Saintek Vol 5, No 2 Tahun 2010

ANALISIS *EDIT DISTANCE* MENGGUNAKAN ALGORITMA *DYNAMIC PROGRAMMING*

Arip Mulyanto

Fakultas Teknik Universita Negeri Gorontalo

Abstract

Edit distance merupakan jumlah minimum *point mutation* yang diperlukan untuk merubah suatu string ke string yang lain. *Point mutation* tersebut adalah mengganti, menambah dan menghapus sebuah karakter. Konsep *edit distance* banyak digunakan dalam proses manipulasi data berbagai aplikasi komputer.

Berbagai algoritma dapat digunakan dalam pencarian dan penentuan *edit distance*. Dalam penelitian ini, dilakukan eksperimen untuk mencari *edit distance* menggunakan algoritma *dynamic programming*. Sedangkan proses perhitungannya dibantu dengan fungsi-fungsi yang ada dalam Microsoft Excel.

Hasil eksperimen menunjukkan kompleksitas waktu (*time complexity*) untuk algoritma *dynamic programming* adalah O(|s1|*|s2|). Jika s1 dan s2 mempunyai panjang yang hampir sama, katakanlah "n", maka kompleksitas waktunya adalah $O(n^2)$. Kompleksitas ruang (*space-complexity*) untuk algoritma ini juga $O(n^2)$, karena keseluruhan dari matriks digunakan untuk menemukan solusi yang optimal.

Kata kunci : *edit distance, dynamic programming,* kompleksitas.

Pendahuluan

Edit distance dalam *computer science* dikenal dengan nama *Levenshtein distance*. Nama tersebut diambil dari nama penemunya, yaitu **Vladimir Levinshtein**. *Levinshtein distance* ditemukan pada tahun 1965. Sejalan dengan waktu *Levinstein distance* lebih dikenal dengan *edit distance. Edit distance* merupakan jumlah minimum *point mutation* yang diperlukan untuk merubah suatu string ke string yang lain. *Point mutation* tersebut adalah mengganti, menambah dan menghapus sebuah karakter.

Edit distance dari dua string s1 dan s2 adalah jumlah minimum dari *point mutations* yang diperlukan untuk merubah s1 menjadi s2. *Point mutations* merupakan salah satu dari kegiatan berikut.

1. Mengganti karakter
   Untuk string 'commuter' dan 'computer', dengan merubah huruf m di string 1 menjadi huruf p maka string 1 akan sama dengan string 2.
2. Menambah karakter
   Sebagai contoh, string 'sort' dengan 'sport'. Dengan menambahkan huruf p di string 1, maka string 1 sama dengan string 2.
3. Menghapus karakter
   Sebaliknya, dengan menghilangkan huruf p di string 'sport', maka string tersebut akan menjadi string 'sort'.

Edit distance merupakan jumlah minimum dari penggantian, penambahan ataupun penghapusan karakter atau huruf di string 1, sehingga string 1 akan berubah menjadi string kedua. Dalam contoh di atas, *edit distance* bernilai 1.

Edit distance mempunyai banyak kegunaan. Beberapa aplikasi yang menggunakan konsep *edit distance* adalah *file revision, remote screen update problem, spelling correction, plagiarism detection, molecular biology* dan *speech recognition.*

Pencarian *edit distance* ini dapat dilakukan dengan menggunakan beberapa algoritma, diantaranya adalah algoritma *dynamic programming* dan algoritma *Hirschberg*. Dalam penelitian ini, penulis menggunakan algoritma *dynamic programming*. *Dynamic programming* merupakan suatu algoritma yang membagi problem menjadi sub-sub problem dimana solusi yang optimal dapat dicari dari sub problem tersebut. Sub-sub problem dipecahkan dan hasilnya kemudian disimpan dalam bentuk tabel. Penggunaan tabel dimana tiap-tiap sel berisi solusi yang dihitung berdasarkan algoritma tersebut, mengingatkan kita akan sel-sel yang ada di Microsoft Excel. Sehingga penulis mencoba menyelesaikan masalah *edit distance* dimana perhitungannya memanfaatkan fungsi-fungsi yang ada di Microsoft Excel.

Metode Penelitian

Penelitian dilakukan dengan cara eksperimen untuk mencari dan menentukan *edit distance* terhadap beberapa kasus dengan menggunakan algoritma *dynamic programming*. Algoritma tersebut diimplementasikan menggunakan fungsi-fungsi yang ada dalam program aplikasi Microsoft Excel.

Hasil

Permasalahan

Perubahan dari string satu ke suatu string yang lain yang hampir sama mempunyai bermacam-macam cara. Terkadang, karena salah memilih operasi yang dilakukan, pekerjaan tersebut bisa dilakukan secara berulang-ulang sehingga mengakibatkan tidak efisien. Sebagai contoh : string ‘SPORT’ dan ‘SORT’. Ada beberapa alternatif untuk merubah ‘SPORT’ dan ‘SORT’, diantaranya:

Alternatif I.

String 1	S	P	O	R	T
	↓ Huruf P dihilangkan				
String 2	S	-	O	R	T

Alternatif II.

String 1	S	P	O	R	T
	↓ P diganti O		↓ O dihilangkan		
String 2	S	O	-	R	T

Alternatif III.

String 1	S	P	O	R	T	
	↓ P diganti O		↓ O diganti R	↓ R diganti T	↓ T dihilangkan	
String 2	S	O	R	T	-	

Perubahan dari string 1 ke string 2 untuk alternatif I adalah 1 langkah, alternatif II adalah 2 langkah dan alternatif III adalah 4 langkah. *Edit distance* adalah jumlah minimum dari perubahan string 1 menjadi string 2. Dalam contoh di atas, *edit distance* d(s1,s2) = 1.

Edit distance dari dua string s1 dan s2 yang dinotasikan dengan d(s1,s2) secara rekursif dapat didefinisikan sebagai berikut :

1. d(‘ ’,’ ‘) = 0 # tanda ‘ ‘ adalah simbol untuk string kosong
2. d(s,’ ‘) = d(‘ ‘,s) =|s| # |s| artinya panjang dari string s
3. d(s1+ch1,s2+ch2) = min (d(s1,s2) + if ch1=ch2 then 0 else 1,

   d(s1+ch1,s2)+1,

   d(s1, s2+ch2)+1)

Aturan 1 dan 2 sudah jelas benar. Untuk aturan yang ketiga, karena kedua string tidak kosong, maka masing-masing akan mempunyai karakter terakhir. Disini karakter tersebut dalam mengedit string 1 menjadi string 2. Jika ch1 = ch2, maka tidak akan ada pinalty (karakter tersebut tidak perlu diedit). Dengan kata lain, *edit distance* tetap d(s1,s2).

Tetapi sebaliknya, jika ch1 ≠ ch2, ch1 dapat diganti menjadi ch2 sehingga costnya d(s1,s2)+1. Kemungkinan yang lain adalah menghapus ch1 dan mengedit s1 menjadi s2+ch2. Dalam hal ini *edit distance* yang diperhitungkan adalah d(s1, s2+ch2)+1. Kemungkinan yang terakhir adalah menambahkan ch2 ke s1, sehingga perhitungan *edit distance* menjadi d(s1+ch1,s2)+1. Karena *edit distance* merupakan jumlah minimum untuk merubah string1 menjadi string 2, maka yang dicari adalah nilai minimum dari ketiga kemungkinan di atas.

Algoritma

Dynamic programming, seperti metode *devide and conquer* memecahkan masalah dengan mengkombinasikan solusi dari sub-sub problem. *Programming* di sini mengacu pada tabel, bukan berarti menulis program komputer. Algoritma *dynamic programming* dapat digunakan jika sub-sub problem tidak *independent*, artinya sub problem tersebut mempunyai irisan dengan sub problem lain. Pemecahan setiap sub-sub problem hanya dilakukan sekali dan kemudian disimpan dalam bentuk tabel. Hal ini dilakukan untuk menghindari *recomputing* jawaban setiap kali subproblem tadi di panggil. *Dynamic programming* biasanya diaplikasikan untuk problem optimasi, dimana problem-problem tersebut solusinya mempunyai banyak kemungkinan. Setiap solusi mempunyai nilai, namun diharapkan dapat ditemukan solusi yang optimal. Dengan melihat karakteristik dari algoritma *dynamic programming*, maka algoritma ini akan cocok jika digunakan untuk memecahkan masalah *edit distance*.

Pada dasarnya algoritma *dynamic programming* dapat dibagi menjadi 4 tahap, yaitu:

1. Mendefinisikan karakteristik struktur solusi yang optimal
2. Secara rekursif mendefinisikan nilai dari solusi yang optimal
3. Menghitung nilai dari solusi yang optimal secara bottom-up
4. Menyusun solusi optimal dari informasi yang telah dihitung

Ditinjau dari relasi yang ada pada problem *edit distance*, pada dasarnya d(s1,s2) tergantung hanya pada d(s1’,s2’) dimana s1’ lebih pendek dari s1 dan/atau s2’ lebih pendek dari s2. Berikut teknik *dynamic programming* yang digunakan :

Matriks dua dimensi m[0..|s1|,0..|s2|] adalah nilai dari *edit distance*

m[i,j] = d(s1[1..i], s2[1..j])

m[0,0] = 0

m[i,0] = i, i=1..|s1|

m[0,j] = j, j=1..|s2|

m[i,j] = min(m[i-1,j-1]+ if s1[i]=s2[j] then 0 else 1 fi,

m[i, j-1] + 1,

m[i-1, j] + 1), i=1..|s1|, j=1..|s2|

Dalam bentuk *pseudocode*, algoritma tersebut dapat dituliskan sebagai berikut:

```
EDIT-DISTANCE-LENGTH(s1,s2)
1.  m ← length [s1]
2.  n ← length [s2]
3.  m[0,0] = 0
4.  for i from 1 to m
5.    do m[i, 0] ← i
6.  for j from 1 to n
7.    do m[0, j] ← j
8.  for i from 1 to m
9.    do for j from 1 to n
10.   do if s1[i] = s2[j] then cost ← 0
11.                 else cost ← 1
12.   m[i, j] ← minimum(
13.                 m[i-1, j-1] + cost,   // substitution
14.                 m[i, j-1] + 1,     // insertion
15.                 m[i-1, j] + 1     // deletion
16.                 )
17. return m[m, n]
```

m[,] dapat dikomputasikan baris per baris. Baris m[i,] tergantung pada baris m[i-1,]. Jika dilihat dari tabel *edit distance* di contoh mencari *edit distance* dari string ‘AND’ dan string ‘SAD’, maka tiap sel per barisnya memang hanya tergantung pada baris sebelumnya dan sel disebelahnya.

Perhatikan tabel 1, misalkan pilih sel m[1,2].

m[1,2] = min (m[0,1] + 0, # karena kedua string menambah huruf yang sama yaitu A, maka ditambah dengan 0
m[1,1] + 1, m[0,2] + 1)

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

Contoh lain, misalkan sel m[2,3].

m[2,3] =min (m[1,2] + 1, # karena N ≠ D, maka ditambah dengan 1

m[2,2] + 1, m[1,3] + 1)

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

Tabel 1. Matriks *edit distance* string ‘AND’ dan ‘SAD’

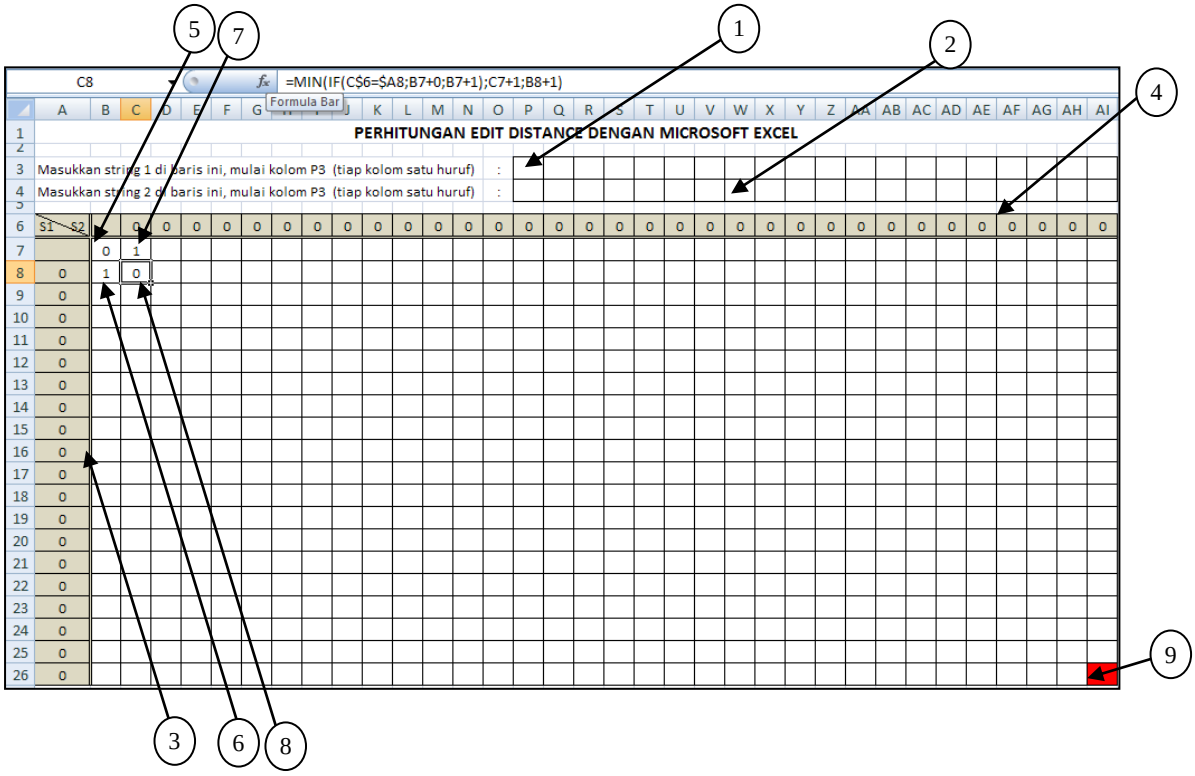
S1 \ S2		‘ ‘	S	A	D
		j = 0	j = 1	j = 2	j = 3
‘ ‘	i = 0	0	1	2	3
A	i = 1	1	1	1	2
N	i = 2	2	2	2	2
D	i = 3	3	3	3	2

m[1,2] = min (1+0, 1+1, 2+1)
= 1

m[2,3] = min (1+1, 2+1, 2+1)
= 2

Hasil Eksperimen

Dalam eksperimen ini, penulis membuat perhitungan *edit distance* dengan menggunakan fungsi-fungsi yang ada di Microsoft Excel berdasarkan algoritma *dynamic programming* yang sudah dijelaskan di bagian sebelumnya. Berikut adalah design program excel untuk menghitung *edit distance*.



Gambar 1. Design program Excel untuk menghitung *edit distance*

Penjelasan:

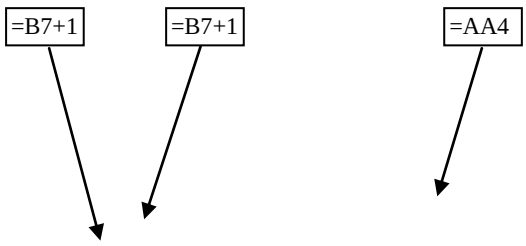
1. *User* akan memasukkan string 1 di baris ke 3 mulai kolom P dimana masing-masing kolom hanya diisi oleh 1 karakter (dari sel P3, Q3, R3, ... dst)

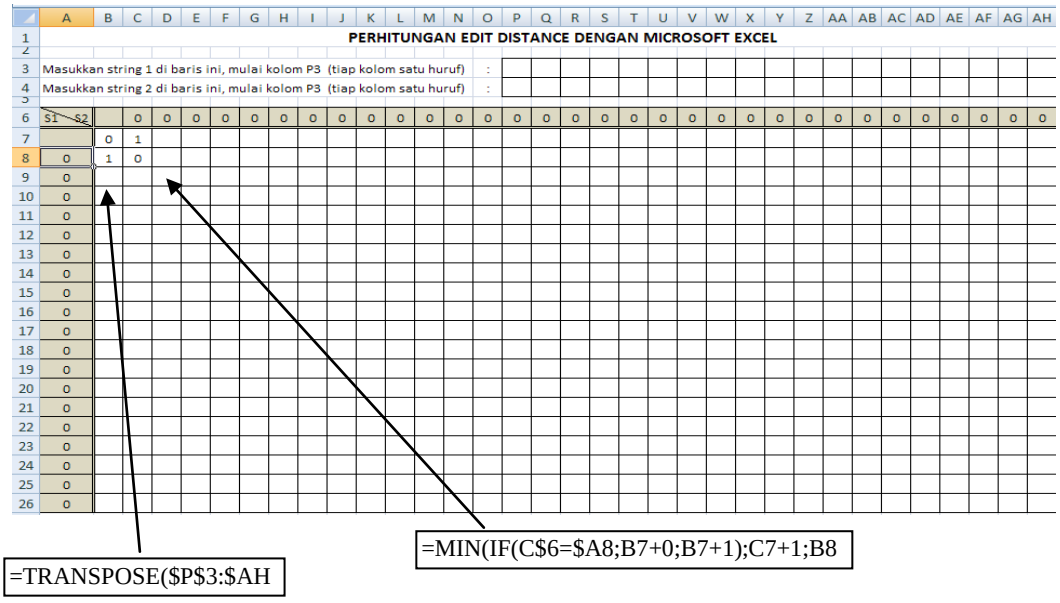
2. *User* memasukkan string 2 di baris ke 4 mulai kolom P dimana masing-masing kolom hanya diisi oleh 1 karakter (dari sel P4, Q4, R4, ... dst)
3. Sel A7 berisi string kosong, sedangkan sel A8 sampai A26 berisi string 1 seperti yang dituliskan *user*. Proses penulisan ini dapat dilakukan secara otomatis dengan cara mengeset sel A8 sampai A26 dengan menggunakan fungsi transpose dari array ($P$3:$AH$3).
4. Sel B6 berisi sel kosong, sedang sel C6 sampai sel AI6 berisi string 2 yang diisikan *user*. Penulisan ini juga dilakukan secara otomatis, dengan cara :
   a. Ketik '=P4' di sel C6
   b. Copy sel C6 ke sel D6 sampai sel AI6
5. Sel di B7 diketik dengan 0. Hal ini sesuai dengan algoritma *dynamic programming* dibaris ke 3 bahwa m[0,0] = 0 (*edit distance* untuk dua string yang kosong = 0).
6. Sel B8 merupakan implementasi dari algoritma *dynamic programming* baris ke 5. Karena i = 1 dimulai di B8 dan tiap baris di bawahnya, i berjalan dengan penambahan 1 angka, maka di sel B8 dapat dituliskan rumus '=B7+1'. Kemudian nilai ini dicopy ke sel B9 sampai B26.
7. Analog dengan 6, sel C7 merupakan implementasi dari algoritma *dynamic programming* baris ke 7. Karena j = 1 dimulai di C7 dan tiap kolom sesudahnya j berjalan dengan penambahan 1 angka, maka di sel tersebut dapat dituliskan rumus '=B7+1'. Kemudian nilai ini dicopy ke sel B9 sampai B26.
8. Sel C8 merupakan implementasi dari algoritma *dynamic programming* baris 10 sampai baris 16. Dalam program Excel, isi algoritma tersebut dapat ditransfer dalam bentuk rumusan seperti yang terlihat di formula bar, yaitu :

   '=MIN(IF(C$6=$A8;B7+0;B7+1);C7+1;B8+1)'

   Nilai ini kemudian dicopykan ke seluruh sel, yaitu range C8 : AI26, kecuali sel C8 itu sendiri. Proses mengcopynya dapat dilakukan dengan 2 cara, yaitu :
   a. Copy sel C8 ke sel-sel di sampingnya terlebih dahulu yaitu sel D8 sampai AI8. Kemudian dari sel C8 sampai AI8 dicopy ke bawah untuk range C9 : AI26.
   b. Copy sel C8 ke sel-sel di bawahnya terlebih dahulu, yaitu sel C9 sampai sel C26. Kemudian sel C8 sampai C26 dicopy ke samping untuk range D8 : AI26.

   Kedua cara ini akan menghasilkan nilai yang sama. Yang terpenting dan perlu diingat, bahwa baris ke i tergantung dari baris sebelumnya, seperti yang sudah dijelaskan di atas. Jadi untuk suatu kolom tertentu, jika ingin mengisi suatu baris, misal baris ke 10 maka baris ke 9 sudah harus terisi. Baris ke 9 tergantung dari baris 8, maka baris ke 8 harus diisi terlebih dulu. Demikian seterusnya, sehingga untuk mengisi baris ke 10 maka baris ke 1 sampai 9 harus sudah terisi.
9. Sel yang berwarna merah (sel AI26) merupakan sel yang menunjukkan nilai *edit distance* untuk string s1 yang panjangnya sampai baris ke 26 dan string s2 yang panjangnya sampai kolom AI. Nilai *edit distance* tentunya tidak selalu berada di sel AI26, tapi tergantung dari panjang string s1 dan s2. Jadi jika panjang string s1 ada di baris Y dan panjang string 2 ada di kolom X, maka *edit distance* kedua string tersebut ada di sel XY. Jika di lihat tiap selnya, maka rumusnya akan tampak seperti gambar dibawah ini.



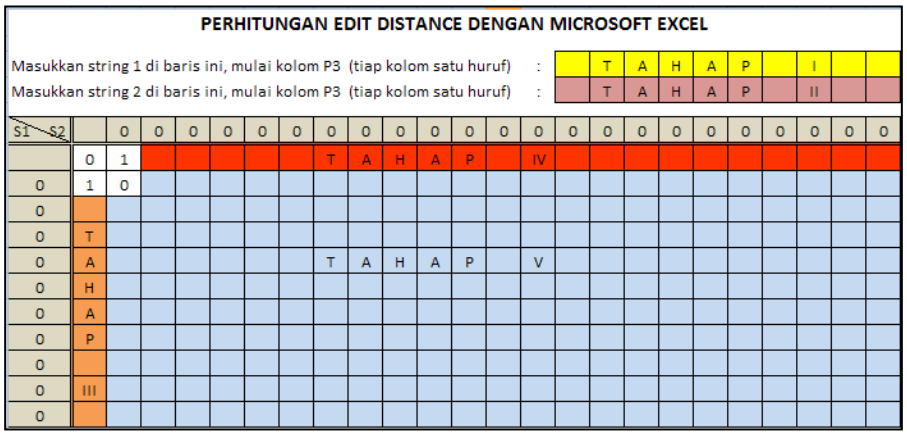


Gambar 2. Rumus di tiap bagian sel

Tahapan yang harus dilakukan *user* untuk mencari *edit distance* dengan program Excel adalah sebagai berikut :

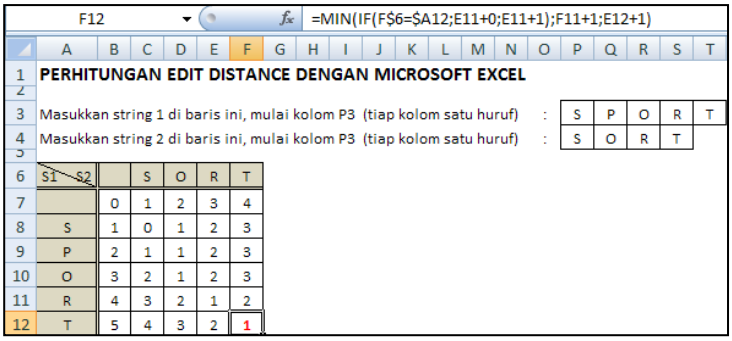
1. Memasukkan string 1 ke kotak yang tersedia, satu kolom diisi satu karakter;
2. Memasukkan string 2 ke kotak yang tersedia, satu kolom juga disi satu karakter;
3. Mengcopy sel B8 ke sel B9 sampai panjang string s1;
4. Mengcopy sel C7 ke sel D7 sampai panjang string s2;
5. Mengcopy sel C8 ke range C8 sampai sel dimana barisnya sama dengan panjang string s1 dan kolomnya sama dengan panjang string s2, terkecuali sel C8 itu sendiri.

Warna-warna dalam program Excel berikut menunjukkan tiap langkah yang harus dilakukan *user*.



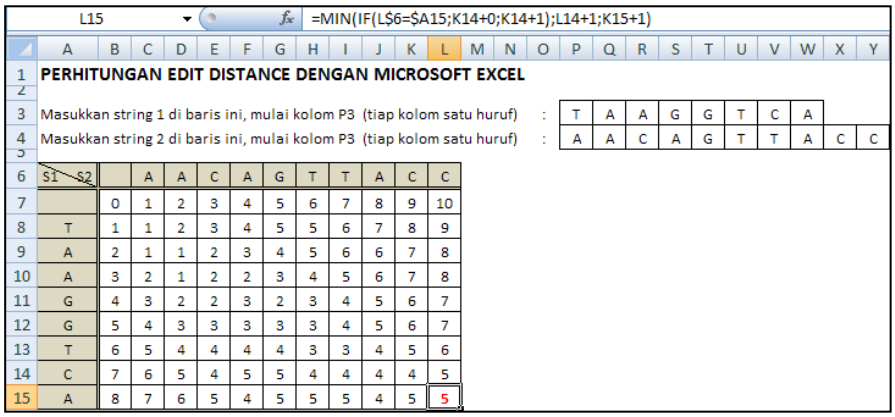
Gambar 3. Tahapan yang harus dilakukan oleh *user*

Berikut beberapa contoh yang di ujicobakan dalam program Excel yang telah dibuat.

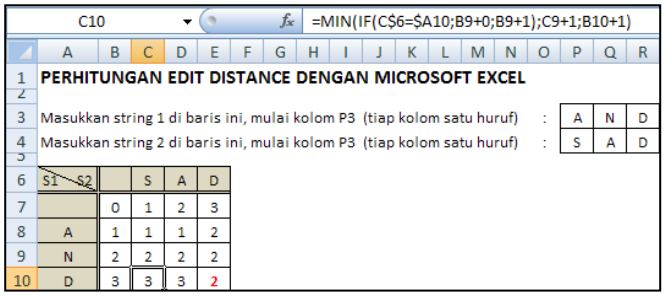


Gambar 4. *Edit distance* string ‘SPORT’ dengan ‘SORT’

Nilai *edit distance* dapat dilihat di kotak pojok kanan bawah yang berwarna merah. Dengan perhitungan program Excel, ternyata *edit distance* string ‘SPORT’ dan ‘SORT’ adalah 1, sama jika dihitung secara manual.



Gambar 5. *Edit distance* string ‘TAAGGTCA’ dengan ‘AACAGTTACC’



Gambar 6. *Edit distance* string ‘AND’ dengan ‘SAD’

Terlihat isi matriks hasil program Excel sama dengan hasil jika dihitung secara manual (lihat sub bab Problem).

PERHITUNGAN EDIT DISTANCE DENGAN MICROSOFT EXCEL

Masukkan string 1 di baris ini, mulai kolom P (tiap kolom satu huruf)	:	A	P	P	R	O	P	R	I	A	T	E		M	E	A	N	I	N	G									
Masukkan string 2 di baris ini, mulai kolom P (tiap kolom satu huruf)	:	A	P	P	R	O	X	I	M	A	T	E		M	A	T	C	H	I	N	G								

S1 \ S2		A	P	P	R	O	X	I	M	A	T	E	O	M	A	T	C	H	I	N	G
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
R	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
O	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P	6	5	4	3	2	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	7	6	5	4	3	2	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I	8	7	6	5	4	3	3	2	3	4	5	6	7	8	9	10	11	12	12	13	14
A	9	8	7	6	5	4	4	3	3	3	4	5	6	7	8	9	10	11	12	13	14
T	10	9	8	7	6	5	5	4	4	4	3	4	5	6	7	8	9	10	11	12	13
E	11	10	9	8	7	6	6	5	5	5	4	3	4	5	6	7	8	9	10	11	12
O	12	11	10	9	8	7	7	6	6	6	5	4	3	4	5	6	7	8	9	10	11
M	13	12	11	10	9	8	8	7	6	7	6	5	4	3	4	5	6	7	8	9	10
E	14	13	12	11	10	9	9	8	7	7	7	6	5	4	4	5	6	7	8	9	10
A	15	14	13	12	11	10	10	9	8	7	8	7	6	5	4	5	6	7	8	9	10
N	16	15	14	13	12	11	11	10	9	8	8	8	7	6	5	5	6	7	8	8	9
I	17	16	15	14	13	12	12	11	10	9	9	9	8	7	6	6	6	7	7	8	9
N	18	17	16	15	14	13	13	12	11	10	10	10	9	8	7	7	7	7	8	7	8
G	19	18	17	16	15	14	14	13	12	11	11	11	10	9	8	8	8	8	8	8	7

Gambar 7. *Edit distance* string ‘APPROPRIATE MEANING’ dengan ‘APPROXIMATE MATHCING’

Untuk string yang sangat panjang

Jika string 1 atau string 2 panjangnya melebihi tampilan yang dibuat penulis, *user* bisa dengan mudah melakukan sedikit modifikasi pada sel inputan (string 1 di sel A8 sampai A26, string 2 di sel C6 sampai AI6). Caranya adalah sebagai berikut.

1. Modifikasi inputan string 1
   a. Hapus sel A8 sampai A26 untuk menghilangkan transpose yang sudah di set di range tersebut;
   b. Blok lagi sel A8 ke bawah sebanyak panjang string 1;
   c. Formulasikan fungsi transpose untuk array string 1 (sel P3 sampai di karakter terakhir) dengan insert function (f_x);
   d. Tekan F2, kemudian CTRL+SHIFT+Enter untuk mendapatkan formula dalam bentuk array. /+-7
2. Modifikasi inputan string 2
   Copy sel AI6 ke sel AJ6, AK6,... dan seterusnya sampai karakter terakhir dari string 2 dapat muncul di inputan string 2.
3. Lakukan proses pengcopyan seperti yang telah dijelaskan dalam tahap-tahap yang harus dilakukan *user* untuk mendapatkan nilai *edit distance* dengan program Excel.

Sebagai contoh untuk string panjang, s1 dan s2 didefinisikan sebagai berikut.
S1 = ‘ACCGGTCGAGTGCGCGGAAGCCGGCCGAA’
S2 = ‘GTCGTTCGGAATGCCGTTGCTCTGTAAA’
Sesudah dimasukkan ke program Excel, hasil *edit distance* kedua string tersebut adalah 14.

Pembahasan

Berdasarkan eksperimen di atas, program Excel memang sangat membantu untuk menghitung *edit distance*. Beberapa eksperimen yang telah dilakukan, ternyata hasil dari program Excel sama dengan hasil jika dihitung secara manual. Hal itu terlihat jelas saat isi matriks *edit distance* string ‘AND’ dan ‘SAD’ sama dengan matriks hasil dari program Excel.

Program ini akan sangat bermanfaat jika stringnya sangat panjang. Apalagi ditunjang dengan kapasitas jumlah sel di Microsoft Excel yang relatif banyak. Satu hal yang menjadi pertanyaan adalah bagaimana dengan kompleksitasnya?

Kompleksitas waktu untuk algoritma ini adalah O(|s1|*|s2|). Jika s1 dan s2 mempunyai panjang yang hampir sama, katakanlah "n", maka kompleksitas waktunya adalah $O(n^2)$. Sedangkan kompleksitas ruang (*space-complexity*) juga $O(n^2)$, karena keseluruhan dari matriks digunakan untuk menemukan solusi yang optimal. Sejalan dengan waktu, Hirschberg menunjukkan bahwa solusi optimal untuk *edit distance* dapat dicari dengan kompleksitas waktu O(|s1|*|s2|) dan kompleksitas ruangnya hanya O(|s2|) dengan menggunakan rekursif-biner (*binary-recursion*). Algoritma ini menggunakan paradigma *devide and conquer*, dan kemudian dikenal dengan nama *Hirschberg's algorithm*.

Ada beberapa algoritma lain yang lebih cepat dalam menyelesaikan persoalan *edit distance* ini. Beberapa algoritma tersebut cepat jika suatu kondisi terpenuhi, misalnya stringnya *similar* atau *dissimilar*. Ukkonen (1983) membuat sebuah algoritma dengan kompleksitas waktu untuk worst case adalah $O(n+d^2)$, dimana n adalah panjang string dan d adalah *edit distance*nya. Algoritma ini akan cepat untuk string yang similar dimana d nya kecil (d<<n).

Simpulan

Dari bahasan di atas dapat disimpulkan bahwa :

1. Microsoft Excel dapat digunakan sebagai tool untuk membantu menghitung *edit distance* dengan menggunakan algoritma *dynamic programming*.
2. Kompleksitas waktu algoritma *dynamic programming* adalah $O(n^2)$ dan telah diperbaiki oleh algoritma-algoritma lain yang kompleksitas waktunya lebih cepat.

Saran

Untuk lebih memahami tentang pencarian *edit distance*, dapat dilakukan dengan pembuktian dengan menggunakan algoritma lain selain *dynamic programming*.

DAFTAR PUSTAKA

D. S. Hirschberg. *A linear space algorithm for computing maximal common subsequences.* Comm. A.C.M. **18**(6) p341-343, 1975.

E. Ukkonen *On approximate string matching*. Proc. Int. Conf. on Foundations of Comp. Theory, Springer-Verlag, LNCS **158** p487-495, 1983.

L. Allison. *Hirschberg's Algorithm*. Faculty of Information Technology, Department of Computer Science, Monash University, Australia. 1999.

L. Allison. *Dynamic Programming Algorithm (DPA) for Edit Distance*. Faculty of Information Technology, Department of Computer Science, Monash University, Australia. 1999.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. Second Edition. The MIT Press, Cambridge, Massachusetts London, 2003.

V. I. Levenshtein. *Binary codes capable of correcting deletions, insertions and reversals*. Doklady Akademii Nauk SSSR **163**(4) p845-848, 1965.