Penerapan Algoritma *Dynamic Programming* dalam *Spell Check*

Sekar Anglila Hapsari / 13514069
Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514069@std.stei.itb.ac.id

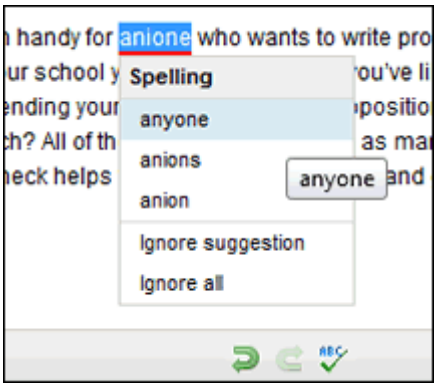
***Abstract*—Manusia sudah tidak asing lagi dengan fitur *Spell Check* yang ada pada aplikasi-aplikasi atau program-program pengolah kata yang ada. *Spell check* mengkomputasi hasilnya dengan Algoritma *Minimum Edit Distance* agar sugesti kata yang diberikan tidak asal-asalan. Algoritma tersebut mengaplikasikan salah satu strategi algoritma yang terkenal, yaitu program dinamis atau *dynamic programming*. Dalam makalah ini, akan dibahas pengaplikasian *dynamic programming* dalam fitur *spell check*.**

Keywords—Spell check*, program dinamis, *Minimum Edit Distance, backtrace, weighted Minimum Edit Distance

I. PENDAHULUAN

Dalam era digital ini, sudah banyak sekali hadir aplikasi-aplikasi atau program-program pengolah kata yang membantu manusia dalam penyusunan, penyuntingan, pemformatan, dan pencetakan dokumen. Contoh programnya antara lain *Microsoft Word*, *Notepad*, dan lainnya. Dalam aplikasi ini hadir fitur *Spell Check* yang membantu pengguna untuk membuat dokumen dengan ejaan yang baik dan benar.

Fitur *Spell Check* ini memeriksa kata yang diketik oleh pengguna. Jika salah atau tidak tepat, akan dicari ejaan atau kata yang benar dan ditampilkan pilihan-pilihan ejaan yang kira-kira dimaksud pengguna. Tidak hanya di perangkat lunak pengolah kata saja, sekarang fitur *Spell Check* juga digunakan dalam mesin pencari. Jika *input* pengguna ada yang menggunakan ejaan yang salah, maka akan ditampilkan ejaan yang benar.



Gambar 1 : Contoh tampilan hasil *spell check* dalam salah satu perangkat lunak pengolah kata[1]

Spell Check menggunakan algoritma *Minimum Edit Distance* dalam komputasinya. Algoritma tersebut menerapkan prinsip *dynamic programming* atau program dinamis yang merupakan salah satu strategi algoritma. Dengan menggunakan strategi algoritma ini, algoritma yang digunakan akan mencari solusi optimal tanpa melakukan repetisi.

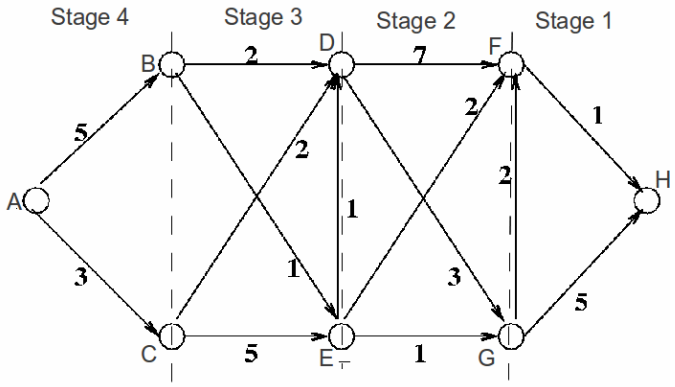
Selain untuk *Spell Check*, Algoritma *Minimum Edit Distance* juga digunakan untuk *Speech Recognition*, *Machine Translation*, dan *Information Extraction*. Penerapan program dinamis dalam Algoritma *Minimum Edit Distance* untuk komputasi Fitur *Spell Check* akan dijelaskan dalam makalah ini.

II. DASAR TEORI

A. *Program Dinamis*

Program dinamis atau *dynamic programming* merupakan salah satu strategi algoritma yang dipakai untuk mencari solusi optimal dengan efektif. Program dinamis adalah teknik algoritma yang biasanya memiliki algoritma rekurens dan penyelesaiannya diuraikan menjadi beberapa *stage* atau tahapan. Sehingga, penyelesaiannya dapat dihasilkan dari serangkaian keputusan yang saling berhubungan berdasarkan *stage* atau tahapan solusinya.

Dalam setiap tahapan, akan ada beberapa *state* atau keadaan. *State* atau keadaan yang dimaksud adalah suatu cara untuk mendeskripsikan suatu situasi yang merupakan sub-penyelesaian dari permasalahan yang ada. *State* atau keadaan ditentukan berdasarkan situasi-situasi yang mungkin terjadi dalam setiap tahapan pencarian solusi dari permasalahan yang ada.



Gambar 2 : Gambaran tahapan yang terdiri dari beberapa *state* dalam program dinamis[2]

Solusi yang dihasilkan dari program dinamis memiliki kompleksitas polinomial sehingga, waktu yang diperlukan untuk mencapai sebuah solusi akan jauh lebih cepat daripada penyelesaian dengan algoritma *brute force*, *backtrack*, *branch and bound*, dan strategi algoritma lainnya.

Beberapa syarat dari penyelesaian yang dihasilkan oleh strategi algoritma program dinamis adalah, antara lain :

1. Terdapat sejumlah pilihan yang mungkin (jumlahnya berhingga)
2. Penyelesaian pada setiap *stage* atau tahap dapat dibangun dari hasil penyelesaian pada tahap-tahap sebelumnya
3. Terdapat persyaratan optimasi dan kendala untuk membatasi jumlah pilihan yang harus dipertimbangkan pada suatu tahap atau keadaan

Penyelesaian dengan strategi algoritma program dinamis seperti penyelesaian dengan strategi algoritma *greedy*. Kedua strategi algoritma tersebut dapat menghasilkan sebuah penyelesaian dari suatu masalah dengan membentuk solusi tersebut secara bertahap. Namun, algoritma *greedy* kurang baik untuk beberapa permasalahan. Hal ini karena algoritma *greedy* tidak mempertimbangkan lebih jauh apakah pilihan yang diambil sudah tepat pada langkah-langkah atau tahapan selanjutnya. Sehingga terkadang, tidak dihasilkan solusi yang terbaik, hanya yang memenuhi persyaratan.

Berbeda dengan strategi algoritma *greedy*, strategi algoritma program dinamis menggunakan Prinsip Optimalitas sehingga, rangkaian keputusan yang dibuat oleh algoritma tersebut adalah keputusan yang optimal. Sebab, Prinsip Optimalitas mengatakan bahwa, jika solusi total optimal, maka solusi pada tahap ke-*n* juga merupakan solusi yang optimal. Jika bekerja dari tahap ke-*n* menuju tahap (*n* + 1), maka keputusan solusi yang digunakan adalah yang optimal, tanpa harus kembali ke tahap awal. Sehingga, ongkos pada tahap (*n* + 1) sama dengan ongkos pada tahap ke-*n* ditambah dengan ongkos dari tahap ke-*n* ke tahap (*n* + 1). Sehingga, program dinamis juga mungkin menghasilkan lebih dari satu solusi optimal dari suatu permasalahan.

Dari uraian diatas, dapat dilihat bahwa persoalan atau permasalahan yang dapat diselesaikan dengan strategi algoritma program dinamis memiliki beberapa karakteristik tertentu, yaitu antara lain:

1. Persoalan dapat dibagi menjadi beberapa tahap atau *stage*. Dalam setiap tahap ini, hanya diambil satu keputusan tertentu.
2. Setiap tahap dalam persoalan tersebut terdiri dari beberapa *state* atau keadaan yang berhubungan dengan tahapan tersebut. *State* atau keadaan adalah kemungkinan-kemungkinan yang mungkin terjadi dalam tahapan penyelesaian permasalahan tersebut. Jumlah keadaan yang mungkin terjadi bisa saja berhingga, atau mungkin juga tidak berhingga.
3. Hasil keputusan dari setiap *state* pada sebuah tahap dihasilkan dari transformasi hasil keputusan *state* sebelumnya pada tahap sebelumnya.
4. Ongkos pada suatu tahap meningkat secara teratur seiring dengan bertambahnya tahapan yang dilewati.
5. Keputusan yang paling optimal yang dihasilkan oleh suatu tahapan bersifat independen dari keputusan yang dihasilkan pada tahap sebelumnya.
6. Terdapat hubungan rekursif dalam algoritma yang membantu menentukan keputusan pada tahap ke-*n* memberikan keputusan terbaik untuk setiap *state* pada tahap selanjutnya.
7. Berlaku prinsip optimalitas pada persoalan yang ingin diselesaikan.

Dalam penyelesaian suatu persoalan dengan strategi algoritma program dinamis, terdapat dua pendekatan. Dua pendekatan itu adalah antara lain: pendekatan maju dan pendekatan mundur. Pendekatan maju menyelesaikan suatu persoalan dengan bergerak dari tahap pertama sampai tahap terakhir, sehingga rangkaian keputusan yang dihasilkan berurut dari tahap awal hingga tahap akhir. Berbeda dengan pendekatan maju, pendekatan mundur melakukan hal yang sebaliknya. Yaitu dengan menyelesaikan suatu persoalan dengan bergerak dari tahap terakhir hingga tahap pertama. Sehingga, rangkaian keputusan yang dihasilkan juga sebaliknya, yaitu berurut mulai dari keputusan terakhir hingga keputusan pertama. Namun, pendekatan apapun yang digunakan akan menghasilkan penyelesaian atau solusi optimal yang sama.

Beberapa langkah yang dilakukan untuk membuat suatu algoritma program dinamis antara lain:

1. Mendefinisikan karakteristik dari solusi optimal yang ingin dihasilkan
2. Mendefinisikan nilai dari solusi optimal permasalahan secara rekursif
3. Menghitung solusi optimal permasalahan dengan pendekatan maju atau pendekatan mundur

4. Mengkonstruksi solusi optimal permasalahan dengan memroses rangkaian keputusan yang telah dihasilkan dari perhitungan rekursif.

Strategi algortima program dinamis dapat menyelesaikan berbagai macam masalah, contohnya masalah-masalah klasik seperti pencarian lintasan terpendek, *knapsack problem*, atau *travelling salesman problem*. Strategi algoritma program dinamis sangat berguna untuk menyelesaikan suatu masalah secara efektif. Berbagai pencarian solusi dari suatu permasalahan dalam dunia nyata mengaplikasikan stratefi ini. Dalam bagian makalah selanjutnya, akan dibahas penerapan strategi algoritma program dinamis dalam *Spell Check* menggunakan algoritma *Minimum Edit Distance*.

III. PEMBAHASAN

Spell Check merupakan salah satu fitur paling berguna dalam perangkat lunak pengolah kata bagi penggunanya. *Spell Check* membantu pengguna dalam membetulkan ejaannya yang salah dengan memberikan beberapa usulan ejaan yang benar. Usulan tersebut juga tidak asal-asalan, usulan yang diberikan merupakan kata-kata terdekat dengan yang akan dibetulkan.

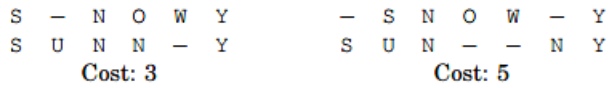
Untuk mendapatkan usulan-usulan kata-kata tersebut, *Spell Check* menggunakan algoritma *Minimum Edit Distance* yang menerapkan strategi algoritma program dinamis. Berikut adalah pembahasannya.

A. Algoritma Minimum Edit Distance

Saat fitur *Spell Check* mengusulkan kata-kata dengan ejaan terdekat, fitur *Spell Check* memeriksa kata-kata dalam kamus yang ejaannya mendekati. Bagaimana fitur *Spell Check* mengomputasi perhitungan kedekatan ini?

Dengan menggunaka algoritma *Minimum Edit Distance*, akan dapat dihitung jumlah perbedaan antara dua kata. Apa yang dimaksud dengan ini? Ketika dua kata disejajarkan, akan terlihat berapa banyak perubahan yang harus dilakukan agar kata pertama dapat diubah menjadi kata kedua, atau sebaliknya. Jumlah total dari perubahan-perubahan inilah yang menjadi tolak ukut kedekatannya. Semakin kecil jumlah perubahan yang dilakukan, maka semakin dekat pula kata tersebut antara satu sama lainnya. Saat dua kata disejajarkan dan dibandingkan, tentu saja ada banyak kemungkinannya, sehingga banyak sekali kemungkinan kata yang dapat dihasilkan.

Berikut adalah contohnya,



Gambar 3 : Contoh kemungkinan-kemungkinan cara untuk menyejajarkan dua kata *Snowy* dan *Sunny*[3]

Menurut Gambar 3, dapat dilihat bahwa ada dua alternatif berbeda dalam perbandingan kata *Snowy* dan *Sunny*, masing-masing dengan ongkos berbeda. Untuk perbandingan sebelah kiri, dapat dilihat untuk merubah kata *Snowy* menjadi *Sunny*, dibutuhkan ongkos sebanyak tiga. Perubahan yang terjadi antara lain menambahkan huruf U setelah S di kata *Snowy*, mengganti huruf O di kata *Snowy* menjadi huruf N, dan menghilangkan huruf W pada *Snowy*.

Sedangkan untuk perbandingan yang yang di sebelah kanan Gambar 3, dapat dilihat bahwa untuk merubah kata *Snowy* menjadi *Sunny*, dapat dilakukan penyejajaran seperti sebelah kanan gambar. Namun, ongkos yang dipakai lebih banyak, yaitu sebanyak 5. Dimana perubahan yang dilakukan antara lain, menambah huruf S di awal kata *Snowy*, merubah huruf *S* pada kata *Snowy* menjadi huruf U, menghapus huruf O dan huruf W pada kata *Snowy*, dan menambahkan huruf N pada kata *Snowy*.

Dengan banyaknya kemungkinan cara untuk menyejajarkan kedua kata, algoritma ini sangat berguna untuk memberikan hasil kata-kata yang paling tepat sebagai usulan dalah fitur *Spell Check*. Hal ini karena algoritma *Minimum Edit Distance* akan mengkomputasi perhitungan ongkos perubahan yang dilakukan, dan ketika melakukan *backtrack* pada hasilnya, akan didapatkan cara untuk menyejajarkan kedua kata yang dibandingkan serta perubahan yang dilakukan.

Untuk melakukan hal itu, ada empat hal yang harus diingat dalam algoritma ini, yaitu:

1. *State* awalnya adalah kata yang akan diubah ejaannya
2. Perubahan yang dilakukan ada tiga macam, yaitu insersi, penghapusan, dan substitusi
3. *State* akhir adalah kata yang menjadi hasil transformasinya
4. Ongkos yang diinginkan adalah ongkos terendah

Sehingga untuk dua kata atau *string*, *x* dan *y*, dengan panjang masing-masing *string* atau kata *n* dan *m*, akan dihitung perbedaannya untuk setiap D(i, j), yaitu ongkos per huruf. Dimana *x*[1..i] dan *y*[1..j], sehingga *Minimum Edit Distance* antara kata *x* dan *y* adalah D(*n*, *m*).

Perubahan yang dilakukan ada tiga macam, yaitu

1. *Insertion* atau insersi, yaitu menginsersi sebuah huruf ke dalam kata

$$\text{Ongkos} = D(i , j) = D(i, j - 1) + 1$$

2. *Deletion* atau penghapusan, yaitu menghapus sebuah huruf dari kata

$$\text{Ongkos} = D(i, j) = D(i - 1, j) + 1$$

3. *Substitution* atau substitusi, yaitu mengganti sebuah huruf dalam kata menjadi huruf lainnya

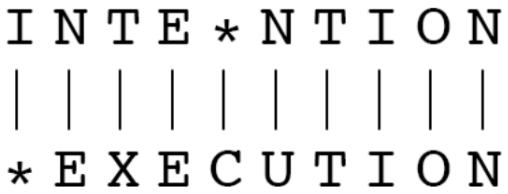
$$\text{Ongkos} = D(i, j) = D(i - 1, j - 1) + 1$$

Atau dalam algoritma Levenshtein

$$\text{Ongkos} = D(i, j) = D(i - 1, j - 1) + 2$$

Berikut adalah contoh kata yang akan kita proses dalam makalah ini, yaitu *Intention* dan *Execution*. Dimana tujuannya

disini adalah untuk mengubah kata *Intention* menjadi *Execution*.



Gambar 4 : Salah satu cara untuk kata *Intention* dan *Execution* disejajarkan[4]

Dapat dilihat dari Gambar 4 bahwa perubahan yang terjadi ada lima, yaitu menghapus huruf I, merubah huruf N menjadi huruf E, merubah huruf T menjadi huruf X, menginsersi huruf C setelah huruf E, dan mengubah huruf N menjadi huruf U dalam kata *Intention*. Sehingga ongkos yang diperlukan adalah lima. Namun, dalam algoritma *Minimum Edit Distance* Levenshtein, ongkos total yang diperlukan adalah delapan.

Mengapa algoritma *Minimum Edit Distance* menghasilkan dua ongkos berbeda? Dalam algoritma Levenshtein, substitusi memakan ongkos dua. Sedangkan dalam algoritma biasa hanya menambahkan satu.

Untuk mengomputasikan hal ini, akan dilakukan dengan menggunaka strategi algoritma program dinamis. Dimana perhitungan setiap D(i, j) dilakukan dalam bentuk tabular. Setiap huruf akan ditempatkan pada sel-sel berbeda. Dengan kata awal sebagai sumbu y, dan kata tujuan sebagai sumbu x.

Berikut adalah algoritma *Minimum Edit Distance* Levenshtein yang akan dibahas dalam makalah ini,

```
Inisialisasi
    D(i , 0) = i;
    D(0 , j) = j;

Rekurens
    for each i = 1 to n
        for each j = 1 to m
            D(i , j) = min (D(i - 1 , j) + 1,

                       D(i , j - 1 ) + 1,

                       if x[i] != y[j]

                            D(i - 1 , j - 1) + 2;

                       else //x[i] = y[j]

                            D(i - 1 , j - 1) + 0;
                       );
Terminasi
    D(n, m) adalah Minimum Edit Distance;
```

Dalam algoritma ini, akan dilakukan pendekatan mundur sehingga tabel yang dihasilkan di bagian inisialisasi adalah sebagai berikut,

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Gambar 5 : Tabel Algoritma *Minimum Edit Distance* saat insialisasi[5]

Dapat diliha bahwa baris dan kolom sudah diisi ongkos 0 hingga 9. Yang dilakukan disini adalah menyamakan kondisi *string* dalam sel tersebut. Contohnya tanda pagar menandakan *string* kosong, sehingga D(0, 0) merupakan perbandingan antara *string* kosong dengan *string* kosong sehingga perubahan yang terjadi 0. D(0, 1) merupakan perbanding *string* kosong dengan huruf E. Maka harus dilakukan satu perubahan, yaitu menginsersi huruf E, sehingga sel tersebut mempunyai ongkos 1. Semua itu dilakukan secara terus menerus hingga tahap inisialisasi selesai.

Lalu, dalam bagian rekurensnya, akan dilakukan pengisian tabel dengan perhitungan sesuai algoritmanya, dimana dicari nilai terkecil antara nilai sel di kiri tambah 1, nilai sel di bawah tambah 1, dan nilai sel diagonal kiri bawah ditambah 0 (jika hurufnya sama) atau ditambah dengan 2 (jika hurufnya berbeda). Pengisian dilakukan secara terus menerus hingga tabel penuh.

Contohnya pada sel D(2, 2) membandingkan huruf I dan huruf E sehingga,

D(1, 1) = min (D(0, 1) + 1, D(1, 0) + 1, atau D(0, 0) + 2))

D(1, 1) = min((1 + 1), (1 + 1), (0 + 2))

D(1, 1) = min (2, 2, 2)

dimana semuanya menghasilkan nilai 2 sehingga,

D(1, 1) = 2

Berikut adalah hasil tabel ketika sudah terisi penuh,

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Gambar 6 : Tabel Algoritma *Minimum Edit Distance* setelah terisi penuh [6]

Sehingga hasil perubahan yang terhitung sesuai Algoritma *Minimum Edit Distance* Levenshtein adalah D(9, 9), yaitu bernilai 8. Perubahan yang terjadi bisa insersi, penghapusan, atau substitusi. Semua ini dapat dicari tahu dengan ditambahkan *backtrace* pada algoritma *Minimum Edit Distance.*

B. Backtrace untuk Algoritma Minimum Edit Distance

Untuk menentukan langkah jenis perubahan yang harus dilakukan untuk mentransformasikan suatu kata menjadi kata lain, harus dilakukan penambahan *backtrace* pada algoritma *Minimum Edit Distance*. Hal yang dimaksud adalah setiap menambahkan nilai baru pada sel table perhitungan ongkos, harus diingat juga jalurnya (nilai minimum yang diambil dari sel kiri, sel bawah, atau sel diagonal kiri bawah). *Backtrace* ini dalam bentuk *pointer*. Sehingga, ketika perhitungan telah selesai, dapat digunakan *pointer* pada sel D(*n*, *m*) untuk melihat kembali perubahan yang dilakukan, serta huruf apa pada kata *x* yang sejajar dengan suatu huruf pada kata *y*.

Sehingga, algoritma *Minimum Edit Distance* menjadi

```
Inisialisasi
    D(i , 0) = i;
    D(0 , j) = j;

Rekurens
    for each i = 1 to n
        for each j = 1 to m
    D(i , j) = min (D(i - 1 , j) + 1, //delesi
                    D(i , j - 1 ) + 1, //insersi
                    if x[i] != y[j] //substitusi
                        D(i - 1 , j - 1) + 2;
                    else //x[i] = y[j], sama
                        D(i - 1 , j - 1) + 0;
                    );
    Ptr(i, j) = if down then
                    insersi;
                if left then
                    delesi;
                if diagonal then
                    if (D[i][j] = D[i – 1][j – 1] + 2) then
                        substitusi;
                    if (D[i][j] = D[i – 1][j – 1]) then
                        sama;

Terminasi
    D(n, m) adalah Minimum Edit Distance;
```

Sehingga, tabel pada Gambar 6 menjadi sebagai berikut,

n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↙ **8**
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↙ **8**	← 9
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↙ **8**	← 9	← 10
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙ **8**	← 9	← 10	←↓ 11
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ **8**	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10
e	4	↙ 3	← 4	↙← **5**	← **6**	← 7	←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9
t	3	↙←↓ 4	↙←↓ **5**	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙ 7	←↓ 8	↙←↓ 9	↓ 8
n	2	↙←↓ **3**	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↙ 7
i	**1**	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙ 6	← 7	← 8
#	**0**	1	2	3	4	5	6	7	8	9
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**

Gambar 7 : Tabel Algoritma *Minimum Edit Distance* dengan *Backtrace*[7]

Dari tabel pada Gambar 7, dapat dilihat bahwa kini sudah jelas huruf pada *string x* sejajar dengan huruf apa pada *string y*, yaitu seperti pada Gambar 4. Selain itu, sudah jelas juga perubahan-perubahan apa saja yang dilakukan pada kata *x* untuk ditransformasikan menjadi kata *y*. Dengan begitu, fitur *Spell Check* dapat memroses kata-kata apa saja yang tepat untuk memperbaiki ejaan kata *x* dan mengusulkannya kepada pengguna.

C. Weighted Minimum Edit Distance

Ada banyak alasan kenapa ejaan yang ditulis pengguna bisa salah. Alasan yang paling umum adalah karena letak huruf dalam *keyboard*, atau karena kesalahan mengeja, khususnya di huruf vokal. Sehingga, terdapat sebuah tabel *weight* dari perbedaan huruf yaitu,

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X \ Y (correct)	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	1	0	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

Gambar 8 : Tabel beban perbedaan huruf[8]

Beban tersebut akan digunakan pada algoritma *Weighted Minimum Edit Distance* yaitu,

Initialization:

```
D(0,0) = 0
D(i,0) = D(i-1,0) + del[x(i)];     1 < i ≤ N
D(0,j) = D(0,j-1) + ins[y(j)];     1 < j ≤ M
```

Recurrence Relation:

```
D(i,j)= min { D(i-1,j)   + del[x(i)]
            { D(i,j-1)   + ins[y(j)]
            { D(i-1,j-1) + sub[x(i),y(j)]
```

Termination:

```
D(N,M) is distance
```

Gambar 9 : Algoritma *Weighted Minimum Edit Distance*[9]

Sehingga, perhitungan ongkos yang dilakukan akan menghasilkan hasil yang berbeda dengan perhitungan-perhitungan pada Gambar 6 dan Gambar 7. Dimana semakin sering ketidakcocokan antara kedua huruf terjadi, semakin besar nilai bebannya, maka semakin besar ongkosnya.

IV. Kesimpulan

Spell Check merupakan salah satu fitur dalam perangkat lunak pengolah kata yang algoritmanya menerapkan strategi algoritma program dinamis agar tidak asal memberikan usulan kata kepada pengguna. Algoritma yang digunakan adalah algoritma *Minimum Edit Distance* yang mencari nilai ongkos perubahan terkecil untuk merubah suatu kata dengan ejaan yang salah menjadi kata dengan ejaan yang benar serta perubahan yang dilakukan harus di huruf mana saja dalam kata tersebut.

Menggunakan prinsip strategi algoritma program dinamis, algoritma ini menggunakan rekurens dan mencari solusi paling optimal dengan membuat keputusan untuk mengisi nilai sel dengan nilai minimum pada setiap tahapnya. Nilai tersebut akan menunjukkan perubahan yang harus dilakukan adalah insersi, atau penghapusan, atau substitusi pada huruf tersebut.

Dengan adanya *backtrace* pada algoritma *Minimum Edit Distance*, dapat diketahui huruf pada kata yang mempunyai ejaan salah sejajar dengan huruf apa pada kata dengan ejaan yang benar. Sedangkan, *Weighted Minimum Edit Distance* menghasilkan perhitungan dengan memperhitungkan seberapa sering perbedaan huruf tersebut terjadi sehingga hasil perhitungan lebih sesuai lagi dengan kebiasaan salah eja pengguna.

Sehingga, dengan strategi algoritma dinamis, perhitungan yang harus dilakukan pada *Spell Check* dapat dilakukan dengan lebih efisien dan mudah. Serta menghasilkan hasil yang optimal.

Ucapan Terima Kasih

Pada kesempatan ini, Penulis mengucapkan terima kasih kepada Tuhan YME yang telah memberi kesempatan dan kemampuan agar Penulis dapat belajar dan menuliskan makalah ini. Penulis juga ingin berterima kasih kepada kedua orangtua Penulis serta teman-teman

Penulis. Selain itu, Penulis juga mengucapkan terima kasih sebesar-besarnya untuk Ibu Dr.Nur Ulfa Maulidevi, ST., M.Sc.. dan Bapak Dr.Ir. Rinaldi Munir, MT. selaku dosen pengajar mata kuliah Strategi Algoritma yang telah mengajar dan membimbing Penulis semester ini. Terakhir, Penulis ingin berterima kasih kepada siapapun yang telah membantu dalam penulisan makalah ini..

Daftar Pustaka

[1] http://www.checkyourtext.com/img/spell-check.gif, diakses pada 8 Mei 2016.

[2] http://astarmathsandphysics.com/a-level-maths/D2/introduction-to-dynamic-programming-html-m3434a800.gif, diakses pada 8 Mei 2016.

[3] https://www.cs.berkeley.edu/~vazirani/algorithms/chap6.pdf, halaman 174, diakses pada 8 Mei 2016.

[4] https://web.stanford.edu/class/cs124/lec/med.pdf, halaman 4, diakses pada 8 Mei 2016.

[5] https://web.stanford.edu/class/cs124/lec/med.pdf, halaman 15, diakses pada 8 Mei 2016.

[6] https://web.stanford.edu/class/cs124/lec/med.pdf, halaman 18, diakses pada 8 Mei 2016.

[7] https://web.stanford.edu/class/cs124/lec/med.pdf, halaman 23, diakses pada 8 Mei 2016.

[8] https://web.stanford.edu/class/cs124/lec/med.pdf, halaman 31, diakses pada 8 Mei 2016.

[9] https://web.stanford.edu/class/cs124/lec/med.pdf, halaman 33, diakses pada 8 Mei 2016.

[10] Munir, Rinaldi. 2009. *Strategi Algoritma*. Bandung: Penerbit Informatika Bandung.

[11] https://www.cs.berkeley.edu/~vazirani/algorithms/chap6.pdf, diakses pada 7 Mei 2016.

[12] Blank, Doug. 2012. *Spelling Checking Algorithms*. http://cs.brynmawr.edu/Courses/cs330/spring2012/SpellingCheckers.pdf , diakses pada 7 Mei 2016.

[13] Khuram, Ali. 2013. *Dynamic Programming – Edit Distance*. https://alikhuram.wordpress.com/2013/04/27/dynamic-programming-edit-distance/, diakses pada 7 Mei 2016.

[14] https://web.stanford.edu/class/cs124/lec/med.pdf, diakses pada 7 Mei 2016.

[15] Dumitru. *Dynamic Programming – From Novice to Advanced*. https://www.topcoder.com/community/data-science/data-science-tutorials/dynamic-programming-from-novice-to-advanced/, diakses pada 8 Mei 2016.

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, 8 Mei 2016

Sekar Anglila Hapsari / 13514069