Penerapan Algoritma Pencocokan String pada Mesin Penerjemah

Muhammad Hendry Prasetya - 13517105
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Bandung, Indonesia
13517105@std.stei.itb.ac.id

***Abstrak*—Algoritma pencocokan string adalah algoritma yang bertujuan untuk menemukan kecocokan dari suatu *pattern* (pola) dengan suatu teks. Panjang sebuah *pattern* umumnya lebih pendek dibandingkan panjang teks. Algoritma pencocokan string sudah sering digunakan dalam kehidupan sehari-hari. Beberapa aplikasi dari *pattern matching* antara lain fitur pencarian pada *text editor*, fitur Find pada Microsoft Office, mesin penelusuran, dan mesin penerjemah. Pada makalah ini, masalah yang akan dibahas adalah penerapan algoritma *brute force*, Knuth-Morris-Pratt, Boyer-Moore, dan Regular Expression dalam pencocokan string untuk mesin penerjemah. Dari keempat algoritma tersebut, algoritma Knuth-Morris-Pratt dan algoritma Boyer-Moore adalah algoritma yang paling tepat untuk diterapkan pada mesin penerjemah.**

Keywords—algoritma, string, pattern, pencocokan, penerjemah.

I. PENDAHULUAN

Algoritma adalah urutan atau langkah-langkah sistematis untuk menyelesaikan suatu masalah. Dalam menyelesaikan masalah tertentu, terdapat banyak pilihan algoritma yang bisa digunakan. Algoritma-algoritma tersebut dapat memberikan hasil yang berbeda-beda. Dalam memilih algoritma, kita sebaiknya memilih algoritma yang paling cocok untuk memecahkan suatu masalah. Parameter kecocokan algoritma dengan suatu masalah dapat dilihat dari terselesaikan atau tidaknya masalah dan efisiensi pemecahan masalah.

Di zaman modern ini, algoritma pemrograman sudah diaplikasikan ke berbagai hal. Hal ini didukung oleh pesatnya perkembangan teknologi informasi dan komunikasi di seluruh negara. Berbagai sumber informasi dan pengetahuan tersedia secara luas dan dapat diakses dengan mudah. Selain itu, komunikasi jarak jauh tidak lagi merupakan suatu hal yang sulit ataupun mahal. Masyarakat dituntut untuk dapat menggunakan teknologi informasi dan komunikasi dengan baik. Namun, terdapat masalah yang membatasi teknologi informasi dan komunikasi tersebut. Salah satunya adalah masalah perbedaan bahasa. Oleh karena itu, solusi yang dapat digunakan adalah menggunakan mesin penerjemah.

Mesin penerjemah adalah sebuah mesin yang dapat menerjemahkan suatu teks dari suatu bahasa ke bahasa lainnya. Mesin penerjemah saat ini sudah sering digunakan, salah satu yang paling sering digunakan adalah Google Translate. Secara sederhana, cara kerja dari mesin penerjemah adalah mencocokkan masukan *pattern* dengan teks yang tersedia di basis data. Jika sebuah *pattern* cocok dengan sebuah teks pada basis data, mesin penerjemah akan memberi keluaran yang sesuai dengan *pattern* tersebut. Hal ini dapat diaplikasikan menggunakan algoritma pencocokan string.

Algoritma pencocokan string adalah algoritma yang bertujuan untuk menemukan kecocokan dari suatu *pattern* dengan suatu teks. Beberapa algoritma yang paling popular adalah *brute force*, Knuth-Morris-Prat, Boyer-Moore, dan Regular Expression.

II. DASAR TEORI

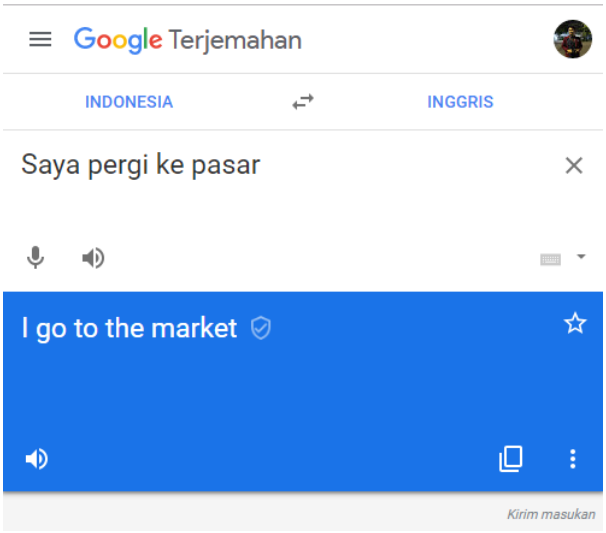
A. Mesin Penerjemah

Mesin penerjemah adalah mesin yang dapat menerjemahkan suatu teks dari suatu bahasa ke bahasa lainnya dalam bentuk formal maupun nonformal. Mesin penerjemah melakukan penggantian atau subtitusi sederhana terhadapa kata-kata pada teks masukan. Akan tetapi, hasil yang didapatkan hanya dengan cara tersebut tidak selalu efektif memberikan terjemahan teks yang baik. Hal ini disebabkan oleh perlunya pengenalan terhadap frasa secara utuh serta menyesuaikan pada konteks kalimat masukan. Selain itu, tata kalimat setiap bahasa dapat berbeda-beda. Sehingga, mesin penerjemah membutuhkan implementasi yang cukup kompleks.

Perangkat lunak mesin penerjemah umumnya memiliki pengaturannya masing-masing untuk meningkatkan hasil terjemahan. Beberapa di antaranya adalah teknik linguistik korpus dan teknik *stemming*. Teknik linguistik korpus adalah teknik menggunakan data dari bahan-bahan bahasa yang terkumpul serta berasal dari berbagai ragam penggunaan bahasa. Teknik ini dapat menerjemahkan suatu frasa secara utuh. Teknik *stemming* atau *lemmatization* adalah teknik pengelompokan bentuk-bentuk kata sehingga dapat dianalisis sebagai kata tunggal. Sebagai contoh, kata “memukul” dan “pukul” terdapat dalam kelompok kata yang sama. Pada bahasa Indonesia, *stemming* umumnya digunakan untuk memangkas imbuhan-imbuhan yang terdapat pada suatu kata. Sehingga, basis data penyimpanan teks tidak perlu menyimpan setiap

kombinasi dari imbuhan-imbuhan yang terdapat pada suatu kata.

Mesin penerjemah tidak hanya mengandalkan perancang atau pengembang aplikasi mesin. Perbaikan kualitas keluaran mesin penerjemah dapat diperoleh melalui kontribusi dari pengguna mesin penerjemah. Pengguna mesin penerjemah dapat menandai kata atau frasa mana saja yang tidak perlu diterjemahkan misalnya nama, alamat, dan sebagainya. Pengguna juga dapat memberikan umpan balik atas kerja mesin penerjemah. Semakin banyak data yang dapat digunakan mesin penerjemah, semakin baik hasil terjemahannya.



Gambar 1 Google Translate

Mesin penerjemah yang populer saat ini:

- Google Translate
- Flitto
- Skype Translator

B. *Pencocokan String*

Pencocokan string adalah proses mencocokan suatu *pattern* (dengan panjang m) dengan sebuah teks (dengan panjang n, n > m). Untuk melakukan pencocokan string, kita dapat menggunakan algoritma-algoritma:

- *Brute force*
- Knuth-Morris-Prat
- Boyer-Moore
- Regular Expression

1. Algoritma *Brute Force*

Algoritma *brute force* adalah algoritma yang memecahkan masalah secara sederhana dengan cara yang jelas (*obvious way*) tanpa memikirkan peningkatan performa penyelesaian masalah. Pada pencocokan string, algoritma *brute force* bekerja secara *straightforward* yang berarti algoritma ini terus mengecek bagian-bagian string dengan perlakuan yang sama hingga akhir bagian string. Berikut merupakan potongan kode dari algoritma *bruteforce* pada pencocokan string.

```
void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++) {
        int j;

        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
            cout << "Pattern found at index "
                 << i << endl;
    }
}
```

Gambar 2 Bruteforce. Sumber: www.geeksforgeeks.com

Langkah-langkah yang dilakukan algoritma *bruteforce* dalam mencocokkan string:

1. String pola (dengan panjang m) dicocokkan dengan string teks (dengan panjang n) pada bagian paling kiri dengan cara diperiksa per karakter string pola.
2. Jika karakter yang dibandingkan bernilai sama, lanjutkan perbandingan karakter string pola dengan karakter string teks berikutnya.
   Sedangkan jika karakter yang dibandingkan bernilai beda, perbandingan karakter dihentikan kemudian string pola digeser tepat satu kali. Setelah itu lakukan pencocokan string dari karakter awal string pola.
3. Jika seluruh karakter string pola bernilai sama dengan karakter-karakter string teks yang sedang dicocokkan, pencarian telah berhasil.
4. Jika hingga akhir teks tidak ditemukan kecocokkan, pencarian gagal.

Kompleksitas algoritma *bruteforce* dapat ditentukan dengan memasukan beberapa kasus uji. Algoritma *bruteforce* memiliki dua definisi kasus terbaik. Kasus yang pertama terjadi jika program tidak perlu melakukan pergeseran sama sekali. Kompleksitasnya sebesar O(m). Kasus yang kedua terjadi jika program tidak membandingkan karakter string pola secara sia-sia.

Contoh masukan dan keluaran:

- Input : txt = “Aku adalah seorang mahasiswa”
  pat = “Aku”
- Output : “Pattern found at index 0”

- Input : txt = “ooooooooooooooooooaaaah”
  pat = “aaaah”
- Output : “Pattern found at index 18”

- Input : txt = “aaaaaaaaaaaaaaaaaaaaah”
  pat = “aaaah”
- Output : “Pattern found at index 18”

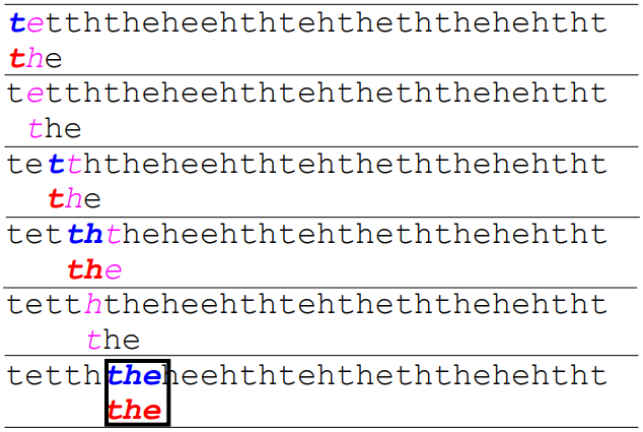
```
Aku adalah seorang mahasiswa
Aku

ooooooooooooooooooaaaah
                    aaaah
```

Gambar 3 Bruteforce Best Case

Pada kasus terburuk, algoritma *brutefoce* memiliki kompleksitas O(mn). Hal ini dapat terjadi jika program terus-menerus melakukan perbandingan karakter yang sia-sia hingga akhir teks.

```
aaaaaaaaaaaaaaaaaaaaah
                    aaaah
```

Gambar 4 Bruteforce Worst Case



Gambar 5 Bruteforce Pattern Matching. Sumber: cs.purdue.edu

2. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan string yang memanfaatkan informasi terkait string pola untuk meminimalisasi banyaknya pergeseran. Informasi tersebut dapat disimpan dalam sebuah senarai atau *list*, kemudian dicek setiap kali melakukan pergeseran. Berikut merupakan potongan kode dari algoritma Knuth-Morris-Pratt pada pencocokan string.

```
void KMPSearch(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    int lps[M]; //Fungsi border
    computeLPSArray(pat, M, lps);

    int i = 0; // index for txt[]
    int j = 0; // index for pat[]
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }

        if (j == M) {
            printf("Found pattern at index %d ", i - j);
            j = lps[j - 1];
        }

        // mismatch after j matches
        else if (i < N && pat[j] != txt[i]) {
            // Do not match lps[0..lps[j-1]] characters,
            // they will match anyway
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
}
```

Gambar 6 Algoritma KMP. Sumber: www.geeksforgeeks.com

Proses yang dilakukan algoritma KMP dalam pencocokan string terbagi menjadi dua bagian. Bagian pertama disebut proses awal (*preprocessing*). Bagian ini mencari fungsi *border* yang mencatat prefix (awalan) dari bagian-bagian pattern. pattern yang juga merupakan suffix (akhiran). Dengan adanya pergeseran tersebut, program akan melakukan pencarian dengan lebih efisien karena tidak melakukan pencarian yang sia-sia.

Langkah-langkah membuat fungsi awalan (*border*) dalam algoritma Knuth-Morris-Pratt:

1. Periksa setiap bagian dari string pola. Bagian-bagian tersebut merupakan himpunan dari suffix pada string pola.
2. Apabila sebuah bagian string pola ke-i memiliki suffix yang juga prefix dengan panjang k, masukkan nilai k ke dalam fungsi awalan pada indeks ke-i.

```
Pattern:    a b a b a a
borderFunc: 0 0 1 2 3 1
```

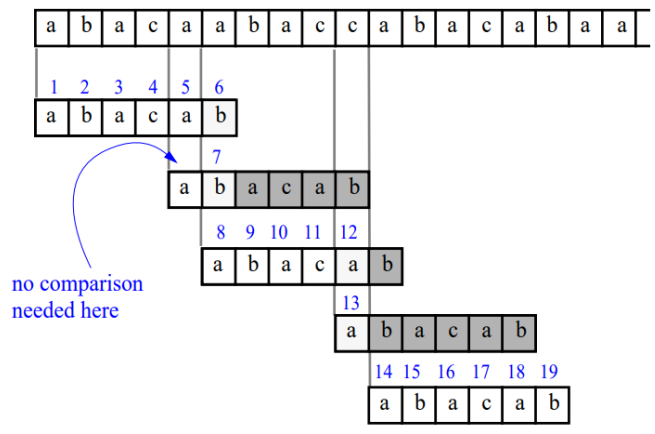
Gambar 7 Contoh Fungsi Awalan KMP

```
void computeLPSArray(char* pat, int M, int* lps)
{
    // length of the previous longest prefix suffix
    int len = 0;

    lps[0] = 0; // lps[0] is always 0

    // the loop calculates lps[i] for i = 1 to M-1
    int i = 1;
    while (i < M) {
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            if (len != 0) {
                len = lps[len - 1];
            }
            else // if (len == 0)
            {
                lps[i] = 0;
                i++;
            }
        }
    }
}
```

Gambar 8 Algoritma Border Function. Sumber: www.geeksforgeeks.com

Proses bagian kedua dalam algoritma KMP adalah proses mencari kecocokan string pola pada string teks. Langkah-langkah yang dilakukan pada bagian kedua algoritma Knuth-Morris-Pratt:

1. Tentukan fungsi border terlebih dahulu dengan parameter string pola.
2. Lakukan pencocokan string karakter per karakter.
3. Jika pada karakter ke-j dari string pola ditemukan ketidakcocokan dengan string teks, lakukan pergeseran sesuai panjang suffix yang juga prefix pada fungsi border dengan indeks ke-j. Hal ini dapat menghindari pengecekan ulang terhadap string yang polanya sudah sama.



Gambar 9 KMP. Sumber: cs.purdue.edu

3. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma pencocokan string yang memanfaatkan karakter penyebab *mismatch* untuk melakukan pergeseran secara optimal. Berbeda dengan kedua algoritma sebelumnya, perbandingan karakter string pola dengan string teks dilakukan dari kanan ke kiri. Namun, algoritma ini memiliki kesamaan dengan algoritma KMP yaitu sama-sama melakukan *preprocessing* (proses awal) pada string pola.

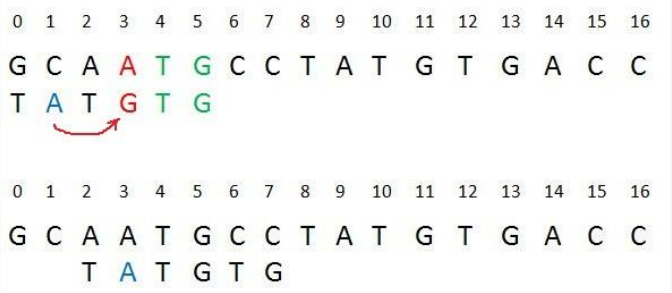
Algoritma Boyer-Moore merupakan kombinasi dari dua pendekatan yaitu *Bad Character Heuristic* dan *Good Suffix Heuristic*. Pendekatan-pendekatan tersebut bekerja dengan cara menyimpan kemunculan terakhir suatu karakter pada string pola. Berikut merupakan potongan kode algoritma Boyer-Moore.

```
void search( string txt, string pat)
{
    int m = pat.size();
    int n = txt.size();

    int badchar[256];
    badCharHeuristic(pat, m, badchar);
    int s = 0; // s is shift of the pattern
    while(s <= (n - m))
    {
        int j = m - 1;
        while(j >= 0 && pat[j] == txt[s + j])
            j--;

        if (j < 0)
        {
            cout << "pattern occurs at shift = " <<  s << endl;
            s += (s + m < n)? m-badchar[txt[s + m]] : 1;
        }
        else
            s += max(1, j - badchar[txt[s + j]]);
    }
}

void badCharHeuristic( string str, int size,
                       int badchar[NO_OF_CHARS])
{
    int i;
    for (i = 0; i < NO_OF_CHARS; i++)
        badchar[i] = -1;
    for (i = 0; i < size; i++)
        badchar[(int) str[i]] = i;
}
```

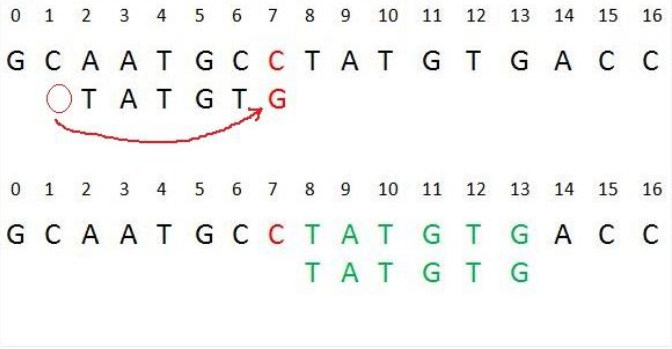
Langkah-langkah yang dilakukan pada algoritma Boyer-Moore:

1. Mula-mula, string pola dicocokkan dengan strik teks pada bagian paling kiri (awal). Pencocokan dilakukan per karakter dari karakter yang paling kanan (akhir).
2. Jika perbandingan karakter string pola dengan string teks menemukan ketidakcocokan pada karakter ‘x’ dalam string teks, geser string pola sedemikian rupa sehingga karakter ‘x’ pada string pola bertepatan pada karakter ‘x’ pada string teks.
   Namun jika karakter ‘x’ tidak ditemukan pada string pola, geser seluruh string pola ke bagian setelah karakter ‘x’ pada string teks.
3. Lakukan pencocokan hingga panjang karakter yang cocok sama dengan panjang string pola (pencarian

berhasil) atau hingga seluruh teks telah selesai diperiksa.



Gambar 10 Pergeseran string pola pada BM. Sumber: geeksforgeeks.com



Gambar 11 Pergeseran string pola pada BM (2). Sumber: geeksforgeeks.com

4. Algoritma Regular Expression

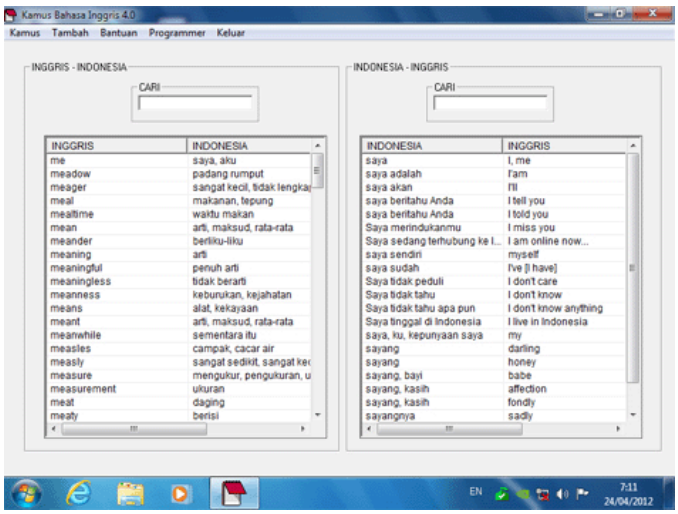
Algoritma *regular expression* adalah algoritma dengan *finite automata* dan notasi-notasi yang mendefinisikan himpunan string. Ketika sebuah string teks dapat didefinisikan dengan notasi *regular expression*, kita dapat mengatakan bawah *regular expression* cocok digunakan. Notasi-notasi pada *regular expression*:

1. Repeaters, yaitu simbol yang menyatakan bahwa karakter sebelumnya akan digunakan lebih dari satu kali. Notasinya '*', '+', dan '{ }'.
2. Asterisk symbol, yaitu simbol yang menyatakan bahwa karakter sebelumnya diulang sebanyak nol sampai dengan tak hingga. Notasinya '*'. Contoh penggunaan: bc*d dapat menotasikan string bd, bcd, bccd, bcccd, dan seterusnya.
3. Plus symbol, yaitu symbol yang menyatakan bahwa karakter sebelumnya diulang sebanyak satu sampa dengan tak hingga. Notasinya '+'. Contoh penggunaan: ab+c dapat menotasikan string abc, abbc, abbbc, dan seterusnya.
4. The curly braces, yaitu simbol yang menyatakan bahwa karakter sebelumnya diulang sesuai isi pada tanda kurung kurawal. Notasinya '{}'. Contoh penggunaan: a{2} memberikan string aa; a{2,4} memberikan string aa, aaa, dan aaaa; a{2,} memberikan string aa, aaa, aaaa, aaaaa, dan seterusnya sampai dengan tak hingga.
5. Wildcard, yaitu simbol yang dapat menotasikan simbol lain apapun. Notasinya '.'. Contoh penggunaan: notasi .* menyatakan bahwa setiap karakter dapat digunakan berkali-kali.
6. Optional character, yaitu simbol yang menyatakan bahwa karakter sebelumnya mungkin saja tidak ada pada string yang ingin dicocokkan. Notasinya '?'. Contoh penggunaan: pada notasi "docx?", karakter 'x' tidak harus ada.
7. The caret symbol, yaitu simbol yang menyatakan bahwa kecocokan harus dimulai dari awal string. Notasinya '^'.
8. The dollar symbol, yaitu simbol yang menyatakan bahwa kecocokan harus muncul pada akhir string atau sebelum newline pada akhir string. Notasinya '$'.
9. Character classes, yaitu himpunan karakter-karakter tertentu yang dapat dicocokkan. Terdapat notasi umum dari *character class*. Notasi "/s" untuk bagian kosong (*whitespace*) karakter dan notasi "/S' untuk karakter yang tidak kosong. Notasi "/d" untuk karakter digit dan notasi "/D" untuk karakter non-digit. Notasi "/w" untuk karakter kata apapun dan notasi "/W" untuk karakter non-kata. Notasi "/b" untuk *word boundary* seperti spasi, strip, koma, titik koma, dan lain-lain.
10. Set of characters, yaitu simbol yang menyatakan kecocokan suatu string yang merupakan kombinasi dari karakter-karakter yang ada pada suatu himpunan karakter. Notasinya '[]'. Contoh penggunaan: [abc] akan cocok dengan karakter a,b, dan c di string apa pun.
11. Character range, yaitu simbol yang menyatakan jangkauan karakter dari awal hingga akhir. Notasinya '-'. Contoh penggunaan: [A-Z] akan cocok dengan karakter dari A sampai Z.
12. The escape symbol, yaitu simbol yang menyatakan suatu karakter setelahnya bukan merupakan suatu notasi. Notasinya '\'. Contoh: "\d + \+ \d+" akan cocok dengan string "2 + 2".
13. Grouping character, adalah himpunan dari simbol *regular expression* yang berperilaku seperti unit tunggal. Notasinya ().
14. Vertical bar, adalah simbol yang menyatakan string cocok dengan setiap elemen yang terpisah oleh garis vertical. Notasinya '|'. Contoh penggunaan: "th(e|is|at)" akan cocok dengan kata "the", "this", dan "that".

Pencocokan string pada *regular expression* dibuat dengan cara menuliskan notasi string masukan dengan string yang ada di dalam basis data. Misalnya kita ingin mencocokan string "Apa ibukota Indonesia?", sedangkan string yang ada di dalam basis data adalah "Apa ibukota Negara Kesatuan Republik Indonesia?". Hal tersebut dapat ditangani dengan *regular expression*. String masukan disisipkan dengan notasi-notasi yang cocok, contohnya "Apa(.*)ibukota(.*)Indonesia(.*)?". Hal ini dapat mencocokkan string masukan dengan string di dalam basis data yang telah dibahas sebelumnya.

III. PEMBAHASAN

Pada mesin penerjemah, kita dapat menerapkan mekanisme sederhana menggunakan algoritma pencocokan string. Jika masukan berupa kata, program melakukan *lemmatization* kata masukan. Kemudian, program mencocokkan kata masukan dengan string yang ada pada basis data berisi kamus kata. Setelah itu, program melakukan metode *exact matching* yaitu mencocokan secara utuh setiap kata yang ada. Jika tidak ada kata yang sama persis, program memberi keluaran arti dari kata yang paling mirip dengan kata masukan serta saran kata masukan. Jika masukannya berupa kalimat, program terlebih dahulu mencocokkan *substring* masukkan pada basis data berisi frasa-frasa bahasa tertentu. Kemudian, program melakukan *lemmatization* pada setiap kata masukan dan menerjemahkannya.



Gambar 12 Aplikasi Kamus Bahasa Inggris-Indonesia. Sumber: micopadorsi.tripod.com

Sebagai contoh, kita akan menggunakan Kamus Bahasa Inggris-Indonesia. Langkah-langkah penerjemahan yang dilakukan adalah:

1. Menentukan bahasa dari string masukan.
2. Melakukan pencarian *substring* dengan frasa-frasa yang ada pada kamus. Jika cocok, terjemahkan frasa tersebut.
3. Melakukan *lemmatization* pada kata kemudian mencari kata tersebut pada kamus. Jika cocok, terjemahkan kata tersebut. Misalnya mengapus imbuhan "me-", "di-" dan sebagainya.
4. Jika tidak ditemukan string yang cocok, artikan kata tersebut ke arti dari kata yang paling mendekati.
5. Membangun sebuah string baru yang berisi frasa-frasa dan kata-kata yang sudah diartikan.

IV. ANALISIS

Setelah membahas tentang cara kerja mesin penerjemah, kita dapat menganalisis apakah suatu algoritma tepat digunakan untuk mesin penerjemah. Algoritma *bruteforce*, Knuth-Morris-Pratt, dan Boyer-Moore dapat memberikan hasil yang tepat untuk permasalahan ini. Hal ini disebabkan oleh sifat algoritma tersebut adalah *exact matching*. Algoritma-algoritma tersebut akan selalu memberikan hasil selama ditemukan string yang paling mirip. Kemiripan suatu string dapat dinilai dari banyaknya karakter yang sama. Namun jika dibandingkan dengan algoritma Knuth-Morris-Pratt dan Boyer-Moore, algoritma *bruteforce* kekurangan dalam performa secara signifikan. Mesin penerjemah yang baik biasanya menggunakan banyak data agar hasil terjemahan semakin baik. Kekurangan performa akan menghambat kecanggihan dari suatu mesin penerjemah.

Algoritma *regular expression* juga dapat menyelesaikan masalah pencarian kata dalam kamus kata. Akan tetapi, pencarian yang dilakukan adalah membandingkan satu string penuh. Hasil yang dikembalikan berupa dua nilai yaitu *true* atau *false*. Sehingga, mesin penerjemah akan sulit memberi nilai pada kemiripan antara masukan pengguna dengan kata di dalam kamus kata. Selain itu, notasi-notasi pada *regular expression* kurang baik untuk digunakan pada pencarian ini. Hal ini disebabkan oleh sifat pencarian yang mencari hanya sebuah kata dengan panjang terbatas dan terdiri atas karakter non-numerik. Notasi-notasi seperti *asterisk*, *wildcard*, dan sebagainya cenderung tidak digunakan dalam pencarian karena dapat mengubah arti sesunggunya dari kata yang diinginkan.

V. KESIMPULAN

Setelah dilakukan analisis kepada setiap algoritma yang dibahas pada pembahasan, kita dapat menarik kesimpulan bahwa algoritma-algoritma tersebut dapat diterapkan pada mesin penerjemah. Akan tetapi, algoritma yang paling tepat untuk digunakan dari keempat algoritma tersebut adalah Knuth-Morris-Pratt dan Boyer-Moore karena memiliki peningkatan performa yang signifikan serta dapat menilai suatu kemiripan string pola dengan string teks.

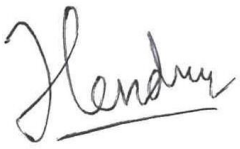
VI. REFERENSI

[1] Matematika Diskrit_ Ed. 3 - Rinaldi Munir
[2] http://www.geeksforgeeks.com. Diakses pada 25 April 2019, 22.26.
[3] www.cs.purdue.edu . Diakses pada 26 April 2019,
[4] www.regexr.com . Diakses pada 26 April 2019, 09.34.
[5] http://nlp.stanford.edu . Diakses pada 25 April 2019, 21.26
[6] www.micopardosi.tripod.com . Diakses pada 26 April 2019, 09.12
[7] www.kbbi.web.id . Diakses pada 26 April 2019, 09.35
[8] www.badanbahasa.kemdikbud.go.id Diakses pada 26 April 2019, 11.30.

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 2019



Muhammad Hendry Prasetya
13517105