Penerapan *String Suggestion* dengan Algoritma *Levenshtein Distance* dan Alternatif Algoritma Lain dalam Aplikasi

Fatardhi Rizky Andhika 13508092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18092@students.if.itb.ac.id

***Abstract*—Makalah ini bertujuan untuk memaparkan bagaimana pengolahan string untuk menghasilkan *string suggestion* pada beberapa aplikasi. *String suggestion* adalah saran yang diberikan oleh aplikasi ketika kita melakukan kesalahan pengetikan atau kita memasukkan kata yang tidak dikenal oleh aplikasi. *String suggestion* juga merupakan pengolahan string yang sangat umum di dalam banyak aplikasi seperti *search engine, spell checker, dictionary*, dan banyak aplikasi lainnya. Untuk mendapatkan *suggestion* dari sebuah string, dilakukan pembandingan untuk menentukan keterbedaan antara dua *string*. Ada banyak metode yang digunakan. Salah satu metodenya adalah dengan algoritma *Levensthein Distance*, yaitu sebuah algoritma pengembangan *dynamic programming* yang mengasilkan keterbedaan dari dua buah string. Algoritma ini melakukan pembandingan dengan menggunakan matriks pembanding dan meng-*output* jumlah langkah pembeda antara dua string tersebut. Selain itu, makalah ini juga menyediakan alternatif algoritma pembanding lain yang juga dapat menghasilkan keterbedaan dari dua string.**

Kata kunci* — aplikasi, *levensthein*, *suggestion*, *string

I. PENDAHULUAN

Dunia keinformatikaan senantiasa berkembang. Perkembangan dunia ini tak pelak karena perkembangan dan kemajuan dari teknologi informasi. Teknologi informasi diciptakan sebagai sebuah solusi dalam meningkatkan kenyamanan dan kemudahan manusia dalam melakukan berbagai aktivitas.

Salah satu aktivitas manusia yang sangat erat hubungannya dengan teknologi keinformatikaan adalah informasi. Dalam dunia teknologi sekarang ini, informasi yang palin dasar bentuknya adalah informasi berupa teks. Informasi berupa teks dapat kita temukan dalam banyak hal, seperti koran, web, dan lain-lain. Teks, yang selanjutnya disebut sebagai string dalam pengolahan dan komputasinya menjadi dasar dalam dunia informasi sekarang ini.

String sudah dapat dikatakan sebagai bagian fundamental dalam penggunaan komputer dan teknologi informasi bagi manusia. Seiring kemajuan teknologi, sudah banyak sekali aplikasi-aplikasi berbasis teks yang digunakan dalam kehidupan sehari-hari seperti *Microsoft Word* yang memiliki kemampuan mengolah teks/string dan menampilkan hasilnya ke layar kita, atau aplikasi *search engine* seperti google yang mampu mencarikan informasi tertentu sesuai ketikan teks/string kita. Semua aplikasi tersebut mrupakan aplikasi yang melakukan pengolahan pada string dan melakukan komputasi terhadapnya sehingga bisa melakukan fungsinya bagi manusia.

Adalah *string suggestion*, sebuah bentuk fasilitas yang diberikan oleh aplikasi-aplikasi tadi. *String suggestion* adalah saran yang diberikan oleh aplikasi ketika kita melakukan kesalahan pengetikan atau kita memasukkan kata yang tidak dikenal oleh aplikasi. Penggunaan fasilitas ini juga merupakan bentuk kemajuan dunia teknologi informasi yang memudahkan manusia melakukan aktivitas dan melakukan pencarian informasi. Penerapan konsep *string suggestion* ini tidak lepas dari kemajuan dunia teknologi informasi yang melahirkan konsep pemrograman dinamis dan algoritma-algoritma efektif dalam pengolahan string. *String suggestion* menerapkan konsep keterbedaan string yang terlahir dari algoritma *levensthein* maupun algoritma lain yang memiliki kemampuan mendeteksi keterbedaan antara dua buah string. Konsep keterbedaan antar string merupakan konsep penting yang umum digunakan dalam aplikasi seperti *spell checker*, kamus, *search engine*, dll. Semua ini merupakan hasil dari perkembangan teknologi informasi untuk menciptakan kemudahan bagi manusia.

II. ALGORITMA LEVENSTHEIN DISTANCE

Dalam terori informasi dan computer science, *levenshtein distance* merupakan metric yang digunakan untuk mengukur keterbedaaan jarak antara dua sekuens. *Levenshtein distance* antara dua string ditentukan berdadsarkan jumlah minimum perubahan/pengeditan yang diperlukan untuk melakukan transformasi dari saru bentuk string ke bentuk string yang lain. Operasi yang dilakukan dan diperbolehkan digunakan dalam menentukan *levenshtein distance* ini ada 3 macam

operasi yaitu :

1. *Insertion*
   Insertion adalah operasi melakukan penyisipan sebuah karakter ke dalam sebuah string tertentu. Misalnya, penulis ingin menyisipkan sebuah karakter 'a' ke dalam sebuah string "kenpa" tepat setelah karakter 'n'. Dengan dilakukannya *insertion*, string yang tadinya "kenpa" akan menjadi "kenapa"
2. *Deletion*
   Deletion adalah operasi melakukan penghilangan atau penghapusan sebuah karakter tertentu dari sebuah string. Misalnya, penulis ingin menghapus karakter 'n' pada string "kenpa". Setelah dilakukan *deletion*, string akan menjadi "kepa".
3. *Substitution*
   Substitution adalah operasi menukarkan sebuah karakter pada string tertentu dengan karakter lain. Misalnya, penulis ingin menukarkan karakter 'n' pada string "kenpa" dengan karakter baru 'm'. Setelah dilakuannya *substitution*, string akan menjadi "kempa".

Algoritma levensthein yang digunakan ini merupakan aloritma buatan Vladimir Levenshtein yang berhasil beliau temukan pada tahun 1965. Pada saat itu, beliau berhasil menemukan "distance" antara dua string masukan.

Yang dimaksud dengan *distance*, seperti yang telah penulis sebutkan di atas, adalah jumlah modifikasi yang dibutuhkan untuk mengubah suatu bentuk string ke bentuk string yang lain. Sebagai contoh hasil penggunaan algoritma ini, string "apabila" dan "pabila" memiliki *distance* 1 karena hanya perlu dilakukan satu operasi saja untuk mengubah satu string ke string yang lain. Dalam kasus dua string di atas, string "pabila" dapat menjadi "apabila" hanya dengan melakukan satu sisipan karakter 'a' sebelum string "pabila". Pada bagian berikutnya akan lebih dijelaskan lagi tentang mekanisme diperolehnya angka *distance* dari kedua string tersebut dengan lebih detail.

III. MEKANISME PEROLEHAN NILAI "DISTANCE" PADA ALGORITMA LEVENSHTEIN

Berikut adalah algoritma levensthein yang digunakan untuk mencari nilai *distance* antara dua masukan string, dengan *m* adalah panjang dari string pertama, dan *n* adalah panjang string kedua. Algoritma ini merupakan perkembangan dari *dynamic programming* dimana proses pencocokan kedua string-nya dipisahkan ke bagian-bagian yang lebih kecil.

```
int LevenshteinDistance(char s[1..m], char t[1..n])
 {
  // d is a table with m+1 rows and n+1 columns
  declare int d[0..m, 0..n]
  for i from 0 to m
    d[i, 0] := i // deletion
  for j from 0 to n
    d[0, j] := j // insertion
  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,  // deletion
        d[i, j-1] + 1,  // insertion
        d[i-1, j-1] + 1 //substitution
        )
    }
  }
  return d[m,n]
}
```

Fungsi *levenshtein distance* di atas menggunakan masukan dua buat string dan menghasilkan nilai "distance"-nya. Seperti yang telah penulis sebutkan pada bagian sebelumnya, nilai "distance" merupakan nilai yang menunjukkan jumlah modifikasi minimum yang harus dilakukan untuk melakukan pengubahan string yang satu ke string yang lain. Untuk lebih jelasnya, perhatikan matriks ilustrasi pencari *levenshtein distance* antara dua string berikut yaitu "penjara" dan "jarak".

		-1	0	1	2	3	4	5	6
			p	e	n	j	a	r	a
-1		0	1	2	3	4	5	6	7
0	j	1	1	2	3	3	4	5	6
1	a	2	2	2	3	4	3	4	5
2	r	3	3	3	3	4	4	3	4
3	a	4	4	4	4	4	4	4	3
4	k	5	5	5	5	5	5	5	4

Tabel di atas adalah matriks yang menunjukkan bagaimana proses pencarian nilai "distance" untuk string "jarak" dan "penjara". Pengecekan dimulai atau diiterasi dari dari awal dari kedua string yaitu karakter karakter paling awal dari kedua string, lalu dilakukan operasi-operasi tertentu pada string seperti *insertion*, *deletion*, atau *substitution*. Bagian yang dilingkari merupakan

jumlah perubahan string minimum secara bertahap yang didapat untuk mencapai keaadaan saat itu juga. Jadi, yang menjadi nilai "distance" akhir adalah yang terdapat pada akhir atau ujung kedua string yang dibandingkan yaitu pada bagian pojok kanan bawah pada matriks.

Apabila kita melakukan pengecekan pada matriks yang terlihat pada gambar tersebut terutama pada bagian yang dilingkari , didapat : pada bagian yang dicek pertama kali, terdapat karakter 'p' pada string 1 sedangkan string 2 masih pada karakter kosong. Karena untuk menyamakan kedua kedua karakter tersebut dapat dilakukan proses insersi yakni mengyisipkan karakter "p" pada "", dilakukanlah proses insersi lalu isi dari matriksnya dijadikan bernilai '1', artinya, untuk mendapatkan karakter yang sama pada bagian tersebut untuk kedua string harus dilakukan satu kali operasi perubahan stiring. Pada lingkaran yang berikutnya, karena string masih berupa "p", dilakukanlah operasi insersi lagi sehingga stringnya dapat menjadi "pe". Karena dilakukan satu kali lagi operasi perubahan string, nilai matriks untuk bagian tersebut diisi dengan nilai 2 (terjadi penambahan 1 dari nilai sebelumnya). Pada lingkaran yang berikutnya, kembali dilakukan insersi pada yakni menginsersi karakter 'n' pada string "pe" tadi. Karena kembali dilakukan operasi pada string, nilai matriks pada bagian tersebut ditambahkan 1 menjadi '3'. Pada lingkaran yang berikutnya, karakter yang dicek adalah sama-sama karakter 'j', sehingga tidak perlu dilakukan operasi pada string dan nilai matriks pada bagian tersebut akan tetap '3'. Begitu pula pada tiga lingkaran yang berikutnya, dimana karakter pada kedua string adalah sama-sama karakter 'a', 'r', 'a' sehingga nilai matriks pada ketiga bagian tersebut adalah tetap '3'. Untuk lingkaran yang terakhir, ditemukan karakter 'k' di akhir string padahal karakter yang dicocokkan telah terpenuhi pada lingkaran yang sebelumnya (sudah menjadi kata "penjara"). Oleh karena itu, karakter 'k' di delesi dari string "penjarak" sehingga hanya menjadi "penjara" dan tepat sama dengan string yang kita tujukan. Isi matriks pada bagian yang dimaksud pun ditambahkan satu menjadi bernilai '4'.

Dari proses pencarian nilai "distance" –nya, didapat bahwa jumlah modifikasi minimum yang dilakukan untuk mengubah string "jarak" menjadi "penjara" adalah sebanyak 4 operasi yaitu : penyisipan karakter 'p', penyisipan karakter 'e', penyisipan karakter 'n', dan penghilangan karakter 'k'.

Algoritma ini pada bagian berikutnya akan diimplementasikan untuk dapat menghasilkan *string suggestion* yang dapat diterapkan pada aplikasi.

IV. IMPLEMENTASI ALGORITMA LEVENSHTEIN DALAM *STRING SUGGESTION*

Seperti yang telah dijelaskan pada bagian pendahuluan, fasilitas *string suggestion* ditentukan berdasarkan algoritma yang digunakan dalam penentuan *string suggestion*-nya. Proses penyusunan *string suggestion* sendiri menitikberatkan pada perbandingan antara string masukan dengan *database* string valid yang tersimpan di media penyimpanan.

Algoritma *levenshtein* sebagai salah satu algoritma yang menghitung keterbedaan antar string dapat diterapkan sebagai dasar dari penyusunan *string suggestion*. Untuk penerapan algoritma ini dalam aplikasi, terdapat beberapa hal yang harus tersedia yaitu :

1. *Query* pengguna, yaitu kumpulan kata berupa string yang dimasukkan pengguna ke dalam sistem.
2. *Database* kata, yakni daftar seluruh kata yang valid dan dapat diterima sistem. *Database* digunakan sebagai pembanding antara *query* masukan pengguna sehingga nantinya bisa dicari kata-kata tertentu yang paling mendekati dengan masukan pengguna.

Pengecekan dilakukan per-kata masukan pengguna, artinya, setiap kata masukan pengguna akan dicari dan dimiripkan dengan setiap kata yang ada di database lalu setiap kata yang paling mirip ditemukan di database akan disusun menjadi *string suggestion*-nya.

Berikut ini adalah *pseudo code* mekanisme penyusunan *string suggestion* dengan menggunakan algoritma *levenshtein distance* yang dapat diimplementasikan dalam aplikasi :

```
List<String> query
//merupakan query masukan user yang terdiri atas kata-kata

List<String> suggestion = ""
//merupakan media ntuk menyimpan string hasil keluaran fungsi levenshtein distance sebagai suggestion yang nantinya akan ditampilkan

List<String> database
//merupakan media untuk menyimpan string kata yang valid dari dari database

int LevenshteinDistance(String query1, String kata1)
// merupakan fungsi levenshtein distance yang diubah seperlunya yaitu masukan fungsinya adalah berupa string

List<String> LoadDatabase(String pathdatabase)
//melakukan load ke database dan menyimpan semua daftar kata yang valid ke list string

String CekDatabase (List<String> database, String query1)
//melakukan pengecekan "kata" di dalam database kata dan mereturn kata yang sama atau kata yang paling
```

```
mendekati
{
   int max=0;
   int i = 0;
   String katamax;
   for (String data : database)
   {
    i = LevenshteinDistance(query1, data)
    if (i==0)
    {return data;}
    else if (max>i)
       { katamax = data;max=i}
   }
   return katamax;
}
//main program

database = LoadDatabase (path)
for (String kata :  query)
{
   suggestion.add(CekDatabase (database, kata))
}

//mengakhiri program dengan mer*eturn suggestion*
return suggestion;
```

V. ALTERNATIF ALGORITMA LAIN UNTUK MENENTUKAN KETERBEDAAN STRING

Berikut ini merupakan alternatif algoritma lain yang dapat digunakan untuk membentuk *string suggestion*. Ide dari algoritma ini adalah memecah string masukan user yang terdiri atas banyak kata menjadi token-token kata yang kemudian setiap kata-kata tersebut dipecahkan lagi menjadi pasangan-pasangan huruf yang hanya terdiri dari dua huruf saja. Dengan demikian, setiap kata yang akan dicocokkan dengan daftar kata akan memiliki substring-substring yang terdiri atas 2 karakter saja. Nantinya, penentuan keterbedaan antara kedua kata akan ditentukan oleh kemiripan pasangan-pasangan karakter yang terdapat dalam kata yang dibandingkan.

Nilai keterbedaan antara dua string kata yang dibandingkan dengan algoritma ini adalah berupa bilangan double. Bilangan yang dihasilkan adalah persentase keterdekatan string yang dibandingkan. Nilai keterbandingan string yang semakin mendekati ‘1’ artinya semakin miriplah kedua kata yang dibandingkan. Apabila nilai yang diterima adalah bernilai tepat ‘1’, artinya string yang dibandingkan memiliki kemiripan 100%.

```
String[] letterPairs(String str) {
        int numPairs = str.length()-1;
        String[] pairs = new String[numPairs];
        for (int i=0; i<numPairs; i++) {
           pairs[i] = str.substring(i,i+2);
        }
        return pairs;
     }

     ArrayList wordLetterPairs(String str) {
        ArrayList allPairs = new ArrayList();
        // Tokenize the string and put the tokens/words
into an array
        String[] words = str.split("\\s");

        // For each word
        for (int w=0; w < words.length; w++) {
           // Find the pairs of characters
           String[] pairsInWord = letterPairs(words[w]);

           for (int p=0; p < pairsInWord.length; p++) {
              allPairs.add(pairsInWord[p]);
           }
        }
        return allPairs;
     }

     double compareStrings(String str1, String str2) {
        ArrayList pairs1 =
wordLetterPairs(str1.toUpperCase());
        ArrayList pairs2 =
wordLetterPairs(str2.toUpperCase());
        int intersection = 0;
        int union = pairs1.size() + pairs2.size();
        for (int i=0; i<pairs1.size(); i++) {
           Object pair1=pairs1.get(i);
           for(int j=0; j<pairs2.size(); j++) {
              Object pair2=pairs2.get(j);
              if (pair1.equals(pair2)) {
                 intersection++;
                 pairs2.remove(j);
                 break;
              }
           }
        }
        return (2.0*intersection)/union;
  }
```

Adapun hal-hal yang harus disediakan untuk implementasi algoritma ini dalam membentuk *string suggestion* adalah sama dengan apa yang dibutuhkan pada penerapan algoritm levenshtein yaitu *query* masukan pengguna dan *database* daftar kata-kata valid yang dapat diterima oleh sistem.

Pengecekan dilakukan per-kata masukan pengguna, artinya, setiap kata masukan pengguna akan dicari dan dimiripkan dengan setiap kata yang ada di database lalu setiap kata yang paling mirip ditemukan di database

akan disusun menjadi *string suggestion*-nya.

Berikut adalah *pseudo code* mekanisme penyusunan *string suggestion* dengan menggunakan algoritma alternatif ini yang dapat diimplementasikan dalam aplikasi :

```
List<String> query
//merupakan query masukan user yang terdiri atas kata-kata

List<String> suggestion = “”
//merupakan media ntuk menyimpan string hasil keluaran fungsi levenshtein distance sebagai suggestion yang nantinya akan ditampilkan

List<String> database
//merupakan media untuk menyimpan string kata yang valid dari dari database
List<String> LoadDatabase(String pathdatabase)
//melakukan load ke database dan menyimpan semua daftar kata yang valid ke list string

double CompareStrings(String str1, String str2)
//menghasilkan nilai perbandingan antara dua string, makin mendekati “satu” makin menunjukkan kedekatan string

String SearchDatabase (List<String> database, String query1)
//melakukan pengecekan “kata” di dalam database kata dan mereturn kata yang sama atau kata yang paling mendekati
{
   double max=0;
   double i = 0;
   String katamax;
   for (String data : database)
   {
     i = CompareStrings(query1, data)
     if (i==1)
     {return data;}
     else if (i>max)
        { katamax = data; max=i}
   }
   return katamax;
}


//main program

database = LoadDatabase (path)
for (String kata :  query)
{
   suggestion.add( SearchDatabase(database, kata))
}
//mengakhiri program dengan mereturn suggestion
return suggestion;
```

VI. KESIMPULAN

Algoritma *levenshtein distance* maupun algoritma alternatif yang ditawarkan dalam makalah ini dapat melakukan pembandingan antara dua string dan menghasilkan nilai tertentu yang menunjukkan keterbedaannya. Namun, kedua algoritma ini menggunakan pendekatan berbeda dalam menghitung nilai keterbedaan string yang mana perbedaan metode ini pastinya juga akan mempengaruhi performanya dalam membandingkan string.

Dalam implementasinya, kedua algoritma ini telah dapat menyelesaikan pembentukan *string suggestion* yakni memberikan saran atau pembenaran terhadap kumpulan string masukan pengguna. Namun, tingkat keberhasilan dari algoritma-algoritma ini untuk memberikan suggestion yang benar-benar diinginkan pengguna selain bergantung pada ketepatan metode perbandingan katanya, juga sangat bergantung pada banyaknya kata-kata valid yang tertampung di *database*-nya. Selain itu, untuk meningkatkan fungsionalitas *string suggestion*nya, *database* harus senantiasa diperbarui dengan kata-kata peristiwa yang terjadi secara *real-time* sehingga apa yang dihasilkan *string suggestion*-nya akan benar-benar menemui ekspektasi pengguna.

REFERENCES

[1] http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node2.html.
[2] http://en.wikipedia.org/wiki/Levenshtein_distance.
[3] http://en.wikipedia.org/wiki/Dynamic_programming.
[4] http://www.merriampark.com/ld.htm.

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 Desember 2010

Fatardhi Rizky Andhika
13508092