PEREDUKSIAN *READS* SEBAGAI PRAPROSES PADA *DNA SEQUENCE ASSEMBLY* DENGAN *PREFIX TREE*

GARY YUTHIAN

DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR
BOGOR
2014

PERNYATAAN MENGENAI SKRIPSI DAN SUMBER INFORMASI SERTA PELIMPAHAN HAK CIPTA

Dengan ini saya menyatakan bahwa skripsi berjudul Pereduksian *Reads* sebagai Praproses pada *DNA Sequence Assembly* dengan *Prefix Tree* adalah benar karya saya dengan arahan dari komisi pembimbing dan belum diajukan dalam bentuk apa pun kepada perguruan tinggi mana pun. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan maupun tidak diterbitkan dari penulis lain telah disebutkan dalam teks dan dicantumkan dalam Daftar Pustaka di bagian akhir skripsi ini.

Dengan ini saya melimpahkan hak cipta dari karya tulis saya kepada Institut Pertanian Bogor.

Bogor, Juni 2014

Gary Yuthian
NIM G64090129

ABSTRAK

GARY YUTHIAN. Pereduksian *Reads* sebagai Praproses pada *DNA Sequence Assembly* dengan *Prefix Tree*. Dibimbing oleh WISNU ANANTA KUSUMA.

Teknologi DNA *sequence assembly* memiliki peran penting dalam bidang ilmu pengetahuan, khususnya genomika. Teknologi ini berfungsi untuk membentuk fragmen DNA yang lebih panjang (*contigs)* dari penggabungan fragmen-fragmen pendek (*reads*). Redudansi bisa terjadi pada *reads* yang dihasilkan oleh DNA *Sequencer*. Redudansi pada *reads* dapat menyebabkan waktu komputasi yang lama pada proses *assembly* DNA. Oleh karena itu, pereduksian *reads* menjadi solusi untuk mengatasi masalah tersebut. Penelitian ini berhasil membuat sebuah perangkat lunak yang mampu mereduksi *redundant reads* pada data sekuens DNA dengan struktur data *prefix tree. Data set* DNA yang mengandung *redundant reads* dibangkitkan menggunakan perangkat lunak MetaSim. Evaluasi dilakukan dengan membandingkan hasil pereduksian dari perangkat lunak yang dibuat dengan hasil pereduksian dengan Edena. Edena adalah perangkat lunak DNA *Sequence Assembly* yang sudah dipublikasikan dan memiliki modul pereduksi *redundant reads*. Hasil evaluasi menunjukkan bahwa perangkat lunak yang dibuat mampu mereduksi *redundant reads*, ditunjukan dengan berkurangnya jumlah *reads* serta kesamaan jumlah *reads* hasil reduksi dengan hasil reduksi yang dilakukan menggunakan Edena.

Kata kunci: DNA *sequence assembly,* fragmen DNA, pereduksian *reads*

ABSTRACT

GARY YUTHIAN. Reduction of Reads as preprocess on DNA Sequence Assembly using *Prefix Tree*. Supervised by WISNU ANANTA KUSUMA.

DNA sequence assembly technology has an important role in the field of science, especially genomics. The technology serves to assembly of short fragments (reads) to yield longer DNA fragments (contigs). Redundancies can occur in reads produced by DNA Sequencer. Redudancies of reads may cause long computation time on the assembly process of DNA. Therefore, reduction of reads can be a solution to solve the problem. This study successfully made a software to reduce redundant reads in DNA sequence data with a prefix tree data structure. Data sets DNA containing redundant reads was yielded using MetaSim software. Evaluation was performed by comparing the results of reads reduction using our software with the result of reads reduction using the Edena. Edena is DNA Sequence Assembly software that has been published and has module of reducing redundant reads. Evaluation results show that the the number of reads after being reduced by this software are the same as those of being reduced by the Edena.

Keywords: DNA *sequence assembly,* DNA fragments, reads reduction

PEREDUKSIAN *READS* SEBAGAI PRAPROSES PADA *DNA SEQUENCE ASSEMBLY* DENGAN *PREFIX TREE*

GARY YUTHIAN

Skripsi
sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer
pada
Departemen Ilmu Komputer

DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR
BOGOR
2014

Penguji : Toto Haryanto SKom, MSi

Karlina Khiyarin Nisa SKom, MT

Judul Skripsi: Pereduksian *Reads* sebagai Praproses pada *DNA Sequence Assembly* dengan *Prefix Tree*
Nama : Gary Yuthian
NIM : G64090129

Disetujui oleh

Dr Wisnu Ananta Kusuma, MT
Pembimbing

Diketahui oleh

Dr Ir Agus Buono, MSi MKom
Ketua Departemen

Tanggal Lulus:

PRAKATA

Puji dan syukur penulis panjatkan kepada Allah *subhanahu wa ta'ala* atas segala berkat rahmat, karunia dan ridho-Nya sehingga karya ilmiah ini berhasil diselesaikan. Penelitian ini difokuskan pada bidang Bioinformatika khususnya mengenai pereduksian *reads* pada DNA *sequence assembly*. Semakin berkembang dan majunya metode serta teknologi dalam dunia Bioinformatika membuat penulis termotivasi sehingga memilih Bioinformatika sebagai topik penelitian.

Terima kasih penulis ucapkan kepada Dr Wisnu Ananta Kusuma selaku pembimbing yang selalu membimbing, mengawasi dan mengingatkan penulis pada proses penyelesaian penelitian ini. Tidak lupa pula penulis menyampaikan terima kasih kepada ayah, ibu dan kedua adik yang selalu mendukung, menyemangati dan mendoakan selama pengerjaan penelitian ini. Ungkapan terima kasih juga disampaikan kepada rekan-rekan satu angkatan Ilmu Komputer angkatan 46 terutama kepada Aziz, Novaldo, Aditrian, Albert yang secara langsung dan tidak langsung membantu penulis pada penelitian ini.

Semoga penelitian ini berguna bagi perkembangan ilmu pengetahuan secara umum dan ilmu komputer pada khususnya di masa depan, serta dapat dijadikan panduan untuk perkembangan penelitian dengan tema sejenis di masa mendatang.

Bogor, Juni 2014

Gary Yuthian

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

PENDAHULUAN

Latar Belakang

Teknologi DNA *sequencing* dan DNA *sequence assembly* memiliki peranan penting dalam beberapa bidang ilmu pengetahuan, khususnya pada cabang biologi, yaitu genomika. Kemajuan paling berpengaruh pada teknologi DNA *sequencing* ialah terciptanya teknologi DNA *sequencing* generasi kedua. Teknologi tersebut dapat menghasilkan *reads* dengan jumlah jauh lebih banyak dibanding teknologi sebelumnya. Namun, *reads* yang dihasilkan jauh lebih pendek. *Reads* merupakan serangkaian fragmen pendek dari molekul DNA yang direpresentasikan ke dalam mesin (Trapnell dan Salzberg 2009).

DNA *sequence assembly* diciptakan untuk menggabungkan *reads* yang dihasilkan oleh mesin *Sequencer* generasi kedua. DNA *sequence assembl*y dapat mengggunakan 2 metode, yaitu dengan memetakan fragmen *DNA* dengan sekuens referensi atau menggunakan teknik *de novo* (Kusuma e*t al.* 2011).

Teknik *de novo* dapat dilakukan dengan dua pendekatan, yaitu *Overlap Layout Consensus* (OLC) dan *Eulerian path* menggunakan graf de Bruijn (Pevzner *et al.* 2001). Tiap pendekatan memiliki banyak aplikasi sebagai implementasi masing-masing untuk melakukan *DNA sequence assembly*, antara lain SHARCGS (Dohm *et al.* 2007), Velvet (Zerbino dan Birney 2008), SSAKE (Warren *et al*. 2007), dan Edena (Hernandez *et al.* 2008).

Edena merupakan suatu program yang berfungsi untuk melakukan *DNA assembly* dengan menggunakan metode *Overlap Layout Consensus* (OLC). Metode ini menggunakan overlap dari potongan-potongan *reads* untuk dirangkaikan satu sama lain. Edena memiliki 4 tahap dalam prosesnya. Tahap pertama sebagai praproses, yaitu menghilangkan redundansi. Tahap kedua adalah membuat suatu *overlap graph.* Tahap ketiga adalah proses pembersihan graf yang dilakukan dengan menghilangkan *transitive edges* dan memotong *node* dari graf yang kedalamannya tidak memenuhi syarat tertentu (Hernandez D *et al.* 2008).

Penelitian ini akan difokuskan pada tahap pertama dalam Edena, yaitu menghilangkan redundansi pada *reads* menggunakan implementasi dari *prefix tree*. Dengan sistem pereduksian *reads* yang akurat, tahap selanjutnya, pembuatan *overlap graph* dapat dilakukan lebih mudah.

Perumusan Masalah

Perumusan masalah pada penelitian ini adalah:

1 Bagaimana cara untuk mengimplementasikan *prefix tree* untuk menghilangkan *redundant* dan *reverse complement reads*?

2 Bagaimana cara menghilangkan *redundant* dan *reverse complement reads* pada *data set*?

3 Bagaimana hubungan keterkaitan antara banyak *reads* dengan waktu eksekusi?

Tujuan Penelitian

Tujuan penelitian ini adalah untuk mengembangkan perangkat lunak yang bertujuan menghilangkan redundansi dan *reverse complement* pada r*eads* dengan menggunakan *prefix tree* sehingga *reads* yang diproses selanjutnya hanya memiliki informasi penting dan ukuran data lebih mudah dikelola.

Manfaat Penelitian

Manfaat dari penelitian ini adalah mempermudah proses menggabungkan *reads* serta mengurangi lama waktu komputasi pada DNA *assembly*.

Ruang Lingkup Penelitian

Ruang lingkup pada penelitian ini adalah sebagai berikut:

1 Data yang direduksi adalah *redundant reads* beserta *reverse complement*nya.
2 *Data set* yang digunakan adalah sekumpulan *reads* yang memiliki panjang sama.
3 *Data set* yang digunakan pada penelitian ini adalah hasil simulasi menggunakan perangkat lunak MetaSim Versi 0.91. *Data set* disimulasikan dengan menggunakan model *error free* (Richter *et al.* 2008).

METODE

Metode yang digunakan pada penelitian ini terdiri atas 3 tahap, yakni menyiapkan *data set*, memproses data set menggunakan *prefix tree* dan evaluasi waktu eksekusi.

Menyiapkan *Data Set*

Data set yang akan digunakan untuk proses menghilangkan redundansi yaitu berupa sekumpulan *reads* dengan panjang yang sama hasil simulasi proses DNA *sequencing* menggunakan perangkat lunak MetaSim. Oleh karena itu, perlu ditetapkan organisme yang akan digunakan untuk proses simulasi tersebut. Pada penelitian ini dipilih 3 organisme yang didapat dari basis data MetaSim. Ketiga organisme beserta informasinya dapat dilihat pada Tabel 1.

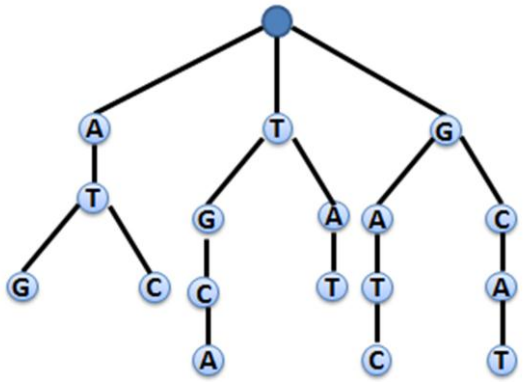
Tabel 1 Spesifikasi organisme yang digunakan pada penelitian

Organisme	Panjang sekuens lengkap (bp)
Staphylococcus aureus subsp. aureus ED98 plasmid pAVY	1 442
Acetobacter pasteurianus IFO 3283-01 plasmid pAPA01-060	1 815
Lactobacillus plantarum WCFS1 plasmid pWCFS101	1 917

Memproses *Data Set* Menggunakan *Prefix Tree*

Setelah *data set* hasil simulasi didapat, data tersebut kemudian diproses dengan *prefix tree. Prefix tree* merupakan struktur data yang menyimpan himpunan *string* dalam bentuk meyerupai pohon dan tiap *node* mengandung awalan (*prefix)* yang sama. Masing-masing *node* dalam *prefix tree* merepresentasikan satu karakter, sedangkan *root* merepresentasikan *string* kosong (*null string*). Setiap keturunan dari sebuah *node* mempunyai prefix yang sama dengan *string* yang diwakilkan oleh *node* tersebut. Pada penelitian ini, cabang *root* dan *node* pada *prefix tree* dibatasi maksimal 4. Hal ini dikarenakan *reads* tersusun dari empat unsur pokok yang disebut nukleutida, yaitu *Adenine* (A), *Guanine* (G), *Cytosine* (C),dan *Thymine* (T) (Maftuhah 2007).

Sebagai contoh, misalkan terdapat himpunan *string* dari basa nukleotida “ATG”, “ATC”, “TGCA”, “TAT”, “GATC”, “GCAT”. *Prefix tree* yang dibentuk dapat dilihat pada Gambar 1.

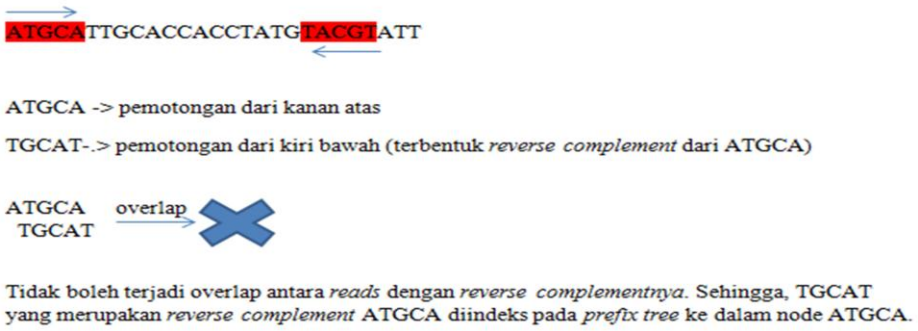


Gambar 1 Contoh prefix tree untuk himpunan *string*

Proses pada tahap ini bertujuan untuk menghilangkan *reads* yang muncul berulang (*redundant)* dan *reverse complement reads* dari *data set. Redundant reads* perlu dihilangkan untuk memudahkan fase selanjutnya pada DNA *assembly*, yaitu dalam proses pembentukan *overlap graph* agar

graf yang terbentuk tidak kompleks. *Reverse complement* dari suatu *reads* adalah urutan kebalikan dari *reads* dan kemudian dikomplemenkan nukleotidanya, *Adenine* (A) menjadi *Thymine* (T) (berlaku sebaliknya) dan *Guanine* (G) menjadi *Cytosine* (C) (berlaku sebaliknya). Sebagai contoh, misalkan terdapat *reads* "ATGGGCC" maka *reverse complement* dari *reads* tersebut "GGCCCAT".

Reverse complement reads bisa terbentuk akibat proses simulasi DNA *sequencing* oleh MetaSim. Pada saat proses sequencing, pemotongan *reads* dilakukan dari dua arah, atas dan bawah. Dari proses pemotongan dua arah tersebut dapat terbetuk *reverse complement reads.* Akan tetapi, *reads* asli tidak boleh *overlap* dengan *reverse complement reads.* Untuk mengatasi hal itu, maka *reverse complement reads* dihilangkan dengan cara diindeks pada *prefix tree* ke dalam *node reads* asli. Ilustrasi dari eliminasi *reverse compelement reads* dapat dilihat pada Gambar 2.



Gambar 2 Ilustrasi eliminasi *reverse complement reads*

Evaluasi

Untuk membuktikan kebenaran perangkat lunak ini, dilakukan evaluasi dengan membandingkan hasil pereduksian dari perangkat lunak yang dibuat dengan hasil pereduksian menggunakan Edena. Apabila jumlah *reads* hasil pereduksian sama, maka perangkat lunak yang dikembangkan terbukti kebenarannya. Selain itu, juga dilakukan evaluasi waktu eksekusi antar tiap organisme, evaluasi *average redundant reads,* evaluasi panjang *reads* tiap organisme dan menghitung kompleksitas algoritma perangkat lunak.

Spesifikasi Perangkat Lunak dan Perangkat Keras

Kebutuhan perangkat lunak yang digunakan untuk pengembangan perangkat lunak pada penelitian ini adalah sebagai berikut:

- Bahasa pemrograman : C++
- Sistem operasi : Microsoft Windows 7 Ultimate 64 bit
- *Integrated development environment* : Dev-C++ 5.4.2
- *Text Editor* : Sublime text 2

HASIL DAN PEMBAHASAN

Menyiapkan *Data Set*

Pada metode di atas telah disebutkan penelitian ini menggunakan tiga organisme yang dipilih dari basis data MetaSim. Alasan memilih tiga organisme tersebut karena panjang *sequence* organisme tersebut relatif pendek dibanding organisme lain yang ada pada basis data. Waktu eksekusi yang dihasilkan tidak terlalu lama.

Dari setiap organisme tersebut dilakukan simulasi masing-masing dua kali dengan konfigurasi panjang tiap *reads* yaitu 35 *base pairs* (bp) dan jumlah *reads* berbeda, yaitu 2000 *reads* dan 5000 *reads*. Hasil dari simulasi tersebut berupa sebuah fail berformat FASTA (.fna). Dari tiga organisme tersebut maka akan dihasilkan enam fail FASTA yang nantinya digunakan sebagai data masukan pada penelitian ini.

Pada simulasi ini digunakan model *exact (error free)* sehingga fail hasil simulasi yang didapat tidak memiliki *error*. *Error* yang dimaksud adalah tidak ada nukleotida pada *reads* yang disubtitusi dengan nukleotida lainnya, namun tetap memiliki *redundant reads.* Informasi parameter yang digunakan untuk simulasi dapat dilihat pada Tabel 2.

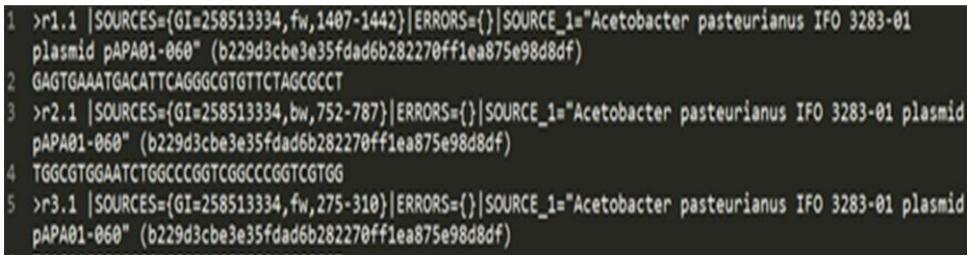
Tabel 2 Paramater data DNA sequence menggunakan MetaSim

No.	Nama Organisme	*Number of Read / Mate Pairs*	*Error Model*	*DNA Clone Size Distribution Type*	*Mean* (bp)	*Second Parameter*
1	*Staphylococcus aureus plasmid*	2000	*Exact*	Normal	35	0.0
2	*Staphylococcus aureus plasmid*	5000	*Exact*	Normal	35	0.0
3	*Acetobacter pasteurianus plasmid*	2000	*Exact*	Normal	35	0.0
4	*Acetobacter pasteurianus plasmid*	5000	*Exact*	Normal	35	0.0
5	*Lactobacillus plantarum plasmid*	2000	*Exact*	Normal	35	0.0
6	*Lactobacillus plantarum plasmid*	5000	*Exact*	Normal	35	0.0

Penjelasan setiap parameter untuk simulasi adalah sebagai berikut:

- ***Number of Read*s** merupakan jumlah *reads* yang dihasilkan dari simulasi menggunakan MetaSim. Pada simulasi ini jumlah *reads* yang dihasilkan adalah 2000 dan 5000.
- ***Error Model*** merupakan jenis error yang digunakan pada simulasi menggunakan MetaSim. Pada simulasi ini menggunakan Model *Exact* sehingga hasil simulasi tidak memiliki *error (error-free).*
- ***DNA Clone Size Distribution Type*** merupakan cara pendistibusian *reads* hasil simulasi. Pada simulasi ini digunakan tipe Normal sehingga *reads* hasil simulasi tersebar urutannya.
- ***Mean* (bp)** merupakan panjang untuk satu *reads* dari hasil simulasi menggunakan MetaSim. Pada simulasi ini mean yang digunakan adalah 35.
- ***Second Parameter*** merupakan standar deviasi dari panjang *reads.* Pada simulasi ini *second paramater* yang digunakan adalah 0.0.

Contoh hasil simulasi menggunakan MetaSim dapat dilihat pada Gambar 3.

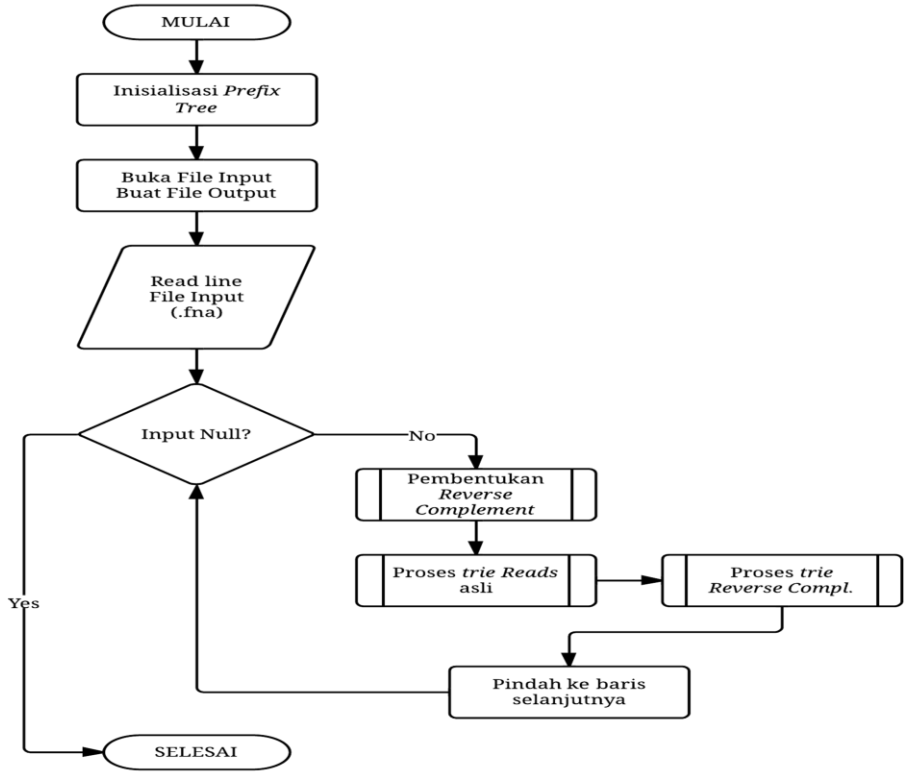


Gambar 3 Acetobacter pasteurianus dalam fail FASTA

Dalam fail FASTA tanda lebih besar “>” bermakna bahwa sebuah *reads* mulai terbaca dan akhir dari urutan *reads* tersebut akan ditandai lagi dengan tanda “>” pada baris yang lainnya (Rahim 2013).

Alur Kerja Perangkat Lunak dalam Mereduksi *Reads*

Pada penelitian ini tercipta sebuah perangkat lunak yang dapat mereduksi *reads* bermasalah dengan menggunakan *prefix tree.* Proses untuk pereduksian ini terdiri atas beberapa tahap, yaitu inisiasi *prefix tree*, membaca *reads* pada fail masukan (.fna) satu per satu, pembentukan *reverse complement reads* satu per satu, pengindeksan *reverse complement reads* dan *reads* asli ke dalam *prefix tree* dan menyimpan *reads* yang telah diproses berupa *reads-reads* tunggal dalam fail keluaran (.fna) sebagai hasil keluaran. Diagram alur tahapan kerja perangkat lunak ini dapat dilihat pada Gambar 4.

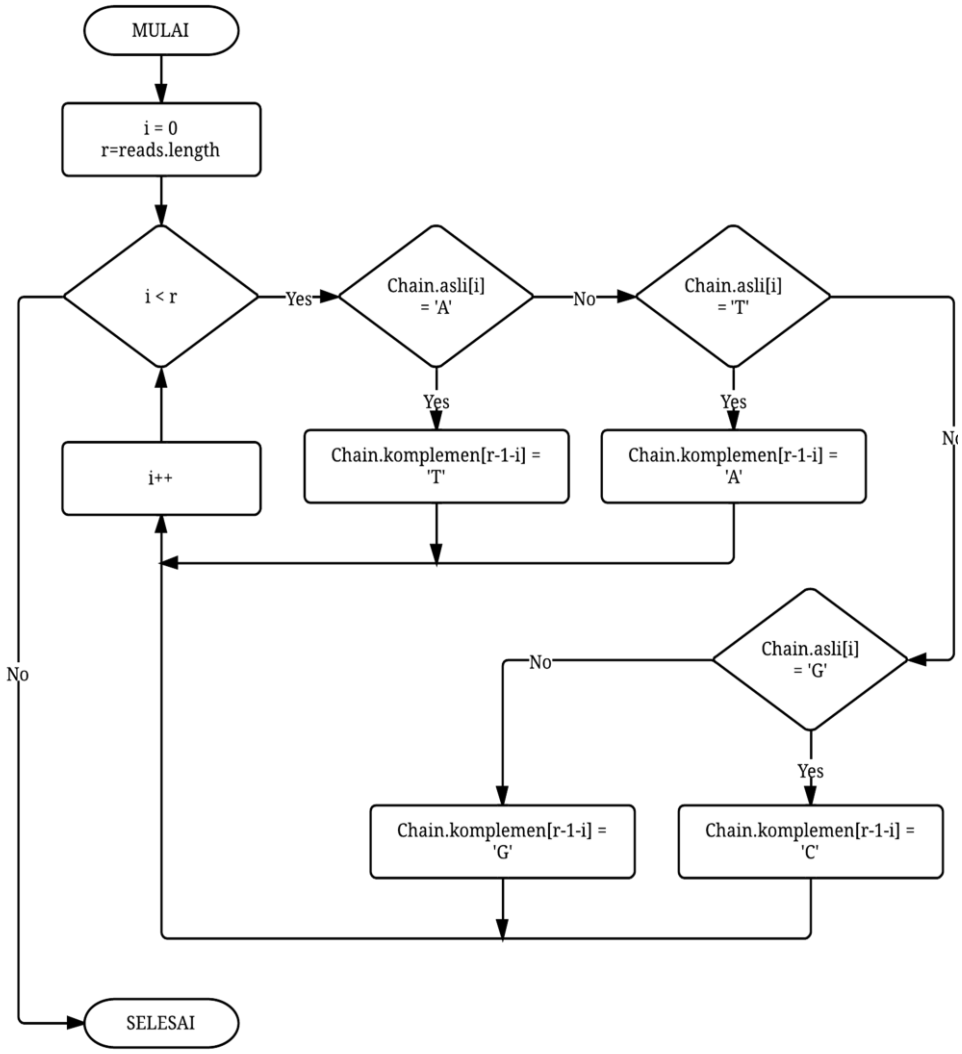


Gambar 4 Diagram alur tahapan kerja perangkat lunak

Saat perangkat lunak dijalankan, hal pertama yang dilakukan sistem adalah inisiasi *prefix tree.* Inisiasi dilakukan untuk membangkitkan *tree root* dan mendeklarasikan *pointer* dari *root* yang nantinya digunakan untuk menunjuk ke *node* selanjutnya. Hal ini dilakukan karena belum adanya *prefix tree* sehingga perlu dilakukan inisiasi terlebih dahulu.

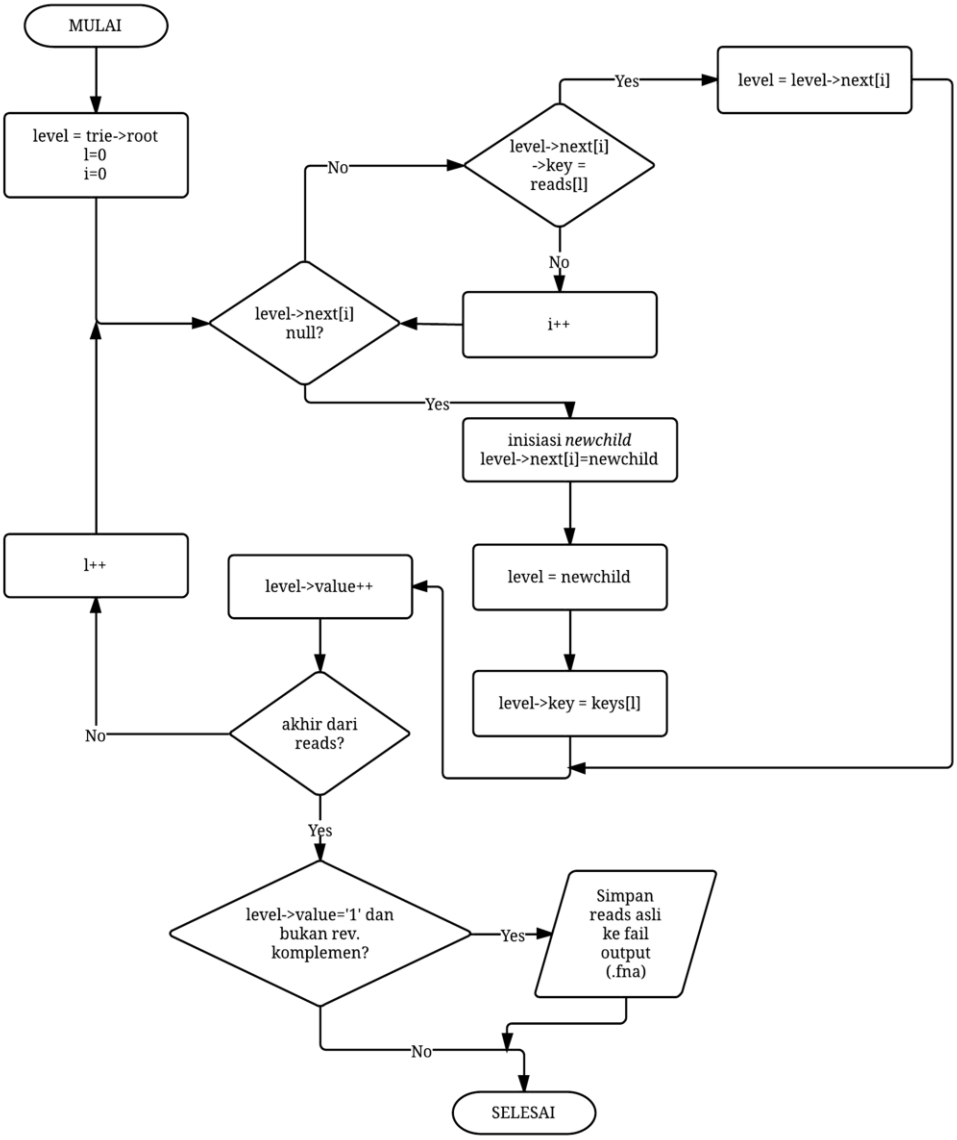
Setelah inisiasi, sistem akan membuka fail data masukan dan juga membuat fail untuk data keluaran. Kemudian, sistem akan membaca data masukan yang merupakan potongan-potongan *reads* dengan panjang 35bp dengan jumlah 2000 dan 5000 buah seperti yang telah ditentukan. Jika, data masukan ternyata tidak terdapat potongan *reads*, program selesai bekerja. Pembacaan fail data masukan dilakukan secara satu per satu. Perangkat lunak memproses *reads* pertama sampai semua proses selesai baru kemudian membaca *reads* selanjutnya.

Selanjutnya, sebelum *reads* diproses dengan *prefix tree*, sistem membentuk *reverse complement* dari *reads* yang dibaca. Hasil *Reverse complement* kemudian diindeks ke dalam *prefix tree* dengan memberikan argumen bernilai 1 yang menandakan bahwa masukan adalah sebuah *reverse complement* sehingga tidak akan disimpan dalam fail keluaran. Fungsi *reverse complement* hanya untuk membandingkan *reads* selanjutnya yang bernilai sama dengan *reverse complement* tersebut lalu dieliminasi dari *data set*. Diagram alur proses pembentukan *reverse complement* dapat dilihat pada Gambar 5.



Gambar 5 Diagram alur proses *reverse complement*

Kemudian, *reads* asli juga diindeks ke dalam *prefix tree*. Pada saat pengindeksan *reads* asli, apabila belum terdapat rantai *reads* yang sama pada *prefix tree* maka *reads* yang sedang diproses langsung disimpan dalam fail keluaran (.fna). Apabila sudah terdapat rantai *reads* yang sama pada tree maka *reads* tersebut hanya diindeks namun tidak ikut disimpan dalam fail keluaran dan langsung memproses *reads* selanjutnya sampai tidak terdapat *reads* pada *data set*. Diagram alur proses pengindeksan pada *prefix tree* dapat dilihat pada Gambar 6.



Gambar 6 Diagram alur proses pengindeksan *reads*

Data keluaran dari perangkat lunak ini adalah waktu eksekusi, jumlah *reads* yang dibaca, jumlah *reads* yang tunggal, *average redundant reads*, serta fail berformat .fna yang berisi *reads* yang tidak berulang serta tidak terdapat *reverse complement.* Hasil eksekusi perangkat lunak yang dihasilkan dari tiga organisme yang telah dipilih sebagai data masukan dapat dilihat pada Tabel 3.

Tabel 3 Hasil eksekusi perangkat lunak yang dihasilkan

Nama Organisme	Jumlah *reads* dibaca	Jumlah *reads* akhir	Waktu Eksekusi (s)	*Average Redundant Reads*
Staphylococcus aureus plasmid	2000	1086	0.057	1.8
Staphylococcus aureus plasmid	5000	1390	0.105	3.6
Acetobacter pasteurianus plasmid	2000	1223	0.061	1.6
Acetobacter pasteurianus plasmid	5000	1672	0.101	3.0
Lactobacillus plantarum plasmid	2000	1267	0.060	1.6
Lactobacillus plantarum plasmid	5000	1747	0.104	2.9

Evaluasi Hasil Reduksi dengan Edena

Setelah perangkat lunak berhasil dibuat, selanjutnya dilakukan pengujian kebenaran perangkat lunak ini. Pengujian dilakukan dengan cara membandingkan hasil reduksi perangkat lunak yang dibuat dengan Edena. Edena yang digunakan adalah Edena versi terakhir, yaitu Edena V3.131028. Untuk menjalankan Edena menggunakan sistem operasi Ubuntu 11.04.

Data masukan yang digunakan sama dengan yang digunakan perangkat lunak, yaitu tiga pasang *data set* dari tiga organisme yang telah ditentukan sebelumnya. Tiap pasang mewakili satu organisme namun berbeda jumlah *reads* nya, 2000 *reads* dan 5000 *reads*. Percobaan dilakukan tiga kali dan setiap percobaan menggunakan *data set* baru yang dibangkitkan menggunakan MetaSim. Hal ini dilakukan karena MetaSim melakukan simulasi *data set* secara acak. Sehingga *reads* yang terdapat padas *data set* untuk satu organisme bisa berbeda. Oleh karena itu, untuk memastikan akurasi hasil reduksi dilakukan tiga kali pengujian. Berikut Hasil perbandingan antara Perangkat lunak dengan Edena disajikan pada Tabel 4, Tabel 5 dan Tabel 6.

Tabel 4 Perbandingan hasil reduksi antara perangkat lunak dengan Edena menggunakan *Data Set* I

Perangkat Lunak	Jumlah reads					
	Staphylococcus aureus		*Acetobacter pasteurianus*		*Lactobacillus plantarum*	
	2000	**5000**	**2000**	**5000**	**2000**	**5000**
Edena	1086	1390	1223	1672	1267	1747
Sistem	1086	1390	1223	1672	1267	1747

Tabel 5 Perbandingan hasil reduksi antara perangkat lunak dengan Edena menggunakan *Data Set* II

Perangkat Lunak	Jumlah reads					
	Staphylococcus aureus		*Acetobacter pasteurianus*		*Lactobacillus plantarum*	
	2000	**5000**	**2000**	**5000**	**2000**	**5000**
Edena	1087	1394	1177	1687	1233	1747
Sistem	1087	1394	1177	1687	1233	1747

Tabel 6 Perbandingan hasil reduksi antara perangkat lunak dengan Edena menggunakan *Data Set* III

Perangkat lunak	Jumlah reads					
	Staphylococcus aureus		*Acetobacter pasteurianus*		*Lactobacillus plantarum*	
	2000	**5000**	**2000**	**5000**	**2000**	**5000**
Edena	1104	1396	1180	1675	1229	1764
Sistem	1104	1396	1180	1675	1229	1764

Pada percobaan I, *data set* organisme *Staphylococcus aureus* dengan 2000 *reads* direduksi menjadi 1086 *reads.* Artinya, *reads* yang tunggal dan tidak terdapat *reverse complement* sebanyak 1086 *reads.* Pada Percobaan II dan III, *data set* organisme yang sama, *Staphylococcus aureus*, dengan jumlah *reads* yang sama, 2000 *reads*, direduksi menjadi 1087 *reads* dan 1104 *reads.* Hal tersebut terjadi karena aplikasi MetaSim membuat *data set* secara acak. *Redundant reads* dan *reverse complement reads* yang terbentuk lebih banyak. Hasil reduksi berkurang menjadi 1087 *reads* dan 1104 *reads.*

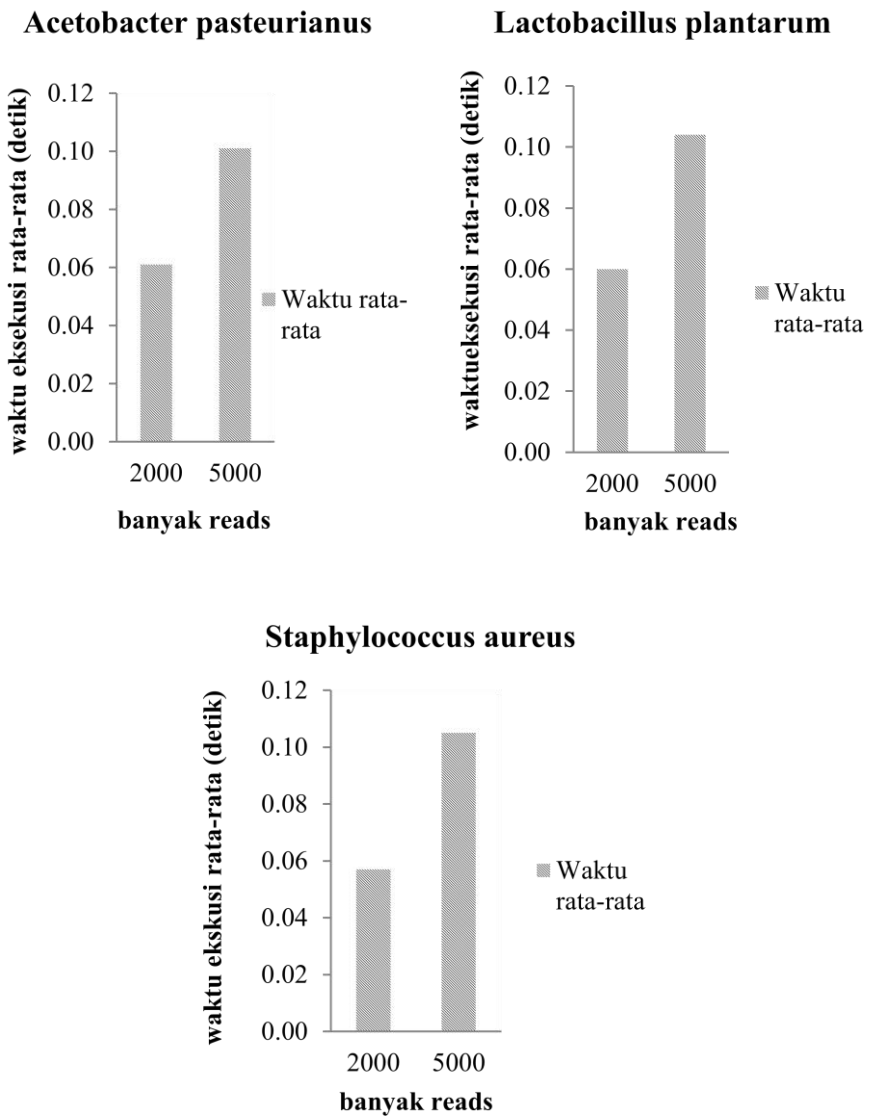
Perbedaan hasil reduksi pada percobaan I, II dan III juga terjadi pada dua organisme lainnya, *Acetobacter pasteurianus* dan *Lactobacillus plantarum.* Dari 3 Tabel di atas dapat terlihat bahwa hasil reduksi antara perangkat lunak dengan Edena menghasilkan jumlah *reads* tunggal yang sama. Dengan demikian, dapat disimpulkan perangkat lunak berhasil mereduksi *redundant reads* dan *reverse complement reads.*

Evaluasi Waktu Eksekusi

Pada evaluasi ini, dilakukan percobaan untuk mencatat waktu eksekusi sebanyak tiga kali untuk tiga pasang *data set* dari tiga organisme. Satu pasang *data set* mewakili satu organisme yang berbeda jumlah *reads* nya. *Data set* yang digunakan sama untuk tiap percobaan. Pada Gambar 7 disajikan grafik waktu eksekusi yang didapat dari ketiga organisme.

Dari grafik pada Gambar 7, memperlihatkan bahwa jumlah *reads* yang ditetapkan, yaitu 2000 *reads* dan 5000 *reads* berbanding lurus dengan lama eksekusi. Semakin banyak jumlah *reads*, waktu yang dibutuhkan program akan relatif semakin lama. Hal ini disebabkan semakin besar *data set*, semakin banyak *reads*, memungkinkan semakin banyak *redundant reads*

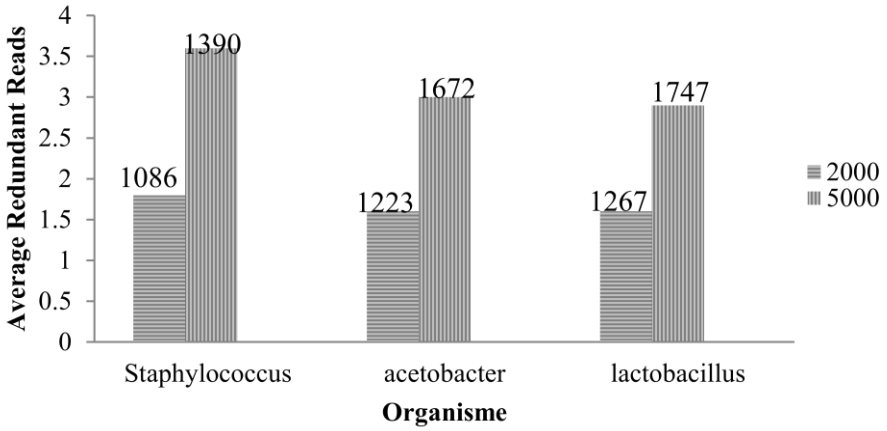
dan *reverse complement* reads sehingga butuh waktu yang lebih banyak untuk memprosesnya.



Gambar 7 Grafik waktu eksekusi

Evaluasi Average Redundant Reads

Pada evaluasi ini, dilakukan percobaan membandingkan *average redundant reads* terhadap jumlah *reads* awal dari ketiga organisme. *Average* re*dudant reads* adalah nilai rata-rata dari kemunculan *redundant reads.* Nilai tersebut didapat dari hasil perbandingan jumlah *reads* awal dengan jumlah *reads* unik. Grafik antara *average redundant reads* dengan jumlah *reads* awal dapat dilihat pada Gambar 8.



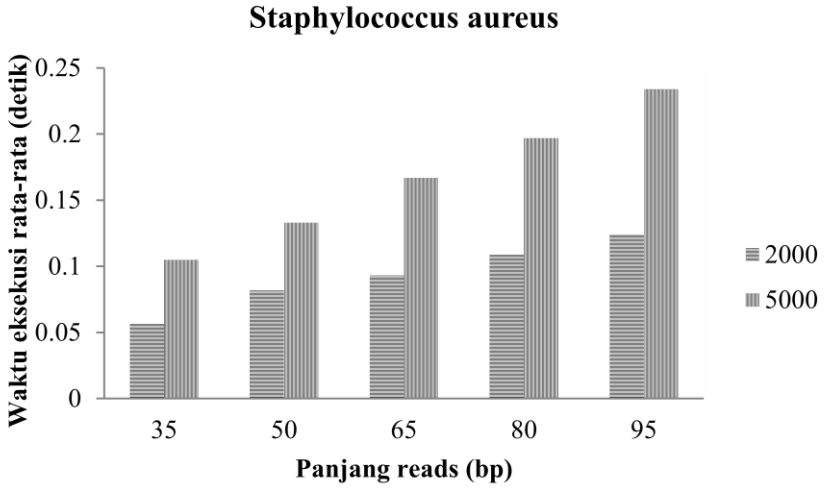
Gambar 8 Grafik *average redundant reads*

Pada Gambar 8 dapat dilihat grafik *reads* awal berjumlah 2000, *average redundant reads* pada organisme *Acetobacter pasteurianus* dan *Lactobacillus plantarum* bernilai sama, yaitu 1.6. Hal tersebut dikarenakan jumlah *reads* tunggal pada kedua organisme tersebut hampir sama, yaitu 1223 *reads* pada *Acetobacter pasteurianus* dan 1267 *reads* pada *Lactobacillus plantarum*. Pada grafik jumlah *reads* awal berjumlah 5000, nilai *average redundant reads* pada organisme *Acetobacter pasteurianus* dan *Lactobacillus plantarum* hanya berselisih sedikit, yaitu 3.0 pada *Acetobacter pasteurianus* dan 2.9 pada *Lactobacillus plantarum.* Hal ini terjadi karena *Acetobacter pasteurianus* memiliki jumlah *reads* tunggal lebih sedikit dibanding *Lactobacillus plantarum.*

Staphylococcus aureus memiliki nilai *average redundant reads* tertinggi diantara ketiga organisme, yaitu 1.8 pada grafik *reads* awal berjumlah 2000 dan 3.6 pada grafik *reads* awal berjumlah 5000. Pada grafik *reads* awal berjumlah 2000 dan grafik *reads* awal berjumlah 5000, *reads* tunggal *Staphylococcus aureus* berjumlah 1086 *reads* dan 1390 *reads.* Jumlah *reads* tunggal tersebut paling kecil dibanding dua organisme lain, *Acetobacter pasteurianus* dan *Lactobacillus plantarum.* Sehingga dapat disimpulkan bahwa semakin kecil jumlah *reads* tunggal maka nilai *average redundant* akan semakin besar.

Evaluasi Panjang *Reads*

Pada evaluasi ini, dilakukan percobaan dengan menggunakan lima pasang *data set* dari satu organisme. Setiap pasang *data set* memiliki dua jenis jumlah *reads* awal, yaitu 2000 *reads* dan 5000 *reads.* Satu pasang *data set* mewakili panjang *reads* yang berbeda, yaitu 35 bp, 50 bp, 65 bp, 80 bp dan 95 bp. Organisme yang digunakan untuk percobaan ini adalah *Staphylococcus aureus.* Evaluasi ini bertujuan untuk mencari hubungan antara panjang *reads* dengan lama waktu eksekusi. Grafik hubungan panjang *reads* dengan lama waktu eksekusi dapat dilihat pada Gambar 9.



Gambar 9 Grafik panjang *reads*

Dari grafik pada Gambar 9, panjang *reads* yang ditetapkan, yaitu 35 bp, 50 bp, 65 bp, 80 bp dan 95 bp berbanding lurus dengan lama waktu eksekusi. Semakin panjang *reads*, waktu yang dibutuhkan program untuk mengeksekusi *data set* semakin lama. Hal ini terjadi karena, semakin panjang *reads* maka proses pembentukan *reverse complement reads* dan pengindeksan *reads* asli dan *reverse complement reads* pada *prefix tree* membutuhkan waktu yang lebih lama.

Kompleksitas

Kompleksitas waktu program untuk membentuk *reverse complement*, memproses *reads* asli ke dalam *prefix tree*, dan memproses *reverse complement reads* ke dalam *prefix tree* adalah konstan, atau O(1). Sedangkan proses-proses tersebut diulang sebanyak inputan (n) di dalam fail. Dengan demikian, kompleksitas waktu program adalah O(n) atau linier terhadap jumlah inputan.

SIMPULAN DAN SARAN

Simpulan

Penelitian ini berhasil menghasilkan perangkat lunak yang mampu melakukan reduksi terhadap *redundant reads* dan *reverse complement reads* pada data sekuens DNA dengan metode *prefix tree*. Perangkat lunak ini juga telah berhasil memberikan informasi waktu eksekusi, jumlah *reads* yang dibaca, jumlah *reads* yang dihasilkan dan *average redundant reads*. Selain itu, perangkat lunak dapat menyimpan hasil *reduksi* ke dalam fail keluaran (.fna).

Saran

Untuk penelitian selanjutnya disarankan menerapkan algoritma yang sama dengan menyertakan pemrosesan paralel agar didapatkan waktu eksekusi yang lebih cepat. Selain itu, *data set* yang digunakan tidak terbatas pada *data set* yang *error-free* tetapi juga bisa memproses *data set* yang memiliki *error*.

DAFTAR PUSTAKA

Dohm, JC, Lottaz, C, Borodina T, and Himmelbauer H. 2007. SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Res.* 17**:** 1697–1706. doi: 10.1101/gr.6435207.

Hernandez D, François P, Farinelli L, Østerås M, Schrenzel J. 2008. De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research. IPSJ Transactions on Bioinformatics.* 18:802-809. doi: 10.1101/gr.072033.107.

Kusuma WA, Ishida T, Akiyama Y. 2011. A combined approach for de novo DNA sequence assembly of very short reads. *IPSJ Transactions on Bioinformatics.* 4( ):21-33. doi: 10.2197/ipsjtbio.4.21.

Maftuhah DN, 2007. Pencarian *string* dengan menggunakan metode indexing pada *data genomic* [skripsi]. Depok (ID): Universitas Indonesia.

Pevzner PA, Tang H, Waterman MS. 2001. An Euler path approach to DNA fragment assembly. Proc. Natl. Acad. Sci. 98(17):9478-9753. doi: 10.1073/pnas.171285098.

Rahim A, 2013. Penyusunan *overlap graph* menggunakan *suffix tree* pada DNA *sequence* [skripsi]. Bogor (ID): Institut Pertanian Bogor.

Richter DC, Ott F, Auch AF, Huson DH, Schmid R. 2008. MetaSim – A sequencing simulator for genomics and metagenomics. *Plos One*. 3(10):e3373. doi:10.1371/journal.pone.0003373.

Trapnell C, Salzberg SL. 2009. How to map billions of short reads onto genomes. *Nat Biotechnol*. 27(5):455-457. doi: 10.1038/nbt0509-455.

Warren RL, Sutton GG, Jones SJ, and Holt RA. 2007. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics.* 23: 500–501. doi:10.1093/bioinformatics/btl629

Zerbino DR, Birney W. 2008. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.* 18(5):821–829. doi: 10.1101/gr.074492.107

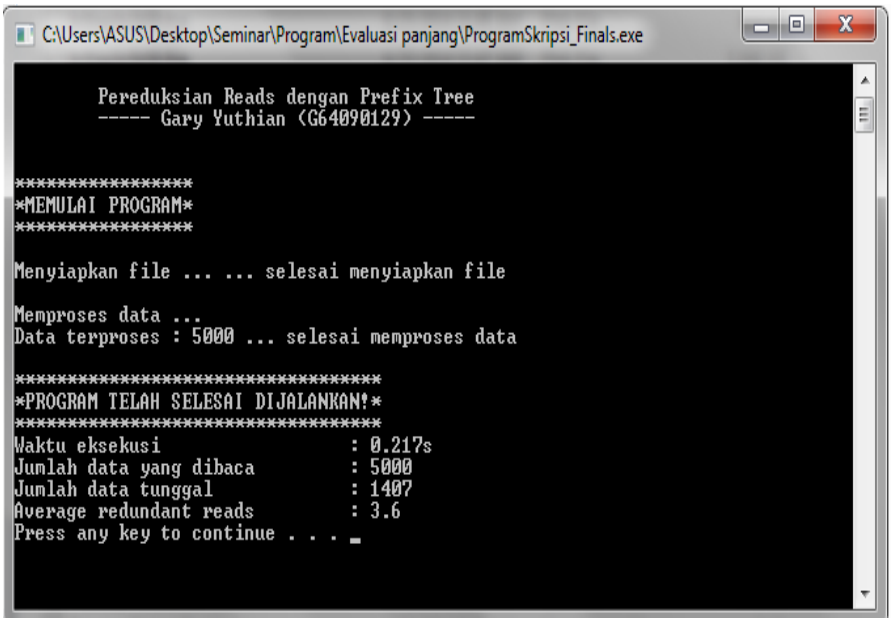
Lampiran 1 *Source code* program

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
#include <iomanip>
#include <time.h>

using namespace std;

typedef struct sTrieNode{
   char key;   //untuk salah satu karakter dari rantai DNA
   int value;  //untuk banyaknya data masuk pada node
   struct sTrieNode *next[4]; //max. child cuman 4 (A, C, G, T)
} trieNode;

typedef struct sTrieRoot{
   trieNode *root;
} *trieRoot;

void resetTrie (trieNode *tNode){   //jaga-jaga dari 'garbage data'
   tNode->key = 0; //key di set NULL
   tNode->value = 0;
   int i; for(i=0; i<4; i++) tNode->next[i] = NULL;     //child nya diset tidak menunjuk
//apa-apa
}

trieRoot trieCreate(){
   trieRoot trie = (trieRoot)malloc(sizeof(trieRoot)); //alokasikan memori buat struct
//Root
   trie->root = (trieNode *)malloc(sizeof(trieNode));    //alokasikan memori buat node
//yang pertama
   resetTrie(trie->root);
   return trie;
}

int trieAdd(trieRoot trie, char keys[], int isRC, int r){
//isRC bernilai 1 (true) jika string yang dipanggil adalah reverse Complement
   trieNode *level = trie->root; //inisiasi struct Node dgn nama  level
   int l;

   for(l=0; l<r; l++){     //looping sesuai panjang rantai
      int i;
      for(i=0; i<4; i++){ //loop buat cek setiap node child
        if(level->next[i] == NULL){
          trieNode *newChild = (trieNode *)malloc(sizeof(trieNode));
          resetTrie(newChild);        //habis bikin node child, reset node-nya

          level->next[i] = newChild;
          level = newChild;       //set variable level jadi node yang baru, jadi proses di node
//yang baru itu
          level->key = keys[l]; //simpan key yang lagi diproses ke  node child yang baru
           break;  //keluar dari loop buat cek tiap node child
         }
        else if(level->next[i]->key == keys[l]){   //kondisi key yang diproses udah ada di
//node child-nya
```

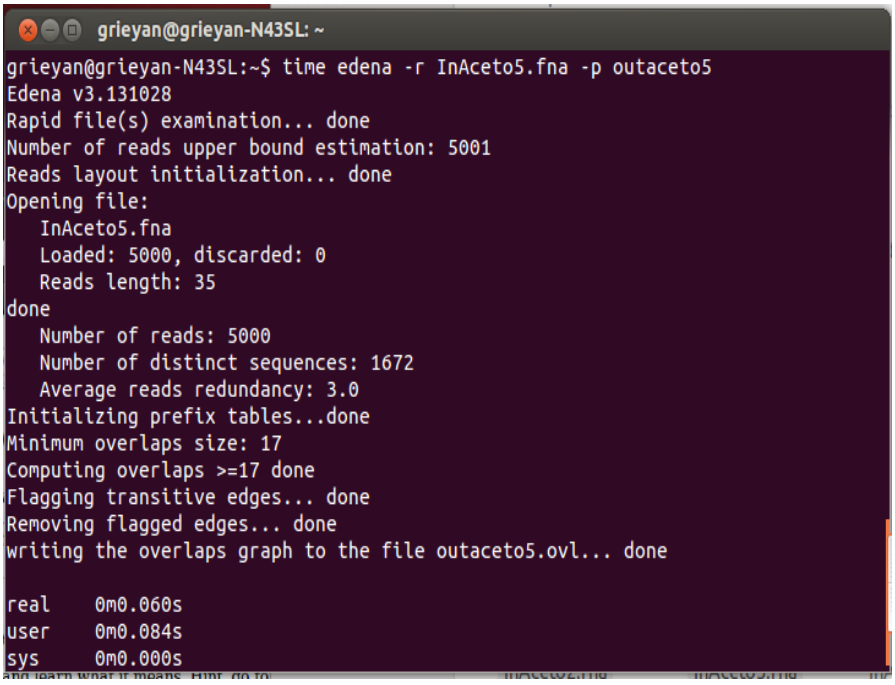
```
          level = level->next[i];
          break; //keluar dari loop buat cek tiap node child
         }
       }
         level->value+=1; //counter buat simpan ada berapa rantai di node itu

       if(l==r-1){
         if(level->value==1 && !isRC) return 1;
         else return 0;
       }
   }
}

typedef struct sChain{
   char asli[256];  //variabel rantai asli
   char komplemen[256]; //variabel reverse complement rantai asli
}Chain;

int main(){
   trieRoot trie = trieCreate();//buat trie
   clock_t tStart = clock();   //start timer

            cout << "\n\tPereduksian Reads dengan Prefix Tree\n";
            cout << "\t----- Gary Yuthian (G64090129) -----\n\n\n";


   cout <<"*****************\n*MEMULAI PROGRAM*\n*****************\n\n";

   cout << "Menyiapkan file ... ";
   FILE *db = fopen("InStapilo5k95.fna","r"),*output = fopen("OutStapilo5k95.fna","w");

   char temp[500];//dummy variable, buat nyimpen line informasi
   Chain chain;//deklarasi struct chain
   int i, counter = 0, n=0;//counter utk print di file output,
//n utk hitung data yang diproses
   int r=0;
   int checkSama = 0;
   float n1, c1, average;

   if (db){
      cout << "... selesai menyiapkan file" << endl;
      cout << "\nMemproses data ...";
      while (fgets(temp, sizeof(temp), db) != NULL){
//selama belum sampai akhir dari file, proses data
         char readTemp;
         int ci=0;
         memset(chain.asli,'\0',256);
         memset(chain.komplemen,'\0',256);
         while (readTemp=fgetc(db),readTemp !=EOF && readTemp != '>'){
           switch (readTemp){
           case 'A': case 'T': case 'G': case 'C':
             chain.asli[ci]=readTemp;
             ci++;
           }

         }

         int rNew = (chain.asli[strlen(chain.asli)-1]=='\n' ? strlen(chain.asli)-1 :
```

```
strlen(chain.asli));       //panjang rantai

          if(r != rNew) {
            checkSama++;
            if(checkSama>1){
              cout << endl << "Data input tidak valid : Terdapat perbedaan pada panjang
rantai" << endl;
              break;
            }
          }
          r = rNew;

          for (i=0; i<r; i++){
             switch (chain.asli[i]){
                case 'A':chain.komplemen[r-1-i] = 'T'; break;
                case 'T':chain.komplemen[r-1-i] = 'A'; break;
                case 'G':chain.komplemen[r-1-i] = 'C'; break;
                case 'C':chain.komplemen[r-1-i] = 'G'; break;
             }
          }
          if(trieAdd(trie, chain.asli, 0, r)){
             counter++;//counter bertambah ketika ada data yang diprint
           chain.asli[r]= (chain.asli[r]=='\n'?'\0':chain.asli[r]);
//buat hilang newline di akhir rantai
           fprintf(output, ">r%d\n%s\n", counter, chain.asli);

          }

          trieAdd(trie, chain.komplemen, 1, r);  //nilai parameter terakhir (1/true) berarti data
//yang kita input itu rantai reverse komplemen-nya


          n++;//data yang selesai di proses nambah 1

       }
       cout << "\nData terproses : "<< n <<" ... selesai memproses data\n";
    }
    else                  printf("\n**********************\n*FILE                  TIDAK
DITEMUKAN!*\n**********************\n");

             average = (float)n/counter;

    cout    <<    "\n************************************\n*PROGRAM    TELAH
SELESAI DIJALANKAN!*\n************************************" << endl;
    cout << "Waktu eksekusi\t\t\t: " << (double)(clock() - tStart)/CLOCKS_PER_SEC <<
"s\n";
    cout << "Jumlah data yang dibaca\t\t: " << n << endl;
    cout << "Jumlah data tunggal\t\t: " << counter << endl;
    cout << fixed << setprecision(1) << "Average redundant reads\t\t: " << average <<
endl;
    fclose(db);
    fclose(output);
    system("pause");
    return 0;
}
```

Lampiran 2 *Screenshot* perangkat lunak



Lampiran 3 *Screenshot* Edena



RIWAYAT HIDUP

Penulis lahir di Jakarta pada tanggal 1 Juli 1992. Penulis merupakan sulung dari 3 bersaudara dari pasangan Bapak Isdoni dan Ibu Renny Har. Penulis menghabiskan masa pendidikan dasar, menengah dan atas di kota kelahirannya, Jakarta. Tahun 2009 menjadi tahun kelulusan penulis dari SMA Negeri 77 Jakarta. Di tahun yang sama pula penulis diterima sebagai mahasiswa Institut Pertanian Bogor di departemen Ilmu Komputer lewat jalur SNMPTN.

Pada Semester 3 berkuliah di IPB, Penulis mengambil Minor Manajemen Fungsional. Penulis melaksanakan kegiatan praktek kerja lapang selama 2 bulan di Badan Pemeriksa Keuangan RI pada tahun 2012.