JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

USULAN TUGAS AKHIR

1. IDENTITAS PENGUSUL

Nama : Mohammad Fajrin Aziz

NRP : 5109100088

Dosen Wali : Ary Mazharuddin S, S.Kom, M.Comp.Sc

2. JUDUL TUGAS AKHIR

"Implementasi dan Perbandingan Performa Suffix Array Statis dan Dinamis untuk String Input yang Mengalami Pembaruan"

"Implementation and Performance Comparison between Static and Dynamic Suffix Array for Edited Input String"

3. URAIAN SINGKAT

Suffix array adalah struktur data yang *simple* dan kuat untuk menyimpan *suffix-suffix* yang sudah terurut dari suatu *string*. Kekuatan dari *suffix array* terletak pada efisiensi memori dan waktu konstruksi yang dibutuhkan. Selain itu, penyimpanan *suffix* secara leksikografi memungkinkan pencarian *substring* (apakah *string* N adalah *substring* dari A) dengan *binary search*.

Permasalahan yang muncul adalah apabila *string* asal/input mengalami pembaruan. Pembaruan disini melingkupi penghapusan, penambahan, dan perubahan sebuah karakter atau factor (*substring*). Terdapat 2 solusi untuk masalah ini, membuat *suffix array* yang bisa diperbarui (dinamis) atau membuat *suffix array* baru.

Membuat *suffix array* adalah solusi yang efisien dan tidak mahal. Algoritma pembentukan *suffix array* yang diusulkan Nong, Zhang, dan Chan berhasil mencapai hasil optimal baik dari segi memori dan waktu konstruksi, yaitu *O(n)* [3]. Disisi lain, *suffix array* dinamis (*dynamic suffix array*) yang diajukan Salson et al. [2] memiliki

kompleksitas pembentukan *suffix array* baru yang lebih buruk, *O(n lg n)*. Berdasarkan teori, terlihat bahwa membuat *suffix array* yang baru lebih cepat dan efisien.

Hal yang membuat diangkatnya topik ini sebagai tugas akhir adalah hasil eksperimen dari Salson et al. yang menyatakan bahwa *dynamic suffix array* mampu mengalahkan algoritma linear dalam beradaptasi dengan pembaruan *string* input. Hasil eksperimen terhadap berbagai kasus dan *constraint* menunjukkan, *dynamic suffix array* memiliki waktu konstruksi *suffix array* baru yang lebih cepat dibandingkan dengan algoritma linear usulan Maniscalco dan Puglisi apabila dilakukan penambahan karakter atau *factor* pada *string* input.

Sayangnya, hasil eksperimen ini hanya berkisar pada penambahan *substring*. Hasil untuk penghapusan dan perubahan tidak disertakan. Selain itu, algoritma yang dijadikan perbandingan bukan algoritma linear yang tercepat. Salah satu algoritma dengan kebutuhan memori dan waktu terbaik saat ini adalah algoritma yang ditulis oleh Nong, Zhang, dan Chan.

Tugas akhir ini akan melakukan pengujian ulang untuk membandingkan dua solusi tersebut. Penulis akan melakukan implementasi dan studi kebenaran serta kompleksitas untuk masing-masing algoritma. Pengujian akan membandingkan waktu konstruksi yang dibutuhkan apabila dilakukan pembaruan teks, baik itu penambahan, pengurangan, maupun perubahan karakter/*factor*. Memori maksimal yang dibutuhkan juga akan dihitung dan dibandingkan.

Proses implementasi akan menggunakan bahasa C++ dengan pustaka STL. *Compiler* yang digunakan adalah GCC 4.2.1. Opsi optimisasi kompilasi yang digunakan adalah '-O3 –formit-frame-pointer –DNDEBUG'. Seluruh proses implementasi dan pengujian dilakukan di komputer MacBook 5.1, dengan CPU 2 GHz Intel Core 2 Duo, memori 8 GB, dan sistem operasi 64 bit Mac OS X Snow Leopard 10.6.8.

Input yang digunakan berasal dari berbagai repositori seperti Manzini's Large Corpus dan Pizza&Chili Corpus. Input random juga digunakan dengan berbagai *constraint*. Dilakukan 10 kali percobaan untuk masing-masing *testfiles* dan diambil rata-ratanya. Pengambilan waktu menggunakan fungsi C *gettimeofday* sedangkan untuk perhitungan kebutuhan memori menggunakan *memusage*.

Diharapkan dengan tugas akhir ini ditemukan suatu *constraint* yang tepat untuk penggunaan dari masing-masing *suffix array*, kapan sebaiknya membuat *dynamic suffix*

array dan sebaliknya, kapan membuat *suffix array* yang baru di setiap pembaruan lebih efisien.

4. PENDAHULUAN

1. LATAR BELAKANG

Suffix array adalah *array* terurut yang berisi *suffix-suffix* dari sebuah *string*. Perkembangan membuat definisi *suffix array* menjadi *array* yang menunjuk pada posisi awal *suffix* terurut pada *string* [1,2]. Gambar-gambar berikut akan menjelaskan pembentukan *suffix array* untuk *string* CTCTGC$. Gambar 1 adalah *array* dari *suffix-suffix* yang belum terurut. *Array* ini kemudian diurutkan secara leksikografi, ditunjukkan oleh Gambar 2. Gambar 3 adalah *suffix array* yang terbentuk.

0	C	T	C	T	G	C	$
1	T	C	T	G	C	$	
2	C	T	G	C	$		
3	T	G	C	$			
4	G	C	$				
5	C	$					
6	$						

Gambar 1. *Array* dari *suffix* yang belum terurut

6	$						
5	C	$					
0	C	T	C	T	G	C	$
2	C	T	G	C	$		
4	G	C	$				
1	T	C	T	G	C	$	
3	T	G	C	$			

Gambar 2. *Suffix* yang telah terurut

0	6
1	5
2	0
3	2
4	4
5	1
6	3

Gambar 3. *Suffix array* yang terbentuk

Suffix array mulai diperhatikan sebagai solusi pengganti *suffix tree* pada awal era 90-an. Kelemahan dari *suffix tree*, yaitu pengaruh dari jumlah alphabet (|Σ|) pada kompleksitas, menjadi masalah apabila dihadapkan dengan problem yang memiliki Σ dalam jumlah besar. Manber dan Myers memberikan solusi untuk masalah ini dengan mengenalkan *suffix array* pada tahun 1991 [1].

Kekurangan dari *suffix array* ini adalah waktu konstruksi yang lebih lambat daripada *suffix tree*. *Suffix array* usulan Manber dan Myers memiliki kompleksitas *O(n lg n)*, sedangkan *suffix tree* mampu dibentuk secara linear (*O(n)*). Penelitian terus dilakukan untuk menutupi kelemahan ini dan mencapai waktu pembentukan yang linear.

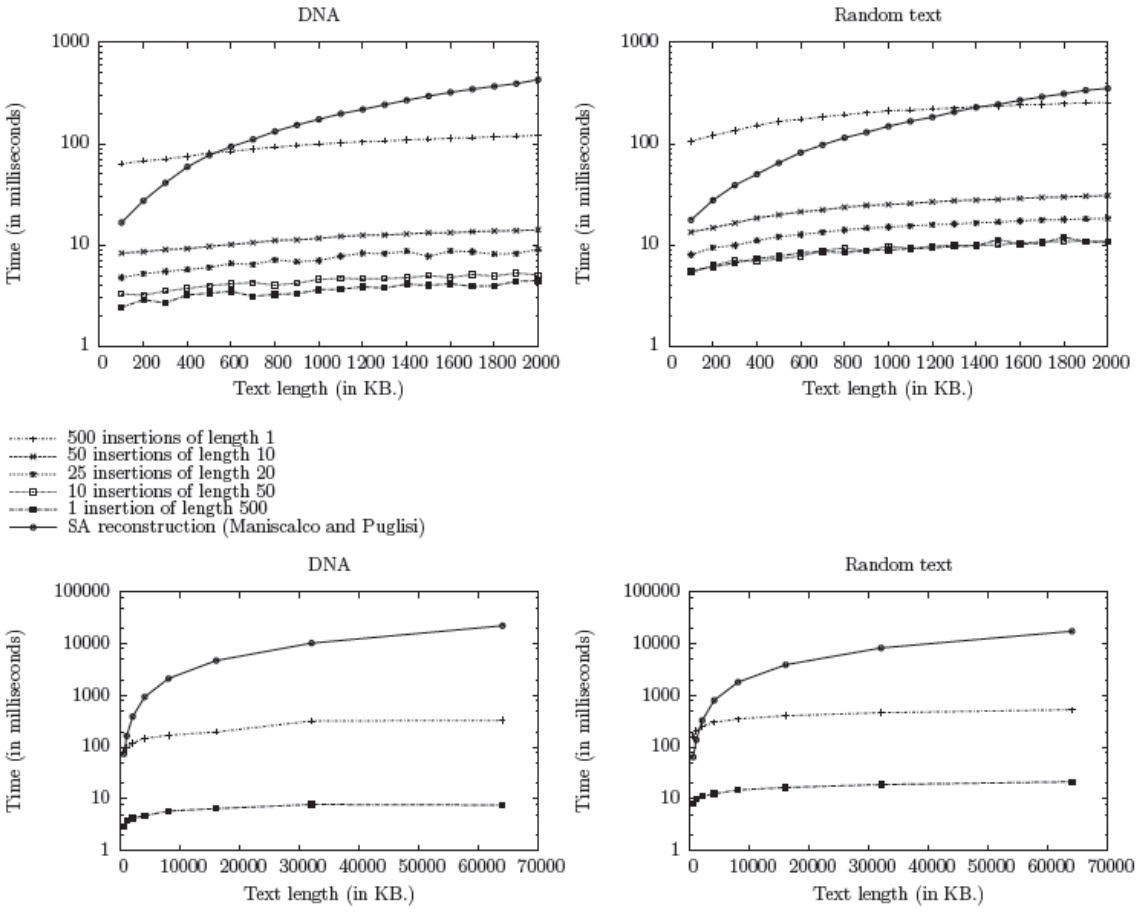
Pada tahun 2003, tiga algoritma pembentukan *suffix array* secara linear diajukan dalam waktu yang hampir bersamaan [4,5,6]. Perkembangan terus berlanjut hingga pada tahun 2009, Nong, Zhang, dan Chan merilis algoritma SA-IS [3]. Algoritma ini menggunakan *induced sort* yang digunakan Aluru dan Ko sebagai bagian algoritma usulan mereka, salah satu algoritma rilisan tahun 2003 [4]. Hasil yang yang sangat baik berhasil dicapai, *O(n)* secara waktu konstruksi dan memori.

Permasalahan selanjutnya adalah ketika *string* yang menjadi dasar dari *suffix array* (*string* input) mengalami pembaruan. Pembaruan ini melingkupi penambahan, penghapusan, dan perubahan karakter atau *factor* (*substring*). Salson et al. mengajukan *dynamic extended suffix array* untuk mengatasi problem ini pada tahun 2010 [2].

Paper Salson et al. pada tahun sebelumnya, mengenai *dynamic Burrows-Wheeler Transforms*, menjadi dasar dari *dynamic suffix array*. *Burrows-Wheeler Transforms*

memiliki bentuk yang hampir mirip dengan *suffix array* sehingga *dynamic suffix array* dapat diajukan.

Sayangnya, waktu konstruksi *suffix array* baru dari struktur data ini secara teoritis lebih buruk dari *O(n)*, yaitu *O(n lg n)*. Salson et al. menutupi hal ini dengan mengajukan hasil eksperimen. Berdasarkan grafik yang diberikan (Gambar 4), terlihat bahwa *dynamic suffix array* lebih cepat daripada *suffix array* linear dalam beradaptasi dengan pembaruan *string* input. Terlihat bahwa algoritma yang diajukan Maniscalco dan Puglisi kalah cepat apabila dilakukan penambahan *factor* pada *string* input.



Gambar 4. Hasil percobaan perbandingan dynamic suffix array dengan suffix array oleh Maniscalco dan Puglisi

Tetapi hasil percobaan ini belum bisa menunjukkan bahwa *dynamic suffix array* lebih baik dalam problem ini. Grafik tersebut hanya menyertakan kasus ketika *string* sebelumnya ditambahkan dengan *factor* dengan panjang tertentu. Salson et al. belum menyertakan performa *dynamic suffix array* untuk kasus perubahan dan penghapusan

factor. Algoritma linear yang dijadikan pembanding juga bukan algoritma yang tercepat saat ini, Maniscalo dan Puglisi mengajukan algoritma tersebut pada tahun 2006, sebelum algoritma SA-IS dirilis.

Tugas akhir ini akan melengkapi hasil percobaan tersebut, dengan menyertakan semua kasus pembaruan, dan algoritma linear tercepat sebagai pembanding. Sebelum melakukan perbandingan, studi algoritma dilakukan untuk kedua algoritma sebagai dasar untuk implementasi. Hasil implementasi ini yang akan menjadi acuan dari performa masing-masing algoritma.

2. RUMUSAN MASALAH

Masalah yang dibahas dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana menjelaskan dan membuktikan kebenaran algoritma pembentukan *suffix array* oleh Nong, Zhang, dan Chan?
2. Bagaimana menjelaskan dan membuktikan kapabilitas *dynamic suffix array* yang diajukan Salson et al. untuk kasus penambahan, penghapusan, dan perubahan karakter atau *factor* pada *string* input?
3. Bagaimana implementasi dari kedua *suffix array* tersebut?
4. Bagaimana performa dari masing-masing *suffix array*?

3. BATASAN MASALAH

Permasalahan yang dibahas dalam tugas akhir ini memiliki batasan – batasan sebagai berikut:

1. Implementasi kedua *suffix array* tersebut menggunakan bahasa C++ dengan bantuan IDE NetBeans 7.1.1.
2. Pustaka yang digunakan saat mengimplementasi algoritma menggunakan pustaka yang terdapat dalam *Standard Template Library (STL)*.
3. Kompilasi akan dilakukan oleh GCC/G++ 4.2.1. Opsi optimasi dalam proses kompilasi adalah ‘-O3 –formit-frame-pointer –DNDEBUG’.

4. Program akan dijalankan di atas computer dengan spesifikasi:
    a. MacBook 5.1
    b. CPU Intel Core 2 Duo 2.00 GHz
    c. Memori utama 8 GB
    d. Sistem operasi Mac OS X Snow Leopard 10.6.8
5. Performa yang diteliti meliputi *construction time* dari *suffix array* yang baru di tiap proses pembaruan dan memori yang digunakan.

4. TUJUAN TUGAS AKHIR

Tujuan dari pengerjaan tugas akhir ini adalah melakukan studi algoritma pada dua bentuk *suffix array*, statis dan dinamis. Studi ini melingkupi *proof of correctness*, perhitungan kompleksitas baik waktu konstruksi maupun memori yang dibutuhkan, serta implementasi. Keseluruhan kegiatan studi ini bertujuan untuk membandingkan performa dari dua *suffix array* tersebut.

5. MANFAAT TUGAS AKHIR

Manfaat yang diharapkan dari hasil tugas akhir ini adalah memberikan referensi dalam memilih algoritma *suffix array* yang sesuai dengan kebutuhan problem. *Suffix array* sangat bermanfaat dalam memproses teks dalam ukuran/panjang yang besar, salah satunya berada dalam lingkup bioinformatika. Selain itu, implementasi juga diberikan sebagai acuan dalam pembuatan aplikasi lain terkait problem tersebut.

5. TINJAUAN PUSTAKA

String adalah sebuah urutan karakter yang berasal dari himpunan alfabet (Σ). Berdasarkan definisi ini, himpunan semua *string* dengan panjang berhingga dapat dilambangkan dengan Σ^*. *Substring* atau *factor* adalah potongan urutan dari suatu *string.* Untuk *string* $S = s_1, s_2, \ldots, s_n$, *substring* dari *string* tersebut adalah $S' = s_{i+1}, s_{i+2}, \ldots, s_{i+m}$ dimana $i \geq 0$ dan $(m+i) \leq n$.

Suffix adalah *substring* yang berakhir dengan huruf terakhir. Sebagai contoh : ABRA$, KADABRA$, dan A$ adalah *suffix-suffix* untuk *string* ABRAKADABRA$. *Suffix array* dari suatu input *string* adalah sebuah *array* dari indeks-indeks yang disusun sedemikian

hingga indeks tersebut menunjuk pada *suffix-suffix* yang telah diurut leksikografi dari *string* input.

Algoritma SA-IS adalah algoritma pembentukan *suffix array* yang mampu berjalan secara linear [3]. Algoritma ini menitikberatkan pada *induced sort*, algoritma *sorting* untuk *suffix-suffix* yang memaksimalkan sifat bahwa suatu *suffix* adalah *suffix* dari *suffix* yang lain, misalnya *suffix* ABRA$ adalah *suffix* dari KADABRA$, keduanya adalah *suffix* dari ABRAKADABRA$.

Seperti algoritma SA-IS yang berdiri di atas algoritma lain, *dynamic suffix array* merupakan modifikasi dari struktur data *dynamic Burrows Wheeler Transform* [2,8]. *Burrows Wheeler Transform* (BWT) [7] adalah suatu algoritma kompresi, misalnya pada aplikasi kompresi bzip2. Seperti *suffix array*, BWT dibentuk dari *suffix-suffix* yang terurut. Tetapi, *suffix* ini juga menyertakan *substring* yang terbuang dan menaruhnya di depannya, sehingga berbentuk seperti *rotation of string*. Misalnya, untuk *suffix* ABRA$ dari *string* ABRAKADABRA, BWT menggunakan ABRA$ABRAKAD. Selanjutnya, semua *rotation of string* ini akan diurutkan – seperti *suffix array* yang membutuhkan urutan leksikografi dari semua *suffix* yang ada – dan diambil deretan huruf terakhirnya.

Kemiripan bentuk dari BWT dan *suffix array* memungkinkan modifikasi dari *dynamic* BWT menjadi *dynamic suffix array*. Salson et al. yang mengusulkan kedua struktur data tersebut menyatakan bahwa kelebihan dari bentuk dinamis ini adalah performa yang lebih dalam pemakaian nyata (*in practice*). Tugas akhir ini akan membuktikan hal tersebut.

6. METODOLOGI

Ada beberapa tahap dalam proses pengerjaan tugas akhir ini, yaitu sebagai berikut:

1. Studi Literatur

Studi literatur mencakup pembelajaran pada algoritma-algoritma terkait, analisa *proof of correctness*, serta perhitungan kompleksitas. Pada tahap ini juga dilakukan studi terhadap beberapa implementasi terdahulu apabila tersedia untuk dijadikan acuan pada tahap selanjutnya.

2. Implementasi dan Uji Coba

Hasil studi pada tahap sebelumnya menjadi dasar pada tahap implementasi. Bahasa yang digunakan adalah C++ dengan bantuan IDE NetBeans 7.1.1. Diharapkan

pada tahap ini didapatkan bentuk implementasi yang cukup optimal untuk kedua *suffix array* agar didapatkan hasil yang relevan pada tahap eksperimen.

Implementasi *suffix array* linear mengacu pada algoritma yang diajukan Nong, Zhang, dan Chan [3]. Gambar 5 memberikan *pseudocode* dasar untuk algoritma tersebut. Detail mengenai *pseudocode* ini akan dijelaskan di tugas akhir.

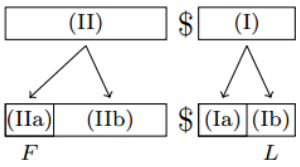
```
SA-IS (S, SA)
  ➢ S is the input string;
  ➢ SA is the output suffix array of S;
  t    : array [0 ... n−1] of boolean;
  S1   : array [0 ... n1−1] of integer;
  P1   : array [0 ... n1−1] of integer;
  B    : array [0 ... || Σ(S) || −1] of integer;

  1. Scan S once to classify all the characters as L- or S-type into t;
  2. Scan t once to find all the LMS-substrings in S into P1;
  3. Induced sort all the LMS-substrings using P1 and B;
  4. Name each LMS-substring in S by its bucket index to get a new shortened string S1;
  5. if Each character in S1 is unique
  6.       then
  7.             Directly compute SA1 from S1;
  8.       else
  9. SA-IS(S1,SA1); //where recursive call happens
  10. Induce SA from SA1;
  11. return
```

Gambar 5. *Pseudocode* dasar untuk algoritma SA-IS

Sedangkan struktur data *dynamic suffix array* didasarkan pada *paper* rujukan Salson et al. Struktur data ini akan memiliki 3 fungsi untuk mengatasi tiap-tiap bentuk pembaruan.

Setiap proses pembaruan *suffix array*, terdapat 4 kondisi kemungkinan yang terjadi. Ke-4 kondisi ini berasal dari 4 kemungkinan penempatan (atau perubahan/penghapusan) sebuah karakter (atau *factor*). Gambar 6 menjelaskan kemungkinan-kemungkinan tersebut [2].

$$T'[j] = \begin{cases} T[j-1\,..\,n-1]\ \$\ T[0\,..\,i-1]\ c\ T[i\,..\,j-2] & \text{if } i+1 < j \le n+1 \quad \text{(Ia)} \\ T[i\,..\,n-1]\ \$\ T[0\,..\,i-1]\ c & \text{if } j = i+1 \quad \text{(Ib)} \\ c\ T[i\,..\,n-1]\ \$\ T[0\,..\,i-1] & \text{if } j = i \quad \text{(IIa)} \\ T[j\,..\,i-1]c\ T[i\,..\,n-1]\ \$\ T[0\,..\,j-1] & \text{if } 0 \le j < i \quad \text{(IIb)} \end{cases}$$



c appears: (I) right to \$, (II) left to \$.
More precisely:
c appears: (Ia) between \$ and L, (Ib) in L.
c appears: (IIa) in F, (IIb) between F and \$.

Gambar 6. 4 kemungkinan penambahan karakter

Dari 4 kemungkinan tersebut, Salson et al. mengajukan algoritma rekonstruksi *suffix array* dengan 4 tahap (*four-stage algorithm*) [2,8]. 4 tahap itu adalah:

Ia	*Ignore*	Tidak ada dampak khusus pada F dan L.
Ib	*Modification*	Untuk baris ke-i pada *suffix array*, huruf pada L disimpan dan diganti dengan c.
IIa	*Insertion*	Baris baru ditambahkan pada posisi LF(SA[i]). L akan menerima huruf yang disimpan pada tahap sebelumnya (Ib) dan F akan menerima c.
IIb	*Reodering*	Huruf-huruf yang lain akan diurutkan sebagaimana semestinya (urutan *suffix* secara leksikografi).

F, L, dan c, seperti yang tertulis pada Gambar 6, adalah variabel-variabel yang terlibat dalam algoritma ini. F adalah *suffix array*, L adalah BWT, c adalah karakter yang ditambahkan. Sedangkan LF adalah *array* yang menghubungkan F dan L.

Implementasi program untuk kebutuhan ujicoba tentu saja berbeda untuk masing-masing algoritma. SA-IS membutuhkan pembuatan *string* input baru yang sesuai dengan pembaruan, lalu kemudian merekonstruksi ulang *suffix array* berdasarkan *string* input yang baru. *Dynamic suffix array* tidak membutuhkan *string* input yang baru, dia akan berubah sendiri sejalan dengan pembaruan *string* input.

3. Eksperimen dan Evaluasi

Pada tahap ini kedua hasil implementasi akan diujikan pada beberapa *testfile* untuk didapatkan performa dari masing-masing *suffix array*. Tes uji mencakup beberapa *corpus* yang tersedia di internet serta kasus-kasus random.

Eksperimen dilakukan dalam 2 tahap. Pertama adalah pengujian bahwa implementasi algoritma pembentukan *suffix array* mampu menghasilkan output (*suffix array*) yang sesuai. Dalam tahap ini, waktu konstruksi dan memori tidak dihitung.

Tahap selanjutnya adalah pengujian performa yang melibatkan pembaruan *string* input. Setiap dilakukan pembaruan, waktu yang dibutuhkan untuk merekonstruksi *suffix array* dicatat dengan *fungsi gettimeofday*. Pengujian dilakukan 4 kali, yaitu untuk kasus penambahan, penghapusan, dan perubahan *factor*. Untuk pengujian terakhir, ketiga kasus pembaruan ikut dilibatkan. Pencarian *constraint* yang tepat dilakukan di setiap kasus untuk menentukan algoritma yang sesuai dengan problem tertentu.

Input yang digunakan berasal dari *corpus-corpus* yang menyediakan teks dalam ukuran besar, seperti Manzini's Large Corpus dan Pizza&Chili Corpus.

7. JADWAL PEMBUATAN TUGAS AKHIR

Tugas akhir ini diharapkan bisa dikerjakan sesuai jadwal, sebagai berikut.

Tahapan	2013																			
	Maret				April				Mei				Juni				Juli			
Penyusunan Proposal																				
Studi Literatur																				
Implementasi dan Uji Coba																				
Eksperimen dan Evaluasi																				

8. DAFTAR ACUAN

[1] Manber, Udi; Myers, Gene (1990). Suffix Arrays: a New Method for On-line String Searches. In Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms 90 (319): 327.

[2] Salson, M.; Lecroq, T.; Léonard, M.; Mouchard, L. (2010). Dynamic Extended Suffix Arrays. Journal of Discrete Algorithms 8 (2): 241.

[3] Nong, Ge; Zhang, Sen; Chan, Wai Hong (2009). Linear Suffix Array Construction by Almost Pure Induced-Sorting. 2009 Data Compression Conference. pp. 193.

[4] P. Ko, S. Aluru, (2005). Space Efficient Linear Time Construction of Suffix Arrays, J. Discrete Algorithms 3 (2-4) 143-156

[5] J. Kärkkäinnen P. Sanders (2006). Simple Linear Work Suffix Array Construction, J. ACM 53 (6) 918-936

[6] D. K. Kim, J. S. Sim, H. Park, K. Park (2005). Constructing Suffix Arrays in Linear Time, J. Discrete Algorithms 3 (2-4) 143-156

[7] M. Burrows, D. J. Wheeler (1994). A Block-sorting Lossless Data Compression Algorithm., Tech. Rep. 124, DEC, Palo Alto, California

[8] Salson, M.; Lecroq, T.; Léonard, M.; Mouchard, L. (2009). A Four-Stage Algorithm for Updating a Burrows-Wheeler Transform. Theoretical Computer Science