Hapus karakter "m". Kata "klaksonmi" menjadi "klaksoni".
5. Hapus karakter "i". Kata "klaksoni" menjadi "klakson".

Jadi, jumlah minimum operasi yang diperlukan untuk mengubah "taksonomi" menjadi "klakson" adalah lima dan jumlah tersebut dikatakan sebagai jarak edit di antara *string* "taksonomi" dan *string* "klakson".

Umumnya algoritma Levenshtein *Distance* hanya digunakan untuk menghitung nilai jarak edit kedua *string* tanpa memedulikan operasi apa saja yang harus dilakukan sehingga algoritma ini sering digunakan untuk melakukan pencarian *string* yang mirip satu sama lain. Namun, dengan mengekstrak informasi mengenai operasi yang dilakukan terhadap *string*, algoritma ini bisa digunakan untuk membuat program perbandingan berkas atau *difftool*.

Umumnya algoritma Levenshtein *Distance* juga hanya dioperasikan terhadap *string* atau sekuens karakter. Namun, sebenarnya algoritma ini dapat dioperasikan terhadap sekuens apa pun selama itu dapat dibandingkan satu sama lain. Misalnya algoritma ini dapat dioperasikan terhadap sekuens bilangan integer karena bilangan integer dapat dibandingkan satu sama lain. Maka dari itu, untuk membuat sebuah *difftool* yang menggunakan mode perbandingan baris per baris, algoritma Levenshtein *Distance* harus dioperasikan terhadap sekuens *string* dan mengekstrak informasi mana baris yang perlu disisipkan, dihapus, dan diganti.

Berikut ini adalah rumus yang digunakan untuk menghitung jarak edit dua buah *string* dengan algoritma Levenshtein *Distance*.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise} \end{cases}$$

Gambar 2.1. Rumus Levenshtein *Distance* (*Sumber: wikipedia.com*)

Berikut ini penjelasan mengenai rumus di atas dengan operasi transformasi *string* yang berkorespondensi.

1. lev(tail(*a*), tail(*b*)) berkorespondensi dengan operasi penggantian (*substitution*) jika kedua buah *string* tidak memiliki karakter pertama yang sama. Jika karakter pertama kedua sama, maka rumus tersebut berarti mengabaikan kedua karakter pertama kedua *string*.
2. lev(tail(*a*), *b*) berarti karakter pertama pada *string* pertama dihilangkan sehingga ekspresi tersebut berkorespondensi dengan operasi penghapusan.
3. lev(*a*, tail(*b*)) berarti karakter pertama pada *string* kedua dihilangkan sehingga ekspresi tersebut berkorespondensi dengan operasi penyisipan.

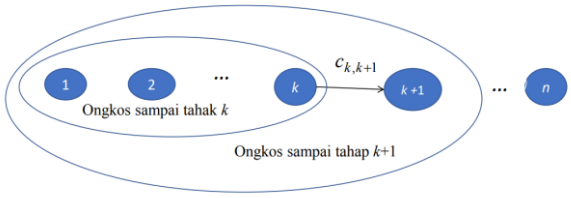
Dengan mengetahui operasi yang berkorespondensi dengan setiap cabang rumus, informasi mengenai operasi tersebut dapat dicatat sehingga algoritma Levenshtein *Distance* dapat digunakan untuk membuat program *difftool.*

C. *Program Dinamis*

Program dinamis merupakan strategi algoritma atau metode pemecahan suatu permasalahan dengan cara menguraikan solusi menjadi sekumpulan tahapan (*stage*) atau upamaasalah sedemikian sehingga solusi persoalan pada setiap upamasalah menjadi serangkaian keputusan yang saling berkaitan. Istilah "dinamis" pada program dinamis berkaitan dengan metode pencarian solusi yang memanfaatkan tabel dinamis sebagai media perhitungan.

Program dinamis sering digunakan untuk menyelesaikan masalah-masalah optimasi (maksimasi atau minimasi). Hal tersebut mirip dengan algoritma *greedy*. Perbedaan antara program dinamis dengan algoritma *greedy* terletak pada jumlah rangkaian keputusan yang dihasilkan dan dipertimbangkan. Program dinamis menggunakan lebih dari satu rangkaian keputusan untuk memutuskan hasil perhitungan pada setiap tahapan.

Program dinamis memanfaatkan prinsip optimalitas dalam membuat serangkaian keputusan yang optimal. Prinsip optimalitas mengatakan bahwa jika solusi total optimal, maka solusi upamasalah pada setiap tahap juga optimal. Dengan kata lain,untuk mendapatkan solusi optimal pada tahap ke-k, cukup gunakan solusi optimal pada tahap ke-*k-1* sehingga tidak perlu kembali ke tahap awal.



Gambar 2.2. Ilustrasi prinsip optimalitas (*Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf)*

Program dinamis memiliki karakteristik sebagai berikut.

1. Permasalahan dapat dibagi menjadi beberapa upamasalah yang disebut sebagai tahapan (*stage*). Selanjutnya pada setiap tahap diambil satu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Status tersebut umumnya merupakan variasi kemungkinan masukan yang ada pada suatu tahap.
3. Hasil keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos pada suatu tahap meningkat secara teratur seiring bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos dari tahap tersebut ke tahap berikutnya.
6. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap keputusan terbaik untuk setiap status pada tahap ke-*k* memberikan keputusan terbaik untuk setiap status pada tahap ke-*k+1*.
7. Prinsip optimalitas berlaku pada persoalan tersebut.

Terdapat dua jenis pendekatan yang digunakan dalam algoritma program dinamis:

1. **Program dinamis maju (*forward* atau *up-down)***
   Program dinamis bergerak mulai dari tahap ke-1, lalu tahap ke-2, ke-3, dan seterusnya hingga tahap ke-n. Program dinamis maju memiliki rangkaian peubah keputusan $x_1, x_2, \ldots, x_n$.
2. **Program dinamis mundur (*backward* atau *bottom-up)***
   Program dinamis bergerak dari tahap ke-n, terus bergerak mundur ke tahap ke-*n-1*, ke-*n-2*, dan seterusnya hingga tahap ke-1. Program dinamis mundur memiliki rangkaian peubah keputusan $x_n, x_{n-1}, \ldots, x_1$.

Selain dua jenis pendekatan, terdapat juga langkah-langkah dalam pengembangan algoritma program dinamis, antara lain:

1. **Karakteristikkan struktur solusi optimal**
   Langkah ini menentukan tahap, peubah keputusan, status, dan sebainya.
2. **Definisikan nilai solusi optimal secara rekursif**
   Pada langkah ini, tentukan hungan nilai optimal suatu tahap dengan tahap sebelumnya.
3. **Hitung nilai solusi optimal secara maju atau mundur**
   Pada langkah ini, tabel digunakan sebagai media perhitungan. Perhitungan dilakukan secara maju, yaitu dari tahap ke-1 ke tahap ke-*n* atau dilakukan secara mundur dari tahap ke-*n* sampai tahap ke-1 dengan tahap ke-*n* merupakan tahap terakhir dari program dinamis.
4. **Rekonstruksi solusi optimal (opsional)**
   Langkah ini melakukan rekonstruksi solusi secara mundur.

Dengan pendekatan dan langkah-langkah untuk melakukan program dinamis, persoalan optimasi dapat diselesaikan menggunakan metode pendekatan program dinamis. Algoritma Levenshtein *Distance* merupakan salah satu dari persoalan optimasi. Dalam makalah ini, algoritma Levenshtein Distance akan diimplementasikan menggunakan metode program dinamis untuk membuat sebuah program *difftool*.

III. Pembahasan dan Implementasi

A. Pemetaan Masalah

Untuk membuat sebuah program *difftool*, persoalan yang perlu diselesaikan adalah bagaimana cara menentukan apa saja perbedaan isi dua buah berkas. Hal ini dapat disederhanakan menjadi persoalan mencari operasi-operasi yang harus dilakukan terhadap berkas pertama sedemikian sehingga isi berkas pertama berubah menjadi sama persis dengan isi berkas kedua. Operasi-operasi tersebut meliputi penyisipan suatu baris, penghapusan suatu baris, dan penggantian suatu baris dengan baris yang lain. Agar operasi yang dilakukan akurat, jumlah operasi yang dilakukan harus minimum sehingga persoalan ini juga termasuk ke dalam persoalan minimasi.

Algoritma Levenshtein *Distance* merupakan algoritma yang cocok untuk menyelesaikan persoalan ini. Algoritma Levenshtein *Distance* dapat menghitung jumlah operasi minimum yang diperlukan untuk mengubah satu *string* menjadi *string* yang lain. Jumlah operasi tersebut sering juga disebut dengan jarak edit. Selain itu, algoritma Levenshtein *Distance* juga dapat digeneralisasi sedemikian sehingga dapat dioperasikan terhadap sekuens *string*, mengubah satu sekuens *string* menjadi sekuens *string* yang lain. Transformasi sekuens *string* diperlukan untuk membuat *difftool* dengan mode perbandingan baris per baris. Informasi mengenai operasi yang dilakukan diperlukan untuk membuat sebuah *difftool.* Informasi ini akan disimpan beriringan dengan proses perhitungan jarak edit.

B. Analisis Persoalan dengan Program Dinamis

Berikut merupakan karakteristik struktur optimal dalam algoritma Levenshtein *Distance*.

- Status (a, b) adalah dua bilangan yang merepresentasikan panjang sekuens yang sedang dibandingkan
- Tahapan pada persoalan ini merupakan status itu sendiri. Dengan kata lain, setiap status dievaluasi pada tahap yang berbeda-beda.

Untuk mengimplementasikan algoritma Levenshtein Distance, diperlukan sebuah relasi rekurens yang tepat untuk merepresentasikan fungsi Levenshtein Distance yang asli. Fungsi Levenshtein Distance yang asli tidak bisa digunakan karena parameter yang dimasukkan merupakan sebuah *string*. Padahal, program dinamis merupakan metode pemecahan masalah yang menggunakan matriks, sedangkan matriks pada umumnya hanya bisa diakses dengan indeks berupa bilangan cacah.

Persoalan ini dapat diselesaikan dengan mengubah tipe parameter yang digunakan menjadi sebuah bilangan, yaitu panjang dari sekuens yang sedang dibandingkan. Hal ini dimungkinkan karena pada fungsi asli Levenshtein *Distance, string* yang menjadi parameternya hanya digunakan untuk membandingkan karakter pada indeks tertentu.

Berikut merupakan fungsi baru Levenshtein *Distance* yang akan digunakan dalam program dinamis ini.

$$
\begin{aligned}
&D(a,b) = a, && \text{jika } a = 0 && \text{(basis)}\\
&D(a,b) = b, && \text{jika } b = 0 && \text{(basis)}\\
&D(a,b) = D(a-1,b-1), && \text{jika } s_1[a-1] = s_2[b-1] && \text{(basis)}\\
&D(a,b) = 1 + \min \begin{cases} D(a-1,b) \\ D(a,b-1) \\ D(a-1,b-1) \end{cases} && && \text{(rekurens)}
\end{aligned}
$$

Gambar 3.1. Relasi rekurens untuk Levenshtein *Distance* (*Sumber: Arsip Penulis*)

Berikut merupakan contoh penyelesaian persoalan perbadingan dua buah *string* (sekuens karakter) untuk kata “taksonomi” dan kata “klakson”.

TABLE I. PROSES ALGORITMA LEVENSHTEIN DISTANCE

a	b	s1[a-1]	s2[b-1]	D(a-1,b)	D(a,b-1)	D(a-1,b-1)	D(a,b)	Operasi
0	0						0	I
0	1		k				1	A
0	2		l				2	A
0	3		a				3	A
0	4		k				4	A
0	5		s				5	A
0	6		o				6	A
0	7		n				7	A
1	0	t					1	R

1	1	t	k	1	1	0	1	S
1	2	t	l	2	1	1	2	A
1	3	t	a	3	2	2	3	A
1	4	t	k	4	3	3	4	A
1	5	t	s	5	4	4	5	A
1	6	t	o	6	5	5	6	A
1	7	t	n	7	6	6	7	A
2	0	a					2	R
2	1	a	k	1	2	1	2	R
2	2	a	l	2	2	1	2	S
2	3	a	a	3	2	2	2	I
2	4	a	k	4	2	3	3	A
2	5	a	s	5	3	4	4	A
2	6	a	o	6	4	5	5	A
2	7	a	n	7	5	6	6	A
3	0	k					3	R
3	1	k	k	2	3	2	2	I
3	2	k	l	2	2	2	3	R
3	3	k	a	2	3	2	3	R
3	4	k	k	3	3	2	2	I
3	5	k	s	4	2	3	3	A
3	6	k	o	5	3	4	4	A
3	7	k	n	6	4	5	5	A
4	0	s					4	R
4	1	s	k	2	4	3	3	R
4	2	s	l	3	3	2	3	S
4	3	s	a	3	3	3	4	R
4	4	s	k	2	4	3	3	R

4	5	s	s	3	3	2	2	I
4	6	s	o	4	2	3	3	A
4	7	s	n	5	3	4	4	A
5	0	o					5	R
5	1	o	k	3	5	4	4	R
5	2	o	l	3	4	3	4	R
5	3	o	a	4	4	3	4	S
5	4	o	k	3	4	4	4	R
5	5	o	s	2	4	3	3	R
5	6	o	o	3	3	2	2	I
5	7	o	n	4	2	3	3	A
6	0	n					6	R
6	1	n	k	4	6	5	5	R
6	2	n	l	4	5	4	5	R
6	3	n	a	4	5	4	5	R
6	4	n	k	4	5	4	5	R
6	5	n	s	3	5	4	4	R
6	6	n	o	2	4	3	3	R
6	7	n	n	3	3	2	2	I
7	0	o					7	R
7	1	o	k	5	7	6	6	R
7	2	o	l	5	6	5	6	R
7	3	o	a	5	6	5	6	R
7	4	o	k	5	6	5	6	R
7	5	o	s	4	6	5	5	R
7	6	o	o	3	5	4	4	I
7	7	o	n	2	4	3	3	R
8	0	m					8	R

8	1	m	k	6	8	7	7	R
8	2	m	l	6	7	6	7	R
8	3	m	a	6	7	6	7	R
8	4	m	k	6	7	6	7	R
8	5	m	s	5	7	6	6	R
8	6	m	o	4	6	5	5	R
8	7	m	n	3	5	4	4	R
9	0	i					9	R
9	1	i	k	7	9	8	8	R
9	2	i	l	7	8	7	8	R
9	3	i	a	7	8	7	8	R
9	4	i	k	7	8	7	8	R
9	5	i	s	6	8	7	7	R
9	6	i	o	5	7	6	6	R
9	7	i	n	4	6	5	5	R

Dari tabel tersebut didapatkan jarak edit dari “taksonomi” dan “klakson” adalah D(9,7) = 5. Selain itu, perhatikan bahwa dengan operasi yang dilakukan pada kata “taksonomi” dapat diperoleh dengan menelusuri kolom operasi secara mundur berdasarkan operasi yang dilakukan. Operasi terakhir adalah R atau penghapusan (*remove*). Operasi penghapusan didapatkan dari rumus D(a-1,b) sehingga kita dapatkan operasi sebelumnya berada pada D(8,7), yaitu R. Operasi sebelumnya lagi berada di D(7,7), yaitu R. Operasi sebelumnya lagi berada berada di D(6,7), yaitu I atau tidak ada operasi (*ignore*). Karena operasi I didapatkan dari rumus D(a-1,b-1), maka operasi sebelumnya berada di D(5,6), yaitu I. Operasi sebelumnya lagi berada di D(4,5), yaitu I. Lalu di D(3,4), yaitu I. Lalu di D(2,3), yaitu I. Lalu operasi sebelumnya berada di D(1,2), yaitu A atau penambahan (*addition*). Karena operasi A diperoleh dari rumus D(a,b-1), maka operasi sebelumnya berada di D(1,1), yaitu S atau penggantian (*substitution*). Operasi S diperoleh dari D(a-1,b-1) sehingga operasi sebelumnya adalah D(0,0) yang tidak perlu dicatat karena hanya sebuah *sentinel*. Dengan demikian, jika kita runut operasi-operasi tersebut, didapatkan urutan operasi S, A, I, I, I, I, I, R, R, dan R. Operasi tersebut berarti bahwa jika dilakukan traversal pada kata “taksonomi” sembari melakukan operasi-operasi tersebut, maka kata “taksonomi” dapat berubah menjadi “klakson”.

Selanjutnya algoritma ini akan digunakan pada sekuens *string* untuk membuat program *difftool*.

C. *Implementasi*

Kode sumber implementasi program *difftool* dengan menggunakan algoritma Levenshtein *Distance* dan program dinamis dilampirkan di akhir makalah ini.

D. *Pengujian*

Berikut contoh keluaran program *difftool* yang telah dibuat.

```
lev-diff on  main [!?] is 📦 v0.1.0 via 🦀 v1.69.0
❯ target/release/lev-diff file1.txt file2.txt
 1  | Lorem ipsum dolor sit amet, officia excepteur ex fugiat
 2  | reprehenderit enim labore culpa sint ad nisi Lorem pariatur
 3 ~| mollit ex esse exercitation. ⇆ commodo ex non excepteur.
 4  | excepteur officia. Reprehenderit nostrud nostrud ipsum
 5  | Lorem est aliquip voluptate amet voluptate dolor minim
 6  | nulla est proident. Nostrud officia pariatur ut officia.
 7 -| Sit irure elit esse ea nulla sunt ex occaecat reprehenderit
 7  | commodo officia dolor Lorem duis laboris cupidatat officia
 9 -| voluptate. Culpa proident adipisicing id nulla nisi laboris
 8  | ex in Lorem sunt duis officia eiusmod. Aliqua reprehenderit
 9  | commodo ex non excepteur duis sunt velit enim. Voluptate
10 +| voluptate. Culpa proident adipisicing id nulla nisi laboris
11  | laboris sint cupidatat ullamco ut ea consectetur et est culpa
12  | et culpa duis.
```

Gambar 3.2. Contoh keluaran program *difftool* (*Sumber: Arsip Penulis*)

IV. PENUTUP

A. *Simpulan*

Program *difftool* dapat dibuat menggunakan algoritma Levenshtein *Distance* meskipun pada umumnya algoritma Levenshtein *Distance* hanya digunakan untuk analisis kemiripan dua buah *string*. Strategi program dinamis juga dapat digunakan untuk mengoptimasi kinerja dari Levenshtein *Distance*.

B. *Saran*

Program *difftool* yang dibuat dalam makalah ini tidak dapat digunakan untuk membandingkan dua berkas yang berukuran besar. Hal ini disebabkan oleh metode program dinamis yang menggunakan matriks dua dimensi sehingga mengonsumsi sumber daya memori yang sangat besar. Oleh karena itu, gunakan lah program yang sudah sering digunakan seperti GNU *diff* atau Git.

LINK KODE PROGRAM

https://github.com/msfir/lev-diff

UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan puji dan syukur kepada Allah swt. atas nikmat dan ksehatan yang telah dikaruniakan-Nya sehingga penulis dapat menyelesaikan tugas makalah mata kuliah IF2211 Strategi Algoritma. Saya mengucapkan terima kasih yang sebesar-besarnya kepada Dr. Ir. Rinaldi Munir, MT. selaku dosen dan pengajar yang telah memberikan ilmu yang bermanfaat bagi penulis sehingga penulis dapat menyelesaikan pembuatan makalah ini. Semoga Allah swt. membalas semua kebaikan dengan kebaikan yang berliapat ganda. Semoga pembahasan pada makalah ini dapat bermanfaat bagi siapa pun yang membaca dan dapat terus dikembangkan lebih lanjut lagi. Terakhir, penulis mohon maaf atas segala kekurangan dan kesalahan yang penulis lakukan dalam makalah ini.

REFERENSI

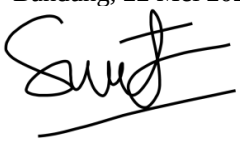
[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf, diakses pada tanggal 18 Mei 2023.

[2] https://www.trivusi.web.id/2022/04/apa-itu-algoritma-levenshtein-distance.html, diakses pada tanggal 22 Mei 2023.

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, 22 Mei 2023



Moch. Sofyan Firdaus - 13521083