Penerapan Algoritma Boyer Moore-Dynamic Programming untuk Layanan Auto-Complete dan Auto-Correct

Christabella Chiquita B. - 13509050
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
christabella.c.b@itb.ac.id

***Abstrak*—Pada text editor, biasanya terdapat layanan auto-complete pada saat mencari suatu kata sehingga dapat membantu user agar tidak perlu mengetikkan secara lengkap kata yang dicari. Selain itu juga terdapat layanan auto-correct sebagai antisipasi user melakukan salah ketik atau kata yang dicarinya tidak ada pada text. Untuk membuat layanan auto-complete dan auto-correct ini, sebenarnya digunakan dua buah algoritma utama, yaitu algoritma Boyer Moore untuk mencari kata yang sama (String Matching) dan algoritma Dynamic Programming untuk menentukan jarak minimum (perbedaan) antara dua string sehingga dapat diketahui string mana saja pada teks yang mendekati pattern yang dicari oleh user.**

***Index Terms*—auto-complete, auto-correct, Boyer Moore, Dynamic Programming.**

I. PENDAHULUAN

Bekerja dengan file yang berbasis teks, seringkali kita membutuhkan pencarian kata tertentu. Apabila file teks tersebut berukuran kecil, pencarian kata dapat dilakukan manual. Namun jika file teks sudah berukuran besar, kita akan kesulitan dalam mencari kata tersebut. Oleh karena itulah, pada teks editor hamper selalu ditemukan layanan *Find*. Dengan layanan ini, kita hanya perlu mengetikkan kata yang ingin kita temukan. Pada layanan Find ini, biasanya terdapat auto-complete dan auto-correct. Keduanya hampir sama dalam tampilannya, yaitu sama – sama memberikan daftar word-suggestions kepada user untuk dipilih. Perbedaannya adalah, pada auto-complete, akan dicari kata yang sama persis dengan input user (pattern), lalu diberikan daftar word-suggestions yang sama atau mengandung substring sama dengan pattern tersebut, sehingga user tidak perlu mengetikkan secara lengkap. Sedangkan pada auto-correct, diberikan word-suggestions yang berisi kata – kata yang mirip dengan kata yang diinput user.

Pada makalah ini, akan dijelaskan pembuatan layanan auto-complete dan auto-correct ini dengan memanfaatkan algoritma Boyer-Moore untuk auto-complete dan auto-correct, serta algoritma dynamic programming untuk layanan auto-correct. Akan disertakan juga contoh source code yang dibuat oleh penulis untuk menguji kedua algoritma tersebut beserta contoh hasil *screenshot* tampilan prototype programnya.

II. DASAR TEORI

1. Algoritma Boyer-Moore
   Algoritma Boyer-Moore adalah salah satu algoritma untuk mencari suatu pattern di dalam teks, dibuat oleh R.M Boyer dan J.S Moore. Ide utama algoritma ini adalah mencari pattern dengan melakukan pembandingan karakter mulai dari karakter paling kanan dari pattern yang dicari. Dengan mengunakan algoritma ini, secara rata-rata proses pencarian akan menjadi lebih cepat jika dibandingakan dengan algoritma lainnya. alasan melakukan pencocokan dari kanan (posisi terakhir pattern yang dicari) ditunjukan dalam contoh berikut :

m	***a***	***k***	***a***	***n***		***j***	***a***	***m***	***b***	***u***
j	a	m	b	u						

pada contoh diatas, dengan melakukan pembandingan dari posisi paling akhir pattern dapat dilihat bahwa karakter “u” pada string “jambu” tidak cocok dengan karakter “n” pada string “makan” yang dicari, dan karakter “n” tidak pernah ada dalam string “jambu” yang dicari sehingga pattern “jambu” dapat digeser melewati string “makan” sehingga posisinya menjadi

m	***a***	***k***	***a***	***n***		***j***	***a***	***m***	***b***	***u***
					j	a	m	b	u	

Dalam contoh terlihat bahwa algoritma Boyer-Moore memiliki loncatan karakter yang besar sehingga mempercepat pencarian pattern karena dengan hanya memeriksa sedikit karakter, dapat langsung diketahui bahwa string yang dicari tidak ditemukan dan dapat digeser ke posisi berikutnya.

Langkah-langkah algoritma Boyer-Moore :

- Buat tabel pergeseran pattern yang dicari (P) dengan pendekatan Match Heuristic (MH) dan Occurence Heuristic (OH), untuk menentukan jumlah pergeseran yang akan dilakukan jika mendapat karakter tidak cocok pada proses pencocokan dengan string pada teks (S).
- Jika dalam proses pembandingan terjadi ketidakcocokan antara pasangan karakter pada P

dan karakter pada S, pergeseran dilakukan dengan memilih salah satu nilai pergeseran dari dua tabel analisa pattern, yang memiliki nilai pergeseran paling besar.

- Dua kemungkinan penyelesaian dalam melakukan pergeseran P, jika sebelumnya belum ada karakter yang cocok adalah dengan melihat nilai pergeseran hanya pada tabel occurence heuristic : Jika karakter yang tidak cocok tidak ada pada P maka pegeseran adalah sebanyak jumlah karakter pada P. dan jika karakter yang tidak cocok ada pada P, maka banyaknya pergeseran bergantung dari nilai pada tabel.
- Jika karakter pada teks yang sedang dibandingkan cocok dengan karakter pada P, maka posisi karakter pada P dan S diturunkan sebanyak 1 posisi, kemudian lanjutkan dengan pencocokan pada posisi tersebut dan seterusnya. Jika kemudian terjadi ketidakcocokan karakter P dan S, maka pilih nilai pergeseran terbesar dari dua tabel analisa pattern yaitu nilai dari tabel match heuristic dan nilai tabel occurence heuristic dikurangi dengan jumlah karakter yang telah cocok.
- Jika semua karakter telah cocok, artinya P telah ditemukan di dalam S, selanjutnya geser pattern sebesar 1 karakter. Lanjutkan sampai akhir pattern S.

Cara Menghitung Tabel Occurence Heuristic :
contoh pattern : manaman
Panjang : 7 karakter
Tabel Occurence Heuristic

Posisi	1	2	3	4	5	6	7
Pattern	m	a	n	a	m	a	n
Pergeseran	2	1	0	1	2	1	0

Lakukan pencacahan mulai dari posisi terakhir pattern sampai ke posisi awal, dimulai dengan nilai 1, catat karakter yang sudah ditemukan (dalam contoh ini karakter “n”)
Mundur ke posisi sebelumnya, nilai pencacah ditambah 1, jika karakter pada posisi ini belum pernah ditemukan, maka nilai pergeserannya adalah sama dengan nilai pencacah. (dalam contoh ini, karakter “a” belum pernah ditemukan sehingga nilai pergeserannya adalah sebesar nilai pencacah yaitu 2)
Mundur ke posisi sebelumnya, karakter “m” nilai pergeserannya 3
Mundur lagi, karakter “a”. karakter “a” sudah pernah ditemukan sebelumnya sehingga nilai pergeserannya sama dengan nilai pergesean karakter “a” yang sudah ditemukan paling awal yaitu 2.
Begitu seterusnya sampai posisi awal pattern.
catatan : untuk karakter selain “m”,”a” dan “n” nilai pergeseran sebesar panjang pattern yaitu 7 karakter.

Cara Menghitung Tabel Match Heuristic :
contoh pattern : manaman
Panjang : 7 karakter

Tabel Match Heuristic

Posisi	1	2	3	4	5	6	7
Pattern	m	a	n	a	m	a	n
Pergeseran	4	4	4	4	7	7	1

Langkah-langkah perhitungannya adalah sebagai berikut :

- jika karakter pada posisi 7 bukan “n” maka geser 1 posisi, berlaku untuk semua pattern yang dicari.
- jika karakter “n” sudah cocok, tetapi karakter sebelum “n” bukan “a” maka geser sebanyak 7 posisi, sehingga posisi pattern melewati teks.karena sudah pasti “manambn” bukan “manaman”
- jika karakter “an” sudah cocok, tetapi karakter sebelum “an” bukan “m” maka geser sebanyak 7 posisi, sehingga posisi pattern melewati teks.karena sudah pasti “manaban” bukan “manaman”
- jika karakter “man” sudah cocok, tetapi karakter sebelum “man” bukan “a” maka geser sebanyak 4 posisi, sehingga posisi pattern berada / bersesuaian dengan akhiran “man” yang sudah ditemukan sebelumnya. karena bisa saja akhiran “man” yang sudah ditemukan sebelumnya merupakan awalan dari pattern “manaman” yang berikutnya.
- jika karakter “aman” sudah cocok, tetapi karakter sebelum “aman” bukan “n” maka geser sebanyak 4 posisi, sehingga posisi pattern berada / bersesuaian dengan akhiran “man” yang sudah ditemukan sebelumnya, karena bisa saja akhiran “man” yang sudah ditemukan sebelumnya merupakan awalan dari pattern “manaman” yang berikutnya.
- selanjutnya sama, pergseran paling mungkin dan aman dalam tabel Match Heuristic adalah pergeseran sebanyak 4 posisi.

2. Algoritma Dynamic Programming

Merupakan metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (step) atau tahapan (stage) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Pada penyelesaian persoalan dengan metode ini:
1. terdapat sejumlah berhingga pilihan yang mungkin,
2. solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya,
3. kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan Prinsip Optimalitas, yaitu jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal. Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap k ke tahap k + 1, kita dapat menggunakan hasil optimal dari tahap k tanpa harus kembali ke tahap awal. ongkos pada tahap k +1 = (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap k + 1). Dengan prinsip optimalitas ini dijamin bahwa pengambilan keputusan pada suatu tahap

adalah keputusan yang benar untuk tahaptahap selanjutnya. Pada metode greedy hanya satu rangkaian keputusan yang pernah dihasilkan, sedangkan pada metode program dinamis lebih dari satu rangkaian keputusan. Hanya rangkaian keputusan yang memenuhi prinsip optimalitas yang akan dihasilkan.

III. APLIKASI

A. Implementasi Penerapan Boyer-Moore

Pada awalnya, dibuat sebuah list kosong bernama Result, yang akan digunakan untuk menyimpan kumpulan indeks awal string yang mengandung pattern yang dicari. Dibuat juga suatu Boolean isFound yang diinisialisasi false. Boolean ini menjadi penanda apakah pattern ditemukan atau tidak dalam teks. Setiap kali user mengetikkan suatu karakter yang akan dicari, secara otomatis program akan menganggap string yang sudah tertulis pada textfield sebagai sebuah pattern untuk dicocokkan dengan isi teks. Pencocokan pattern dengan string ini menggunakan algoritma Boyer-Moore yang keluarannya adalah indeks ditemukannya pattern pada teks. Setelah menemukan string yang sama dengan pattern, maka indeks awal string pada teks tersebut dimasukkan ke antrian paling belakang listOfFound dan isFound diisi dengan true. Jika menemukan lagi string yang sama dengan pattern, maka indeks awal string itu diantrikan lagi ke akhir list. Begitu seterusnya sampai end of file, dan diperoleh list Result yang berisi indeks – indeks awal string pada teks yang mengandung pattern.

Source Code penerapan Boyer-Moore :

```
public class BM {
    public String text; //isi file txt
    public int PatLength; //panjang string yang dicari
    public String Pattern; //string yang mau dicari
    public char[] PatArray; //array of char pattern
    public int[] OH; //tabel pergeserah OH
    public int[] MH; //tabel pergeseran MH
    public LinkedList<Integer> Result;
    public Boolean isFound = false;
```

Pembuatan tabel OH :

```
public void makeOH(String pattern){
    OH = new int[pattern.length()];
    PatArray = pattern.toCharArray();
    int i, found;
    //System.out.println("length = "+i);
    for (i = PatArray.length-1; i>=0; i--){

        found = searchEqualChar(PatArray[i], i, PatArray);
        //System.out.println("found at : "+found);
        if (found!=-1){
            OH[i] = OH[found];
            //System.out.println("OH["+i+"] = "+OH[i]);
        }
        else{
            OH[i]=PatArray.length - 1 - i;
            //System.out.println("OH["+i+"] = "+OH[i]);
        }

    }
}
```

Pembuatan tabel MH

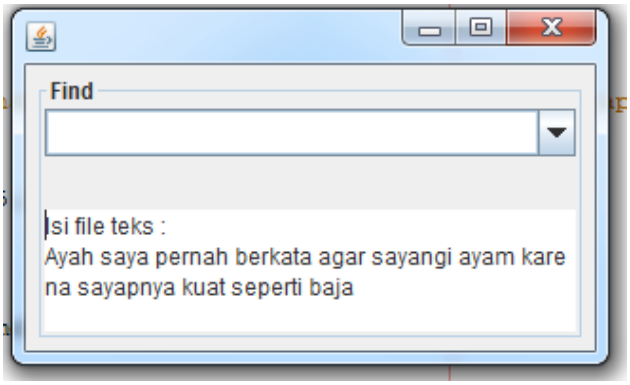
```
public void makeMH(String pattern){
    MH = new int[pattern.length()];
    int i, at;
    for (i=pattern.length()-1 ; i>=0; i--){
        if (i==pattern.length() - 1){
            MH[i] = 1;
        }
        else{
            String s = pattern;
            int x = i+1;
            int cc = shift(s.substring(x, (pattern.length())),pattern);
            if(cc!=-1){
                MH[i] = cc;
                border = true;
            }
            else {
                if(border){
                    int founddpn = searchEqualCharDpn(pattern.charAt(i+1), i+1, patte
                    if(founddpn!=-1){
                        MH[i] = pattern.length()-1-founddpn;
                    }
                    else{
                        MH[i] = MH[i+1];
                    }
                }
                else{
                    int founddpn = searchEqualCharDpn(pattern.charAt(i+1), i+1, patte
                    if(founddpn!=-1){
                        MH[i] = pattern.length()-1-founddpn;
                    }
                    else{
                        MH[i] = pattern.length();
                    }
                }
            }
        }
    }
}
```

Pencarian pattern pada teks

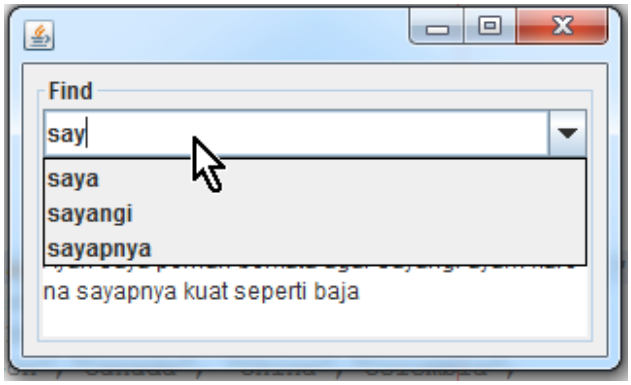
```
public void searchBM(String patterna, String texta){
    String text = texta.toLowerCase();
    String pattern = patterna.toLowerCase();
    int i, j;
    int patlen = pattern.length();
    int txtlen = text.length();
    char[] patchar = new char[patlen];
    patchar = pattern.toCharArray();
    char[] txtchar = new char[txtlen];
    txtchar = text.toCharArray();
    makeOH(pattern);
    makeMH(pattern);
    for (i = 0; i<txtlen; i++){
        if(txtchar[i]!='\0'){
            for (j = patlen-1; j>=0 && (i+j)<txtlen; j--){
                if(patchar[j]!=txtchar[i+j]){
                    int max = Math.max(MH[j],OH[j]);
                    i = i-1+ Math.max(MH[j],OH[j]);

                    break;
                }
                else {
                    if(j==0){
                        Result.addLast(i);
                        isFound = true;
                    }
                }
            }
        }
    }
}
```

Apabila pada end of file (ketika i = txtlen), isFound bernilai true, berarti pattern ditemukan dalam teks. Oleh karena itu, string – string pada teks dengan indeks awal yang ada dalam list Result boleh dimasukkan ke dalam daftar word suggestions untuk layanan auto-complete. Dibuat sebuah list kosong bernama listOfFound. List ini diisi dengan kumpulan string pada teks dengan indeks awal yang terdapat pada list Result. Jadi misalkan pada list Result ke-1 adalah 13, berarti pattern ditemukan pada indeks ke 13 teks, maka karakter dari indeks ke-13 dan seterusnya dimasukkan ke dalam sebuah variable array of char sampai ditemukan spasi. Array of char ini akan diubah menjadi String untuk diantrikan ke listOfFound. Begitu seterusnya sampai indeks terakhir pada list Result. Dengan ini, diperoleh semua string pada teks yang mengandung pattern yang dicari. Maka semua isi listOfFound ini dimasukkan ke dalam daftar word suggestions.



Gambar 1. Tampilan awal menampilkan isi teks



Gambar 2. Layanan auto-complete untuk kata "say"

Namun apabila ketika i=txtlen (end of file) isFound masih bernilai false, berarti pattern tidak ditemukan pada teks.Kemungkinan yang terjadi adalah user salah mengetikkan pattern atau ia tidak tahu bahwa pattern yang dicari memang tidak ada pada teks. Maka dibutuhkan layanan auto-correct.

B. Implementasi Penerapan Dynamic Programming

Pada algoritma dynamic programming, persoalan dibagi – bagi menjadi beberapa step / langkah. Pada kasus ini, langkah / step adalah minimum jumlah karakter antara pattern dan string pada teks. Pada setiap tahap, diambil cost (perbedaan) minimal dengan tujuan ketika tahap terakhir diperoleh solusi optimal, yaitu yang memiliki total perbedaan minimal.

Pada persoalan ini,

- Misalkan x1, x2, …, xn adalah simpul - simpul yang dikunjungi pada tahap k (k = 1, 2, 3, n).
- Maka rute yang dilalui adalah (s-p)→x1→x2→x3→..→xn, yang dalam hal ini xn merupakan salah satu kondisi basis.
- Tahap (k) adalah proses memilih simpul tujuan berikutnya (ada n tahap).
- m adalah panjang string, j adalah panjang pattern

Relasi rekurens di bawah ini menyatakan perbedaan tersedikit dari status s ke xn pada tahap k:

f_n = Proses("", "") = 0
f_n = Proses (s[x..m], "") = m-x
f_n = Proses ("". p[y..j]) = j-y
} (Basis)

f_k = Proses (s[1..m], p[1..m]) s[1] = p[1]
f_k = 1+min(Proses(s, p[1..j]),
Proses(s[1..m], p), s[1] != p[1]
Proses(s[1..m], p[1..j]))

Dapat dilihat bahwa basis dari proses ini adalah ketika string dan pattern sudah merupakan string kosong (""), atau ketika salah satu antara string dan pattern saja yang berupa string kosong (""). Ketika mencapai basis, proses akan mengembalikan integer bernilai selisih panjang dari

sisa karakter string dan pattern, karena jika salah satu dari mereka sudah kosong, berarti jumlah berbedaan karakternya pasti sejumlah sisa karakter string / pattern yang belum kosong.

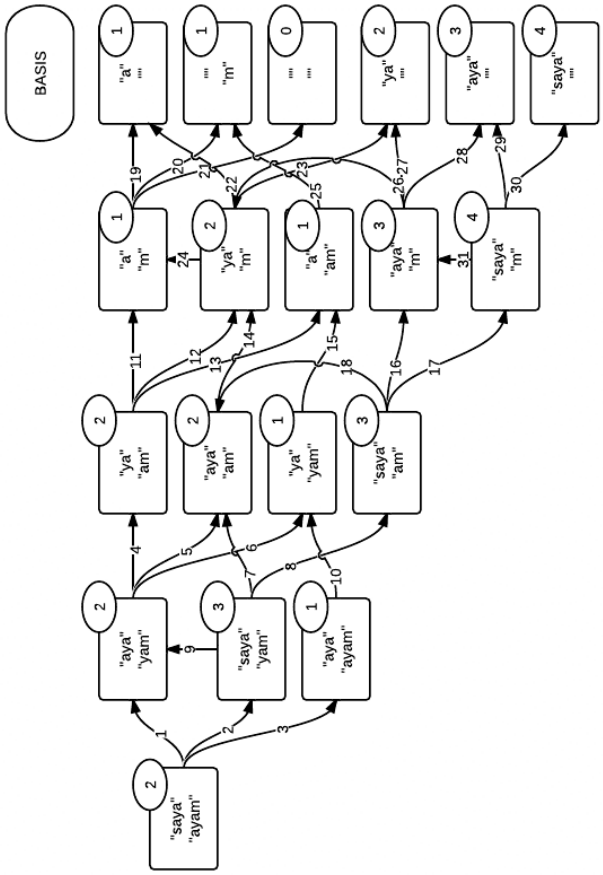
Sedangkan untuk proses rekurens setiap tahap, selalu dicari nilai minimum dari tiga jenis proses, yaitu proses untuk string.substring(1) dan pattern.substring(1), proses untuk string dan pattern.substring(1), dan proses untuk string.substring(1) dan pattern. Proses ini akan melakukan rekursi sampai mencapai basis. Karena pada setiap tahap dilakukan method Proses, maka pada setiap tahap akan diperoleh perbedaan minimum, dan pada akhirnya, akan dicapai perbedaan total yang minimum.

Source Code untuk Proses

```
public int Proses(String str1, String str2)
{
    if ( str1.length() == 0 )
        return str2.length();
    if ( str2.length() == 0 )
        return str1.length();

    String substring1 = str1.substring(1);
    String substring2 = str2.substring(1);
    if ( str1.charAt(0) == str2.charAt(0) )
        return Proses(substring1, substring2);
    else
    {
        int one = Proses(substring1, substring2);
        int two = Proses(substring1, str2);
        int three = Proses(str1, substring2);
        return (1+Math.min(one, Math.min(two, three)));
    }
}
```

Jadi, misalkan pattern yang dicari oleh user adalah “saya”, dan string yang sedang dibaca pada teks adalah “ayam”. Diagram yang terbentuk adaah sebagai berikut.



Gambar 3. Diagram tahap – tahap Dynamic Programming mencari perbedaan dua string

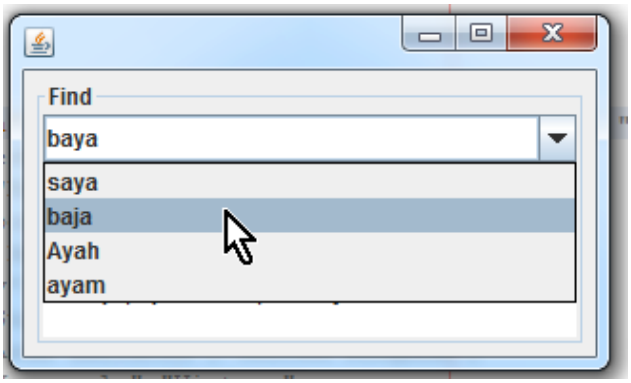
Pada diagram tersebut, dijabarkan seluruh proses rekursif yang terjadi. Angka yang terdapat di dalam lingkaran pada setiap kotak menunjukkan cost (jumlah perbedaan). Cara penghitungan cost pada setiap kotak adalah sebagai berikut.

- Pada ujung paling kanan (tahap 5), merupakan kondisi basis. Jika kedua string telah kosong (“”), cost diisi dengan 0. Jika hanya salah satu antara kedua string sudah kosong (“”), maka cost diisi dengan sisa panjang string yang belum kosong. Misalkan “aya” dan “”, berarti cost = length(“aya”) = 3
- Dilaknjutkan ke tahap sebelumnya (tahap 4). Apabila karakter pertama antara kedua string sama, maka cost diisi dengan cost minimum anaknya. Misalkan “a” dan “am”, maka cost nya diisi 1 karena cost anaknya hanya 1.
- Sedangkan jika karakter pertama kedua string beda, cost diisi dengan cost minimum anaknya + 1. Misalkan “ya” dan “m” costnya diisi dengan 2 karena dari ketiga anaknya, cost minimumnya 1
- Begitu seterusnya dilakukan untuk tahap – tahap berikutnya (tahap 3 sampai 1). Pada tahap 1 akan diperoleh cost minimum.

Dengan prinsip optimasi, karena pada setiap tahap, selalu diambil cost minimal sehingga pada akhirnya diperoleh cost akhir yang minimal juga, yaitu 2. Dapat

disimpulkan bahwa untuk pattern “saya” dan string “ayam” perbedaan minimum yang diperlukan adalah 2 karakter, yaitu menghilangkan huruf ‘s’ di awal “saya” dan menambah huruf ‘m’ di belakang “aya”.

Dengan algoritma ini, setiap pattern dicocokkan dengan string pada teks, akan dilakukan Proses yang mengembalikan integer sebagai jumlah karakter yang berbeda antara pattern dan string. Pada pengujian ini, yang masuk ke daftar string untuk ditampilkan pada daftar word suggestion hanya string yang mempunyai perbedaan sampai 2 karakter saja. Jadi misalkan pattern “saya” tidak ditemukan dalam teks, maka word suggestion akan berisi string – string yang jumlah perbedaannya maksimal 2, seperti “ayam”, “baja”, dll.



Gambar 4. Layanan auto-correct untuk kata “baya”

VII. SIMPULAN

Algoritma Boyer-Moore merupakan algoritma yang cocok untuk mencari string yang sama dengan pattern (string matching) dan termasuk algoritma yang cukup mangkus dibandingkan algoritma string matching lainnya. Sehingga algoritma Boyer-Moore ini cocok untuk membuat layanan auto-complete.

Algoritma Dynamic Programming merupakan algoritma yang cocok untuk mencari jumlah perbedaam karakter antara pattern dan string teks karena string dapat dipecah menjadi karakter – karakter yang mewakilkan setiap tahap. Dengan prinsip optimasi pada algoritma Dynamic Programming, makan akan diperoleh cost (perbedaan) minimum antara kedua string. Oleh karena itulah algoritma ini cocok dimanfaatkan untuk membuat layanan auto-correct.

REFERENSI

[1] Munir, Rinaldi, “Strategi Algoritmik”, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2007

[2] Edward G.R., Algoritma boyer-moore, http://edwardgr.wordpress.com/2009/01/06/algoritma-boyer-moore/, Januari 2009.

[3] Michael Gilleland, Levenshtein Distance, in Three Flavors, http://www.merriampark.com/ld.htm#WHATIS.

[4] http://en.wikipedia.org/wiki/Dynamic_programming

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 Desember 2011

Christabella C. B - 13509050