Pemanfaatan Algoritme Pencocokan String pada Permainan Dalam-Jaringan (*Online*) TypeRacer

Andreas Halim - 13516003
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
Stale.fx@gmail.com

Abstrak—Pengetikan merupakan hal yang lumrah terjadi pada interaksi antara manusia dengan komputer, karena untuk menginstruksikan kepada komputer untuk menuliskan kata-kata atau kalimat yang ingin disampaikan pada komputer. Setiap pengguna komputer memiliki kecepatan mengetik berbeda dan perbedaan ini dimanfaatkan oleh orang untuk dijadikan permainan dan satu sama lain dilombakan. Dalam membuat permainan di komputer tentu diperlukan algoritme agar dapat dimengerti komputer. Algoritme ini disebut algoritme pencocokan string. Algoritme ini bertugas untuk mencocokan setiap string dengan string lainnya sebagai pembanding.

Kata Kunci—Pencocokan-string (*string matching*), Algoritme Brute-Force, Algoritme Knuth-Morris-Pratt (KMP), Algoritme Boyer-Moore

I. PENDAHULUAN

Dewasa ini, hampir semua orang telah mengenal yang namanya teknologi. Menurut KBBI (Kamus Besar Bahasa Indonesia) daring, teknologi adalah keseluruhan sarana untuk menyediakan barang-barang yang diperlukan bagi kelangsungan dan kenyamanan hidup manusia. Sejak zaman dahulu, teknologi sebenarnya sudah ada, namun tidak secanggih sekarang. Tahun 2000-an, masih dapat dikatakan jarang bahwa orang memiliki telepon genggam. Namun, di tahun 2010-an hingga saat ini, hampir semua orang telah memiliki telepon genggam dari yang telepon genggam biasa hingga telepon genggam cerdas. Dapat dilihat bahwa teknologi telah berubah dengan pesat. Manusia zaman dulu masih bisa ingat terhadap waktu yang sedang berjalan, tapi sekarang hampir semua orang sudah lupa terhadap waktu bekerja. Tidak sedikit pekerja yang melebihi jam standar kerja hingga lembur, termasuk mahasiswa Teknik Informatika di Institut Teknologi Bandung ini yang terus-menerus dicekok ilmu dan tugas hingga bergadang. Hal ini lumrah terjadi pada semua orang tidak terkecuali mahasiswa teknik informatika Institut Teknologi Bandung demi memenuhi keinginan para pengguna dan mengembangkan inovasi teknologi yang terpendam.

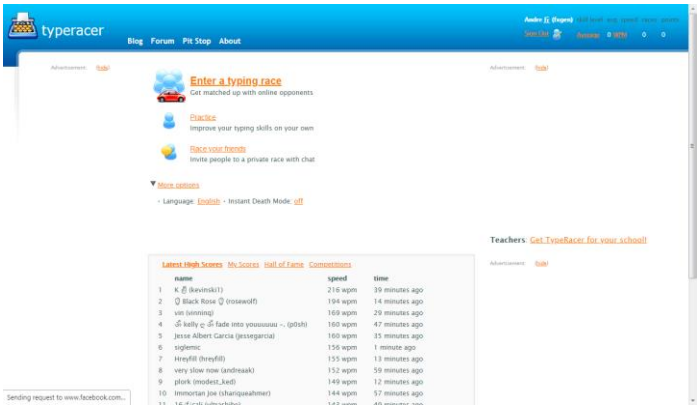
Manusia cenderung ingin terus-menerus terhibur demi mengurangi atau menghindari stres dan depresi di zaman yang berat akan batin ini. Banyak cara untuk mengurangi atau menghindari stres, umumnya adalah mendengarkan musik, menonton film, bertamasya ke tempat wisata, bermain permainan, dan lainnya. Sebagai mahasiswa teknik informatika yang baik walaupun terkadang dirinya juga pernah mengalami stres dan depresi, ingin senantiasa selalu menghibur manusia melalui teknologi yang telah ada yang tinggal dimanfaatkan. Karena sejauh ini hanya terlintas dan dapat dikatakan mampu membuat permainan, maka pada makalah ini akan membahas mengenai permainan yang sekarang ini sepertinya agak terkenal di kalangan mahasiswa teknik informatika Institut Teknologi Bandung. Permainan ini sekaligus melatih tangan para mahasiswa teknik informatika dan tidak terkecuali mahasiswa program studi lain yang diharuskan mengetik, baik untuk mengetik kode program, maupun membuat laporan setiap tugas kecil, tugas besar, dan dokumen lainnya.

Permainan yang akan melatih tangan para mahasiswa jurusan apa pun adalah TypeRacer. TypeRacer merupakan permainan mengetik dalam-jaringan berbasis web. Untuk mengunjungi lamannya langsung dapat mengunjungi alamat http://play.typeracer.com. Permainan ini merupakan permainan yang cara bermainnya adalah dengan mengetik kumpulan kalimat yang telah tercetak di layar. Permainan ini dapat langsung dimainkan tanpa harus mendaftar terlebih dahulu. Akan tetapi jika ingin akses penuh dalam bermain, pemain diharuskan mendaftar agar mendapat hak akses bermain secara penuh. Permainan ini harus dimainkan di laptop atau komputer, agar dapat melakukan kemampuan mengetik dengan maksimal. Permainan ini dapat dimainkan sendiri dan dimainkan bersama dengan individu lainnya (wajib mendaftar terlebih dahulu). Dari cara bermainnya, tersembunyi algoritme untuk menghitung persen kesalahan pengetik dalam mengetik, durasi mengetik, dan yang terutama adalah pencocokan string yang tercetak di layar dengan string yang diketik dari pengguna melalui *keyboard*.

Demi kelancaran permainan ini, haruslah ada algoritme yang dapat berjalan sesuai dengan mekanisme seperti di atas. Sungguh beruntungnya para pemain yang ingin bermain permainan ini, karena algoritme ini telah ditemukan dan wajib dipelajari oleh mahasiswa teknik informatika Institut Teknologi Bandung yang judul mata kuliahnya adalah Strategi Algoritma dengan kode mata kuliah IF2211. Dalam mata kuliah ini, telah dipelajari cara mencocokan string dapat dilakukan dengan berbagai cara dari yang tidak mangkus hingga yang mangkus. Untuk yang tidak mangkus, dapat digunakan algoritme brute-force yang cara kerjanya mengandalkan "otot" saja. Untuk yang mangkus, dapat digunakan algoritme KMP dan kawan-kawannya yang cara kerjanya mengandalkan "otak" dan logika.

Berikut ini merupakan tampilan antar-muka TypeRacer, permainan daring berbasis web yang telah hadir sejak 2008

hingga sekarang.



Gambar 1.1
Tampilan antar-muka halaman depan (beranda) TypeRacer

Sumber Gambar: Dokumentasi Pribadi

Sekadar informasi, permainan ini dibawa dengan menggunakan Bahasa Inggris sebagai bahasa utamanya. Namun, dalam pengetikan saat bermain terdapat banyak bahasa yang salah satunya adalah Bahasa Indonesia, walaupun masih menggunakan mesin terjemahan yang mengandalkan Google Translate.

II. DASAR TEORI

A. Algoritme *Pattern Matching*

Pattern matching adalah sebuah proses pencarian string pada suatu teks. Ada dua elemen dalam persoalan pencarian string, yaitu:

- Teks, merupakan string yang memiliki panjang n karakter.
- *Pattern*, merupakan string yang memiliki panjang m karakter, yang akan dicari keberadaannya pada teks (m < n).

Pada saat pencocokan, akan dicari lokasi pertama dimana *pattern* terletak pada teks. Ada beberapa metode yang digunakan dalam *pattern* matching, yaitu algoritme Brute-Force, algoritme Knuth-Morris-Pratt (KMP), dan algoritme Boyer-Moore.

1. Algoritme Brute-Force

Algoritme Brute-Force adalah algoritme yang memeriksa semua kemungkinan yang ada tanpa memedulikan efisiensi, karena itu algoritme ini sering disebut sebagai algoritme naif. Algoritme Brute-Force ini memiliki langkah – langkah pengerjaan sebagai berikut:

a. Sejajarkan `P[i]` (*pattern*) dengan `T[j]` (teks) dimana `i = j = 1`.

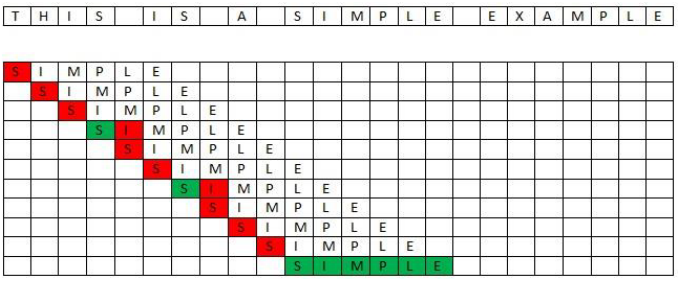
b. Pencocokan *pattern* akan bergerak dari kiri ke kanan dengan mencocokan setiap karakter dari *pattern* yaitu `P[i]` dengan `T[j]`, kemudian `P[i + 1]` dengan `T[j + 1]`, dan seterusnya sampai:

i. *pattern* sudah ditemukan didalam teks, atau

ii. terdapat perbedaan karakter saat pencocokan atau *mismatch*

c. Apabila terjadi *mismatch*, maka ulangi langkah 2 dengan `i = 1`, dan `j = j + 1`.

d. Apabila teks sudah habis dan *pattern* masih belum ditemukan di dalam teks, maka akan mengembalikan nilai yang melambangkan *pattern* tidak ditemukan (*not found*).



Gambar 2.1
Contoh proses Algoritme Pattern String dengan Brute-Force.

Sumber Gambar: **http://1.bp.blogspot.com/-YJDalyxz6XY/UFCYnd2_nBI/AAAAAAAAAB4/uewJpXgs9Mc/s1600/Brute+Force.jpg**

Berikut adalah penerapan algoritme Brute-Force dalam bahasa Java:

```
public static int brute(String text,String pattern)
{ int n = text.length();    // n is length of text
  int m = pattern.length(); // m is length of pattern
  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;   // match at i
  }
  return -1;   // no match
} // end of brute()
```

Gambar 2.2
Algoritme Brute-Force Pattern Matching dalam Bahasa Java

Sumber Gambar: Davidson, Andrew. 2006. *Pattern Matching slide*. Songkhla, Thailand: Prince of Songkhla University (PSU). Diakses melalui www.slideshare.net pada tanggal 10 Mei 2018.

Kompleksitas waktu yang dihasilkan oleh Algoritme Brute-Force berbeda – beda tergantung dari kasus yang ada. Pada kasus terburuk, kompleksitas waktunya adalah $O(mn)$ dengan jumlah pencocokannya atau perbandingan sebanyak $m(n-m+1)$. Kasus tersebut dapat terjadi apabila semua karakter pada *pattern* selalu cocok dengan teks kecuali pada karakter akhir *pattern*, sehingga terdapat

banyak pencocokan atau perbandingan antara *pattern* dengan teks. Contohnnya ketika:

Teks	: aaaaaaaaaaaaaaaaaaaaaa**aaah**
Pattern	: **aaah**

Pada kasus terbaik, kompleksitas waktu yang dihasilkan adalah $O(n)$, terjadi apabila karakter pertama *pattern* tidak pernah sama dengan karakter teks yang dicocokan, kecuali ketika posisi j berada pada karakter ke m – n + 1, Contoh:

Teks	: String ini berakhir dengan **zzz**
Pattern	: **zzz**

Pada kasus biasa, kompleksitas waktunya adalah $O(m + n)$. Pencocokan string dengan Algoritme Brute-Force biasanya cocok pada kalimat yang memiliki karakter yang bervariasi seperti pada teks yang memiliki banyak alfabet yang bervariasi.

2. Algoritme Knuth-Morris-Pratt (KMP)

Algoritme *pattern matching* dengan KMP memiliki sifat yang sama dengan Brute-Force yaitu menelusuri karakter dari kiri ke kanan, tetapi Algoritme KMP memiliki cara yang lebih efisien dibanding dengan Algoritme Brute-Force. Prinsip pada algoritme KMP adalah menentukan jumlah pergeseran *pattern* saat mencocokan karakter dengan teks yang nantinya ada proses tambahan saat mencocokan karakter yaitu dengan memakai fungsi pinggiran.

Sebelum kita melakukan perbandingan antar karakter *pattern* dengan teks, algoritme ini akan menghitung fungis pinggiran tiap – tiap karakter pada *pattern* terlebih dahulu. Fungsi pinggiran nantinya akan membalikan atau *return* sebuah angka yang merepresentasikan ukuran dari prefik terbesar *pattern* yang juga merupakan sufiks dari *pattern*, guna dari fungsi pinggiran adalah untuk menghindari pengulangan pencocokan atau perbandingan prefiks yang ada pada sufiks. Berikut merupakan algoritme untuk menghitung fungsi pinggiran dalam Bahasa Java:

```
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)){  //j+1 chars match
      fail[i] = j + 1;
      i++;
      j++;
    }
    else if (j > 0) // j follows matching prefix
      j = fail[j-1];
    else {     // no match
      fail[i] = 0;
      i++;
    }
  }
    return fail;
} // end of computeFail()
```

Gambar 2.3
Fungsi Pinggiran pada Algoritme KMP

Sumber Gambar: Davidson, Andrew. 2006. *Pattern Matching slide*. Songkhla, Thailand: Prince of Songkhla University (PSU). Diakses melalui www.slideshare.net pada tanggal 10 Mei 2018.

Setelah membentuk fungsi pinggiran, barulah algoritme KMP mulai dijalankan, langkah – langkahnya adalah sebagai berikut:

a. Sejajarkan `P[i]` (*pattern*) dengan `T[j]` (teks) dimana `i = j = 1`.

b. Pencocokan *pattern* akan bergerak dari kiri ke kanan dengan mencocokan setiap karakter dari *pattern* yaitu `P[i]` dengan `T[j]`, kemudian `P[i + 1]` dengan `T[j + 1]`, dst sampai:

   i. *Pattern* sudah ditemukan didalam teks, atau

   ii. Terdapat perbedaan karakter saat pencocokan atau *mismatch*

c. Apabila menemui *mismatch*, *pattern* akan melakukan pergeseran sebanyak indeks karakter terakhir yang cocok dikurangi dengan fungsi pinggirannya karakter terakhir yang cocok tersebut. Jika *mismatch* yang terjadi adalah pada P, maka *pattern* akan geser ke kanan sebanyak 1 karakter.

d. Apabila teks sudah habis dan *pattern* masi belum ditemukan di dalam teks, maka akan membalikan atau return nilai yang melambangkan *pattern not found*.

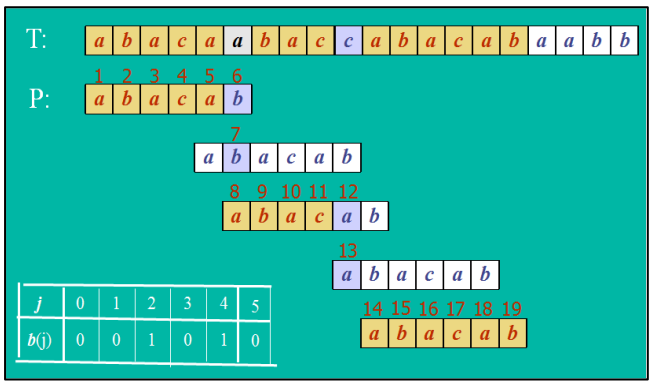
Berikut adalah Algoritme KMP dalam Bahasa Java:

```
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()
```

Gambar 2.4
Algoritme KMP dalam Bahasa Java

Sumber Gambar: Davidson, Andrew. 2006. *Pattern Matching slide*. Songkhla, Thailand: Prince of Songkhla University (PSU). Diakses melalui www.slideshare.net pada tanggal 10 Mei 2018.

Berikut adalah contoh kasus *pattern matching* yang diselesaikan dengan Algoritme KMP:



Gambar 2.5
Contoh kasus yang diselesaikan dengan Algoritme KMP

Sumber Gambar: Davidson, Andrew. 2006. *Pattern Matching slide*. Songkhla, Thailand: Prince of Songkhla University (PSU). Diakses melalui www.slideshare.net pada tanggal 10 Mei 2018.

Kompleksitas waktu yang dihasilkan oleh Algoritme KMP adalah **O(m + n)** dengan **O(m)** untuk mencari fungsi pinggiran dan **O(n)** untuk pencocokan *pattern* dengan teks. Algoritme KMP bagus digunakan ketika variasi karakter pada teks tidak terlalu banyak seperti pencarian sub-DNA pada keseluruhan DNA, tetapi ketika variasi karakter pada teks bertambah, waktu kerja KMP pun semakin lama dan menjadi tidak efisien.

3. Algoritme Boyer-Moore

Algoritme Boyer-Moore merupakan algoritme yang dinilai cukup cepat untuk *pattern matching*. Algoritme ini terdiri dari dua teknik, yaitu yang pertama teknik *looking-glass*, yang memiliki arti bahwa pencocokan *pattern* dengan teks dimulai dari karakter *pattern* akhir ke karakter yang depan atau dapat bilang dari P[m] ke P. Teknik kedua pada Algoritme Boyer-Moore adalah *character-jump*. Teknik ini adalah teknik dalam pergeseran saat pencocokan string dan terbagi menjadi 3 macam kasus, yaitu:

a. Kasus pertama ketika `P[i]` terjadi *mismatch* dengan T[i], tetapi T[i] ada pada sebelah kiri `P[i]`, sehingga dilakukan pergeseran sehingga `P[i]` dan T[i] menjadi sejajar dan tidak lagi *mismatch*.

b. Kasus kedua adalah ketika `P[i]` terjadi *mismatch* dengan T[i], tetapi T[i] tidak ada pada sebelah kiri `P[i]`, melainkan berada pada sebelah kanan `P[i]`, sehingga dilakukan pergesaran sebanyak ke kanan sebanyak 1 kali atau 1 karakter.

c. Kasus ketiga adalah ketika kasus yang terjadi bukanlah kasus pertama dan kasus kedua yaitu ketika `P[i]` terjadi *mismatch* dengan T[i], dan T[i] tidak ada pada sebelah kiri maupun kanan `P[i]`, sehingga dilakukan pergesaran dengan posisi indeks awal *pattern* (P[0]) menjadi indeks ke i + 1 pada T .

Berikut merupakan Algoritme Boyer-Moore dalam Bahasa Java:

```
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;  // no match if pattern is
               // longer than text
 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)];  //last occ
         i = i + m - Math.min(j, 1+lo);
         j = m - 1;
      }
    } while (i <= n-1);

    return -1; // no match
} // end of bmMatch()
```

Gambar 2.6
Algoritme Boyer-Moore dalam Bahasa Java

Sumber Gambar: Davidson, Andrew. 2006. *Pattern Matching slide*. Songkhla, Thailand: Prince of Songkhla University (PSU). Diakses melalui www.slideshare.net pada tanggal 10 Mei 2018.

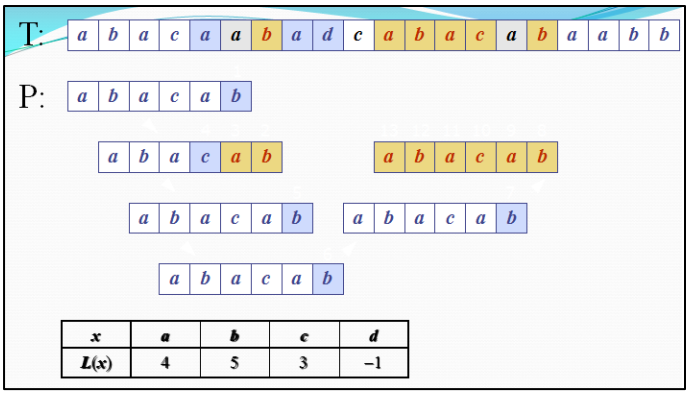
Di sebelumnya, disebutkan bahwa dalam menentukan pergeseran ketika harus tau akan posisi `T[i]` dan `P[i]`, apakah T[i] yang *mismatch* berada di sebelah kiri `P[i]` atau berada disebelah kanan `P[i]` atau bahkan tidak ada di keduanya. Oleh karena itu, dibutuhkan suatu fungsi untuk menghitung letak dari `P[i]`. Fungsi tersebut dinamakan sebagai Fungsi Last Occurrence, yaitu fungsi yang menghitung letak atau posisi terakhir indeks dari karakter – karakter yang ada pada *pattern* dan akan mengembalikan -1 apabila tidak ada pada *pattern*. Berikut adalah Algoritme Fungsi Last Occurrence dalam Bahasa Java:

```
public static int[] buildLast(String pattern)
/* Return array storing index of last
  occurrence of each ASCII char in pattern. */
{
  int last[] = new int[128]; // ASCII char set

  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()
```

Gambar 2.7
Algoritme Fungsi Last Occurrence dalam Bahasa Java

Sumber Gambar: Davidson, Andrew. 2006. *Pattern Matching slide*. Songkhla, Thailand: Prince of Songkhla University (PSU). Diakses melalui www.slideshare.net pada tanggal 10 Mei 2018.

Berikut adalah contoh kasus *pattern matching* yang diselesaikan dengan Algoritme Boyer-Moore:



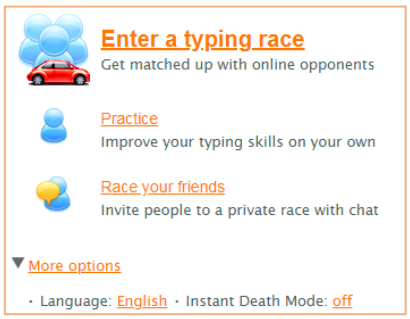
Gambar 2.8
Contoh kasus yang diselesaikan dengan Algoritme Boyer-Moore

Sumber Gambar: Davidson, Andrew. 2006. *Pattern Matching slide*. Songkhla, Thailand: Prince of Songkhla University (PSU) dengan perubahan tampilan.

Kompleksitas waktu terburuk pada Algoritme Boyer-Moore adalah $O(nm + A)$ dengan A merupakan jumlah alphabet yang ada. Algoritme Boyer-Moore bagus digunakan untuk *pattern matching* pada kalimat – kalimat atau artike yang terdapat banyak variasi karakter dan lebih cepat dari Algoritme Brute-Force, tetapi jelek ketika digunakan untuk pencarian seperti pencocokan binary.

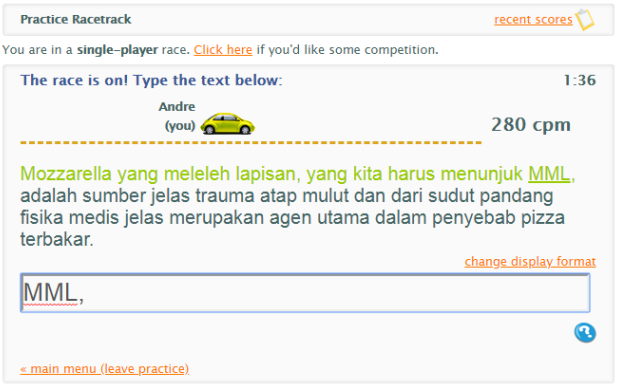
III. PEMBAHASAN PERMAINAN PADA TYPERACER

Untuk dapat memainkan permainan ini, pemain diharuskan mengunjungi laman web TypeRacer. Apabila telah mengunjungi laman web TypeRacer, pemain dapat memilih jenis permainan yang akan dimainkan nantinya. Jika pemain ingin bertanding dengan pemain lainnya, maka pilih "*Enter a typing race*" (Bahasa Indonesia: masuk ke perlombaan mengetik). Akan tetapi, apabila pemain ingin bermain sendiri (berlatih), maka pemain dapat memilih *"Practice"* (Bahasa Indonesia: berlatih). Pemain dapat memilih berbagai bahasa yang dikehendaki. Pada gambar terlihat bahwa Bahasa Inggris (*English*) terpilih sebagai bahasa yang akan digunakan untuk pengetikan nanti. Akan tetapi, karena bahasa nasional negara Indonesia adalah Bahasa Indonesia, maka akan Bahasa Indonesia akan digunakan dalam permainan ini.



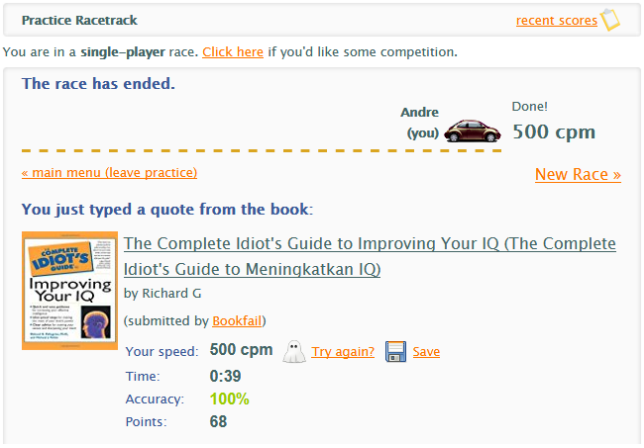
Gambar 3.1
Pilihan jenis permainan pada bagian tengah laman web TypeRacer

Sumber Gambar: Dokumentasi Pribadi



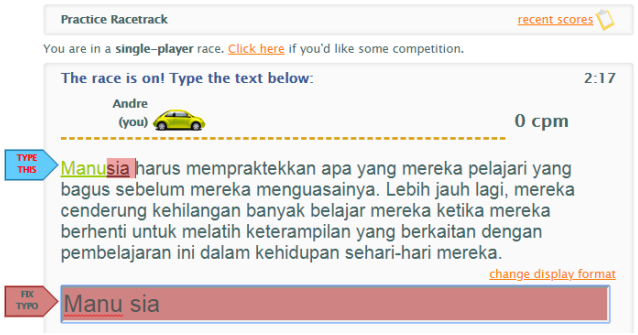
Gambar 3.2
Tampilan permainan saat permainan sedang berlangsung

Sumber Gambar: Dokumentasi Pribadi



Gambar 3.3
Tampilan permainan saat permainan sudah berakhir, beserta dengan hasil penilaian dari TypeRacer

Sumber Gambar: Dokumentasi Pribadi



Gambar 3.4
Tampilan permainan saat permainan berlangsung, namun ditemukan ketidaksamaan (memberi pesan kesalahan)

Sumber Gambar: Dokumentasi Pribadi

Prinsip algoritme pada permainan ini adalah memeriksa setiap karakter untuk setiap katanya (*string*). Untuk setiap kata, akan diperiksa karakter per karakter. Apabila ditemukan kesamaan (tidak ada masalah), maka permainan tidak akan memberi pesan kesalahan dan mewarnai teks yang telah diketik dengan benar dengan warna hijau lengkap dengan garis-bawah (*underline*). Namun, apabila ditemukan kesalahan yakni berupa ketidaksamaan untuk satu atau lebih karakter pada kata, maka permainan ini akan memberikan pesan kesalahan dengan cara menandai daerah yang salah dengan cara menyoroti warna merah untuk karakter yang salah dan seterusnya untuk setiap karakter apabila tidak diperbaiki, kotak teks untuk pengetikan berubah menjadi warna merah, dan akan diberikan anak panah yang memberitahukan pemain untuk mengetik dengan sesuai (pada permainan bertuliskan "*type this*" atau ketik ini).

Seperti yang sudah dijelaskan sebelumnya, terdapat tiga algoritme untuk pencocokan string, walaupun bisa menjadi empat untuk ekspresi reguler (*regular expression*). Namun, ekspresi reguler tidak dimasukkan karena tidak masuk sebagai bagian dari pembelajaran. Pada permainan ini, dipastikan menggunakan dua algoritme, salah satunya adalah algoritme Boyer-Moore dan lainnya lagi kemungkinan algoritme Brute-Force atau Knuth-Morris-Pratt (KMP).

Algoritme Boyer-Moore ini dapat digunakan untuk pengetikan dalam bahasa yang penulisannya menuliskan dari kanan, seperti bahasa Arab. Berikut ini merupakan contoh penulisan bahasa yang menggunakan penulisan dari kanan (bahasa Arab):

Gambar 3.5
Pengetikan yang dimulai dari kanan

Sumber Gambar: Dokumentasi Pribadi

Pada gambar tersebut, jelas terdapat pesan kesalahan yang diawali dari kanan. Di sinilah algoritme Boyer-Moore diimplementasikan.

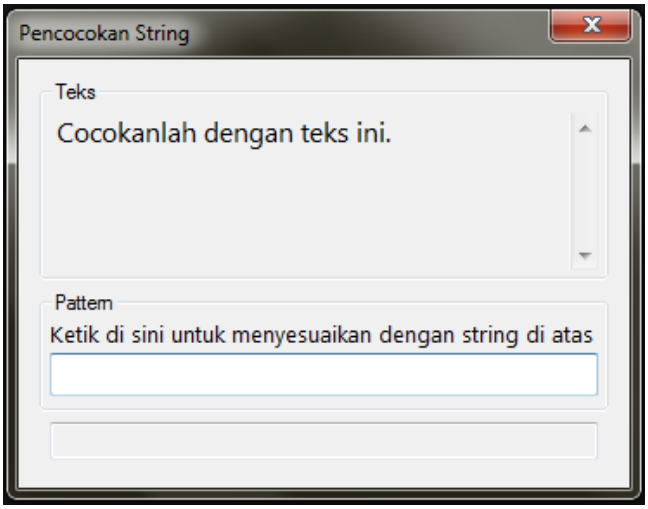
Di dunia ini, umumnya penulisan bahasa dimulai dari kiri. Sehingga, algoritme Boyer-Moore tidak dapat digunakan, karena algoritme Boyer-Moore memeriksa string dimulai dari kanan. Algoritme yang cocok untuk penulisan dari kiri adalah algoritme Brute-Force atau algoritme Knuth-Morris-Pratt. Namun, tidak menutup kemungkinan algoritme Brute-Force dapat digunakan.

IV. IMPLEMENTASI PADA PROGRAM KECIL

Prinsip pada program kecil yang dibuat ini adalah mencocokan setiap string yang diketik dengan setiap string yang tersedia. Pembacaan diakhiri dengan mengetik spasi sebagai akhir dari sebuah string.

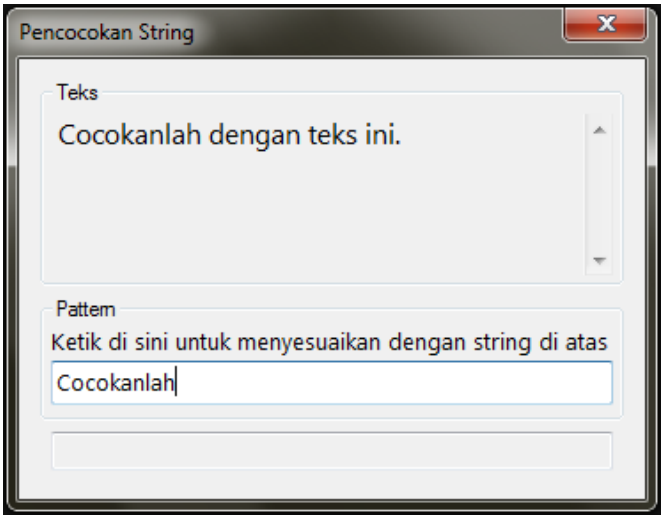
Penentuan benar atau salah untuk setiap string dilakukan di akhir saat mengetik spasi. String dikatakan benar apabila saat tepat mengetik spasi untuk semua string yang diketik dengan string yang ada pada layar tepat sama dan string yang tadi telah diketik akan disisipkan ke sebuah kotak teks. Namun, string dikatakan tidak benar atau salah apabila ada satu atau lebih karakter yang tidak sama dengan string yang tertera. Apabila hal ini terjadi, pengguna harus mengulang atau menghapus tulisan yang telah diketik tadi dan mengganti dengan yang benar dan otomatis kotak teks akan berubah menjadi warna putih kembali seperti semula.

Perlu diketahui bahwa program ini tidak sesempurna dan sebaik TypeRacer. Program ini dibuat hanya untuk mencocokan string saja, tidak menghitung berapa kecepatan mengetik pengguna dalam semenit atau pun persen kesalahan yang dilakukan. Berikut ini merupakan program sederhana yang dibuat menyerupai dengan TypeRacer dengan menggunakan pemrograman Bahasa C# berbasis antar-muka pengguna grafis (*graphical user interface*):



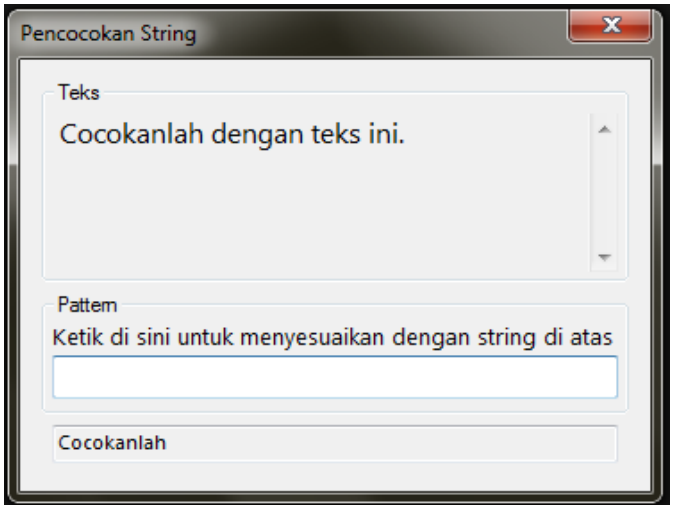
Gambar 4.1
Tampilan antar-muka beranda beserta dengan teks yang telah ditulis lebih dahulu

Sumber Gambar: Dokumentasi Pribadi



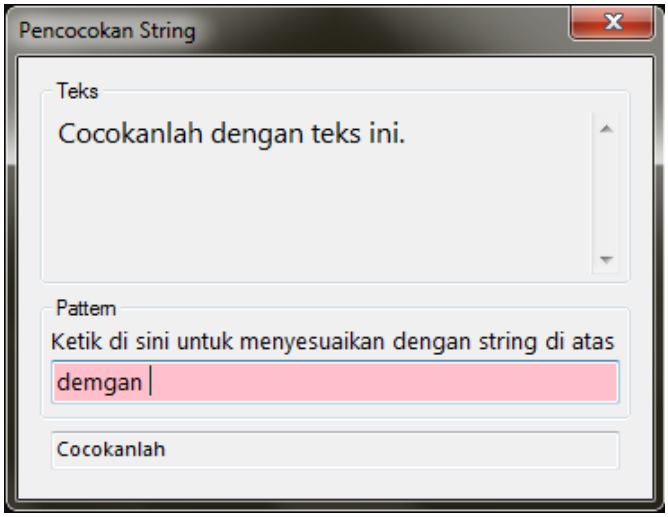
Gambar 4.2
Tampilan saat proses pengetikan

Sumber Gambar: Dokumentasi Pribadi



Gambar 4.3
Tampilan saat pengguna telah mengetik spasi sebagai akhir dari sebuah string dan disisipkan (ditampung) ke sebuah kotak teks (yang bawah)

Sumber Gambar: Dokumentasi Pribadi



Gambar 4.4
Tampilan saat ditemukan ketidaksesuaian yang menyebabkan kotak teks berubah menjadi warna merah (pertanda kesalahan)

Sumber Gambar: Dokumentasi Pribadi

V. SIMPULAN

Salah satu kegunaan yang nyata untuk algoritme pencocokan string ini adalah membuat orang lain bahagia dengan menciptakan sebuah permainan berlatih mengetik seperti yang telah dijelaskan sebelumnya. Meskipun demikian, algoritme pencocokan string ini juga turut membantu orang-orang dalam menyelesaikan berbagai hal yang umumnya adalah pengetikan pada aplikasi pengolah kata. Pada aplikasi pengolah kata, algoritme ini hadir pada pencarian kata yang fiturnya bernama *find* dan pasangannya, *replace*. Tidak hanya untuk pengolah kata, algoritme ini ternyata juga turut berkontribusi dalam ilmu yang mempelajari kehidupan dan organisme hidup dalam bidang membaca kode DNA (*deoxyribonucleic acid*).

VI. Ucapan Terima Kasih

Sebelumnya, saya ingin mengucapkan terima kasih kepada Tuhan Yang Maha Esa yang telah mengizinkan penulis untuk menulis makalah ini dan dapat selesai dengan kata jauh dari sempurna; tidak lupa juga untuk berterima kasih kepada dosen, terutama untuk Pak Rinaldi Munir, selaku dosen pengajar mata kuliah IF2211 Strategi Algoritma yang telah menurunkan segala ilmu yang dimiliki oleh beliau; orang tua yang telah mengizinkan penulis untuk berkuliah di salah satu tempat berkuliah terbaik di Indonesia; rekan-rekan yang turut membantu penulis dalam menyelesaikan makalah ini sebagai referensi; terakhir adalah internet dan google sebagai pencari yang turut mempertemukan penulis dengan berbagai referensi lainnya.

Daftar Pustaka

[1] Pratama, Aditya. 2017. *Penerapan Algoritma Pattern Matching dalam Game Typer Shark Deluxe*. Makalah Program Studi Teknik Informatika, Institut Teknologi Bandung. Bandung: Tidak diterbitkan.

[2] Davidson, Andrew. 2006. *Pattern Matching slide*. Songkhla, Thailand: Prince of Songkhla University (PSU).

[3] Munir, Rinaldi. 2004. *Bahan Kuliah ke-3 IF2251 Strategi Algoritmik Algoritma Greedy*. Bandung: Institut Teknologi Bandung.

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, 11 Mei 2018

Andreas Halim
13516003