# String Matching dalam Aplikasi Memrise

Bryan Rinaldo 13519103
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519103@std.stei.itb.ac.id

***Abstract*—Pada dunia pekerjaan sekarang, menguasai banyak bahasa menjadi nilai lebih sehingga banyak orang yang berlomba untuk menguasai banyak bahasa. Kebanyakan metode yang dipakai untuk belajar bahasa membosankan sehingga orang-orang malas untuk memahaminya. Tetapi ada aplikasi yang dapat membantu kita memahami bahasa asing dan dibawakan secara menarik, salah satunya adalah aplikasi Memrise. Dengan Memrise, kita bisa mempelajari beberapa bahasa seperti Jerman, Perancis, Mandarin, dan masih banyak bahasa lain yang bisa kita pelajari. Pada dasarnya aplikasi ini beroperasi dengan menggunakan String Matching.**

***Kata Kunci—Bahasa, String Matching, Memrise***

## I. PENDAHULUAN

Dimana pun kita berad, pasti kita harus berkomunikasi, karena pada dasarnya kita manusia adalah makhluk sosial. Setiap hari kita pasti berkomunikasi, baik dalam lingkugan keluarga maupun lingkungan pekerjaan bahkan pendidikan. Tetapi terkadang bahasa menjadi penghalang dari komunikasi yang kita jalani. Dalam keluarga mungkin tidak mungkin terjadi penghalang ini, tetapi dalam dunia pendidikan bahkan yang lebih penting yaitu dunia pekerjaan, penghalang ini sangat mungkin terjadi jika kita tidak bisa mengikuti perkembangan Zaman.

Pada Zaman sekarang,kita dituntut untuk setidaknya menguasai 2 bahasa yaitu bahasa Indonesia dan bahasa Inggris. Pada kurikulum Indonesia, bahasa inggris sudah menjadi bahasa yang wajib untuk dikuasai. Tetapi sekarang tidak hanya bahasa Inggris saja yang dipakai di seluruh dunia, banyak negara yang sudah mulai masuk ke Indonesia dan secara tidak langsung menuntut kita untuk bisa berbahasa asing selain bahasa Inggris.

Kita yang dituntut untuk bisa menguasai bahasa asing dipaksa untuk belajar agar bisa menguasai bahasa tersebut. Banyak cara untuk memahami bahasa asing mulai dari belajar dari youtube, les privat, atau bahkan dari buku. Tetapi dengan perkembangan zaman, kita bisa belajar bahasa asing dengan cara yang menarik dengan menggunakan aplikasi belajar bahasa seperti Memrise.

Memrise adalah aplikasi yang dapat berjalan di platform web maupun mobile atau yang biasa kita sebut dengan multiplatform app. Dengan adanya multipaltform ini memudahkan kita saat kita ingin belajar di rumah dengan menggunakan komputer atau saat kita berada di luar dan ingin belajar kita bisa menggunakan aplikasi mobile.


Gambar 1.1 Logo Memrise Sumber :
https://images.app.goo.gl/HEYLoTX4uaMf5MYp6

Dalam mempelajari bahasa apapun, kita tidak akan terlepas dari mempelajari cara pengucapan kata ataupun kalimat dan cara penulisan dari bahasa yang kita pelajari. Memrise menyediakan layanan pembelajaran bahasa dengan metode yang menarik dan mudah dimengerti. Dengan Memrise kita bisa belajar kosakata melalui gambar, kalimat sehari-hari, dan bahkan cara untuk mengucapkan kata-kata pada bahasa yang ingin kita pelajari.

Dengan Memrise, kita akan diberikan soal-soal agar bisa melatih bahasa yang kita pelajari. Pertanyaan yang diberikan di Memrise sangat beragam ada yang menanyakan arti dari sebuah kosakata yang terdapat di layar, ada yang menanyakan kosakata dari suara, dan ada juga yang meminta kita untuk memasukkan tulisan untuk menjawab soal. Dari banyaknya fitur yang diberikan oleh Memrise kepada pengguna, makalah ini akan membahas secara spesifik pada soal yang mengaharuskan pengguna Memrise memasukkan jawaban latihan.

Dari soal tersebut, makalah ini akan membahas secara lengkap penerapan String Matching pada aplikasi Memrise ini.

## II. DASAR TEORI

Algoritma string matching atau yang biasa disebut pencocokan string adalah algoritma untuk melakukan pencarian semua kemunculan string pendek atau pattern yang muncul dalam teks. Teks adalah long string yang panjangnya n karakter, sedangkan pattern adalah string pendek dengan panjang m < n.

Sebenarnya banyak sekali algoritma string matching seperti Algoritma Finite automata, Rabin-Karp, Boyer-Moore-Horspool, KnuthMorris-Pratt, dan banyak lainnya. Akan tetapi, di makalah ini algoritma yang akan dibahas adalah algoritma yang telah diajarkan pada saat perkuliahan IF, yaitu Brute Force, Knuth-Morris-Pratt, dan Boyer-Moore.

### 2.1 Algoritma Brute Force

Algoritma pencocokan string atau string matching yang paling simpel dan paling mudah untuk dimengerti adalah algoritma brute force. Seperti yang kita ketahui Algoritma Brute Force ini dikerjakan tanpa memikirkan kinerja atau performa dari suatu sistem. Sama seperti pada pencocokan string, performa tidak dipedulikan dalam algoritma brute force. Algoritma ini akan terus menysuri teks hingga ditemukan pattern yang cocok. Algoritma brute force akan melakukan cek pada setiap posisi didalam teks apakah ada pattern yang muncul pada posisi yang di cek.

Andaikan diberikan sebuah teks (T) dan sebuah pattern (P) yang ingin dicari. Kita bisa melihat cara kerja dari algoritma brute force adalah sebagai berikut:

1. Pertama, pattern P dicocokkan dengan awal teks T.
2. Pencocokan dilakukan untuk setiap karakter P dengan teks T dari kiri ke kanan sampai seluruh karakter dalam P yang dicocokkan dengan T sudah sama atau ditemukan ketidakcocokan antara karakter dalam P dan karakter dalam T.
3. Jika ketidakcocokan antara P dengan T ditemukan dan karakter dalam T belum habis, P digeser sebanyak 1 karakter, kemudian langkah 2 dilakukan kembali (pencocokan string kembali dilakukan).

Algoritma ini bisa dikatakan sebagai algortima yang paling dan sangat naif karena algoritma ini akan melakukan pencocokan setiap karakter dalam pattern dengan setiap karakter yang terdapat di dalam teks hingga ditemukan sebuah kecocokan antara pattern dan string yang ada di dalam teks. Algoritma ini akan memiliki waktu komputasi yang cukup lama bahkan sangat lama karena harus melakukan pencocokan karakter satu persatu.

Dengan menggunakan algortima brute force kita bisa sangat memakan waktu apalagi ketika kita dipertemukan dengan kasus terburuk. Kasus terburuk dalam penggunaan algoritma brute force terjadi apabila sebagian karakter dalam pattern yang ingin kita cari selalu cocok dengan teks yang diberikan. Ketika semua sudah cocok, di akhir pattern akan ada sebuah huruf yang membuat proses pencarian diulang. Contoh dari kasus terburuk pada brute force adalah sebagai berikut:

```
Teks : AOAOAOAOAOAOAOAOAOAOH
Pattern : AOAOH
```

Pada kasus ini, akan selalu diemukan kecocokan dalam pencocokan huruf A dan O dengan teks yang diberikan sebanyak 2 kali lalu selalu gagal dalam pencocokan huruf H. Ini berarti pencocokan akan selalu terjadi sebanyak 5 kali yaitu sepanjang pattern yang dicari dikalikan dengan banyaknya pattern maju untuk menemukan kecocokan dengan substring dalam teks, yaitu sebanyak n-m+1 kali.

Meskipun begitu, terdapat juga kasus terbaik yang dapat diterima oleh algoritma brute force yang bergantung pada panjang karakter dari teks yang diberikan. Kasus terbaik pada algoritma brute force bisa terjadi apabila karakter pertama pada pattern yang diberikan tidak muncul sama sekali dengan karakter yang ada pada teks dan karakter tersebut hanya muncul pada string yang kita ingin cari . Contoh dari kasus terbaik pada brute force adalah sebagai berikut:

```
Teks : Nama saya adalah Charles Xavier
Pattern : Xavier
```

Huruf X di dalam teks pasti tidak ditemukan hingga ditemukan string dalam teks yaitu “Xavier” pada teks yang bersangkutan. Jumlah perbandingan yang terjadi disini adalah sebanyak n kali. Oleh karena itu, ini adalah kasus terbaik yang mungkin terjadi dalam penggunaan algoritma brute force.

Kasus terbaik dan kasus terburuk sudah dibahas dan sudah dijelaskan secara rinci. Berikut adalah kasus biasa yang bisa saja terjadi dari penggunaan algortima brute force:

```
Teks: NOBODY NOTICED HIM
Pattern: NOT

  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

Gambar 2.1 penggunaan algoritma brute force Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Setiap karakter dalam teks akan dicocokan secara satu persatu. Ketika ada karakter yang berbeda maka pencocokan akan maju pada teks. Tetapi jika pada karakter dalam teks cocok dengan karakter dalam pattern, maka karakter akan maju pada pattern.

Berikut adalah potongan kode program dalam bahasa java untuk melakukan string matching menggunakan algoritma brute force:

```
public static int brute(String
text,String pattern)
{
   int n = text.length();
   int m = pattern.length();
   int j;
   for(int i=0; i <= (n-m); i++)
   {
      j = 0;
      while((j<m)&&(text.charAt(i+j)==
      pattern.charAt(j)))
      {
         j++;
      }
      if (j == m)
      {
          return i;
      }
   }
  return -1;
}
```

## 2.2 Algoritma Knuth-Morris-Pratt (KMP)

Algortima Knuth-Morris-Pratt atau yang biasa disingkat menjadi KMP merupakan algoritma string matching atau pencocokan string yang jauh lebih pintar dan lebih efektif dibandingkan dengan algoritma brute force. Algortima Knuth-Morris-Pratt atau KMP ini pertama kali diperkenalkan oleh seorang *computer scientist* sekaligus professor di Universitas Stanford bernama Donald Ervin Knuth bersama-sama dengan J. H. Morris dan V. R. Pratt.

Algoritma ini memiliki kemiripan dengan algoritma brute force. Bisa dikatakan bahwa Algoritma ini merupakan hasil modifikasi dari algoritma Brute Force. Pada algoritma Brute Force, setiap kali ditemukan ketidakcocokan pattern dengan teks, maka pattern digeser satu karakter ke kanan. Sedangkan pada algoritma Knuth-MorrisPratt, kita memanfaatkan informasi yang digunakan untuk melakukan sejumlah pergeseran. Algoritma menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter per karakter seperti pada algoritma Brute Force. Dengan memanfaatkan algoritma Knuth-Morris-Pratt ini, waktu pencarian menjadi lebih singkat dan pencarian menjadi lebih efektif dan efisien.

Kita bisa melihat cara kerja dari algoritma KMP adalah sebagai berikut:

1. Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.
2. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   - Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
   - Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser pattern berdasarkan tabel next, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

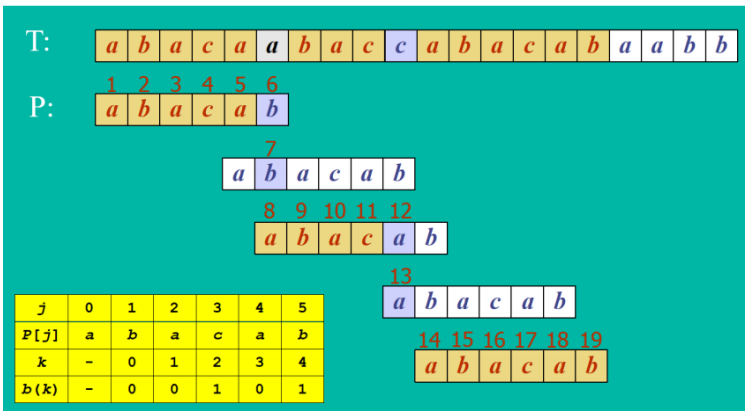
Dalam algoritma KMP, kita membutuhkan sebuah tabel next atau yang biasa disebut sebagai fungsi pinggiran (Border Function) yang menjadi patokan kita kemana kita harus menggeser pattern yang kita cari pada teks. Untuk mendapatkan fungsi pinggiran tersebut kita hanya perlu mencari dimana karakter pada pattern muncul dalam pattern pada karakter awal.

Andaikan diberikan pattern “ababaa”, kita bisa melihat fungsi pinggirannya adalah sebagai berikut:

| j | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| P[j] | a | b | a | b | a | a |
| b(j) | 0 | 0 | 1 | 2 | 3 | 1 |

Tabel 2.1 fungsi pinggiran dengan pattern “ababaa”

Dari tabel di atas kita bisa melihat pada indeks 3 bahwa “a” sudah muncul pada indeks 1 oleh karena itu pada b(j) ditulis indeks ke 1. Lalu lanjut ke indeks ke 4 kita mencari apakah setelah indeks 1 yaitu “a” muncul “b” seperti indeks 4, jika iya maka b(j) dituliskan indeks 2. Pada indeks 5 setelah “ab” muncul a pada indeks 1,2,dan 3 maka indeks 5 diberikan indeks 3. Tetapi pada indeks 6 tidak karena setelah “aba” bukan “a” melainkan “b”.




Gambar 2.2 penggunaan algoritma KMP Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Berikut adalah potongan kode program dalam bahasa java untuk melakukan string matching menggunakan algoritma KMP yang dibagi dua menjadi kmpMatch dan computeFail:

```
public static int kmpMatch(String text,
String pattern)
{
   int n = text.length();
   int m = pattern.length();
   int fail[] = computeFail(pattern);
   int i=0;
   int j=0;
   while (i < n)
   {
      if (pattern.charAt(j) ==
      text.charAt(i))
      {
         if (j == m - 1)
         {
            return i - m + 1; // match
         }
         i++;
         j++;
      }
      else if (j > 0){
         j = fail[j-1];
      }
      else{
      i++;
      }
   }
   return -1; // no match
} // end of kmpMatch()
```

```
public static int[] computeFail(String
pattern)
{
   int        fail[]        =        new
   int[pattern.length()]; fail[0] = 0;
   int m = pattern.length();
   int j = 0;
   int i = 1;
   while (i < m)
   {
      if (pattern.charAt(j) ==
      pattern.charAt(i))
      {
         fail[i] = j + 1;
         i++;
         j++;
      }
      else if (j > 0){
      j = fail[j-1];
      }
      else{
      fail[i] = 0;
      i++;
      }
   } return fail;
} // end of computeFail()
```

Dengan menggunakan algoritma KMP dalam mencari sebuah pattern dalam teks, tidak perlu lagi melakukan penggeseran mundur dalam teks seperti yang dilakukan pada algoritma brute force. Dengan begini aloritma menjadi lebih baik dalam memproses data yang sangat besar. Tetapi dengan KMP ketika size dari banyaknya huruf meningkat, maka algoritma ini bisa menjadi kacau dan menjadi salah.

### 2.3 Algoritma Boyer-Moore

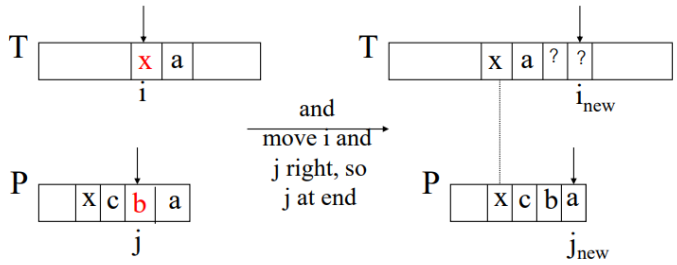
Algoritma Boyer-Moore adalah salah satu algoritma yang menurut banyak orang dianggap sebagai algoritma pencocokan string paling efisien pada penggunaan biasa. Tidak seperti algoritma string matching yang telah dibahas sebelumnya, algoritma Boyer-Moore mulai mencocokkan karakter dari sebelah kanan karakter. Dengan mencocokkan karakter dari kanan, akan lebih banyak informasi yang didapat.

Algoritma pencocokan string Boyer-Moore didasarkan atas dua teknik :

1. Teknik looking-glass, menemukan pattern di dalam teks dengan menggerakan pattern mundur dimulai dari akhir teks.
2. Teknik character-jump, pergeseran karakter yang dilakukan saat terjadi ketidak cocokan

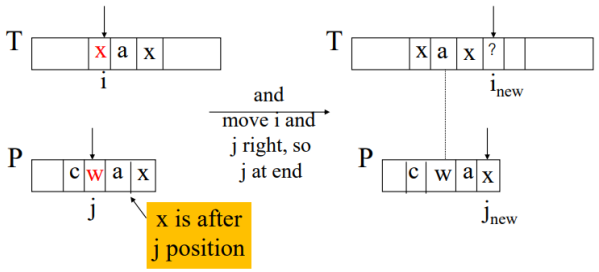
Untuk melakukan teknik character-jump terdapat kondisi-kondisi yang menentukan berapa banyak dilakukan penggesaran. Tidak seperti algoritma brute force yang melakukan penggeseran secara satu persatu, boyer-moore memiliki kondisi tertentu agar penggeseran lebih efisien sehingga waktu yang dibutuhkan lebih sedikit. Kondisi yang menentukan character-jump:

1. Jika pattern P memiliki karakter x di indeks sebelah kiri dari j, maka geser P ke kanan sedemikian sehingga indeks P sejajar dengan indeks i dimana x terdekat berada.



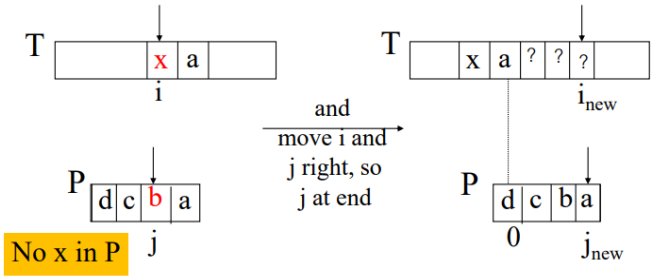

Gambar 2.3 kondisi pertama character-jump Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

2. Jika pattern P memiliki karakter x bukan di indeks sebelah kiri dari j sehingga tidak memungkin untuk menggeser hingga sejajar, maka geser P ke kanan sebanyak 1 karakter.




Gambar 2.4 kondisi kedua character-jump Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

3. Jika yang terjadi tidak termasuk kasus 1 maupun kasus 2, geser pattern sehingga P[1] sejajar dengan T[i+1].




Gambar 2.5 kondisi ketiga character-jump Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Sama seperti algoritma brute force, algoritma boyer-moore juga memiliki kasus terbaik dan kasus terburuk. Kasus terbaik terjadi jika karakter pada teks dan pada pattern tidak pernah sama dengan karakter teks yang dicocokkan, kecuali pada pencocokkan yang terakhir. Contoh kasus terbaik algoritma booyer-moore:

> Teks : Xavier dipanggil sebagai professor X
> Pattern : Xavier

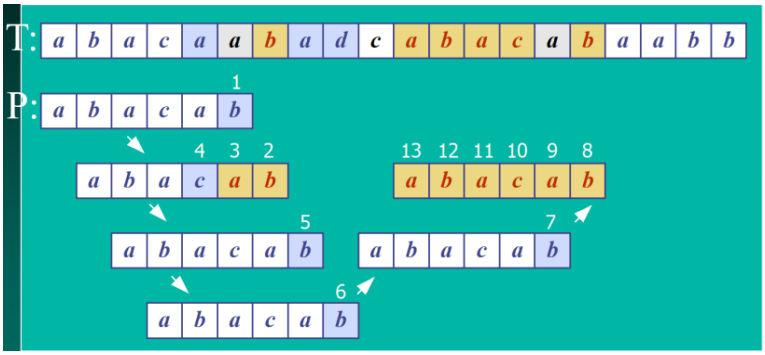
Huruf X di dalam teks pasti tidak ditemukan hingga karakter terakhir dari teks tersebut. Jumlah perbandingan yang terjadi disini adalah sebanyak n kali. Oleh karena itu, ini adalah kasus terbaik yang mungkin terjadi dalam penggunaan algoritma booyer-moore.

Kasus terburuk dalam penggunaan algoritma booyer-moore terjadi apabila sebagian karakter dalam pattern yang ingin kita cari selalu cocok dengan teks yang diberikan. Karena pencocokan dilakukan dari kanan ke kiri, ketika karakter pertama berbeda, maka akan dilakukan penggeseran terus hingga ditemukan. Contoh dari kasus terburuk pada boyer-moore adalah sebagai berikut:

> Teks : haaaaaaaaaaaaaaaaaaaaaaaaaa
> Pattern : haaaa

Pada kasus ini, akan selalu diemukan kecocokan dalam pencocokan huruf "a" dengan teks yang diberikan sebanyak 4 kali lalu selalu gagal dalam pencocokan huruf "h". Ini berarti pencocokan akan selalu terjadi sebanyak 5 kali yaitu sepanjang pattern yang dicari lalu digeser.

Berikut adalah contoh dari pencocokan string atau string matching pada kasus biasa menggunakan algoritma boyer-moore:




Gambar 2.6 contoh penggunaan algoritma boyer-moore Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

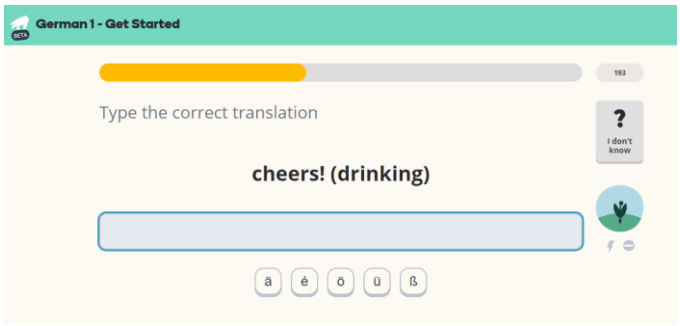
Berikut adalah potongan kode program dalam bahasa java untuk melakukan string matching menggunakan algoritma boyer-moore yang dibagi dua menjadi bmMatch dan buildLast:

```
public static int bmMatch(String text,
String pattern)
{
   int last[] = buildLast(pattern);
   int n = text.length();
   int m = pattern.length();
   int i = m-1;
   if (i > n-1)
      return -1
   int j = m-1;
   do {
      if (pattern.charAt(j) ==
      text.charAt(i))
      {
         if (j == 0){
            return i; // match
         }
         else    {    //    looking-glass
         technique
            i--;
            j--;
         }
      }
      else{ // character jump technique
         int lo = last[text.charAt(i)];
         i = i + m - Math.min(j, 1+lo);
         j = m - 1;
      }
   }
   while (i <= n-1);
   return -1; // no match
} // end of bmMatch()
```

```
public static int[] buildLast(String
pattern)
{
   int last[] = new int[128];
   for(int i=0; i < 128; i++){
      last[i] = -1; // initialize array
   }
   for (int i = 0; i <
   pattern.length(); i++){
      last[pattern.charAt(i)] = i;
   }
   return last;
} // end of buildLast()
```

## III. Analisis

Pada aplikasi Memrise sendiri, memiliki banyak jenis pertanyaan yang harus kita jawab. Tetapi seperti yang sudah dibahas pada pendahuluan, makalah ini akan fokus pada pertanyaan yang meminta input dari user untuk menjawab. Dari input user, aplikasi akan mencoba untuk mencocokan dengan jawaban yang benar.




Gambar 3.1 contoh pertanyaan memrise yang meminta input dari user untuk menjawab pertanyaan

Dari gambar 3.1 di atas kita bisa melihat, aplikasi Memrise memberikan kalimat “cheers!(drinking)” sebagai pertanyaan untuk dijawab oleh user. Jika kita menuliskan jawaban kedalam kotak yang disediakan, aplikasi akan melakukan pengecekan apakah kata yang diketikkan benar atau salah.


Gambar 3.2 contoh pertanyaan memrise yang meminta input dari user untuk menjawab pertanyaan dengan jawaban salah

Jika ternyata input yang dimasukkan pengguna tidak sesuai dengan jawaban seharusnya, maka aplikasi akan mengurangi poin yang didapatkan dan menandai dengan warna merah. Dalam kasus di atas, “cheers!(drinking)” dalam bahasa Jerman bukan disebut sebagai “pröst”.


Gambar 3.1 contoh pertanyaan memrise yang meminta input dari user untuk menjawab pertanyaan dengan jawaban benar

Tetapi jika kita memasukkan kata yang benar yaitu “prost”, maka aplikasi Memrise akan menandai dengan warna hijau dan tanda centang. Tidak hanya itu, poin juga akan ditambahkan ke pengguna dan pengguna dapat melanjutkan ke pertanyaan selanjutnya.

Pada saat dilakukan pengecekan pada input yang diberikan user, disinilah terjadi string matching. Apabila input yang kita berikan tidak cocok dengan jawaban yang sudah ditetapkan oleh aplikasi, maka aplikasi akan mengeluarkan tanda tidak cocok atau salah.

Jika melihat dari algoritma string matching yang telah dibahas pada bab 2, saat menggunakan algoritma brute force untuk mencocokan input user dengan jawaban yang ada dalam database Memrise maka karakter akan dicocokan satu per satu dan memakan waktu yang cukup lama. Jika mendapatkan kasus terburuk, maka waktu yang dibutuhkan akan sangat lam dan sangat tidak efisien.

Apabila pencocokkan string atau string matching dilakukan dengan algoritma KMP yang jauh lebih efisien jika dibandingkan dengan algoritma brute force, maka string matching akan jauh lebih cepat karena penggeseran yang dilakukan tidak satu persatu dan lebih efisien. Hal itu juga berlaku jika digunakan string matching dengan Algoritma Boyer-Moore, pencarian akan lebih cepat walaupun ada kemungkinan bahwa menemukan kasus terburuk seperti algoritma brute force.

## IV. Kesimpulan

Dari makalah ini kita bisa melihat bahwa ilmu yang didapatkan dari strategi algoritma dapat mempermudah kita dalam menjalankan kehidupan sehari-hari. Dalam hal ini berfokus pada komunikasi yaitu bahasa. Dengan ada nya string matching kita bisa belajar banyak bahasa asing menggunakan aplikasi belajar bahasa seperti Memrise. Banyak algoritma yang bisa digunakan untuk string matching, masing-masing dari algoritma yang dibahas memiliki kelebihan dan

kekurangan masing-masing. Untuk kasus string matching pada Memrise yang hanya melakukan string matching pada string yang pendek, pemilihan algoritma yang dipakai tidak terlalu berpengaruh.

## V. PENUTUP

Penulis mengucapkan syukur kepada Tuhan yang Maha Esa karena berkat rahmat-Nya penulis dapat menyelesaikan Makalah IF2211 Strategi Algoritma – Sem. II Tahun 2020/2021 makalah ini dengan tepat waktu. Penulis juga mengucapkan terima kasih kepada keluarga dan teman-teman yang sudah memberikan dukungan baik melalui kata-kata maupun doa. Tidak lupa juga penulis mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen mata kuliah IF2211 Strategi Algoritma yang telah memberikan materi untuk penulisan makalah ini. Akhir kata, penulis meminta maaf jika terdapat kesalahan dan kekurangan pada makalah. Penulis juga berharap makalah ini dapat digunakan sebaik mungkin agar berguna bagi masyarakat.

## REFERENSI


[1] https://catatananalgo.wordpress.com/2016/10/02/algoritma-string-matching-pencocokan-string/

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf


## LINK YOUTUBE

https://youtu.be/gxHt2m5M95I

## 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, 26 April 2021

Bryan Rinaldo - 13519103