Algoritma Program Dinamis *Edit distance* untuk Pengecekan Ejaan

Samsu Sempena

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha 10,Bandung, Jawa Barat
e-mail: if17088@students.if.itb.ac.id

ABSTRAK

Makalah ini membahas mengenai penerapan strategi algoritma untuk membantu pekerjaan manusia yaitu pengecekan ejaan. Fitur ini dapat kita temui pada perangkat lunak khusus untuk pengecekan ejaan maupun sebagai fitur yang terdapat pada aplikasi seperti *email client*, *word processor*, maupun *search engine*. Pada makalah ini pembahasan dibatasi pada pengecekan ejaan terhadap kata-kata yang salah eja saja dan tidak termasuk salah konteks.

Algoritma yang menjadi pokok bahasan merupakan kategori program dinamis (*dynamic programming*) yaitu *edit distance*. Algoritma *edit distance* dapat menghitung perbedaan antara dua string berdasarkan operasi sisip, ubah, dan hapus dengan memanfaatkan kelebihan program dinamis yaitu solusi dibangun dari solusi pada tahap sebelumnya sehingga kompleksitas yang didapat jauh lebih baik dibandingkan algoritma *brute force*. Algoritma *edit distance* memiliki banyak manfaat diantaranya untuk melakukan pengecekan ejaan,revisi file, deteksi plagiarisme, biologi molekular, dan pengenalan suara,bentuk,dsb. Dalam pengecekan ejaan, dibutuhkan sebuah basis data berisi kata-kata legal yang terdaftar untuk dibandingkan dengan kata-kata yang dimasukkan pengguna.

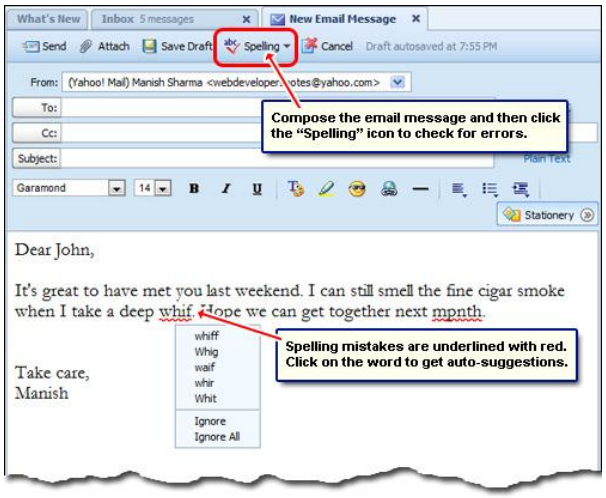
Pengembangan pengecekan ejaan lebih lanjut yaitu pengecekan terhadap ejaan berdasarkan konteks, yang menjadi bagian dari kajian NLP (*Natural Language Processing*).

Kata kunci: program dinamis, *edit distance*, deteksi, koreksi,ejaan

1. PENDAHULUAN

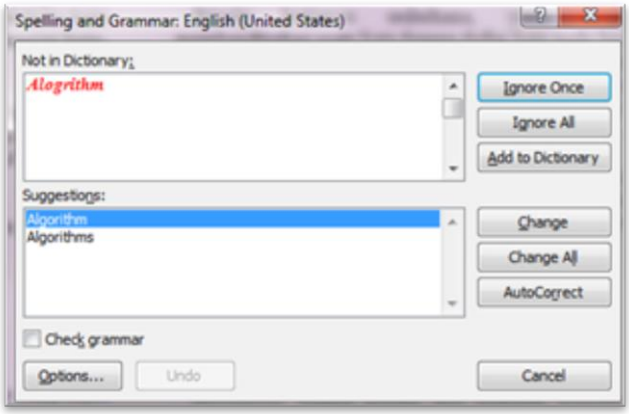
Pengecek ejaan adalah aplikasi komputer yang dapat menandai sebuah kata dalam dokumen yang kemungkinan tidak tereja dengan benar. Aplikasi ini pada umumnya dapat diatur untuk secara otomatis melakukan perubahan maupun dengan memunculkan pilihan perubahan yang dapat dipilih pengguna. Beberapa contoh penerapan pengecekan ejaan ini antara lain pada :

1. *Email Client*



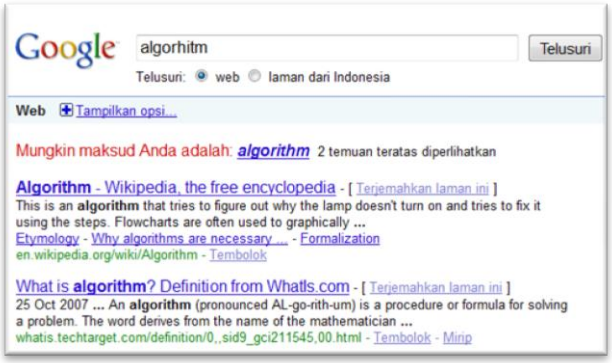
Gambar 1. Tampilan pengecekan ejaan pada *email client* Yahoo Mail

2. *Word Processor*



Gambar 2. Tampilan pengecekan ejaan pada *word processor* Microsoft Word

3. *Search Engine*



Gambar 3. Tampilan pengecekan ejaan pada *Google Search Engine*

Prinsip kerja pengecekan ejaan ini sebenarnya sederhana, yaitu dengan membandingkan suatu kata dengan daftar kata pada basis data kata-kata yang telah tereja dengan benar dan jika tidak terdapat kata yang tepat maka akan ditampilkan usulan kata-kata dengan perbedaan terkecil yang diukur dalam *edit distance*.

Edit distance antara 2 string merupakan jumlah langkah perubahan minimum untuk mengubah 1 string ke string lainnya dengan perubahan yang diijinkan sebagai berikut :

1. Mengubah sebuah huruf
   Contoh : algori**n**ma → algori**t**ma
   Penjelasan : huruf 'n' diganti dengan 't'
2. Menyisipkan sebuah huruf
   Contoh : algorima → algori**t**ma
   Penjelasan : huruf 't' disisipkan
3. Menghapus sebuah huruf
   Contoh : algoritma**n** →algoritma
   Penjelasan : huruf 'n' dihapus

String merupakan untaian dari karakter. Pada kasus pengecekan ejaan, string yang akan dibandingkan adalah kata yang dimasukkan pengguna dan kata yang telah tersimpan dalam kamus (basis data) kata.

2. METODE

2.1 Program Dinamis

Program Dinamis adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah atau tahapan sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Dalam Program Dinamis, sebuah persoalan dapat diselesaikan dengan metode yang memiliki ciri:

1. Terdapat sejumlah berhingga pilihan yang mungkin
2. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya
3. Menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan setiap tahap
4. Ongkos (*cost*) pada suatu tahap bergantung pada ongkos tahap-tahap sebelumnya dan meningkat seiring bertambahnya jumlah tahapan
5. Jika solusi total optimal, maka solusi pada tahap sebelumnya juga optimal

Misalkan $x_1,x_2,\ldots,x_n$ menyatakan peubah keputusan yang harus dibuat untuk tahap 1,2,…,n maka program dinamis memiliki 2 pendekatan :

1. Maju (*forward* atau *up-down*)
   Program dinamis bergerak mulai dari tahap 1, ke tahap 2,dan seterusnya sampai tahap ke-n. Runtutan peubah keputusan adalah $x_1,x_2,\ldots,x_n$.
2. Mundur (*backward* atau *bottom-up*)
   Program dinamis bergerak mulai dari tahap n, ke tahap n-1,dan seterusnya sampai tahap 1. Runtutan peubah keputusan adalah $x_n,x_{n-1},\ldots,x_1$.

2.2 Algoritma *Edit distance*

Algoritma *edit distance* dapat diimplementasikan dengan beberapa pendekatan diantaranya :

1. *Exhaustive search*
2. *Greedy*
3. *Dynamic Programming* (Program dinamis)

Misalkan D(s,t) sebagai *edit distance* antara string s dan t dengan |s| = m dan |t| = n.

Maka, algoritma *exhaustive search* untuk mencari *edit distance* antara 2 string dengan panjang m dan n akan memiliki kompleksitas $O(3^{m+n})$ yang merupakan kompleksitas eksponensial. Adapun algoritma *exhaustive search* yang digunakan memanfaatkan prinsip rekursif yang memiliki aturan rekurens sebagai berikut :

```
1.  D('','') = 0
2.  D(s,'') = D('',s) = |s|
3.  D(s+ch1,t+ch2) =
    Min(D(s,t) +if (ch1=ch2) then 0 else 1,
         D(s+ch1,t) + 1,
         D(s,t+ch2) + 1)
```

'' = string kosong

Sedangkan algoritma *greedy* belum tentu mendapatkan solusi optimal untuk *edit distance* (tingkat akurasinya rendah).

Program dinamis memperbaiki kelemahan-kelemahan tersebut. Adapun notasi algoritma untuk program dinamis ini sebagai berikut :

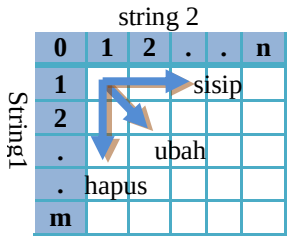
```
/*
Fungsi EditDistance akan menerima 2 buah string s
dan t,dan akan mengembalikan sebuah integer
sebagai nilai edit distance dari kedua string
tersebut.
*/

int EditDistance(string s, string t)
 {
   // d adalah sebuah array 2 dimensi (matriks)
   dengan m+1 baris and n+1 kolom
   declare int d[0..m, 0..n]

   // inisialisasi
   for i from 0 to m
     d[i, 0] := i              // penghapusan
   for j from 0 to n
     d[0, j] := j              // penyisipan

   for j from 1 to n
   {
     for i from 1 to m
     {
       if s[i] = t[j] then
         d[i, j] := d[i-1, j-1]
       else
         d[i, j] :=
              minimum
              (
                 d[i-1, j] + 1,  // hapus
                 d[i, j-1] + 1,  // sisip
                 d[i-1, j-1] + 1 // ubah
               )
     }
   }
   return d[m, n]
 }
```

Matriks nilai edit distance akan memiliki format sebagai berikut :



Misalkan kita ingin membandingkan kata “hujan” dengan “ujian”. Maka tahap-tahap pengisian matriks *edit distance* pada program dinamis adalah sebagai berikut :

Tabel 1. Inisialisasi matriks *edit distance*

		u	j	i	a	n
	0	1	2	3	4	5
h	1					
u	2					
j	3					
a	4					
n	5					

Tabel 2. Pengisian nilai *edit distance* untuk baris ke 1

		u	j	i	a	n
	0	1	2	3	4	5
h	1	1	2	3	4	5
u	2					
j	3					
a	4					
n	5					

Tabel 3. Pengisian nilai *edit distance* untuk baris ke 2

		u	j	i	a	n
	0	1	2	3	4	5
h	1	1	2	3	4	5
u	2	1	2	3	4	5
j	3	0	0	0	0	0
a	4	0	0	0	0	0
n	5	0	0	0	0	0

Tabel 4. Pengisian nilai *edit distance* untuk baris ke 3

		u	j	i	a	n
	0	1	2	3	4	5
h	1	1	2	3	4	5
u	2	1	2	3	4	5
j	3	2	1	2	3	4
a	4					
n	5					

Tabel 5. Pengisian nilai *edit distance* untuk baris ke 4

		u	j	i	a	n
	0	1	2	3	4	5
h	1	1	2	3	4	5
u	2	1	2	3	4	5
j	3	2	1	2	3	4
a	4	3	2	2	2	3
n	5					

Tabel 6. Pengisian nilai *edit distance* untuk baris ke 5

		u	j	i	a	n
	0	1	2	3	4	5
h	1	1	2	3	4	5
u	2	1	2	3	4	5
j	3	2	1	2	3	4
a	4	3	2	2	2	3
n	5	4	3	3	3	2

Setelah seluruh matriks *edit distance* terisi nilai,maka *edit distance* untuk perbandingan string "hujan" dan "ujian" terletak pada baris ke 5 dan kolom ke 5, yaitu 2.

Jika solusi optimal tersebut ditelusuri maka perubahan yang terjadi adalah :

		u	j	i	a	n
	0	1	2	3	4	5
h	1	1	2	3	4	5
u	2	1	2	3	4	5
j	3	2	1	2	3	4
a	4	3	2	2	2	3
n	5	4	3	3	3	2

```
huj-an
1  2
-ujian
```

Penjelasan : 'h' dihapus, dan 'i' disisipkan

Kompleksitas algoritma program dinamis *edit distance* adalah O(m.n), dapat dilihat dari perulangan 2 lapis pada algoritma *edit distance* masing-masing sebanyak m dan n kali.

2.3 Batas atas dan bawah *Edit Distance*

Algoritma *edit distance* untuk perbandingan kata dalam jumlah yang sangat banyak dapat dioptimasi dengan prinsip *branch and bound* dengan batas-batas nilai *edit distance* sebagai berikut :

1. paling kecil sebesar perbedaan panjang kedua string yang dibandingkan
2. paling besar sebesar panjang string yang lebih panjang
3. nilainya 0 jika dan hanya jika kedua string yang dibandingkan identik
4. jika kedua string memiliki panjang yang sama, maka jarak Hamming adalah batas atas dari nilai *edit distance*

2.4 Pengembangan *Edit distance*

Beberapa pengembangan yang dapat dilakukan pada algoritma *edit distance* antara lain :

1. Kompleksitas algoritma ini dapat diperkecil menjadi O(m) dengan cara hanya menyimpan baris saat ini dan baris sebelumnya.
2. Nilai *edit distance* dapat dinormalisasi ke dalam rentang [0,1] yaitu jika 0 berarti kata yang dibandingkan identik dan menuju 1 jika sangat berbeda.
3. Jika kita hanya menginginkan nilai *edit distance* saat nilai tersebut di bawah suatu batas tertentu, maka algoritma ini dapat lebih dioptimasi.
4. Kita dapat menghitung nilai *edit distance* dengan memberikan nilai berbeda pada operasi ganti,hapus,maupun sisip. Bahkan kita dapat memberikan nilai khusus untuk operasi yang melibatkan huruf-huruf tertentu.
5. Dapat dieksekusi secara paralel untuk mempercepat waktu pemrosesan, walaupun algoritma ini kurang bagus untuk diproses secara paralel mengingat banyaknya ketergantungan data, namun *cost*(biaya) dapat dihitung secara paralel.
6. Dapat dimodifikasi untuk melakukan pencarian string fuzzy (*fuzzy string search*)

3. *SPELL CHECKER*

Algoritma program dinamis *edit distance* yang dibahas pada bagian sebelumnya dapat dimanfaatkan untuk melakukan pengecekan ejaan, yaitu dengan cara membandingkan kata yang diketikkan pengguna dengan basis data kata. Jika kata tersebut tidak dapat ditemukan, maka diusulkan daftar kata yang memiliki *edit distance* terkecil.

Notasi algoritma untuk fungsi pengecekan ini:

```
/*variabel global*/
list<string> kamus
//daftar kata yang telah memiliki ejaan yang
benar
int batasMinimal
//nilai minimal edit distance suatu pasangan kata
masih dianggap mirip

/*
   procedure cekEjaan menerima kata masukan
   pengguna yang akan dicocokan dengan daftar
   kata pada basis data.
   Apabila tidak ditemukan kata yang serupa,maka
   akan ditampilkan daftar kata-kata yang
```

```
   memiliki nilai edit distance paling kecil
   dengan kata masukan pengguna
*/
void cekEjaan (string kata) {
   list<string> usulan;
   bool ketemu = false;

   for i from 0 to kamus.length
   {
        if (kamus[i] = kata) {
           break;
        }
        else {
           if (editDistance(kata,kamus[i]) <=
                batasMinimal) {
                usulan.add(kamus[i])
           }
        }
   }
   /*program akan menampikan usulan kata dengan
   edit distance tidak lebih besar dari batas
   minimal jika kata tidak ditemukan di basis
   data*/
   tampilkanUsulan()
}
```

4. KESIMPULAN

1. Pengecekan ejaan merupakan suatu fitur yang sangat membantu pekerjaan manusia diantaranya dapat ditemui pada perangkat lunak *email client*, *search engine*, dan *word processor*. Salah satu cara pengecekan ejaan dengan menggunakan algoritma program dinamis *Edit distance*.
2. Algoritma program dinamis *edit distance* ini memiliki kompleksitas O(m.n) yang jauh lebih baik daripada algoritma pengecekan secara *exhaustive search* $O(3^{m+n})$ maupun *greedy* yang tidak selalu menghasilkan *edit distance* yang optimal.
3. Untuk kemampuan pengecekan ejaan secara konteks dibutuhkan pengembangan lebih lanjut yang menjadi kajian NLP (*Natural Language Processing*).
4. Algoritma *edit distance* dapat digunakan untuk banyak hal lain diantaranya pendeteksian plagiarisme, pencocokan bentuk, pemanfaatan dalam biologi molekuler seperti perbandingan DNA, dan juga pengenalan suara (*speech recognition*).

REFERENSI

[1] http://en.wikipedia.org/wiki/Levenshtein_distance., diakses tanggal 28 Desember 2009.

[2] http://en.wikipedia.org/wiki/Spell_checker, diakses tanggal 28 Desember 2009.

[3] http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Dynamic/Edit/, diakses tanggal 28 Desember 2009.

[4] http://www.panl10n.net/Presentations/Cambodia/Sarmad/StatisticalSpellCheckers.pdf, diakses tanggal 28 Desember 2009.

[5] Munir,Rinaldi, "Diktat Kuliah IF2251 Strategi Algoritmik", Program Studi Informatika ITB, 2007.

[6] Skiena, Steven. Programming Challenges. Springer. 2002