Penerapan Algoritma Pencocokan String pada Fitur Rekomendasi Stiker dalam Aplikasi Perpesanan

Muhammad Helmi Hibatullah - 13520014
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520014@std.stei.itb.ac.id

***Abstrak*—Kini pengguna dapat mengirimkan stiker pada aplikasi perpesanan untuk mengekspresikan pikiran dan emosinya. Stiker menjadi sebuah alternatif visual dari sebuah pesan teks. Dengan banyaknya stiker yang dapat dimiliki oleh suatu pengguna, banyak aplikasi perpesanan menambahkan fitur rekomendasi stiker ke dalamnya. Algoritma pencocokan string jarak Levenshtein dapat digunakan untuk mengimplementasikan fitur ini dengan mencocokkan teks yang diketik pengguna dengan label stiker yang ditentukan ke setiap stiker secara manual. Kelebihan dari algoritma ini adalah dapat mencocokkan teks dengan tingkat kecocokan tertentu. Kekurangannya adalah tidak dapat memprediksi dan menampilkan rekomendasi stiker sesuai dengan konteks yang dibicarakan pengguna dan lawan bicaranya.**

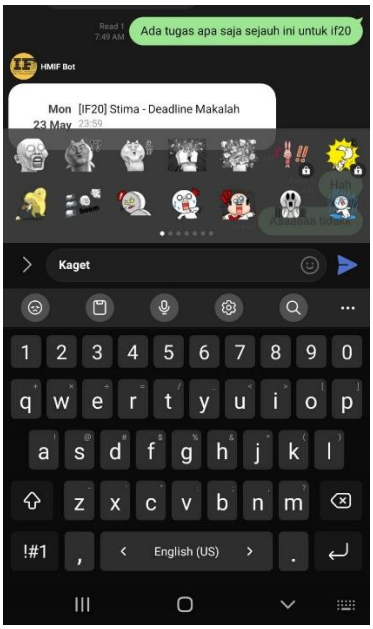
Keywords—Stiker, Rekomendasi, Pencocokan String, Jarak Levenshtein.

I. PENDAHULUAN

Aplikasi bertukar pesan kini sudah berkembang dengan pesat. Pengguna saat ini tidak hanya dapat bertukar pesan dengan teks, namun pengguna juga dapat mengirimkan emoji dan stiker untuk mengekspresikan pikiran, emosi atau perasaan, dan bahasa tubuhnya yang mungkin tidak dapat disampaikan dengan hanya sekedar teks. Emoji digunakan bersamaan dengan teks, sedangkan stiker dapat memberikan alternatif visual dari sebuah pesan teks. Pada aplikasi bertukar pesan, seperti LINE, ribuan stiker tersedia untuk diunduh oleh pengguna secara gratis maupun berbayar. Set stiker yang sudah diunduh pengguna akan ditambahkan ke dalam koleksi stiker miliknya dan dapat digunakan saat mengirim pesan dengan mengaksesnya dari kotak obrolan.

Pada bulan September 2014 lalu, LINE merilis pembaruan versi aplikasi ke 4.6.0 yang berisi penambahan fitur-fitur baru pada saat itu. Salah satu fitur yang baru ditambahkan saat itu adalah fitur rekomendasi stiker dan emoji untuk membantu pengguna menemukan stiker dan emoji yang cocok hanya dengan menuliskan teks. Contohnya, dengan mengetikkan kata "senang," pengguna akan ditunjukkan beberapa stiker dan emoji yang memiliki ekspresi senang yang ada pada koleksi milik pengguna. Pada fitur ini, pembuatan rekomendasi stiker harus dilakukan bersamaan dengan ketikan teks dari pengguna sehingga kecepatan dalam pencarian menjadi hal yang harus diperhatikan untuk menghindari penundaan yang cukup terlihat saat mengetik. Selain itu, relevansi stiker yang direkomendasikan juga menjadi hal yang perlu dipertimbangkan.

Algoritma pencocokan string dapat digunakan untuk mengimplementasikan fitur rekomendasi stiker seperti yang sudah dijelaskan sebelumnya. Algoritma ini dipakai untuk mencocokkan teks yang diketik oleh pengguna dengan label stiker yang ditentukan ke tiap stiker secara manual. Beberapa algoritma pencocokan string juga dapat menentukan tingkat kesamaan dari dua buah string sehingga tidak meninggalkan faktor relevansi dari stiker yang ditampilkan.



Gambar 1 Tampilan Fitur Rekomendasi Stiker pada Aplikasi Bertukar Pesan LINE

II. TEORI DASAR

A. *Algoritma Pencocokan String*

Algoritma pencocokan string adalah algoritma yang digunakan untuk mencari lokasi di dalam teks T yang bersesuaian dengan pola P. Pada algoritma ini, panjang dari teks T diasumsikan jauh lebih panjang dari pada pola P. Berikut adalah contoh dari pencocokan string dengan teks T dan pola P.

T: saya sedang senang hari ini
P: senang

Penerapan dari algoritma pencocokan string yang dapat kita temukan sehari-hari adalah fitur pencarian teks di dalam aplikasi editor teks dan mesin pencarian web di internet. Pada aplikasi editor teks, pengguna dapat memasukkan kata kunci yang ingin dicari di dalam dokumen yang sedang dibuka. Dengan ini pengguna tidak perlu membaca keseluruhan untuk mencari sebuah kata. Pada mesin pencarian web di internet, pengguna juga dapat memasukkan kata kunci yang ingin dicari di internet. Tidak seperti fitur pencarian kata pada aplikasi editor teks, teks yang dimasukkan pada mesin pencari web tidak harus sama persis dan tidak perlu terurut. Pencarian kata dengan algoritma pencocokan string ini tentunya perlu dilakukan penyesuaian agar pencarian dapat dilakukan dalam waktu yang singkat.

B. Algoritma Brute Force

Algoritma *brute force* dapat digunakan untuk melakukan pencocokan string. Algoritma ini melakukan pencarian dengan mengecek kesamaan tiap karakter pada teks T dengan pola P. Proses pencarian memiliki arah dari kiri ke kanan dan dilakukan dengan menggeser satu per satu pola P sehingga tiap karakter yang ada di teks T cocok dengan P. Penggeseran ini akan akan terus dilakukan sampai P ditemukan atau sampai P mencapai akhir dari T. Berikut adalah contoh pencarian pola dengan algoritma *brute force* dengan P adalah "sedang" dan T adalah "saya sedang senang hari ini."

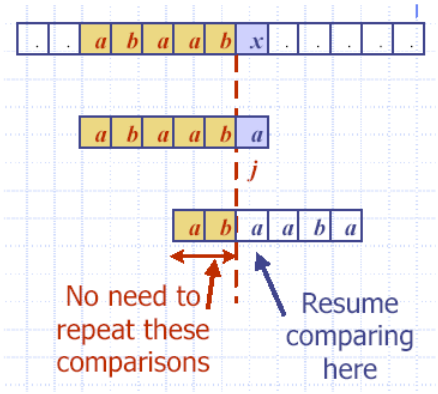
```
   saya sedang senang hari ini
1 sedang
2  sedang
3   sedang
4    sedang
5     sedang
6      sedang
```

Kasus terbaik pada algoritma ini terjadi ketika karakter pertama pola P tidak pernah sama dengan teks T yang dicocokkan. Sedangkan kasus terburuknya adalah ketika semua karakter pada P, kecuali karakter terakhir, selalu cocok dengan T. Contohnya adalah ketika P berisi "ooooh" dan T berisi "ooooooooooooooooooooooh." Algoritma ini cocok untuk digunakan ketika jumlah alfabet dari teks T cukup besar, tetapi tidak cocok jika jumlah alfabetnya kecil. Contoh dari alfabet yang besar adalah seperti alfabet bahasa Indonesia yang terdiri dari 26 karakter, sedangkan alfabet kecil adalah biner yang hanya terdiri dari dua karakter (nol dan satu).

C. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencocokan string. Sama seperti *brute force*, proses pencarian memiliki arah dari kiri ke kanan. Namun, penggeseran yang dilakukan pada algoritma ini mempunyai cara yang lebih pintar dibandingkan algoritma *brute force*.

Jika terjadi sebuah ketidakcocokan pada pola P di teks T pada P[j] dan T[j], penggeseran yang dilakukan pada pola untuk menghindari perbandingan yang berguna adalah penggeseran prefiks (awalan) terbesar pada P[0..j-1] yang merupakan sufiks (akhiran) dari P[1..J-1].



Gambar 2 Contoh Pergeseran Pola pada Algoritma KMP
Sumber: [2]

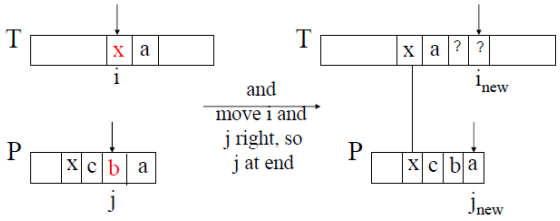
Dalam algoritma ini, terdapat fungsi untuk menghitung banyaknya pergeseran yang dilakukan, yaitu Fungsi Pinggiran KMP. Fungsi ini mengawali proses pada pola untuk mencari kecocokan pada prefiks pola dengan pola itu sendiri. Jika *j* adalah posisi ketidakcocokan pada P[] dan *k* adalah posisi sebelum ketidakcocokan terjadi (*k* = *j*-1), maka fungsi pinggiran *b(k)* didefinisikan sebagai ukuran terbesar prefiks dari P[0..*k*] yang juga merupakan sufiks dari P[1..*k*].

Jika dibandingkan dengan algoritma *brute force*, algoritma KMP memiliki kompleksitas waktu yang jauh lebih cepat. Algoritma ini tidak perlu untuk bergerak mundur pada sebuah input sehingga membuat algoritma ini cocok untuk memproses berkas yang sangat besar yang dibaca melalui perangkat eksternal atau melalui jaringan. Sayangnya, KMP tidak bekerja dengan baik ketika ukuran alfabetnya bertambah.

D. Algoritma Boyer-Moore

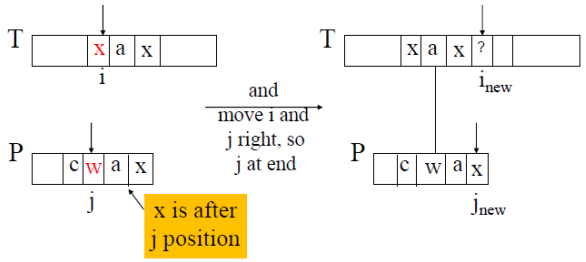
Seperti algoritma KMP, algoritma Boyer-Moore juga merupakan salah satu dari algoritma pencocokan string. Algoritma ini didasarkan pada dua teknik, yaitu *looking-glass* dan *character-jump*. Pada teknik *looking-glass*, pencarian dilakukan dengan bergerak mundur melalui pola P, dimulai dari karakter terakhirnya. Pada teknik *character-jump*, dilakukan lompatan ketika terjadi ketidakcocokan pada T[i] == x, yaitu ketika karakter pada P[j] dan T[i] tidak sama.

Pada algoritma ini, terdapat tiga kemungkinan kasus yang dapat terjadi selama proses pencarian berlangsung. Kasus pertama adalah kondisi ketika terdapat karakter *x* di suatu tempat pada P. Pada kondisi ini akan dicoba untuk menggeser P ke kanan sehingga x pada P sejajar dengan x pada T[i].



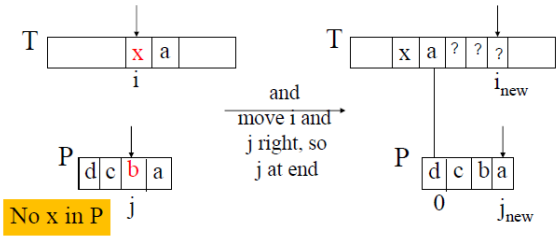
Gambar 3 Kasus 1 Algoritma Boyer-Moore
Sumber: [2]

Kemudian, kasus kedua terjadi ketika terdapat karakter x di suatu tempat pada P, tetapi tidak memungkinkan untuk menggeser P ke kanan hingga kemunculan terakhir x. Pada kasus ini akan dicoba untuk menggeser P ke kanan sebanyak satu karakter hingga T[i+1].



Gambar 4 Kasus 2 Algoritma Boyer-Moore
Sumber: [2]

Terakhir, kasus ketiga terjadi ketika kasus pertama dan kedua tidak berlaku. Pada kasus ini, akan dicoba untuk menggeser P agar P[0] sejajar dengan T[i+1].



Gambar 5 Kasus 3 Algoritma Boyer-Moore
Sumber: [2]

Seperti pada algoritma KMP, algoritma Boyter-Moore juga memiliki fungsi yang dipanggil terlebih dauhlu sebelum melakukan pencarian, yaitu Fungsi Kemunculan Terakhir L(). Fungsi ini memetakan seluruh karakter pada P ke bilangan bulat sesuai dengan kemunculan terakhirnya di P. Fungsi akan mengembalikan -1 jika tidak menemukan suatu karakter pada P.

Algoritma Boyer-Moore lebih cepat secara signifikan dibandingkan *brute force* pada pencarian dengan alfabet besar. Namun, cenderung lambat ketika alfabetnya kecil. Hal ini berkebalikan dengan algoritma KMP.

E. Algoritma Jarak Levenshtein

Algoritma Jarak Levenshtein adalah algoritma yang paling sering dipakai untuk menghitung jarak kedekatan antara dua string. Algoritma ini juga dikenal sebagai algoritma berbasis jarak edit karena algoritma ini menghitung jumlah pengeditan yang dibutuhkan untuk mengubah suatu string ke string lainnya. Pengeditan ini terhitung dengan melakukan operasi penambahan karakter, penghapusan karakter, dan penggantian karakter. Semakin sedikit pengeditan yang diperlukan, maka kedua string tersebut semakin mirip. Contohnya pada string “senang” dan “sedang”, keduanya memiliki jarak Levenshtein sebesar satu karena hanya perlu mengganti karakter *n* dan *d*. Dengan menggunakan algoritma ini, program dapat menentukan tingkat kemiripan dua buah string.

III. PEMBAHASAN

A. Pembuatan Struktur Data Stiker

Data-data yang diperlukan untuk membuat program dibuat dalam bentuk *array of object* yang merepresentasikan kumpulan stiker dalam bahasa Javascript. Objek-objek yang dibuat memiliki atribut *id*, *tags*, dan *path* yang semuanya bertipe data string. *id* adalah angka unik yang dimiliki setiap stiker, *tags* adalah kata kunci yang mendeskripsikan stiker, dan *path* adalah letak dimana berkas gambar stiker berada. Nantinya, pencarian akan dilakukan berdasarkan *tags* atau label yang dimiliki tiap stiker. Pada percobaan ini disediakan 42 stiker contoh yang memiliki berbagai label untuk eksperimen. Gambar-gambar stiker yang disediakan didapat dari koleksi stiker milik penulis yang sudah secara legal dibeli sendiri oleh penulis. Berikut contoh beberapa objek yang dibuat.

```
  {
    id: "34",
    tags: "sedih, sad, nangis, nangis teriak, scream, teriak, nangis,
cry",
    path: cat1,
  },
  {
    id: "35",
    tags: "senang, happy, senyum, smile, heheh",
    path: cat2,
  },
  {
    id: "36",
    tags: "bingung, kaget, terkejut, shock, gasp, hah",
    path: cat3,
  },
```

Gambar 6 Contoh Struktur Objek Stiker

B. Perancangan Program

Algoritma yang digunakan pada program rekomendasi stiker sederhana ini adalah algoritma jarak Levenshtein. Algoritma ini digunakan dengan mempertimbangan relevansi antara stiker yang ditampilkan dengan teks yang diketik oleh pengguna. Dengan algoritma jarak Levenshtein, program dapat menentukan tingkat kecocokan dua buah string sehingga stiker yang ditampilkan bisa diatur dari tingkat kecocokannya. Jika program menggunakan algoritma Boyer-Moore ataupun KMP, ketika pengguna hanya mengetikkan satu karakter, misalnya *s*, nantinya program akan menampilkan semua stiker dengan label yang memiliki karakter s di dalamnya, dan hal ini sudah melenceng dari tujuan dibuatnya fitur rekomendasi stiker. Walaupun algoritma ini terkesan lambat karena kompleksitas waktunya O(*mnp)* dengan *m* dan *n* adalah masing-masing panjang string dan *p* adalah jumlah label yang ada, tetapi jarak Levenshtein ini sesuai dengan tujuan dibuatnya fitur rekomendasi stiker, yaitu memudahkan pengguna memilih stiker berdasarkan teks yang diketiknya. Selain itu, nilai *m* dan *n* ini cenderung tidak terlalu besar sehingga kecepatannya seharusnya tidak terlalu tertinggal jauh dibandingkan menggunakan Boyer-Moore dan KMP.

Keuntungan lain pada penggunaan algoritma jarak Levenshtein adalah memungkinkan program untuk mencari label dengan teks yang tulisannya tidak sama persis dengan label yang ada. Teks yang dimasukkan oleh pengguna sangat mungkin terjadi kesalahan pengetikan, tetapi dengan algoritma

ini, label tetap dapat terdeteksi sampai batas tertentu. Selain itu, dengan algoritma ini juga dapat memungkinkan pencarian dengan teks yang lebih panjang daripada label yang ada.

Program rekomendasi stiker sederhana ini diimplementasikan dengan bahasa Javascript. Terdapat beberapa fungsi yang dibuat untuk mengimplementasikan program ini, yaitu fungsi levenshteinDist, fungsi similarityTest, dan fungsi findStcikers.

```javascript
/**
 * Menghitung jarak Levenshtein antara dua string
 * @param {String} string1
 * @param {String} string2
 * @returns {Integer} Jarak Levenshtein
 */
function levenshteinDist(string1, string2) {
  let m = string1.length;
  let n = string2.length;

  const distance = Array(n + 1)
    .fill(null)
    .map(() => Array(m + 1).fill(null));

  for (let i = 0; i <= m; i++) {
    distance[0][i] = i;
  }

  for (let j = 0; j <= n; j++) {
    distance[j][0] = j;
  }

  var subsCost;
  for (let j = 1; j <= n; j++) {
    for (let i = 1; i <= m; i++) {
      if (string2.charAt(j - 1) === string2.charAt(i - 1)) {
        subsCost = 0;
      } else {
        subsCost = 1;
      }

      distance[j][i] = Math.min(
        distance[j][i - 1] + 1, // deletion
        distance[j - 1][i] + 1, // insertion
        distance[j - 1][i - 1] + subsCost // substitution
      );
    }
  }

  return distance[n][m];
}
```

Gambar 7 Implementasi Fungsi levenshteinDist

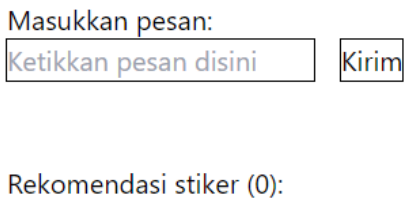
```javascript
/**
 * Mencari tingkat kesamaan antara label dan teks
 * @param {String} label Label stiker
 * @param {String} text  Teks yang ditulis pengguna
 * @returns {Integer} Tingkat kesamaan dalam range 0-100
 */
function similarityTest(label, text) {
  let dist = levenshteinDist(label, text);
  if (label.length > text.length) {
    return ((label.length - dist) / label.length) * 100;
  } else {
    return ((text.length - dist) / text.length) * 100;
  }
}
```

Gambar 8 Implementasi Fungsi similarityTest

```javascript
/**
 * Mencari stiker yang paling cocok dengan teks yang ditulis pengguna
 * @param {String} text Mencari stiker yang cocok dengan teks yang ditulis pengguna
 * @param {String} stickersGallery Daftar stiker yang ada
 * @returns {Array} Daftar stiker yang cocok dengan teks yang ditulis pengguna
 */
function findStickers(text, stickersGallery) {
  let result = [];
  let minSimilarity = 50;

  if (text !== "") {
    for (let sticker of stickersGallery) {
      let tags = sticker.tags.split(", ");

      for (let tag of tags) {
        let similarity = similarityTest(tag, text);
        if (similarity >= minSimilarity) {
          result.push(sticker);
          break;
        }
      }
    }
  }

  return result;
}
```

Gambar 9 Implementasi Fungsi findStickers

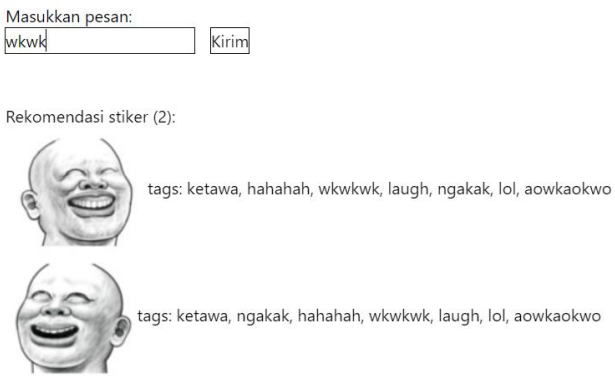
Agar memudahkan percobaan, dibuat juga program berbasis web sisi klien sederhana dengan menggunakan *framework* Reactjs. Berikut adalah tampilan dari program rekomendasi stiker sederhana.



Gambar 10 Tampilan Awal Program Rekomendasi Stiker Sederhana

Pada Gambar 10 dapat dilihat tampilan program terdiri dari dua bagian. Pertama, bagian kotak obrolan sederhana yang hanya dapat menerima masukkan teks. Tombol Kirim di sini hanya bersifat dekorasi dan tidak memiliki fungsionalitas apapun. Bagian kedua, yang terletak di bawah kotak obrolan, adalah daftar rekomendasi stiker yang akan ditampilkan berdasarkan teks yang dimasukkan.

Ketika pengguna mengetikkan teks pada kotak obrolan, akan muncul nol hingga beberapa stiker yang salah satu labelnya memiliki tingkat kecocokan lebih dari atau sama dengan 50%. Gambar dari stiker dan juga *tags* atau label-label yang dimiliki masing-masing stiker juga ditampilkan untuk memudahkan pengamatan.



Gambar 11 Tampilan Saat Mengetikkan Sesuatu pada Kotak Obrolan

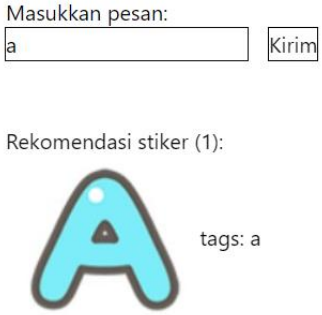
B. Percobaan

Program rekomendasi stiker sederhana ini dapat dijalankan secara lokal dengan mengunduhnya terlebih dahulu melalui tautan yang ditulis pada akhir makalah ini. Setelah program diunduh dalam bentuk zip, ekstrak berkas zip tersebut. Kemudian, buka *command line* dan arahkan direktori ke folder *sticker-suggestions*. Terakhir, jalankan perintah berikut pada *command line*.

```
npm start
```

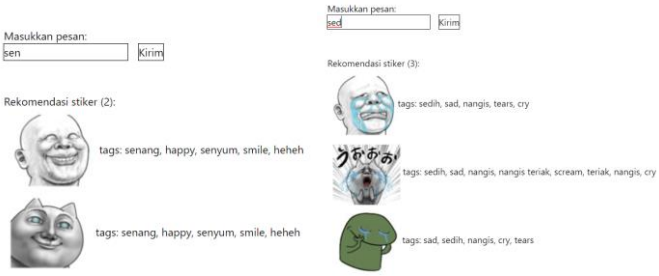
Setelah perintah dijalankan, komputer akan secara otomatis membuka halaman web di dalam browser. Untuk menghentikan program, kembali ke *command line* dan tekan CTRL + C pada keyboard untuk sistem operasi Windows. Percobaan baru dapat dilakukan setelah program terbuka di browser.

Untuk percobaan pertama, program diuji dengan memasukkan string berisi satu karakter. Hal ini dilakukan untuk membuktikan bahwa dengan algoritma jarak Levenshtein, stiker-stiker yang direkomendasikan bukanlah semua stiker dengan label yang memiliki satu karakter tersebut di dalamnya, melainkan benar-benar stiker dengan label satu karakter. Sesuai harapan, ketika kotak obrolan dimasukkan karakter “a”, program hanya merekomendasikan stiker yang memiliki gambar huruf A dengan label “a”, dan bukan semua stiker dengan label yang memiliki huruf “a” di dalamnya.



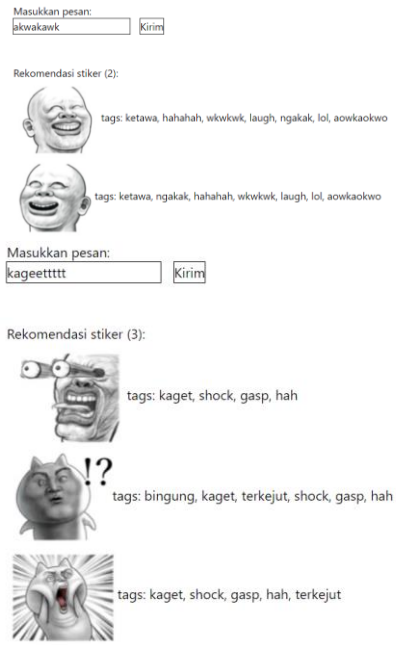
Gambar 12 Percobaan Satu Memasukkan Teks Satu Karakter

Untuk percobaan kedua, program diuji dengan memasukkan kata yang belum lengkap. Percobaan ini dilakukan untuk membuktikan bahwa algoritma Levenshtein juga dapat memprediksi secara kasar kata yang akan pengguna ketik. Kotak obrolan dicoba untuk dimasukkan kata “sen” dan “sed” yang harapannya akan memprediksi kata “senang” dan “sedih”. Ketika dicoba, masing-masing kata memunculkan rekomendasi stiker yang salah satu labelnya, secara berturut-turut, adalah “senang” dan “sedih”. Seperti yang sudah diharapkan, program dapat memprediksi secara kasar apa yang akan pengguna ketik dan langsung menampilkan rekomendasi stiker yang sesuai.



Gambar 13 Percobaan Dua Memasukkan Potongan Teks yang Belum Lengkap

Untuk percobaan ketiga, program diuji dengan memasukkan kata yang isi karakternya tidak sama persis dengan label yang ada dan memiliki panjang karakter yang mungkin melebihi dari label yang ada. Percobaan ini dilakukan untuk membuktikan bahwa program masih dapat mentoleransi perbedaan ketikan pengguna dengan label yang ada. Ketika kotak obrolan dimasukkan kata “akwakawk”, program masih dapat merekomendasikan stiker-stiker dengan label “wkwkwk” dan “aowkaokwo”. Selain itu, ketika kotak obrolan dimasukkan kata “kageetttttt”, program juga masih dapat menampilkan stiker-stiker dengan label “kaget”. Dengan algoritma jarak Levenshtein, perbedaan atau kesalahan pengguna dalam mengetik masih dapat ditoleransi sampai batas tertentu sehingga perekomendasian stiker ini terkesan lebih natural dan tidak kaku.



Gambar 14 Percobaan Tiga Memasukkan Teks yang Mendekati Suatu Label

Terakhir, progran diuji dengan memasukkan kata yang merupakan substring dari dua label berbeda. Ketika kotak obrolan dimasukkan kata “nang”, pengguna mungkin mengharapkan akan direkomendasikan stiker-stiker dengan salah satu labelnya adalah “nangis”. Ternyata, program tidak hanya menampilkan stiker-stiker dengan label “nangis”, tetapi juga menampilkan stiker-stiker dengan label “senang”. Hal ini dikarenakan program hanya bergantung pada algoritma Levenshtein sebagai pencocok teks dengan label sehingga stiker yang direkomendasikan tidak melihat konteks yang sedang dibicarakan oleh pengguna dan lawan bicara.

Gambar 15 Percobaan Empat Memasukkan Potongan Teks yang Merupakan Substring dari Dua Label Berbeda

IV. Kesimpulan

Algoritma pencocokan string jarak Levenshtein dapat digunakan dalam merekomendasikan stiker pada aplikasi perpesanan dengan mencocokkan teks yang dimasukan pengguna dengan label-label yang dimiliki tiap stiker. Kelebihan dari penggunaan algoritma jarak Levenshtein adalah dapat mencocokkan teks dengan tingkat kecocokan tertentu. Dengan ini, program dapat hanya menampilkan stiker dengan label satu karakter, program dapat memprediksi secara kasar kata yang akan diketikkan pengguna, dan teks yang dimasukkan untuk mencari stiker tidak perlu sama persis dengan label yang ada. Kekurangannya adalah walaupun program dapat memprediksi kata yang baru sebagian dimasukkan, tetapi prediksi yang ditampilkan tidak akan sesuai dengan konteks yang dibicarakan pengguna dan lawan bicaranya.

V. Tautan Unduh Program dan Video Youtube

Program rekomendasi stiker sederhana dapat diunduh melalui tautan berikut https://github.com/mhelmih/sticker-suggestions. Sedangkan video penjelasan singkat terhadap makalah ini dapat diakses melalui tautan berikut https://youtu.be/7jMaAZ04HxI.

VI. Ucapan Terima Kasih

Penulis ucapkan terima kasih kepada Allah Swt. karena berkat rahmat dan karunianya penulis mampu menyelesaikan makalah yang berjudul “Penerapan Algoritma Pencocokan String pada Fitur Rekomendasi Stiker dalam Aplikasi Perpesanan”. Kemudian, penulis ingin berterima kasih kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen mata kuliah IF2211 Strategi Algoritma atas segala ilmu dan pedoman yang telah diberikan. Tak lupa juga penulis berterima kasih kepada kedua orang tua serta teman-teman penulis yang selalu mendukung dalam menjalani perkuliahan di kampus.

Referensi

[1] *Announcing the Release of LINE Ver. 4.6.0 With the New Suggest Feature That Helps You Find the Perfect Stickers for Your Chats!* (2 September 2014). LINE official blog. Diakses pada 22 Mei 2022, dari https://official-blog-en.line.me/archives/1007281568.html

[2] Munir, R. (2021). *Pencocokan String (String/Pattern Matching)*. Program Studi Teknik Informatika STEI-ITB. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

[3] Seth, N., & Guide, S. (24 Februari 2021). *A Simple Guide to Metrics for Calculating String Similarity*. Analytics Vidhya. Diakses 22 Mei 2022, dari https://www.analyticsvidhya.com/blog/2021/02/a-simple-guide-to-metrics-for-calculating-string-similarity/

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, 22 Mei 2022

Muhammad Helmi Hibatullah
13520014