Simulasi Pencarian Pola Kata Pada File Teks Berdasarkan Multi Pattern Matching Dengan Metode Wu-Manber

Yudi[1)] , Tintin Chandra[2)]
STMIK IBBI Medan
Jl. Sei Deli No. 18 Medan, 061-4567111
Email : ynn_linc@yahoo.com

Abstrak

Multi-pattern matching problem dapat dideskripsikan sebagai berikut, anggap P = {p_1, p_2, …, p_k} merupakan sebuah set atau kumpulan pola (pattern), yang merupakan string karakter dari sekumpulan alfabet tertentu ∑. Anggap T = t_1, t_2, …, t_N merupakan sebuah teks panjang, yang juga terdiri dari karakter yang diambil dari sekumpulan alfabet tertentu ∑. Sasarannya adalah menemukan semua kemunculan dari semua pattern dari P pada T. Salah satu algoritma yang dapat diterapkan untuk menyelesaikan multi-pattern matching problem ini adalah algoritma Sun Wu dan Udi Manber. Proses kerja dari algoritma Sun Wu dan Udi Manber ini terbagi menjadi dua tahapan, sedangkan, metode penelitian yang digunakan adalah model Waterfall. Perangkat lunak ini akan menampilkan prosedur penyelesaian multi-pattern matching problem dengan menggunakan algoritma Sun Wu dan Udi Manber secara tahap demi tahap dan menyediakan fasilitas pencarian sekumpulan pola pada sekumpulan file yang terdapat pada lokasi pencarian tertentu.

Kata kunci: *simulasi pencarian, multi pattern matching, Algoritma Sun Wu*

Abstract

Matching problem Multi-pattern earn the following dideskripsikan, assume P = { p1, p2 …, pk} representing a setting or pattern corps (pattern), representing character string from a group of certain alfabet ∑. Assume T = t1, t2 …, tN represent a long text, which also consist of taken away from character a group of certain alfabet . Its target is to find all apparition from all pattern of P at T. One of the algorithm able to be applied to finish this problem matching multi-pattern is algorithm of Sun Wu and of Udi Manber. Process work from algorithm of Sun Wu and of Udi this Manber divided to become two step, while, research method the used Waterfall method. This Software will present procedure of[is solving of problem matching multi-pattern by using algorithm of Sun Wu and of Udi Manber phasely for the shake of phase and provide seeking facility a group of pattern at a group of file found on certain seeking location.

Keywords*: Seeking Simulation, matching pattern multi, Algorithm Sun Wu*

1. Pendahuluan

Problema *multi-pattern matching* dapat diilustrasikan sebagai berikut, anggap P = {p_1, p_2, …, p_k} merupakan sebuah set atau kumpulan pola (*pattern*), yang merupakan *string* karakter dari sekumpulan alfabet tertentu ∑. Anggap T = t_1, t_2, …, t_N merupakan sebuah teks panjang, yang juga terdiri dari karakter yang diambil dari sekumpulan alfabet tertentu ∑. Sasarannya adalah menemukan semua kemunculan dari semua *pattern* dari P pada T. Problema *multi-pattern matching* ini memiliki banyak aplikasi, seperti digunakan pada aplikasi sekuritas untuk mendeteksi *keyword* tertentu yang mencurigakan, dan juga digunakan pada pencarian DNA dengan mentranslasikan sebuah pencarian rata-rata ke sebuah pencarian untuk *pattern* tertentu yang berjumlah besar. Aplikasi lainnya dari *multi-pattern matching* ini adalah proses pencarian *file* yang memiliki isi yang hampir sama pada sekumpulan *file* yang banyak dan fasilitas '*match and replace*' (mar) dimana setiap pola dihubungkan dengan pola yang akan diubah. Ketika sebuah pola ditemukan, maka akan diganti dengan pola yang akan diubah pada outputnya. Problema *multi-pattern matching* ini muncul karena algoritma *string matching* tidak efisien apabila diterapkan untuk mencari beberapa pola sekaligus.

Aho dan Corasick menyediakan sebuah algoritma berwaktu linier untuk menyelesaikan problema ini, dengan berdasarkan pada pendekatan automata. Algoritma berwaktu linier dapat bekerja optimal pada kasus terburuk (*worst case*), tetapi seperti pada algoritma *string-matching* biasa yang didemonstrasikan

oleh Boyer dan Moore, terdapat kemungkinan untuk mengabaikan sekumpulan teks dengan porsi besar pada saat pencarian, yang memberikan waktu eksekusi yang lebih cepat daripada algoritma linier pada kasus rata-rata (*average case*). Sun Wu dan Udi Manber memberikan sebuah pendekatan yang menggunakan ide dari Boyer dan Moore tersebut. Algoritma Sun Wu dan Udi Manber yang cukup sederhana ini dipublikasikan pada sekitar bulan Mei 1994 [*A Fast Algorithm for Multi-Patterns Searching*]. Desain dari algoritma ini berkonsentrasi pada pencarian biasa daripada keadaan kasus terburuk. Hal ini diyakini oleh pembuatnya mampu membuat algoritma ini menjadi jauh lebih cepat daripada algoritma lainnya pada prakteknya.

Isu kunci dari simulasi termasuk akuisisi dari informasi sumber yang sah tentang referen, pemilihan kunci, karakteristik dan sifat, penggunaan pencapaian yang disederhanakan dan asumsi di dalam simulasi, serta ketelitian dan validasi dari hasil simulasi. Simulasi sering dipakai di dalam pelatihan dari sipil dan personel militer. Hal ini biasanya muncul saat peralatan mahal dan terlalu bahaya untuk memungkinkan pelatih menggunakan peralatan riil di dalam dunia nyata. Di dalam situasi tertentu akan dilakukan simulasi di dalam lingkungan virtual yang "aman".

Isu dasar lainnya adalah kompleksitas dari sebuah model. Sebagai contoh, jika pemodelan dari penerbangan dari sebuah pesawat, maka dapat ditempelkan setiap bagian mekanikal dari pesawat ke dalam model kita dan memerlukan hampir model kotak putih dari sebuah sistem. Bagaimanapun, ongkos dari komputasional untuk penambahan sedemikian besar dari detail dapat secara efektif menghambat penggunaan dari sebuah model. Lebih jauhnya, ketidakpastian dapat meningkat karena sistem kompleks berlebihan, karena setiap bagian terpisah memasukkan sejumlah variansi ke dalam model. Oleh karena itu biasanya pencapaian yang cocok untuk mengurangi model ke dalam ukuran yang sensibel. Ilmuwan sering dapat menerima beberapa pencapaian dengan tujuan untuk mendapatkan model yang sederhana. Sebagai contoh mekanik klasikal Newton merupakan model pencapaian dari dunia nyata. Model Newton cukup untuk beberapa situasi kehidupan nyata, selama kecepatan partikel berada di bawah kecepatan cahaya.

Sebuah masalah dipecahkan dengan mendeskripsikan langkah-langkah penyelesaiannya. Misalnya masalah pengurutan (*sorting*) berikut: diberikan sejumlah bilangan bulat, tuliskan semua bilangan bulat tersebut dalam urutan yang menaik. Metode untuk pengurutan data diskrit sudah banyak ditemukan orang. Metode pengurutan sering digambarkan dalam sejumlah langkah terbatas yang mengarah pada solusi permasalahan. Urutan langkah-langkah penyelesaian masalah ini dinamakan algoritma.

Kata algoritma berasal dari nama seorang ahli matematika berkebangsaan Persia yang hidup pada abad ke-9 yang bernama Abu Abdullah Muhammad bin Musa Al-Khawarizmi. Pada awalnya, kata 'algorism' diartikan sebagai aturan-aturan untuk melakukan proses aritmatika menggunakan numerik Arab. Kata 'algorism' diubah menjadi kata 'algorithm' pada abad ke-18. Sekarang, pengertian dari kata ini mencakup semua prosedur terhingga untuk menyelesaikan problema atau melakukan pekerjaan.

Penerapan pertama dari algoritma yang ditulis untuk sebuah komputer adalah '*Ada Byron's notes on the analytical engine*' yang ditulis pada tahun 1842, dimana Ada Byron dianggap oleh kebanyakan orang sebagai *programmer* pertama di dunia. Walaupun, sejak Charles Babbage tidak menyelesaikan *analytical engine*-nya, algoritma ini tidak pernah diimplementasikan lagi.

Multi-pattern matching problem dapat dideskripsikan yaitu Anggap $P = \{p_1, p_2, \ldots, p_k\}$ merupakan sebuah set atau kumpulan pola (*pattern*), yang merupakan *string* karakter dari sekumpulan alfabet tertentu $\sum$. Anggap $T = t_1, t_2, \ldots, t_N$ merupakan sebuah teks panjang, yang juga terdiri dari karakter yang diambil dari sekumpulan alfabet tertentu $\sum$. Sasarannya adalah menemukan semua kemunculan dari semua *pattern* dari P pada T. Sebagai contoh, program *fgrep* dan *egrep* pada UNIX mendukung *multi-pattern matching* melalui opsi –f.

Problema *multi-pattern matching* ini memiliki banyak aplikasi, seperti digunakan pada aplikasi sekuritas untuk mendeteksi *keyword* tertentu yang mencurigakan, dan juga digunakan pada pencarian DNA dengan mentranslasikan sebuah pencarian rata-rata ke sebuah pencarian untuk *pattern* tertentu yang berjumlah besar. Aplikasi lainnya dari *multi-pattern matching* ini adalah proses pencarian *file* yang mirip pada sekumpulan *file* yang banyak dan fasilitas '*match and replace*' (mar) dimana setiap pola dihubungkan dengan pola yang akan diubah. Ketika sebuah pola ditemukan, maka akan diganti dengan pola yang akan diubah pada outputnya. Problema *multi-pattern matching* ini muncul karena algoritma *string matching* tidak efisien apabila diterapkan untuk mencari beberapa pola sekaligus.

Aho dan Corasick menyediakan sebuah algoritma berwaktu linier untuk menyelesaikan problema ini, dengan berdasarkan pada pendekatan automata. Algoritma ini merupakan dasar dari *tool fgrep* pada UNIX. Algoritma berwaktu linier dapat bekerja optimal pada kasus terburuk (*worst case*), tetapi seperti pada algoritma *string-matching* biasa yang didemonstrasikan oleh Boyer dan Moore, terdapat kemungkinan untuk mengabaikan sekumpulan teks dengan porsi besar pada saat pencarian, yang memberikan waktu eksekusi yang lebih cepat daripada algoritma linier pada kasus rata-rata (*average case*). Commentz-Walter menyediakan sebuah algoritma untuk *multi-pattern matching problem* yang

mengkombinasikan teknik Boyer-Moore dengan algoritma Aho-Corasick. Algoritma Commentz-Walter ini jauh lebih cepat daripada dari algoritma Aho-Corasick pada prakteknya. Hume mendesain sebuah *tool* yang disebut *gre* berdasarkan pada algoritma ini dan versi 2.0 dari *fgrep* dari proyek GNU menggunakan algoritma Hume ini. Baeza-Yates juga memberikan sebuah algoritma yang mengkombinasikan algoritma Boyer-Moore-Horspool (yang sedikit berbeda dengan algoritma Boyer-Moore klasik) dengan algoritma Aho-Corasick.

Sun Wu dan Udi Manber menyediakan sebuah pendekatan berbeda yang juga menggunakan ide dari Boyer dan Moore. Algoritma ini jauh lebih sederhana. Versi awal dari algoritma ini merupakan bagian dari versi kedua dari *agrep*. Versi sekarang (terbaru) ini digunakan pada *glimpse*. Desain dari algoritma ini berkonsentrasi pada pencarian biasa daripada keadaan kasus terburuk. Hal ini diyakini oleh pembuatnya mampu membuat algoritma ini menjadi jauh lebih cepat daripada algoritma lainnya pada prakteknya.

Ide dasar dari algoritma *string-matching* Boyer-Moore adalah yaitu asumsikan panjang pola adalah sebesar m. Proses akan dimulai dengan membandingkan karakter terakhir dari pola dengan t_m, karakter ke-m dari teks. Jika terdapat sebuah ketidakcocokan (dan pada kebanyakan teks, kemungkinan dari ketidakcocokan jauh lebih besar daripada kemungkinan dari kecocokan), maka ditentukan kemunculan paling kanan (*rightmost occurrence*) dari t_m pada pola dan geserlah sesuai dengan konsep itu. Sebagai contoh, jika t_m tidak terdapat pada pola sama sekali, maka dapat langsung digeser sebanyak m karakter dan pengamatan berikutnya pada t_{2m}, jika t_m coock hanya karakter ke-4 dari pola, maka dapat digeser sebanyak (m – 4) dan seterusnya. Pada teks berbahasa alami, pergeseran dengan ukuran m atau mendekati nilai tersebut akan muncul pada banyak kasus, sehingga algoritma akan bekerja sangat cepat.

Sun Wu dan Udi Manber ingin menggunakan ide yang sama untuk problema *multi-pattern matching*. Walaupun demikian, jika terdapat banyak pola, dan Wu-Manber ingin agar algoritmanya mendukung hingga 10 ribu pola, kemungkinannya adalah kebanyakan karakter pada teks cocok dengan karakter terakhir dari beberapa pola, maka hanya terdapat sedikit pergeseran saja jika menggunakan jenis pergeseran dari Boyer-Moore. Wu-Manber mampu mengatasi masalah ini dan menjaga pokok dan kecepatan dari algoritma Boyer-Moore.

Tahapan pertama adalah sebuah tahapan proses persiapan (*preprocessing*) kumpulan pola. Aplikasi yang menggunakan sebuah kumpulan pola tertentu untuk banyak pencarian akan memperoleh keuntungan dari menyimpan hasil tahapan proses persiapan pada sebuah *file* (atau bahkan pada memori). Walaupun demikian, tahapan ini cukup efisien, dan pada kebanyakan kasus dapat diselesaikan secepat kilat. Tiga tabel dibangun pada tahapan *preprocessing* ini, sebuah tabel SHIFT, HASH dan PREFIX. Tabel SHIFT adalah mirip, namun tidak sama persis dengan tabel SHIFT biasa pada algoritma Boyer-Moore. Tabel ini digunakan untuk menentukan berapa banyak karakter pada teks yang dapat digeser (diabaikan) ketika teks dibaca. Tabel HASH dan PREFIX digunakan ketika nilai pergeseran (*shift*) sebesar 0. Kedua tabel ini digunakan untuk menentukan popla mana yang merupakan kandidat untuk pencocokan dan untuk memverifikasi pencocokan. Detail dari proses akan dijabarkan pada pembahasan berikut ini.

Hal pertama yang dilakukan adalah menghitung panjang minimum dari sebuah pola, disimbolkan dengan m, dan anggap hanya m buah karakter dari setiap pola. Dengan perkataan lain, ditentukan sebuah persyaratan bahwa semua pola memiliki panjang yang sama. Persyaratan ini sangat penting untuk efisiensi dari algoritma. Cermati bahwa apabila salah satu pola sangat pendek, misalkan memiliki panjang 2, maka pergeseran tidak dapat lebih dari 2, sehingga dengan menggunakan pola yang pendek akan membuat pendekatan ini menjadi kurang efisien.

Daripada mengamati karakter dari teks satu per satu, algoritma ini mengasumsikan teks dalam blok dengan ukuran B. Anggap M adalah total ukuran dari semua pola, M = k * m dan anggap c adalah ukuran dari alfabet. Penemu algoritma merekomendasikan bahwa nilai B yang bagus adalah nilai yang berorder $\log_c 2M$, pada prakteknya, penemu algoritma menyarankan untuk menggunakan B = 2 atau B = 3. Tabel SHIFT menggunakan aturan yang sama dengan algoritma Boyer-Moore biasa, kecuali bahwa tabel ini menentukan pergeseran berdasarkan pada B karakter terakhir daripada hanya menggunakan satu karakter. Sebagai contoh, jika string dari B buah karakter pada teks tidak muncul pada sembarang pola, maka dapat digeser sebanyak m – B + 1. Asumsikan bahwa tabel SHIFT mencakup sebuah entri untuk setiap kemungkinan *string* dengan ukuran B, maka ukurannya adalah $|\sum|^B$. Secara aktual, penemu algoritma menggunakan sebuah tabel terkompresi dengan beberapa *string* dipetakan ke entri yang sama untuk menghemat ruang. Setiap *string* dengan ukuran B dipetakan (dengan menggunakan fungsi *hash* yang akan dibahas pada pembahasan selanjutnya) ke sebuah integer yang digunakan sebagai indeks ke tabel SHIFT. Nilai dari tabel SHIFT menentukan seberapa jauh pergeseran (lompatan) ke depan dapat dilakukan ketika teks sedang dibaca. Anggap $X = x_1 \ldots x_B$ adalah B buah karakter pada teks yang akan dibaca sekarang, dan asumsikan X dipetakan ke entri ke-i pada SHIFT, maka terdapat dua kasus:

1. X tidak muncul sebagai sebuah *substring* pada sembarang pola dari P.
   Pada kasus ini, dapat dilakukan pergeseran m – B + 1 buah karakter pada teks. Sembarang pergeseran yang lebih kecil akan membandingkan B buah karakter terakhir dari teks dengan sebuah *substring* dari salah satu dari pola yang merupakan sebuah ketidakcocokan. Nilai m – B + 1 ini disimpan dalam SHIFT[i].
2. X muncul pada beberapa pola.
   Pada kasus ini, ditemukan kemunculan paling kanan dari X pada beberapa pola, misalkan diasumsikan bahwa X berakhir pada posisi q dari P_j dan bahwa X tidak berakhir pada posisi yang lebih besar dari q pada pola lainnya, maka nilai m – q disimpan pada SHIFT[i].

Untuk menghitung nilai dari tabel SHIFT, diasumsikan setiap pola $p_i = a_1a_2 \ldots a_m$ secara terpisah. Setiap *substring* dari p_i dengan ukuran B: $a_{j-B+1} \ldots a_j$ dipetakan pada SHIFT dan set nilai yang berkorespondensi ke nilai minimumnya sekarang (nilai awal untuk semua substring adalah m – B + 1) dan m – j (jumlah pergeseran yang diperlukan untuk memperoleh *substring* ini).

Nilai pada tabel SHIFT adalah nilai teraman maksimum yang mungkin untuk pergeseran. Mengganti sembarang entri pada tabel SHIFT dengan nilai yang lebih rendah akan membuat pergeseran menjadi lebih sedikit dan akan memerlukan waktu lebih banyak, tetapi tetap akan aman, dimana tidak ada pencocokan yang akan terlewatkan. Sehingga dapat digunakan sebuah tabel terkompresi untuk SHIFT, memetakan beberapa *string* berbeda ke entri yang sama asalkan di-set pergeseran minimal dari setiap *string* sebagai nilainya. Pada agrep, dilakukan kedua proses tersebut. Ketika jumlah pola sedikit, dipilih B = 2 dan digunakan tabel sebenarnya untuk SHIFT, sebaliknya akan dipilih B = 3 dan menggunakan sebuah tabel terkompresi untuk SHIFT. Pada kedua kasus, algoritma tetap akan berjalan, tidak peduli apakah tabel SHIFT terkompresi atau tidak.

Selama nilai pergeseran lebih besar daripada 0, maka pergeseran dapat dilakukan dan proses pembacaan dapat dilanjutkan. Hal inilah yang paling sering terjadi. Jika tidak, maka terdapat kemungkinan bahwa *substring* yang bersangkutan pada teks cocok dengan beberapa pola pada daftar pola. Untuk mencegah perbandingan *substring* dengan setiap pola pada daftar pola, digunakan sebuah teknik hash untuk meminimumkan jumlah pola yang akan dibandingkan. Sebelumnya telah dihitung sebuah pemetaan dari B buah karakter ke sebuah integer yang akan digunakan sebagai indeks ke tabel SHIFT. Nilai integer yang sama juga digunakan sebagai indeks ke tabel lainnya, yang disebut HASH. Entri ke-i dari tabel HASH, HASH[i], terdiri dari sebuah penunjuk (*pointer*) ke sebuah daftar pola dimana B buah karakter di-*hash* ke i. Tabel HASH secara khusus akan sangat jarang, karena tabel ini hanya akan menyimpan pola sementara tabel SHIFT menyimpan semua kemungkinan string dengan ukuran B. Hal ini merupakan penggunaan memori yang kurang efisien, tetapi terdapat kemungkinan untuk menggunakan kembali fungsi *hash* (pemetaan), sehingga dapat menghemat banyak waktu.

Anggap h adalah nilai *hash* dari suffiks pada teks dan asumsikan SHIFT[h] = 0. Nilai dari HASH[h] adalah sebuah penunjuk p yang akan menunjuk ke dua tabel terpisah pada waktu yang sama, sebuah daftar dari penunjuk ke pola, PAT_POINT, diurutkan berdasarkan nilai *hash* dari B buah karakter terakhir dari setiap pola. Penunjuk p menunjuk ke awal dari daftar pola dimana nilai *hash*-nya adalah sebesar h. Untuk menemukan akhir dari daftar ini, akan dilakukan penambahan terhadap penunjuk ini hingga nilainya sama dengan nilai pada HASH[h+1], karena keseluruhan daftar diurutkan berdasarkan nilai *hash*. Sebagai contoh, jika SHIFT[h] $\neq$ 0 maka HASH[h] = HASH[h+1], karena tidak ada pola yang memiliki suffiks yang di-*hash* ke h. Sebagai tambahan, juga disimpan sebuah tabel yang disebut PREFIX.

Teks pada bahasa alami tidak bersifat acak. Sebagai contoh, suffiks ‘ion’ atau ‘ing’ adalah sangat umum pada teks berbahasa Inggris. Suffiks ini tidak hanya akan sering muncul pada teks, tetapi juga mungkin muncul pada beberapa pola. Hal ini akan menyebabkan tabrakan pada tabel HASH, yaitu semua pola dengan suffiks yang sama akan dipetakan ke entri yang sama pada tabel HASH. Ketika menemukan suffix yang bersangkutan pada teks, akan ditemukan bahwa nilai SHIFT-nya sebesar 0 (asumsikan merupakan suffix dari pola tertentu) dan dicoba secara terpisah, semua pola dengan suffiks iniuntuk melihat apakah mereka cocok dengan teks. Untuk mempercepat proses ini, diperkenaklan tabel lainya yang disebut PREFIX.

Sebagai tambahan untuk memetakan B buah karakter terakhir dari semua pola, juga dipetakan B’ buah karakter pertam dari semua pola ke tabel PREFIX. Para penemu algoritma menyarankan bahwa nilai B’ = 2 merupakan nilai yang bagus. Ketika ditemukan sebuah SHIFT bernilai 0, dan akan menuju ke tabel HASH untuk menentukan apakah terdapat sebuah kecocokan, maka akan dicek nilai pada tabel PREFIX terlebih dahulu. Untuk setiap suffiks, tabel HASH tidak hanya mencakup daftar dari semua pola dengan suffiks ini, tetapi juga mencakup nilai *hash* dari prefiksnya. Prefiks yang bersangkutan pada teks akan dihitung dengan menggeser m – B’ buah karakter ke kiri dan menggunakannya untuk memfilter pola yang memiliki suffiks yang sama tetapi prefiksnya berbeda. Ini merupakan metode pemfilteran yang efektif karena terdapat sedikit kemungkinan untuk memiliki pola berbeda yang menggunakan prefiks dan suffiks yang sama. Hal ini juga merupakan tindakan penyeimbangan yang bagus dimana kerja tambahan

pada perhitungan fungsi *hash* dari prefiks hanya dilakukan jika nilai SHIFT sebesar 0, dimana hal ini hanya akan muncul ketika terdapat banyak pola dan terdapat banyak kesamaan.

Tahapan proses permulaan ini kelihatan cukup panjang, tetapi pada prakteknya akan dilakukan dengan cepat. Pada implementasi dari penemu algoritma, diset ukuran dari ketiga tabel sampai 2^{15} = 32768. Menjalankan algoritma pencocokan pada sebuah *file* teks yang hampir kosong, hanya diperlukan sekitar 0,16 sekon untuk 10 ribu pola.

Perulangan utama dari algoritma ini terdiri dari beberapa langkah yaitu:

1. Hitung sebuah nilai *hash* h berdasarkan B buah karakter yang bersangkutan dari teks (dimulai dengan t_{m-B+1} ... t_m).
2. Periksa nilai dari SHIFT[h]: jika > 0, maka geser teks dan kembali ke langkah 1, jika tidak maka lanjut ke langkah 3.
3. Hitung nilai *hash* dari prefiks dari teks (mulai dari m buah karakter ke kanan dari posisi sekarang), misalkan disebut sebagai text_prefix.
4. Periksa untuk setiap p, jika HASH[h] ≤ p < HASH[h+1], apakah PREFIX[p] = text_prefix. Jika sama, maka periksa pola aktual (diberikan oleh nilai PAT_POINT[p]) terhadap teks secara langsung.

Penggalan *code* dari algoritma ini dalam bahasa C dapat dijabarkan sebagai berikut:

```
while (text <= textend) {
  h = (*text<<Hbits)+(*(text-1));  /* The hash function (we use Hbits=5) */
  if (Long) h = (h<<Hbits)+(*(text-2)); /* Long=1 when the number of patterns warrants B=3 */
  shift = SHIFT[h];  /* we use a SHIFT table of size 2^15 = 32768 */
  if (shift == 0) { /* ''h = h & mask_hash'' can be used here if HASH is smaller then SHIFT */
    text_prefix = (*(text-m+1)<<8) + *(text-m+2);
    p = HASH[h];
      p_end = HASH[h+1];
      while (p++ < p_end) { /* loop through all patterns that hash to the same value (h) */
          if(text_prefix != PREFIX[p]) continue;
          px = PAT_POINT[p];
          qx = text-m+1;
          while (*(px++) == *(qx++)); /* check the text against the pattern directly */
          if (*(px-1) == 0) { /* 0 indicates the end of a string */
              report a match
          }
          shift = 1;
      }
   text += shift;
}
```

2. Metode Penelitian

Algoritma yang digunakan untuk merancang perangkat lunak pemahaman dan implementasi algoritma Sun Wu dan Udi Manber untuk melakukan *Multi-Pattern Matching* ini dapat dibagi menjadi beberapa bagian yaitu:

1. Algoritma Fungsi Pengecekan Data yang Sama.

   Algoritma fungsi pengecekan data yang sama ini digunakan untuk melakukan pengecekan apakah data yang di-*input* sudah tercantum pada daftar atau belum. Fungsi ini akan menghasilkan sebuah nilai *Boolean*, dimana TRUE berarti data sudah terdaftar sebelumnya, FALSE berarti data belum terdaftar. Prosedur kerja dari algoritma fungsi pengecekan data yang sama ini dapat dijabarkan sebagai berikut:

   ```
   Fungsi SameData(pcText)
     Jika Data.Count = 0 maka Keluar dari fungsi
              Untuk nI = 0 sampai Data.Count - 1
              Jika Data.Item(nI) = pcText maka
             SameData = True
             Keluar dari fungsi
     SameData = False
   ```

2. Algoritma *Multi-Pattern Matching*
   Algoritma ini memerlukan input berupa lokasi pencarian dokumen dan kumpulan pola yang akan dicari. Output dari algoritma ini adalah posisi dari setiap pola yang dicari dalam sekumpulan dokumen teks. Perincian dari algoritma ini adalah sebagai berikut:
   a. Algoritma tahapan *Preprocessing*.
      1. Tahapan *Preprocessing*, dapat dibagi menjadi dua bagian, yaitu:
         a. Tahapan pengambilan variasi panjang dari semua pola
            Untuk i = 1 sampai [Jumlah Pola]
            Jika Panjang(Pola(i)) belum dimasukkan ke dalam *array* Panjang maka
            Masukkan Panjang(Pola(i)) ke dalam *array* Panjang
            Untuk i = 1 sampai [Jumlah Pola] – 1
                Untuk j = i + 1 sampai [Jumlah Pola]
                    Jika *array* Panjang(i) > *array* Panjang(j) maka
                        Tukar nilai Panjang(i) dan Panjang(j)
                        Indeks = 0

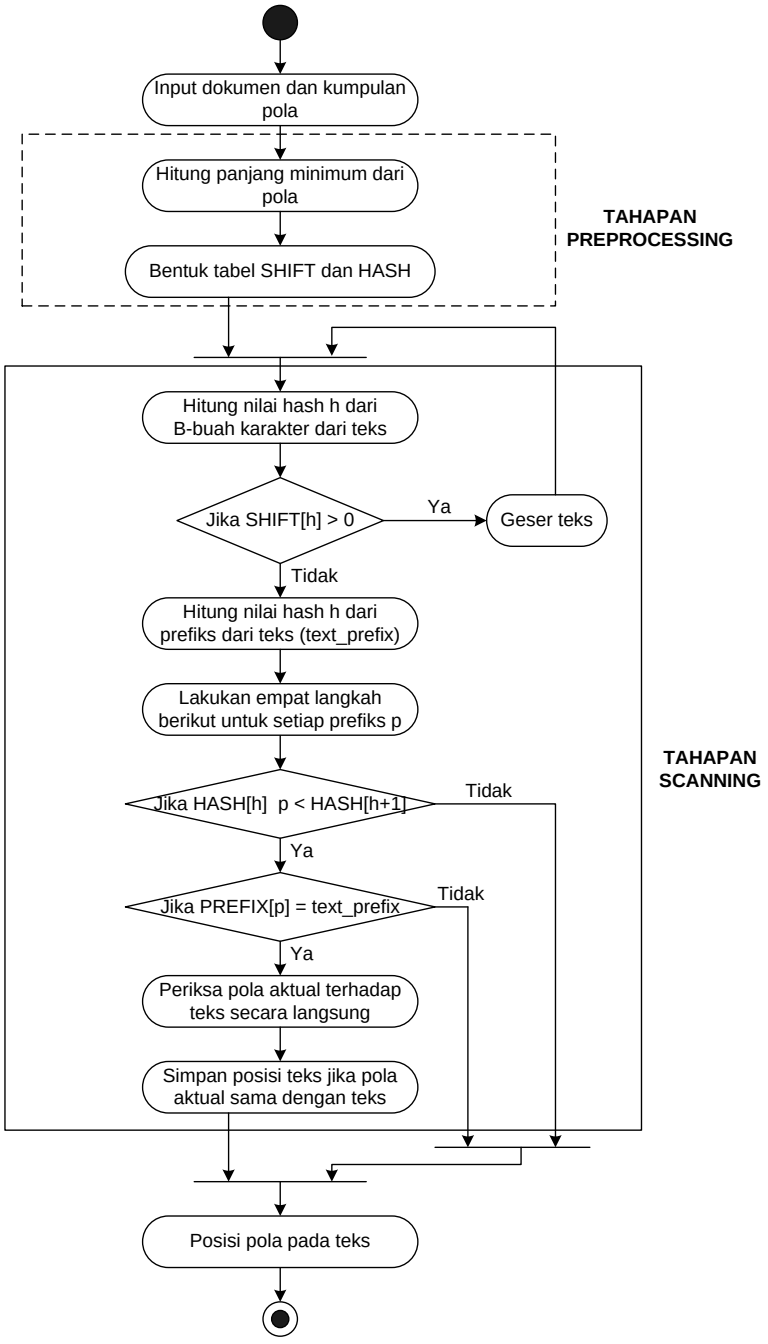
         b. Tahapan pembentukan tabel
            Indeks = Indeks + 1
            m = Panjang(Indeks)
            B = 2 (dapat juga diganti menjadi B = 3)
            Bentuk tabel SHIFT
            Bentuk tabel HASH
            Bentuk tabel PREFIX
   b. Algoritma tahapan *Scanning.*
      pointer = 1
      Selama (text masih ada yang belum di-scan), lakukan proses berikut :
            h = lakukan proses pergeseran pointer ke kanan terhadap teks sebanyak
                    m karakter dan ambil posisi tempat pointer berhenti
            MatchWindow = ambil teks pada posisi pointer berhenti sebanyak m buah
      Suff = ambil B buah karakter *suffix* dari MatchWindow
      Shift = ambil nilai dari tabel SHIFT sesuai dengan *suffix* Suff
      Jika nilai Shift tidak ditemukan, maka
      Set nilai Shift = B – m + 1
      Jika shift = 0 maka
      Pos = cari posisi pada list HASH yang merupakan kemungkinan terdapat teks MatchWindow
      text_prefix = ambil B buah karakter *prefix* dari teks MatchWindow
      Cek pada list HASH pada posisi Pos
      Untuk semua isi pada list HASH, lakukan proses berikut
          Jika text_prefix = PREFIX[p] maka
      Jika text pada posisi px + 1 = salah satu pola maka
      Simpan teks yang dicocokkan tadi
      pointer = pointer + 1
            Jika tidak, maka
          pointer = pointer + Shift

Sesuai dengan metodologi yang digunakan yang terdiri dari empat fase, yaitu fase permulaan (*inception*), fase perluasan (*elaboration*), fase konstruksi (*construction*) dan fase peralihan (*transition*), maka proses analisis akan dimulai dari analisis permasalahan yang menjadi landasan dan tujuan dari pembuatan perangkat lunak. Selain itu, juga perlu dijabarkan ruang lingkup dari perangkat lunak. Setelah itu, proses analisis dilanjutkan dengan perumusan persyaratan fungsional yang harus dipenuhi oleh perangkat lunak dengan menggunakan *use case* dan analisis non-fungsional yang berhubungan dengan kualitas sistem yang sesuai dengan kebutuhan pemakai.

Fase permulaan ini terdiri dari dua bagian meliputi analisis permasalahan, Pada tahapan ini, akan dilakukan proses peninjauan ulang dan pengkonfirmasian tujuan dari pembuatan perangkat lunak, yaitu untuk apa penelitian ini dilakukan. Perangkat lunak yang akan dirancang terbagi menjadi dua segmen yaitu segmen pemahaman, yang berfungsi untuk menampilkan detail dari proses kerja dari algoritma Sun Wu dan Udi Manber secara tahapan demi tahapan. Perancangan segmen ini menggunakan beberapa rumusan ruang lingkup antara lain panjang *string input* dibatasi maksimal 100 karakter, jumlah pola dibatasi maksimal 10 buah, jumlah *file* yang dibuka hanya sebanyak satu buah saja, f*ile* yang dapat dibuka hanya *file* teks yang berekstensi *.txt, *.doc dan *.rtf, disediakan sebuah fasilitas pengaturan kecepatan sehingga pemakai dapat melakukan pengaturan kecepatan transisi antar langkah sesuai

keinginan dan kebutuhan, hasil perhitungan dapat disimpan ke dalam sebuah *file* teks berekstensi *.txt dan segmen aplikasi, yang dapat digunakan untuk melakukan proses pencarian kata (*string*) pada sekumpulan dokumen. Perancangan segmen ini menggunakan beberapa rumusan ruang lingkup yaitu panjang *string* input dibatasi maksimal 1 juta karakter, jumlah pola dibatasi maksimal 1000 buah, f*ile* yang dapat dibuka hanya *file* teks yang berekstensi *.txt, *.doc dan *.rtf, jumlah *file* yang akan dilakukan proses pencarian tidak dibatasi dan hanya perlu dilakukan penginputan lokasi pencarian saja. Perangkat lunak akan membaca *file-file* yang terdapat di dalamnya secara otomatis.

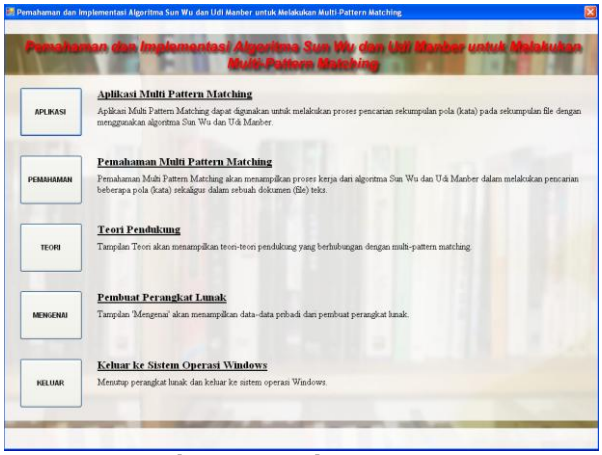
Secara garis besar, langkah kerja dari algoritma *multi-pattern matching* yang dibuat oleh Sun Wu dan Udi Manber dapat dilihat dalam bentuk *flowchart* seperti pada gambar 1.



Gambar 1. *Activity Diagram* dari Algoritma Sun Wu dan Udi Mander

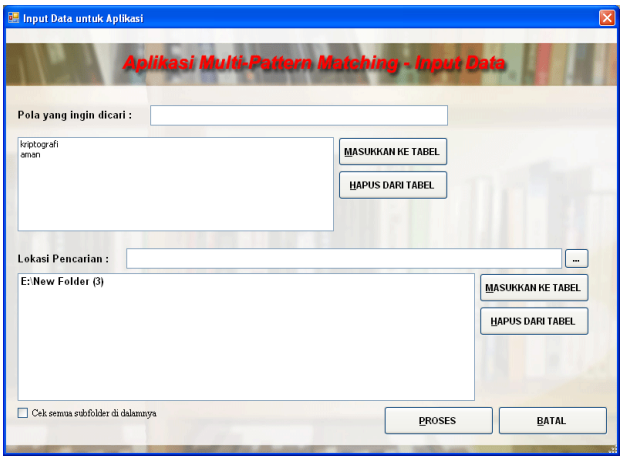
3. Hasil dan Analisis

Perangkat lunak pemahaman dan implementasi algoritma Sun Wu dan Udi Manber untuk melakukan *multi-pattern matching* ini memiliki beberapa tampilan *output* meliputi tampilan form utama seperti pada gambar 2., form *input* aplikasi seperti pada gambar 3., form aplikasi seperti pada gambar 4., form *input* pemahaman seperti pada gambar 5. dan form pemahaman seperti pada gambar 6.



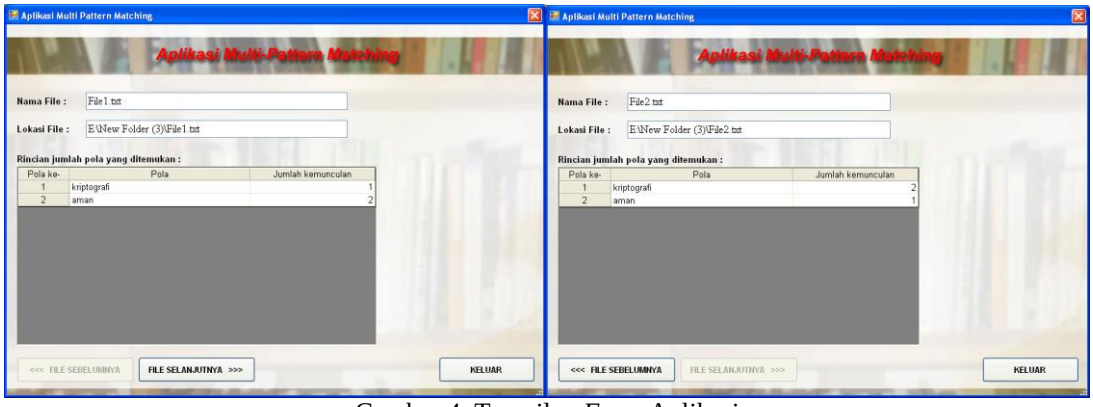
Gambar 2. Tampilan *Form Main*

Pada gambar 2., terlihat bahwa ada sebanyak 5 buah button dengan fungsi yang telah disesuaikan. Selain itu pada tampilan awal ini, terdapat salah satu button yang berisi tentang teori pendukung dari algoritma Sun Wu.



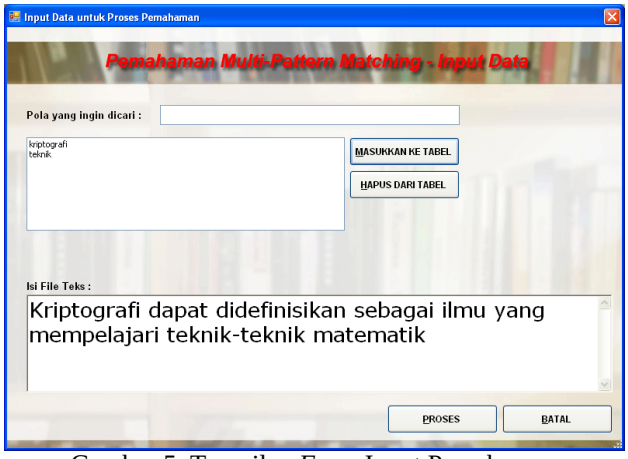
Gambar 3. Tampilan *Form* Input Aplikasi

Untuk mencari pola pada file yang berdasarkan *mutli pattern matching*, maka pada Gambar 3. tersebut, bagian *textbox* isikanlah pola yang ingin dicari selain itu juga isikanlah *textbox* pada lokasi pencarian. Pada form ini dilengkapi fasilitas button untuk memasukkan ke tabel maupun untuk menghapus dari tabel.



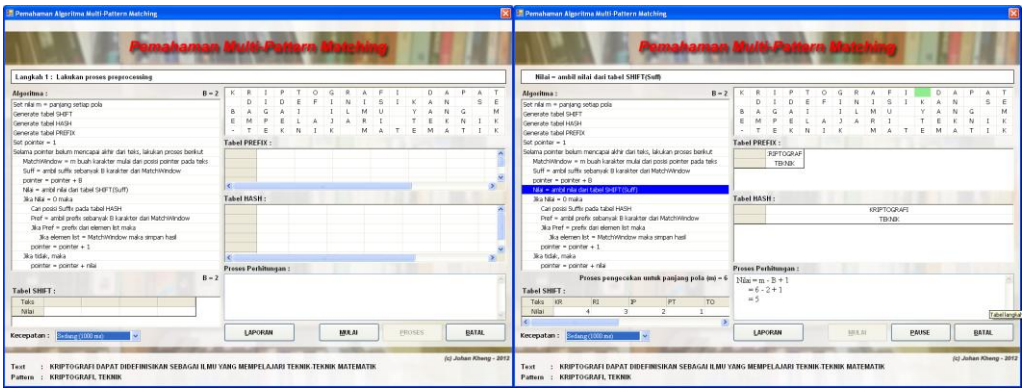
Gambar 4. Tampilan *Form* Aplikasi

Pada gambar 4. nama file yang ingin ditemukan pola katanya dimasukkan ke bagian *textbox* pada nama file dan tempat file yang ingin ditemukan polanya berada. Pada bagian *datagrid* akan ditampilkan hasil temuan kata beserta dengan banyaknya kata tersebut yang sama.



Gambar 5. Tampilan *Form* Input Pemahaman

Pada gambar 5. terdapat 4 button yang berfungsi untuk memasukkan ke tabel, menghapus dari tabel, proses pencarian dan pembatalan. Button ”masukkan ke tabel” berfungsi untuk memasukan pola kata yang ingin dicari ke dalam *listbox*, sedangkan button ”hapus dari tabel” berfungsi kebalikan dari masukkan ke tabel. Untuk button ”proses” akan menampilkan hasil pencarian yang ditemukan pada file text yang ditentukan.



Gambar 6. Tampilan *Form* Pemahaman

Pada gambar 6. akan diperlihatkan langkah demi langkah dalam proses pencarian pola kata pada file teks yang dipilih. Selain itu, pada bagian kiri gambar 6., juga ditunjukan proses animasi sampai dimana proses pencarian pola kata dengan multi pattern matching dengan metode wu-manber.

4. Kesimpulan

Berdasarkan hasil dan analisis yang dilakukan, maka dapat diambil kesimpulan dari Simulasi Pencarian Pola Kata Pada File Teks Berdasarkan Multi Pattern Matching Dengan Metode Wu-Manber yaitu bahwa simulasi ini dapat digunakan untuk membantu pemahaman mengenai proses kerja dari algoritma Sun Wu dan Udi Manber dalam mencari solusi dari problema multi-pattern matching. Kemudian algoritma Algoritma Sun Wu dan Udi Manber ini menggunakan tabel SHIFT, HASH dan PREFIX dalam proses kerjanya dengan tujuan untuk meningkatkan efisiensi dari algoritma dan Algoritma Sun Wu dan Udi Manber ini mampu melakukan proses pencarian sekumpulan pola dengan waktu eksekusi yang relatif cepat.

Daftar Pustaka

[1] Cormen, H., Leiserson, E., Rivest, L., 1990, Introduction to Algorithms, Mc Graw Hill Book Company.
[2] Hariyanto, B., 2003, Struktur Data, Edisi Kedua, Informatika, Bandung.
[3] Munir, R., Lidia L., 2002, Algoritma dan Pemrograman, Edisi Kedua.
[4] Roger S. Pressman, Ph.D., 2002, Rekayasa Perangkat Lunak : Pendekatan Praktisi (Buku Satu), Mc Graw-Hill Companies, Inc, Penerbit Andi.
[5] Wu, S. dan U. Manber, 1994, A Fast Algorithm for Multi-Patterns Searching.
[6] Wahana Komputer, 2010, Buku Tutorial 5 Hari: Belajar Pemograman Visual Basic, Penerbit Andi.