Penerapan Algoritma Pencocokan *String* dan Runut Balik untuk Pendeteksian Kesalahan dan Sugesti Kata

Erick Chandra/13513021[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13513021@std.stei.itb.ac.id

***Abstrak*—Dalam dunia komputer, sering ditemukan banyak masalah khususnya dalam kesalahan pengetikan kata-kata. Kadang kala, mata manusia kurang jeli untuk melihat dan memeriksa setiap kata yang telah ditik apalagi bila teks berupa paragraf yang sangat panjang dan sangat melelahkan. Namun, dengan adanya algoritma runut balik, teks dapat diidentifikasi oleh komputer untuk dicocokkan pada kamus, sehingga bila ditemukan teks tersebut berarti kata tersebut telah benar sesuai dengan kamus. Jika ternyata terdapat kesalahan pada pengetikan kata tertentu, mungkin saja pengguna memang kurang paham penulisan yang benar atau kata lain yang mirip yang dimaksud. Dengan bantuan algoritma pencocokan *string*, kini pengguna dapat menikmati fitur sugesti kata yang mirip dengan kata tersebut.**

***Kata Kunci*—Algoritma, Pencocokan *String*, Runut Balik, Sugesti.**

I. PENDAHULUAN

Seiring perkembangan zaman, sekarang komputer telah dapat membantu manusia untuk melakukan penyuntingan dokumen melalui perangkat lunak pemroses kata. Kadang-kadang, manusia sering mengetikkan kata yang salah, dalam arti salah eja maupun silap pada saat pengetikan. Pada kenyataannya, manusia memiliki kemampuan pengecekan yang terbatas untuk keakuratan 100%. Oleh karena itu, dengan hadirnya algoritma runut balik, pengguna dapat menikmati perangkat lunak yang menerapkan algoritma runut balik untuk memeriksa ketepatan ejaan setiap kata yang dicocokkan pada kamus.

Selain itu, ada beberapa faktor yang menyebabkan pengguna melakukan kesalahan dalam pengetikan. Faktor pertama adalah pengguna kurang teliti pada saat pengetikan. Faktor yang lainnya adalah pengguna memang kurang paham terhadap kata yang ditik, misalkan pengguna sedang belajar bahasa tersebut. Oleh sebab itu, dengan algoritma pencocokan string, pengguna dapat menggunakan perangkat lunak yang dilengkapi dengan algoritma pencocokan string yang dapat membantu menampilkan seluruh sugesti (saran) dari kata-kata yang ada pada kamus yang mirip dengan kata yang ditik.

II. DASAR TEORI

A. Algoritma Runut Balik

Algoritma runut balik adalah langkah-langkah untuk mencari semua atau sebagian masalah komputasi, yang memiliki pembatas kesesuaian yang dicari, yang secara meningkat membangun kandidat ke solusinya, dan meninggalkan setiap kandidat yang tidak mungkin lagi menjadi penyelesaian yang valid.

Algoritma runut balik dapat dipandang sebagai salah satu dari dua hal berikut.

1. Sebagai sebuah fase di dalam algoritma traversal DFS.
2. Sebagai sebuah metode pemecahan masalah yang mangkus, restruktur, dan sistematis.

Algoritma runut balik banyak diterapkan untuk aplikasi permainan, di antaranya:

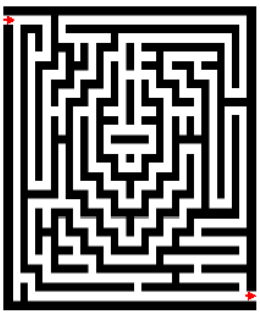
- Permaintan "*Tic Tac Toe*",
- Penemuan jalur keluar dalam sebuah labirin,
- Catur, teka teki silang, *sudoku*, dan masalah-masalah pada bidang kecerdasan buatan.

Misalkan kita harus membuat rangkaian keputusan di antara beberapa pilihan, di mana:

- Kita tidak punya cukup informasi untuk mengetahui apa yang akan dipilih
- Tiap keputusan mengarah pada sekumpulan pilihan baru
- Beberapa sekuens pilihan (bisa lebih dari satu) mungkin merupakan solusi persoalan.

Algoritma runut balik adalah cara yang metodologis mencoba beberapa sekuens keputusan, sampai kita menemukan sekuens yang "bekerja".

Sebagai contoh, diberikan sebuah labirin (*maze*), kita diminta untuk menemukan lintasan dari titik awal sampai titik akhir.

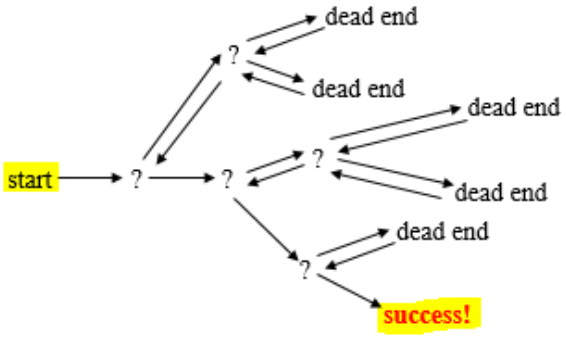


Gambar 1. Permainan Maze.
Sumber: Munir, Rinaldi. Slide Kuliah Strategi Algoritma 2015.

Pada tiap perpotongan, kita harus menemukan satu di antara tiga pilihan, yaitu:
- maju terus
- belok kiri
- belok kanan.

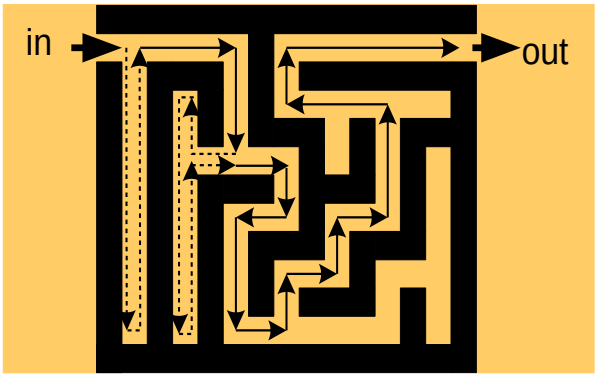
Kita tidak memiliki informasi yang cukup untuk memilih pilihan yang benar, yang mengarah ke titik akhir solusi. Setiap pilihan mengarah ke sekumpulan pilihan lain. Jadi, algoritma runut balik dapat digunakan untuk persoalan seperti ini.

Contoh gambaran tentang algoritma runut balik.



Gambar 2. Ilustrasi Runut Balik.
Sumber: Munir, Rinaldi. Slide Kuliah Strategi Algoritma 2015.

Contoh runut balik pada sebuah labirin dapat dilihat pada Gambar 3.



Gambar 3. Runut Balik pada Labirin.
Sumber: Munir, Rinaldi. Slide Kuliah Strategi Algoritma 2015.

Adapun penyelesaian dengan algoritma runut balik adalah sebagai berikut.
- Bagi lintasan menjadi sederetan langkah.
- Sebuah langkah terdiri dari pergerakan satu unit sel pada arah tertentu.
- Arah yang mungkin: lurus (*straight*), kiri *(left*), ke kanan (*right*).

Garis besar algoritma runut baliknya adalah sebagai berikut.

```
while belum sampai pada tujuan do
      if terdapat arah yang benar sedemikian sehingga kita belum pernah berpindah ke sel pada arah tersebut then
            pindah satu langkah ke arah tersebut
      else
            runut balik langkah sampai terdapat arah seperti yang disebutkan di atas
      endif
endwhile
```

Ada dua penyelesaian untuk mengetahui langkah yang mana yang perlu dijejaki kembali adalah sebagai berikut.
1. Simpan semua langkah yang pernah dilakukan, atau
2. Gunakan rekursi (yang secara implisit menyimpan semua langkah). Rekursi adalah solusi yang lebih mudah.

Algoritma runut balik merupakan perbaikan dari *exhaustive search*. Pada *exhaustive search*, semua kemungkinan solusi dieksplorasi satu per satu. Pada algoritma runut balik, hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi, yaitu dengan memangkas (*pruning*) simpul-simpul yang tidak mengarah ke solusi.

Properti umum metode runut balik adalah sebagai berikut.
1. Solusi persoalan.
   Solusi dinyatakan sebagai vektor dengan *n-tuple*: $X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$.
   Mungkin saja $S_1 = S_2 = \ldots = S_n$.
   Contoh: $S_i = \{0, 1\}$, $x_i = 0$ atau 1.
2. **Fungsi pembangkit** nilai x_k
   Dinyatakan sebagai predikat: *T(k)*.
   T(k) membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.
3. **Fungsi pembatas**
   Dinyatakan sebagai predikat $B(x_1, x_2, \ldots, x_k)$.
   B bernilai *true* jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi.
   Jika *true*, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika *false*, maka $(x_1, x_2, \ldots, x_k)$ dibuang.

Ruang solusi diorganisasikan ke dalam struktur pohon. Tiap simpul pohon menyatakan status (*State*) persoalan, sedangkan sisi (cabang) diberi label dengan nilai-nilai x_i. Lintasan dari akar ke daun menyatakan solusi yang mungkin. Seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi diacu sebagai pohon ruang status (*State space tree*).

Ada tiga macam simpul, yaitu:
- Simpul akar
- Simpul dalam
- Simpul daun

Prinsip pencarian solusi dengan metode runut balik adalah sebagai berikut.
- Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan *depht-first order* (DFS).
- Simpul-simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*).
- Simpul hidup yang *sedang* diperluas dinamakan **simpul-E** (*Expand-node*).
- Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.
- Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi **simpul mati** (*dead node*).
- Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan **fungsi pembatas** (*bounding function*).
- Simpul yang sudah mati tidak akan pernah diperluas lagi.
- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian *backtrack* ke simpul aras diatasnya
- Lalu, teruskan dengan membangkitkan simpul anak yang lainnya.
- Selanjutnya simpul ini menjadi simpul-E yang baru.
- Pencarian dihentikan bila kita telah sampai pada *goal node*.

B. Algoritma Pencocokan String

Definisi pencocokan *string* adalah sebagai berikut.
- Definisi: Diberikan
  - *T*: teks (*text*), yaitu (*long*) *string* yang panjangnya *n* karakter
  - *P: pattern*, yaitu *string* dengan panjang *m* karakter (asumsi $m <<< n$) yang akan dicari di dalam teks.
- Carilah (*find* atau *locate*) lokasi pertama di dalam teks yang bersesuaian dengan *pattern*.
- Contoh:
  - T: "the rain in spain stays mainly on the plain"
  - P: "main"

Konsep *string* adalah sebagai berikut.
- Asumsi S adalah sebuah *string* dengan ukuran m. $S = x_1x_2 \ldots x_m$
- Sebuah awalan dari S adalah *substring* S[1..k-1].
- Sebuah akhiran dari S adalah *substring* S[k-1..m].
  - k adalah indeks apapun di antara 1 dan m (eksklusif).
  - S[0] adalah karakter *null*, simbolnya adalah ∅
- Contohnya:

a	n	d	r	e	w

  - Semua awalan dari S yang mungkin adalah:
    - "∅", "a", "an", "and", "andr", "andre"
  - Semua akhiran dari S yang mungkin adalah:
    - "∅", "w", "ew", "rew", "drew", "ndrew"

Algoritma pencocokan *string* ada beberapa macam, di antaranya:
- *Brute Force*
- *Knuth-Morris-Pratt (KMP)*
- *Boyer Moore*

Algoritma Brute Force

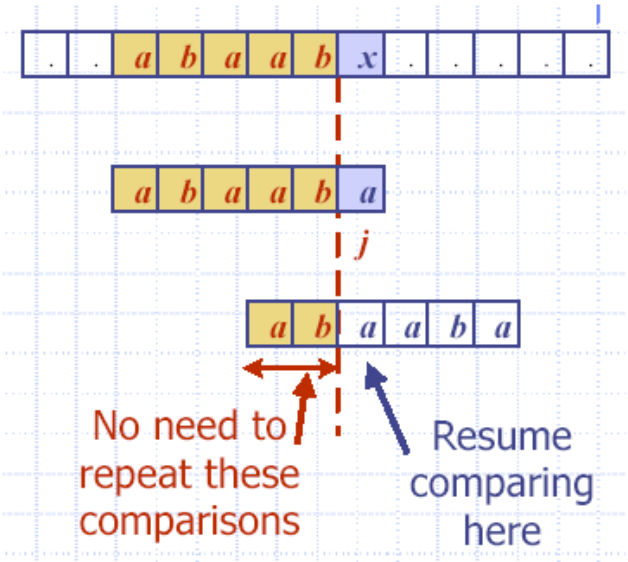
Algoritma Brute Force memeriksa setiap posisi dari teks T untuk melihat jika ada pola P yang sesuai pada posisi tersebut. Contohnya adalah sebagai berikut.

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

Algoritma KMP

Algoritma Knuth-Morris-Pratt (KMP) mencari pola di teks dengan urutan kiri ke kanan (seperti algoritma Brute Force). Tetapi, ia menggerakkan polanya lebih cerdik dibanding dengan algoritma Brute Force.

Jika ketidakcocokan terjadi di antara teks dan pola P pada P[j], maka dimaksimalkan pergeseran pola untuk menghindari perbandingan yang sia-sia sebesar awalan terbesar P[1..j-1] yang juga akhiran dari P[1..j-1].



Gambar 4. Demonstrasi Pencocokan String dengan Algoritma KMP.
Sumber: Munir, Rinaldi. Slide Kuliah Strategi Algoritma 2015.

Fungsi pinggiran KMP adalah sebagai berikut.
- KMP memproses lebih dahulu pola untuk mencari kecocokan *string* dari awalan dari pola tersebut dengan pola itu sendiri.
- j adalah posisi ketidakcocokan di *P[]*.
- k adalah posisi sebelum ketidakcocokan (*k=j-1*).
- Fungsi pembatas *b(k)* didefinisikan sebagai ukuran dari awalan terbesar dari *P[1..k]* yang juga merupakan akhiran dari *P[1..k]*.
- Nama lainnya adalah *failure function* (disingkat *fail*).

Contoh fungsi pembatas adalah sebagai berikut.
Misalkan P: "abaaba" dan j: 123456

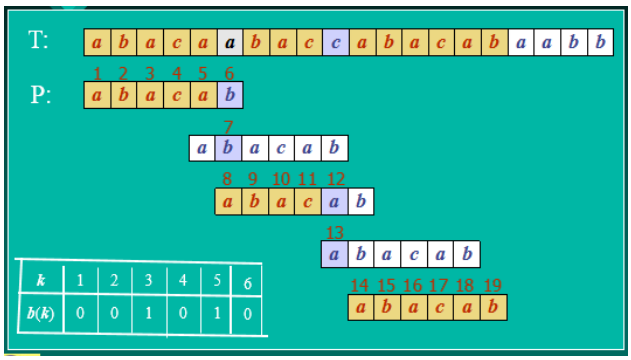
j	1	2	3	4	5	6
P[j]	a	b	a	a	b	a
b(j)	0	0	1	1	2	3

b(j) adalah ukuran dari *border* terbesar.

Algoritma KMP memodifikasi algoritma Brute Force, yaitu

```
if a mismatch occurs at P[j]
(i.e. P[j] != T[i]), then
   k = j-1;
   j = b(k) + 1; // obtain the new j
```

Contoh demonstrasi penggunaan algoritma KMP secara keseluruhan adalah sebagai berikut.



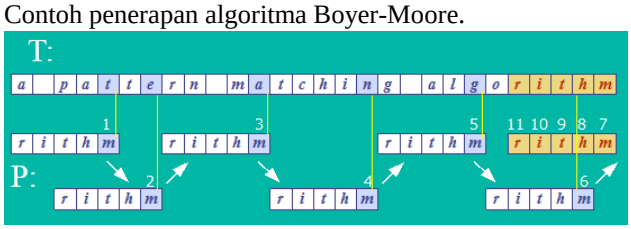
Gambar 5. Contoh Pencocokan String utuh dengan Algoritma KMP.
Sumber: Munir, Rinaldi. Slide Kuliah Strategi Algoritma 2015.

Algoritma Boyer-Moore

Algoritma Booyer-Moore berdasarkan dua buah teknik.
1. Teknik *looking-glass*
   Mencari P di dalam T dengan memundurkan dari P, mulai dari ekornya.
2. Teknik *character-jump*
   Ketika sebuah ketidakcocokan terjadi pada T[i]==x, karakter di pola P[j] tidak sama dengan T[i].

Ada tiga buah kemungkinan ketidakcocokan, yaitu
1. Jika P mengandung x di suatu tempat, maka coba geser P ke kanan untuk di-*align* ke kemunculan terakhir dari x di P dengan T[i].
2. Jika P mengandung x di suatu tempat, tetapi pergeseran ke kanan untuk kemunculan terakhir tidak dimungkinkan, maka geser P ke kanan tepat satu karakter ke T[i+1].
3. Jika kasus 1 dan 2 tidak bisa, maka geser P untuk di-*align* P[1] dengan T[i+1].

Contoh penerapan algoritma Boyer-Moore.



Gambar 6. Pencocokan String dengan Algoritma BM.
Sumber: Munir, Rinaldi. Slide Kuliah Strategi Algoritma 2015.

Fungsi kemunculan terakhir adalah sebagai berikut.
- Algoritma Boyer-Moore memproses lebih dahulu pola P dan alfabet A untuk membangun kemunculan terakhir dari fungsi L().
  L() memetakan semua huruf di A ke bilangan bulat.
- L(x) didefinisikan sebagai indeks terbesar i sedemikian sehingga P[i]==x, atau -1 bila tidak ada indeks yang muncul.

III. Analisis Permasalahan

Di dunia nyata, contoh kesalahan yang dapat dilakukan oleh pengguna komputer adalah hampir tidak berhingga banyaknya. Oleh karena itu, dipilih satu contoh untuk dianalisis. Asumsi bahasa yang digunakan adalah Bahasa Indonesia, metode pemasukan papan tik juga Bahasa Indonesia, dan kamus yang dipadankan adalah Kamus Besar Bahasa Indonesia yang sudah dalam bentuk pohon.

Misalkan s*tring* yang diambil adalah "latiha".

Pertama-tama, yang dilakukan adalah mencari apakah kata "latiha" ada di kamus dengan memanfaatkan algoritma runut balik. Kamus sudah dalam bentuk pohon yang diuraikan huruf demi huruf. Aras yang lebih tinggi menunjukkan huruf sebelumnya. Algoritma runut balik akan menelusuri sambil mencocokkan, apakah pada aras pertama terdapat huruf yang tepat seperti huruf pertama kata yang dicari. Bila bertemu, lakukan lagi pencarian untuk huruf-huruf selanjutnya. Bila tidak bertemu, berarti kata tidak ditemukan di dalam kamus. Pencarian dan penelusuran akan berhenti bila kata tersebut telah ditemukan atau ketika kata tersebut tidak ditemukan.

Komputer akan melakukan pencarian setiap kata yang ditikkan. Pemisah antarkata biasanya menggunakan spasi sehingga mempermudah pemroses kata untuk memilah kata-kata.

Bila kata tidak ditemukan, akan dicari pula sugesti kata yang mirip dengan kata yang salah itu. Misalkan "latiha" mirip dengan:

- "latih"
- "latihan"
- "pelatih"
- "pelatihan"
- "perlatihan"
- "terlatih"
- "latihkan"
- "berlatih"

Algoritma pencocokan *string* dapat digunakan untuk hal ini. Dari kata "latiha", dibuat semua *substring* termasuk "latiha" sendiri untuk kemudian diperiksa dan dicocokkan ke kamus kedekatan *string* tersebut dengan *string* yang ada di dalam kamus.

Seperti yang telah dijelaskan pada bab dasar teori, algoritma yang dapat dimanfaatkan di antaranya algoritma brute force, algoritma KMP, atau algoritma Boyer-Moore. Namun algoritma brute force sangat tidak disarankan karena algoritma tersebut sangat primitif dan sangat menguras sumber daya dan waktu untuk hal yang sebenarnya bisa dilangkahi.

Sebagai contoh, kita manfaatkan algoritma KMP. Maka, untuk setiap *string* yang ada pada kamus, akan dicocokkan terhadap setiap *substring* yang telah dibuat dari kata salah yang ditik oleh pengguna.

Algoritma KMP maupun Boyer-Moore akan sangat efektif dalam pencocokan *string* dalam konteks kamus karena seperti yang kita ketahui, kata tergolong pendek dan ketika terjadi ketidaksesuaian, banyak yang dapat dilangkahi dan dilanjutkan ke pemrosesan *string* berikutnya.

IV. Implementasi

Algoritma runut balik dapat dengan mudah diselesaikan. *Pseudocode*-nya adalah sebagai berikut.

```
procedure RunutBalikR (input k:integer)
{ Mencari semua solusi persoalan dengan
metode runut balik; skema rekursif.
Masukan: k, yaitu indeks komponen vektor
solusi, x[k]
Keluaran: solusi x = (x[1], x[2], ...,
x[n])
}
Algoritma:
       for tiap x[k] yang belum dicoba
       sedemikian sehingga (x[k] ← T[k])
       Ana B(x[1], x[2], ..., x[k]) = true
       do
              if (x[1], ..., x[k]) adalah
              lintasan dari akar ke daun
              then
                     CetakSolusi(k)
              endif
              RunutBalikR(k+1)
       endfor
```

Untuk algoritma KMP, program contoh adalah dalam bahasa Java sebagai berikut.

Bagian 1: Inti Pencocokan KMP

```
// Filename: kmpMatch.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()
```

Bagian 2: Untuk fungsi pembatas KMP

```
// Filename: computeFail.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-11];
      else {     // no match
        fail[i] = 0;
        i++;
      }
    }
    return fail;
  } // end of computeFail()
```

Bagian 3: Sampel Program Utama

```
// Filename: main.java
  public static void main(String args[])
  { if (args.length != 2) {
      System.out.println("Usage:       java
KmpSearch
                           <text>
<pattern>");
      System.exit(0);
    }
    System.out.println("Text: " + args[0]);
    System.out.println("Pattern:     "     +
args[1]);

    int posn = kmpMatch(args[0], args[1]);
    if (posn == -1)
      System.out.println("Pattern       not
found");
    else
      System.out.println("Pattern starts at
posn "
                                        +
posn);
  }
```

Untuk algoritma Boyer-Moore, dapat diimplementasi seperti sebagai berikut yang memanfaatkan bahasa Java.

Bagian 1: Inti program pencocokan dengan BM.

```
// Filename: bmMatch.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()
```

Bagian 2: Fungsi Kemunculan Terakhir

```
//Filename: buildLast.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()
```

Bagian 3: Sampel Program Utama

```
// Filename: main.java
  public static void main(String args[])
  { if (args.length != 2) {
      System.out.println("Usage:       java
BmSearch
                           <text>
<pattern>");
      System.exit(0);
    }
    System.out.println("Text: " + args[0]);
    System.out.println("Pattern:     "     +
args[1]);

    int posn = bmMatch(args[0], args[1]);
    if (posn == -1)
      System.out.println("Pattern       not
found");
    else
      System.out.println("Pattern starts at
posn "
                                        +
posn);
```

V. Analisis Lanjutan: Kompleksitas

Untuk algoritma runut balik, setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif. Jika jumlah simpul dalam pohon ruang status adalah 2^n atau $n!$, maka untuk kasus terburuk, algoritma runut-balik membutuhkan waktu dalam $O(p(n)2^n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

Untuk algoritma brute force pada pencocokan string, kasus terburuknya adalah dengan jumlah perbandingan $m(n - m + 1) = O(mn)$. Sebagai contoh, T: "aaaaaaaaaaaaaaaaaaaaaaaaaah", dan P: "aaah". Kasus terbaiknya adalah dengan kompleksitas *O(n)* di mana terjadi bila karakter pertama pola P tidak pernah sama dengan karakter teks T yang dicocokkan. Jumlah perbandingannya menjadi maksimal *n* kali. Kasus rata-ratanya adalah dengan kompleksitas *O(m+n)*, yang sangat cepat.

Untuk algoritma pencocokan string dengan KMP, untuk menghitung fungsi pinggiran memerlukan *O(m)*, sedangkan untuk pencarian string sendiri adalah *O(n)*. Kompleksitas waktu algoritma KMP menjadi *O(m+n)*. Ini akan sangat cepat dibandingkan brute force. Kelemahan KMP adalah ketika alfabetnya semakin banyak. Ini dikarenakan peluang untuk bertemu dengan kecocokan semakin menipis seiring banyaknya alfabet yang meningkat.

Untuk algoritma pencocokan string dengan Boyer-Moore, kasus terburuknya adalah *O(nm+A)*. Tetapi, BM cepat ketika alfabet A sangat besar, dan akan menjadi lambat ketika alfabet sedikit. Contohnya, algoritma BM baik untuk teks yang beralfabet Latin, tetapi kurang untuk alfabet biner.

VI. Kesimpulan

Pada akhirnya, kesimpulannya adalah pendeteksian kata yang salah dan sugesti kata yang berkenaan dapat diterapkan algoritma runut balik dan juga algoritma pencocokan *string* dengan KMP maupun Boyer-Moore. Namun, dengan adanya algoritma ini, tentu dapat dikembangkan lebih lanjut mengenai kemangkusan dan kesangkilan algoritma ini dengan perangkat lunak pemroses kata itu sendiri, tergantung juga lingkungan pemakaiannya. Adapun aspek yang harus ditinjau bila diimplementasi pada kenyataan, yaitu memori. Kadang kala, untuk algoritma tertentu, memori yang diperlukan menjadi banyak dan mungkin juga menimbulkan pemanggilan fungsi atau prosedur yang pada dasarnya lebih menguras memori dan waktu. Untuk hal yang sederhana, algoritma runut balik dan algoritma pencocokan KMP dan BM adalah cukup mangkus dan sangkil, dengan kelebihan, kelemahan, dan kompleksitas tersendiri.

VII. Ucapan Terima Kasih

Saya ingin mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas segala yang diberikan-Nya sehingga makalah ini dapat selesai tepat waktu. Saya juga ingin mengucapkan terima kasih kepada kedua orang tua saya yang telah membesarkan dan mendidik saya sehingga saya dapat menjadi mahasiswa di Institut Teknologi Bandung. Saya juga ingin mengucapkan terima kasih kepada dosendosen mata kuliah IF2211 tercinta, yaitu Bapak Dr. Ir. Rinaldi Munir dan Ibu Dr. Nur Ulfa Maulidevi, S.T., M.T. atas segala ilmu yang telah dibagikan melalui mata kuliah ini sehingga saya mampu menyelesaikan makalah ini. Saya juga ingin berterima kasih kepada teman-teman seperjuangan saya atas semangat dan dukungan yang diberikan selama penulisan makalah ini.

Referensi

[1] Donald E. Knuth (1968). *The Art of Computer Programming*. Addison-Wesley.
[2] Thomas H. Cormen; Charles E. Leiserson; Ronald R. Rivest; Cliff Stein (1990). *Introduction to Algorithms*. McGraw-Hill.
[3] Gurari, Eitan (1999). Backtracking algorithms "CIS680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms".
[4] Munir, Rinaldi. 2015. Slide Kuliah Strategi Algoritma 2015. http://informatika.stei.itb.ac.id/~rinaldi.munir
[5] Munir, Rinaldi. 2004. IF2251 Strategi Algoritmik – Algoritma Pencarian String (*String Matching*). 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, 2 Mei 2015

Erick Chandra
13513021