Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE

DESY NURBAITI RAHMI
NRP 5114 100 030

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Abdul Munif, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

TUGAS AKHIR - KI141502

IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE

DESY NURBAITI RAHMI
NRP 5114 100 030

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Abdul Munif, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

UNDERGRADUATE THESES - KI141502

IMPLEMENTATION OF ROPE DATA STRUCTURE ON CASE STUDY IN SPOJ PROBLEM ALPHABETIC ROPE

DESY NURBAITI RAHMI
NRP 5114 100 030

Supervisor 1
Rully Soelaiman, S.Kom., M.Kom.

Supervisor 2
Abdul Munif, S.Kom., M.Sc.

INFORMATICS DEPARTMENT
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
Desy Nurbaiti Rahmi
NRP: 5114 100 030

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom. .....
NIP: 197002131994021001 (Pembimbing 1)

Abdul Munif, S.Kom., M.Sc. .....
NIP: 198608232015041004 (Pembimbing 2)

SURABAYA
JANUARI 2018

Halaman ini sengaja dikosongkan

IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE

Nama	: DESY NURBAITI RAHMI
NRP	: 5114 100 030
Departemen	: Informatika FTIF-ITS
Pembimbing I	: Rully Soelaiman, S.Kom., M.Kom.
Pembimbing II	: Abdul Munif, S.Kom., M.Sc.

Abstrak

Permasalahan alphabetic rope merupakan sebuah permasalahan yang melibatkan sebuah rentang pencarian. Tipe query secara umum dibagi menjadi dua yaitu, operasi pencarian dan perubahan. Operasi perubahan pada suatu rentang akan menyebabkan perubahan hasil pencarian selanjutnya. Untuk menangani berbagai permasalahan pada operasi alphabetic rope yang harus dilakukan sehingga dibutuhkan struktur data yang mampu mendukung operasi-operasi tersebut dengan efisien.

Pada Tugas Akhir ini akan dirancang penyelesaian permasalahan alphabetic rope antara lain operasi pencarian karakter pada indeks ke-y pada konfigurasi rope saat ini, operasi memotong segmen rope pada indeks ke-x sampai y dan menggabungkan pada bagian depan rope, dan operasi memotong segmen rope pada indeks ke-x sampai y dan menggabungkan pada bagian belakang rope. Struktur data klasik yang biasa digunakan dalam penyelesaian permasalahan ini adalah stuktur data String. Namun penggunaan algoritma String masih kurang efisien dalam hal kecepatan dan kebutuhan memori.

Pada Tugas Akhir ini digunakan struktur data Rope untuk menyelesaikan operasi-operasi tersebut. Hasil uji coba menunjukkan program menghasilkan jalur yang benar dan memiliki pertumbuhan waktu secara logaritmik dengan kompeksitas waktu sebesar $O(\log N)$ *per query.*

Kata Kunci: concatenation, rope, split, string

IMPLEMENTATION OF ROPE DATA STRUCTURE ON CASE STUDY IN SPOJ PROBLEM ALPHABETIC ROPE

Name	: DESY NURBAITI RAHMI
NRP	: 5114 100 030
Major	: Informatics Department Faculty of IT-ITS
Supervisor I	: Rully Soelaiman, S.Kom., M.Kom.
Supervisor II	: Abdul Munif, S.Kom., M.Sc.

Abstract

Alphabetic rope computation is a problem which involves a specific range of data. Generally it divide by two type of query, range search and update. An update operation would have an impact for the following range search query. To handle various problem in alphabetic rope, an efficient data structure is needed to support the operations.

This undergraduate thesis will be designed problem solving for alphabetic rope query variation such as find the y-th position on current rope, cut the rope segment from x-th to y-th to join at the front of rope and cut the rope segment from x-th to y-th and join at the back of rope. Well known data structures, e.g string are commonly used for solving the this problem. But those algorithm were not efficient enough in the term of running time and memory usage.

In this undergraduate thesis Rope data structure will be used instead for solving those alphabetic rope variations. The program had been tested and proved that it produces correct result and running in $O(\log N)$ *per query.*

Keywords: concatenation, rope, split, string

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur Penulis panjatkan kepada Allah SWT. atas pimpinan, penyertaan, dan karunia-Nya sehingga Penulis dapat menyelesaikan Tugas Akhir yang berjudul :

IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE.

Penelitian Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan Penulis dapat memberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri Penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama Penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

- Papa, Mama dan keluarga Penulis yang selalu memberikan perhatian, dorongan dan kasih sayang yang menjadi semangat utama bagi diri Penulis sendiri baik selama penulis menempuh masa perkuliahan maupun pengerjaan Tugas Akhir ini.
- Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah banyak meluangkan waktu untuk memberikan ilmu, nasihat, motivasi, pandangan dan bimbingan kepada Penulis baik selama Penulis menempuh masa kuliah maupun selama pengerjaan Tugas Akhir ini.

- Bapak Abdul Munif, S.Kom., M.Sc. selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada Penulis.
- Seluruh tenaga pengajar dan karyawan Jurusan Teknik Informatika ITS yang telah memberikan ilmu dan waktunya demi berlangsungnya kegiatan belajar mengajar di Jurusan Teknik Informatika ITS.
- Seluruh teman Penulis di Jurusan Teknik Informatika ITS yang telah memberikan dukungan dan semangat kepada Penulis selama Penulis menyelesaikan Tugas Akhir ini.
- Teman-teman, Kakak-kakak dan Adik-adik *administrator* Laboratorium Arsitektur dan Jaringan Komputer yang selalu menjadi teman untuk berbagi ilmu.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui Tugas Akhir ini Penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, Januari 2018

Desy Nurbaiti Rahmi

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR KODE SUMBER

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Di era modern ini, sudah banyak algoritma dan struktur data yang ditemukan. Masing-masing algoritma seringkali membutuhkan struktur data yang tepat untuk membantu menyelesaikan permasalahan tersebut. Salah satunya pada permasalahan *string*.
Topik Tugas Akhir ini mengacu pada permasalahan *string* pada permasalahan klasik *Alphabetic Rope*[1]. Pada permasalahan ini terdapat sekumpulan data berupa *string* yang diberikan di awal. Dari sekumpulan *string* ini akan dilakukan sejumlah pertanyaan dan perubahan atau *update* pada data awal yang diberikan. Struktur data yang diduga dapat membantu penyelesaian permasalahan ini adalah struktur data Rope.
Hasil Tugas Akhir ini diharapkan dapat memberi gambaran mengenai performa dari struktur data Rope dalam penyelesaian permasalahan di atas secara optimal dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana menganalisis serta menentukan algoritma dan struktur data yang tepat untuk menyelesaikan permasalahan klasik *Alphabetic Rope* dengan optimal?
2. Bagaimana hasil perbandingan performa dari struktur data Rope yang dibangun dalam menjawab permasalahan klasik *Alphabetic Rope* yang diberikan?

1.3 Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas jumlah maksimum data awal yang diberikan adalah 100.000 karakter.
3. Batas maksimum jumlah *query* dan/atau *update* adalah 100.000 perintah.
4. Batas waktu eksekusi program adalah 1 detik.
5. Dataset yang digunakan adalah dataset pada permasalahan klasik *Alphabetic Rope*.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Melakukan analisis dan mendesain algoritma dan struktur data untuk menyelesaikan permasalahan klasik *Alphabetic Rope* dengan pembuktian (*proofing*) yang jelas.
2. Membandingkan performa dari algoritma dan struktur data yang dapat digunakan untuk menyelesaikan permasalahan klasik *Alphabetic Rope* dengan analisis yang jelas.

1.5 Manfaat

Manfaat dari Tugas Akhir ini adalah sebagai berikut:

1. Mengetahui penggunaan struktur data yang tepat untuk menyelesaikan permasalahan klasik *Alphabetic Rope* secara optimal dan efisien.

1.6 Metodologi

Metodologi yang digunakan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir yang berisi gagasan untuk menyelesaikan permasalahan *rope* pada studi kasus permasalahan klasik *Alphabetic Rope*.
2. Studi literatur
   Pada tahap ini dilakukan pencarian informasi dan studi literatur mengenai pengetahuan atau metode yang dapat digunakan dalam penyelesaian masalah. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma yang digunakan untuk penyelesaian permasalahan ini, materi-materi tersebut didapatkan dari buku, jurnal, maupun internet.
3. Desain
   Pada tahap ini dilakukan desain rancangan algoritma yang digunakan dalam solusi untuk pemecahan permasalahan klasik *Alphabetic Rope*.
4. Implementasi perangkat lunak
   Pada tahap ini dilakukan implementasi atau realiasi dari rancangan desain algoritma yang telah dibangun pada tahap desain ke dalam bentuk program.
5. Uji coba dan evaluasi

Pada tahap ini dilakukan uji coba kebenaran implementasi. Pengujian kebenaran dilakukan pada sistem penilaian daring SPOJ sesuai dengan masalah yang dikerjakan untuk diuji apakah luaran dari program telah sesuai.

6. Penyusunan buku Tugas Akhir
   Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi hasil pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

Berikut adalah sistematika penulisan buku Tugas Akhir ini:

1. BABI: PENDAHULUAN
   Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan Tugas Akhir.
2. BAB II: DASAR TEORI
   Bab ini berisi dasar teori mengenai permasalahan dan algoritma penyelesaian yang digunakan dalam Tugas Akhir
3. BAB III: DESAIN
   Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian permasalahan.
4. BAB IV: IMPLEMENTASI
   Bab ini berisi implementasi berdasarkan desain algortima yang telah dilakukan pada tahap desain.
5. BAB V: UJI COBA DAN EVALUASI
   Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.
6. BAB VI: PENUTUP
   Bab ini berisi kesimpulan dan saran yang didapat dari hasil uji coba yang telah dilakukan.

BAB II
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar pengerjaan Tugas Akhir ini.

2.1 Deskripsi Permasalahan

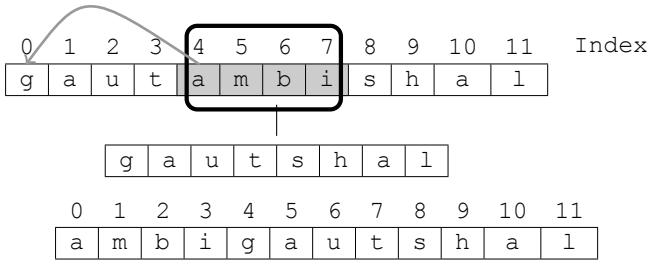
Terdapat sebuah *alphabetic rope*[1] yang terdiri dari karakter-karakter yang terlihat seperti serangkaian *string* dengan karakter yang terdiri dari huruf kecil. Diberikan beberapa operasi perubahan dan atau pertanyaan. Berikut adalah macam-macam tipe operasi:

- 1 **X Y** : memotong segmen *rope* dari indeks ke-X sampai Y dan menggabungkan pada bagian depan *rope*
- 2 **X Y** : memotong segmen *rope* dari indeks ke-X sampai Y dan menggabungkan pada bagian belakang *rope*
- 3 **Y** : mencetak pada baris baru huruf pada indeks ke-Y dari konfigurasi *rope* saat ini

Mula-mula terdapat sejumlah N data berupa *string* sebagai data awal. Kemudian terdapat serangkaian Q buah operasi yang akan dilakukan pada data awal yang telah diberikan. Sebuah operasi dapat berupa sebuah tindakan yang mengubah kondisi data. Variasi dari operasi-operasi yang menjadi permasalahan pada Tugas Akhir ini akan dijelaskan pada subbab-subbab di bawah.

2.1.1 Operasi Memotong Segmen *Rope* dari Indeks ke-X sampai Y dan Menggabungkan pada Bagian Depan *Rope*

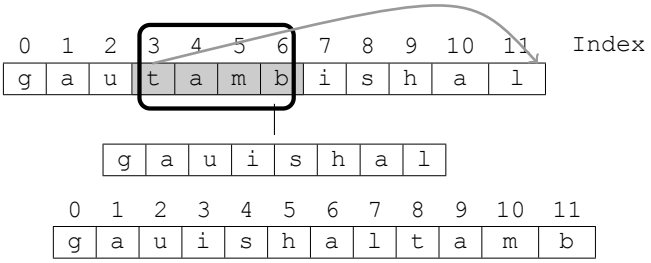
Terdapat 3 parameter pada operasi ini yaitu $type$, x, dan y yang berupa posisi dari indeks dari *string* yang akan dipotong dan digabungkan pada posisi awal *rope*. Misalkan terdapat *string* "$gautambishal$", dilakukan operasi 1 pada indeks ke-4 sampai 7. Maka *string* pada indeks tersebut akan dipotong dan digabungkan pada posisi awal dari *rope*. Ilustrasi permasalahan ini dapat dilihat pada Gambar 2.1.



Gambar 2.1 Ilustrasi Operasi Memotong Segmen *Rope* dari Indeks ke-4 sampai 7 dan Menggabungkan pada Bagian Depan *Rope*

2.1.2 Operasi Memotong Segmen *Rope* dari Indeks ke-X sampai Y dan Menggabungkan pada Bagian Belakang *Rope*

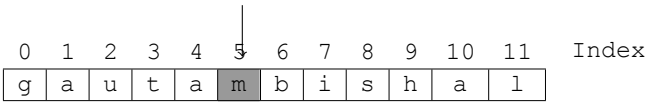
Sama dengan operasi pada subbab 2.1.1, operasi ini memiliki 3 parameter yaitu $type$, x dan y. Namun potongan *string* akan digabungkan pada posisi akhir *rope*. Misal terdapat *string* "$gautambishal$", dan dilakukan operasi 2 pada indeks ke-3 sampai 6. Maka *string* pada indeks tersebut akan digabungkan pada bagian akhir dari *rope*. Ilustrasi permasalahan ini dapat dilihat pada Gambar 2.2.



Gambar 2.2 Ilustrasi Operasi Memotong Segmen *Rope* dari Indeks ke-3 sampai 6 dan Menggabungkan pada Bagian Belakang *Rope*

2.1.3 Operasi Mencari Huruf pada Indeks ke-Y dari Konfigurasi *Rope* Saat Ini

Terdapat dua buah parameter pada operasi ini yaitu $type$ dan y. Nilai y merupakan posisi indeks huruf yang akan dicetak. Misalkan terdapat sebuah *rope* yang berisi *string* "$gautambishal$", dilakukan operasi 3 pada indeks ke-5. Maka jawaban dari pertanyaan ini adalah m. Ilustrasi operasi ini dapat dilihat pada Gambar 2.3.



Gambar 2.3 Ilustrasi Mencari Huruf pada Indeks ke-5 pada Konfigurasi *Rope* Saat Ini

2.2 Deskripsi Umum

Pada subbab ini akan dijelaskan mengenai deskripsi-deskripsi umum yang terdapat pada Tugas Akhir ini.

2.2.1 *String*

Pada dunia ilmu komputer, *string* didefinisikan sebagai sebuah rangkaian karakter, baik sebagai konstanta literal atau semacam sebuah variabel. Sebagai variabel, *string* dapat dimungkinkan unsur-unsurnya dapat dimutasi dan panjangnya menjadi berubah. *String* pada umumnya dipahami sebagai sebuah struktur data dan diimplementasikan menggunakan struktur Array[2].

2.2.2 *Struct*

Struct merupakan suatu struktur data yang menggabungkan sekumpulan data yang memiliki tipe data yang berbeda. Dalam penerapannya, *struct* digunakan untuk membuat *node* yang berfungsi menyimpan informasi-informasi berupa karakter, berat *node*, *pointer node* dan konstruktor. Kode Sumber 2.1 adalah struktur *Struct* yang digunakan dalam implementasi *rope*.

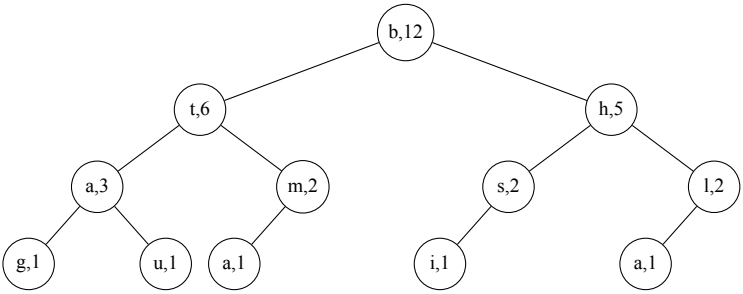
```
1  struct Node {
2    Node *left, *right;
3    int size;
4    char value;
5    Node(char v);
6    Node* update();
7  };
```

Kode Sumber 2.1 Struktur Struct Node

2.2.3 *Binary Search Tree*

Sebuah konsep penyimpanan data, dimana data disimpan dalam bentuk *tree* yang setiap *node* dapat memiliki anak maksimal 2 *node*. BST(*Binary Search Tree*) merupakan susunan *tree* yang sudah terurut. Pencarian data yang tersusun dalam BST dapat dilakukan dengan mudah dan memiliki rata-rata kompleksitas sebesar $O(\log n)$, dan pada kondisi terburuknya dimana BST tidak

berimbang dan berbentuk seperti *linked list* memerlukan waktu sebesar $O(n)$[3].
Pengaplikasian BST pada *rope*, setiap *node* pada *tree* merupakan urutan dari operasi *insert* dari *tree* itu sendiri. Masing-masing *node* hanya menyimpan satu karakter. Berat *leaf node* merupakan panjang *string* yang dimiliki. Sedangkan *parent node* beratnya jumlah berat anak kanan dan anak kirinya ditambah dengan berat dirinya sendiri. Ilustrasi *binary tree* dapat dilihat pada Gambar 2.4.



Gambar 2.4 Contoh *Binary Tree* yang Berisi *String* "$gautambishal$". Setiap *Node* Berisi Sebuah Karakter dan Berat Masing-Masing *Node*

2.3 Struktur Data Rope

Pada permasalahan klasik *Alphabetic Rope*, penyelesaian menggunakan struktur data String dirasa tidak cukup menyelesaikan permasalahan dimana rata-rata kompleksitas yang dimiliki sebesar $O(n)$. Solusi ini masih kurang efisien untuk mengatasi batasan masukan yang cukup besar pada setiap dataset.
Penyelesaian permasalahan yang ditawarkan adalah dengan menggunakan struktur Rope. Struktur data Rope adalah sebuah struktur *binary tree* yang setiap *node*nya menyimpan potongan-potongan dari sebuah *string* yang panjang. Struktur data Rope memiliki beberapa operasi yang dapat dilakukan antara lain

concat, *insert*, *split*, *delete* dan *report*[5].

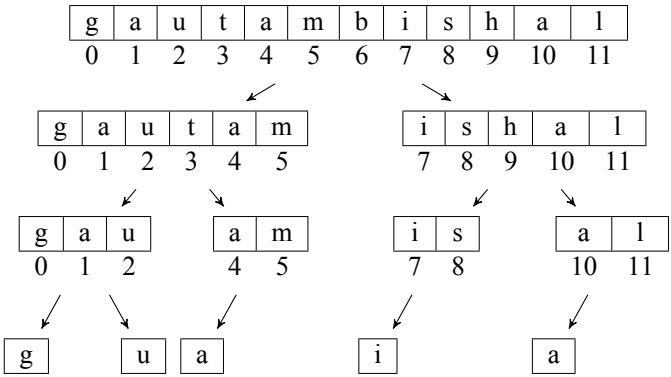
2.3.1 *Build Rope*

Struktur dasar dari *rope* merupakan sebuah *complete binary tree*. Setiap *node* akan berisi satu potongan karakter dari *string* masukan. *Node* pada *rope* akan dibangun dari indeks *string* yang berada di posisi tengah, sampai pada ujung-ujungnya. Setiap *node* pada *rope* akan memiliki anak kiri dan anak kanan jika dan hanya jika masih terdapat *string* yang tersisa. *Pseudocode* fungsi *build* dapat dilihat pada Gambar 2.5.

```
build(*st, *en)
1  if st = en return NULL
2  *mid = st + (en − st)/2
3  return newnode(*mid, build(st, mid), build(mid + 1, en))
```

Gambar 2.5 *Pseudocode* Fungsi Build

Proses ini dilakukan secara rekursif untuk setiap *subtree* yang terbentuk dan berhenti bila mencapai *node leaf* dimana tidak ada lagi *string* yang tersisa. Pada Gambar 2.6 ditunjukkan bagaimana *rope* dibangun berdasarkan *string* "$gautambishal$".
Mula-mula *string* "$gautambishal$" memiliki panjang 12 kemudian dibagi menjadi dua partisi dengan rentang indeks masing-masing. Proses pembangunan sebuah *rope* dimulai dari *string* yang berada pada posisi tengah. *Root* berisi satu karakter pada posisi tengah dari *string* masukan. Bagian kiri *rope* berisi *string* pada indeks 0 sampai posisi tengah. Dan bagian kiri *rope* berisi *string* pada indeks tengah+1 sampai posisi akhir. Dengan secara terus menerus membagi dua panjang *string*, terbentuklah sebuah *complete binary tree*.



Gambar 2.6 Struktur *Rope* Setelah Proses *Build* dengan *String* Awal "$gautambishal$". Pada Ilustrasi Ini Ditampilkan Karakter pada Masing-Masing *Node* Untuk Memberikan Gambaran Umum. Namun Data Tersebut Tidak Disimpan pada Struktur *Rope*

Theorema 2.1

Jika R adalah sebuah *rope* dari *string* dengan panjang N maka *rope* adalah sebuah *complete binary tree* dengan tinggi $H(R) = O(\log N)$.

Proof Misalkan $H(N)$ menyatakan ketinggian *rope* yang berisi *string* sepanjang N, akan terdapat N buah *node* pada *rope*. Misalkan *node* b terpilih menjadi *root*, maka semua *node* dengan posisi indeks kurang dari *root* akan ditempatkan sebagain *subtree* anak kiri, dan yang lebih besar akan di tempatkan sebagai *subtree* anak kanan. Kemudian masing-masing *subtree* kiri dan kanan dari *node* b akan mengikuti pola yang sama secara rekursif. Jadi, ketinggian dari *rope* adalah 1 ditambah ketinggian maksimal dari *subtree* kiri dan *subtree* kanan. Maka,

$$H(N) = \begin{cases} |N|, & |N| \leq 1. \\ \frac{1}{|N|} \sum_{a \in N} (1 + max(H[N_{<}a], H[N_{>}a])), & |N| > 1. \end{cases}$$

Dengan *master theorm* maka rekurens ini terselesaikan menjadi $H(\log N)$.

2.3.2 *Split*

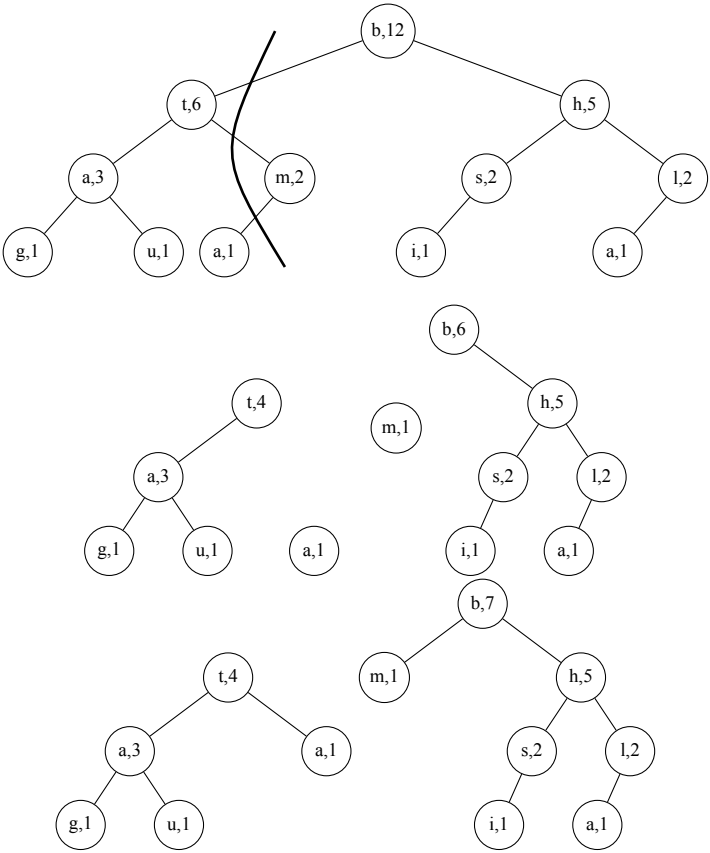
Operasi split(R, i) membagi *rope* R menjadi dua buah *rope* baru R_1 dan R_2, dimana $R_1 = C_0, .., C_{i-1}$ dan $R_2 = C_i, .., C_n$. Sebagai contoh, *rope* yang berisi 12 karakter yang digambarkan pada Gambar 2.7 akan dibagi pada indeks ke-$i = 5$. Langkah pertama mencari *node* pada indeks ke-$i = 5$. Kemudian hapus sambungan antara *node* m dengan *node* a dan *node* t. Telusuri apabila terdapat sebuah node yang terlepas dari induknya. Apabila terdapat *node* yang terlepas dari induknya, bangun kembali *rope* dengan masing-masing komponen barunya.

2.3.3 *Concate*

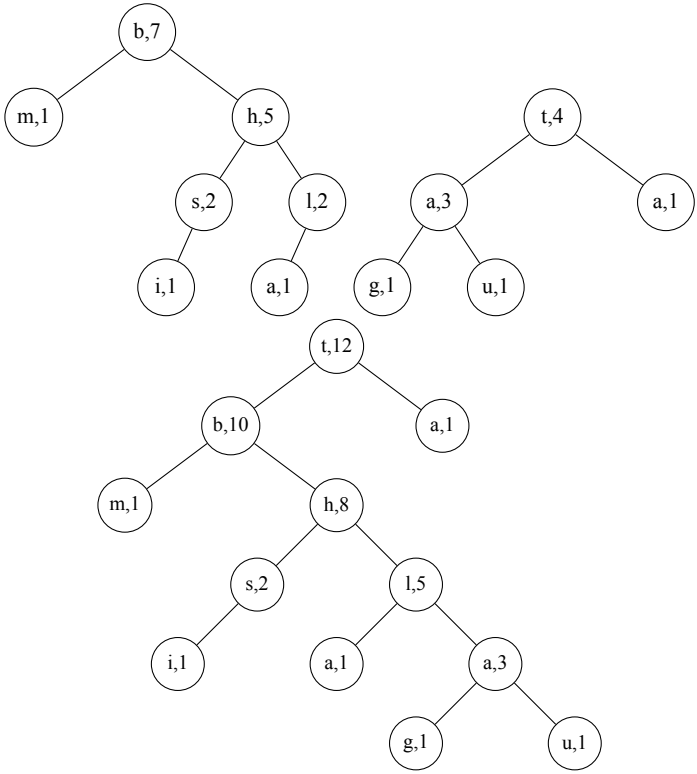
Operasi Concate(R_1, R_2) menggabungkan dua buah *rope*, R_1 dan R_2 menjadi sebuah *rope* utuh. Penggabungan dua buah *rope* R_1 dan R_2 dilakukan dengan menghubungkan *root* R_1 dengan R_2 tergantung pada nilai acak yang didapatkan berdasaarkan hasil modular nilai acak dari kedua *root*. Apabila hasil modular nilai acak lebih kecil dari berat *root* R_1 maka *rope* R_2 akan menjadi anak kanan dari *rope* yang baru terbentuk. Sebaliknya jika mendapatkan hasil modular nilai acak lebih besar atau sama dengan berat *root* R_1 maka R_2 akan menjadi *root* dari *rope* yang baru terbentuk dan *rope* R_1 menjadi anak kirinya. Ilustrasinya dapat dilihat pada Gambar 2.8.

2.3.4 Index

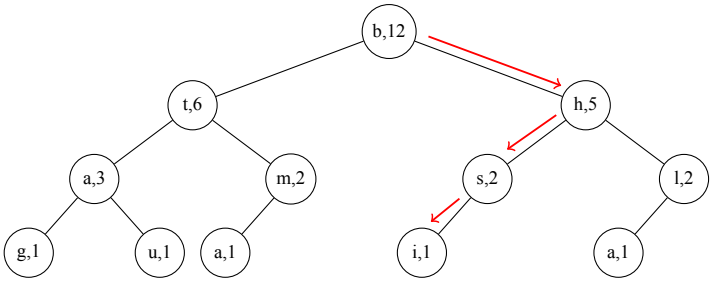
Operasi index(i) menghasilkan sebuah karakter dari *rope* pada posisi ke-i. Sebagai contoh, untuk mencari karakter pada indeks



Gambar 2.7 Split *Rope* pada Indeks ke-5. Setiap *Node* Berisi Sebuah Karakter dan Berat Masing-Masing *Node*



Gambar 2.8 Concate *Rope*. Setiap *Node* Berisi Sebuah Karakter dan Berat Masing-Masing *Node*



Gambar 2.9 Mencari Karakter pada Indeks ke-i=7

ke-i=7 dari *rope* pada Gambar 2.9, dimulai dari *root* b. Kemudian dilakukan pengecekan pada berat *node* anak kiri b. Dikarenakan 6 lebih kecil dari 7, maka pengecekan dilakukan ke anak kanan b dan *node* sekarang menjadi h. Lakukan perubahan pada nilai i menjadi i – berat anak kiri $b - 1 = 0$. Kemudian karena nilai i saat ini adalah 0, maka pencarian dilanjutkan ke anak kiri dan berakhir dengan mengembalikan karakter pada *node* tersebut.

Halaman ini sengaja dikosongkan

BAB III
DESAIN

Pada bab ini akan dibahas tentang desain dan algoritma untuk menyelesaikan permasalahan-permasalahan pada Tugas Akhir ini.

3.1 Permasalahan *Alphabetic Rope*

Pada permasalahan ini sistem akan dibangun untuk menyelesaikan tiga jenis operasi yaitu, memotong segmen *rope* dari indeks ke-x sampai y dan menggabungkan pada bagian depan *rope*, memotong segmen *rope* dari indeks ke-x sampai y dan menggabungkan pada bagian belakang *rope* dan mencetak huruf pada indeks ke-y dari konfigurasi *rope* saat ini.

3.1.1 Deskripsi Umum Sistem

Program akan diawali dengan menerima masukan berupa *string* sebagai data awal dan banyak data uji. Untuk setiap data uji berupa sebuah baris yang diawali dengan data masukan berupa tipe operasi yang akan dilakukan. Setiap data uji, selain tipe operasi 3 diikuti dengan dua data masukan lain yaitu x dan y yang dipisahkan oleh sebuah spasi. Pada data masukan dengan tipe operasi 1 dan 2, nilai x dan y merupakan indeks posisi awal dan indeks posisi akhir dari *string* pada *rope* yang akan diolah. Sedangkan pada tipe operasi 3, data masukan lain hanya berupa y yang merupakan indeks dari huruf yang dicari pada konfigurasi *rope* saat ini. Hasil dari pencarian data pada *rope* akan ditampilkan di layar. Secara garis besar, Fungsi Main dapat dilihat pada Gambar 3.1.

```
Main()
1  r = insert(r, st)
2  while Q − −
3      if type = 3
4          print(r, y)
5      else
6          if type = 1
7              r = mutable_begin(r, x, y − x + 1)
8          else
9              r = mutable_end(r, x, y − x + 1)
```

Gambar 3.1 *Pseudocode* Fungsi Main

3.1.2 Desain Fungsi Newnode

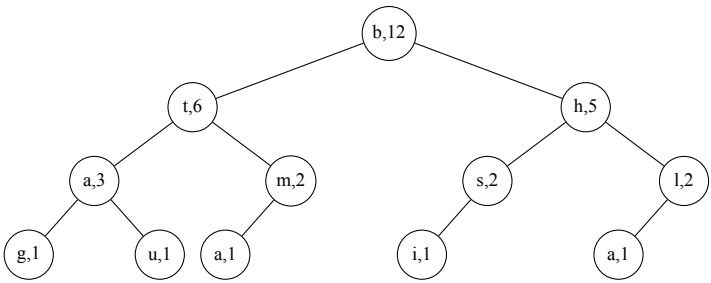
Fungsi Newnode digunakan untuk membentuk sebuah *node* yang berisikan karakter yang diberikan dan relasi terhadap karakter pada posisi yang bersebelahan. *Pseudocode* fungsi Newnode ditunjukkan pada Gambar 3.2.

```
newnode(ch, left, right)
1  r = new Node(ch)
2  r.left = left
3  r.right = right
4  r.update()
5  return r
```

Gambar 3.2 *Pseudocode* Fungsi Newnode

3.1.3 Desain Fungsi Build

Fungsi Build digunakan untuk membangun struktur *tree* dari *rope*. Pembuatan *tree* pada *rope* ini akan dilakukan dari indeks *string* yang berada di posisi tengah, sampai pada ujung-ujungnya. Setiap *node* akan memiliki anak kiri dan anak kanan jika dan hanya jika masih terdapat *string* yang tersisa seperti pada Gambar 2.6. Hasil pembuatan *tree* pada Gambar 2.6 dapat dilihat pada Gambar 3.3. *Pseudocode* Fungsi Build ditunjukkan pada Gambar 3.4.



Gambar 3.3 Build *Tree* Berdasarkan *String* "$gautambishal$". Setiap *Node* Berisi Sebuah Karakter dan Berat Masing-Masing *Node*

build$(*st, *en)$
1 **if** $st = en$ **return** NULL
2 $*mid = st + en - st)/2$
3 **return** newnode$(*mid,$ build$(st, mid),$ build$(mid + 1, en))$

Gambar 3.4 *Pseudocode* Fungsi Build

3.1.4 Desain Fungsi Split

Fungsi Split bertujuan untuk membagi *rope* menjadi dua buah *rope* baru R_1 dan R_2. Terdapat sebuah parameter i yang merupakan

posisi indeks *rope* yang akan dibagi. R_1 berisi semua *node* dengan indeks lebih kecil dari parameter. Sedangkan R_2 berisi semua *node* dengan indeks lebih besar sama dengan parameter. Operasi ini memanfaatkan struktur data Pair yang berupa pasangan dari dua *pointer* menuju *node*. *Pseudocode* Fungsi Split ditunjukkan pada Gambar 3.5.

```
split(r, pos)
 1  if !r return make_pair(R1, R2)
 2  idx = getSize(r.left)
 3  if idx < pos
 4      temp = split(r.right, pos − idx − 1)
 5      r.right = temp.first
 6      R2 = temp.second
 7      R1 = r
 8  else
 9      temp = split(r.left, pos)
10      R1 = temp.first
11      r.left = temp.second
12      R2 = r
13  r.update()
14  return make_pair(R1, R2)
```

Gambar 3.5 *Pseudocode* Fungsi Split

3.1.5 Desain Fungsi Concate

Fungsi Concate bertujuan untuk menggabungkan dua buah *rope* menjadi sebuah *rope* utuh. Penggabungan dua buah *rope* R_1 dan R_2 dilakukan dengan menghubungkan *root* R_1 dengan R_2 tergantung pada nilai acak yang didapatkan berdasaarkan hasil modular nilai acak dari kedua *root*. Apabila hasil modular nilai acak lebih kecil

dari berat *root* R_1 maka *rope* R_2 akan menjadi anak kanan dari *rope* yang baru terbentuk. Sebaliknya jika mendapatkan hasil modular nilai acak lebih besar atau sama dengan berat *root* R_1 maka R_2 akan menjadi *root* dari *rope* yang baru terbentuk dan *rope* R_1 menjadi anak kirinya. *Pseudocode* Fungsi Concate ditunjukkan pada Gambar 3.6.

```
concate(R1, R2)
1  if (!R1||!R2) return R1?R1 : R2
2  if (random(R1.size, R2.size))
3       R1.right = concate(R1.right, R2)
4       return R1.update()
5  else
6       R2.left = concate(R1, R2.left)
7       return R2.update()
```

Gambar 3.6 *Pseudocode* Fungsi Concate

3.1.6 Desain Fungsi Insert

Fungsi Insert digunakan untuk memasukkan *string* ke dalam *rope*. *Pseudocode* Fungsi Insert ditunjukkan pada Gambar 3.7.

```
insert(str, pos)
1  x = build(str, str + strlen(str))
2  return concate(r, x)
```

Gambar 3.7 *Pseudocode* Fungsi Insert

3.1.7 Desain Fungsi Print

Fungsi Print bertujuan untuk mendapatkan karakter pada suatu indeks di dalam *rope*. *Pseudocode* Fungsi Print ditunjukkan pada Gambar 3.8.

```
print(r, pos)
1  idx = getSize(r.left)
2  if (idx = pos) return r.value
3  if (pos ⩽ idx) return print(r.left, pos)
4  else return print(r.right, pos − idx − 1)
```

Gambar 3.8 *Pseudocode* Fungsi Print

3.1.8 Desain Fungsi Mutable Begin

Fungsi Mutable Begin bertujuan untuk menggabungkan hasil potongan *rope* pada bagian depan *rope*. Fungsi ini memanfaatkan dua buah operasi *split* dan dua buah operasi *concate*. *Pseudocode* Fungsi Mutable Begin ditunjukkan pada Gambar 3.9.

```
mutable_begin(r, pos, len)
1  fir = split(x, pos)
2  sec = split(fir.second, len)
3  return concate(sec.first, concate(fir.first, sec.second))
```

Gambar 3.9 *Pseudocode* Fungsi Mutable Begin

3.1.9 Desain Fungsi Mutable End

Fungsi Mutable End bertujuan untuk menggabungkan hasil potongan *rope* pada bagian belakang *rope*. Fungsi ini memanfaatkan dua buah operasi *split* dan dua buah operasi *concate*. *Pseudocode* Fungsi Mutable End ditunjukkan pada Gambar 3.10.

```
mutable_end(r, pos, len)
1  fir = split(x, pos)
2  sec = split(fir.second, len)
3  return concate(concate(fir.first, sec.second), sec.first)
```

Gambar 3.10 *Pseudocode* Fungsi Mutable End

Halaman ini sengaja dikosongkan

BAB IV
IMPLEMENTASI

Pada bab ini dijelaskan mengenai implementasi dari desain dan algoritma penyelesaian permasalahan klasik *Alphabetic Rope*.

4.1 Lingkungan Implementasi

Lingkungan implementasi dalam pembuatan Tugas Akhir ini meliputi perangkat keras dan perangkat lunak yang digunakan untuk penyelesaian permasalahan klasik *Alphabetic Rope* adalah sebagai berikut:

1. Perangkat Keras:
    - *Processor* Intel(R) Core(TM) i3-3240 CPU @ 3.40GHz x 4.
    - Memori 6 GB.
2. Perangkat Lunak:
    - Sistem operasi Ubuntu 14.04 LTS 64 bit.
    - *Text editor* Sublime Text 3.
    - *Compiler* g++ versi 4.3.2.

4.2 Rancangan Data

Pada subbab ini dijelaskan mengenai desain data masukan yang diperlukan untuk melakukan proses algoritma, dan data keluaran yang dihasilkan oleh program.

4.2.1 Data Masukan

Data masukan adalah data yang akan diproses oleh program sebagai masukan menggunakan algoritma dan struktur data yang telah dirancang dalam Tugas Akhir ini.
Data masukan berupa berkas teks yang berisi data dengan format yang telah ditentukan pada deskripsi permasalahan klasik *Alphabetic Rope*. Pada masing-masing berkas data masukan, baris pertama berupa sebuah bilangan bulat yang merepresentasikan jumlah kasus uji yang ada pada berkas tersebut. Untuk setiap kasus uji dengan tipe 1 dan 2, masukan berupa sebuah baris masukan yang terdiri dari dua buah parameter posisi berupa x dan y. Sedangkan pada kasus uji dengan tipe 3, masukan berupa sebuah parameter y yang merupakan indeks *string* yang dicari pada konfigurasi *rope* saat ini.

4.2.2 Data Keluaran

Data keluaran yang dihasilkan oleh program hanya berupa satu nilai, yaitu huruf pada indeks ke-y untuk setiap kasus uji dengan tipe 3.

4.3 Implementasi Algoritma

Pada subbab ini akan dijelaskan tentang implementasi proses algoritma secara keseluruhan berdasarkan desain yang telah dijelaskan pada Bab III.

4.3.1 *Header* yang Diperlukan

Implementasi algoritma dengan pemanfaatan struktur data Rope untuk menyelesaikan permasalahan klasik *Alphabetic Rope* membutuhkan empat buah *header* yaitu stdio.h, cstdlib, cstring dan utility, seperti yang terlihat pada Kode Sumber 4.1.

```
1  #include <stdio.h>
2  #include <cstdlib>
3  #include <cstring>
4  #include <utility>
```

Kode Sumber 4.1 *Header* yang diperlukan

Header stdio.h berisi modul untuk menerima masukan dan memberikan keluaran. *Header* cstdlib berisi modul untuk manajemen memori dinamis, generasi bilangan acak, pemilahan dan konversi. *Header* cstring berisi modul yang memiliki fungsi-fungsi untuk melakukan pemrosesan *string*. *Header* utility mencakup berbagai modul yang menyediakan fungsionalitas mulai dari aplikasi perhitungan bit hingga aplikasi fungsi parsial. Contoh implementasinya penggunaan *pair*.

4.3.2 Variabel Global

Variabel global digunakan untuk memudahkan dalam mengakses data yang digunakan lintas fungsi. Kode sumber implementasi variabel global dapat dilihat pada Kode Sumber 4.2.

```
1  using namespace std;
2
3  const int N = 1e5 + 10;
```

Kode Sumber 4.2 Variabel Global

4.3.3 Implementasi Fungsi Main

Fungsi Main adalah implementasi algoritma yang dirancang pada Gambar 3.1. Setiap tipe memiliki operasi yang berbeda-beda. Untuk operasi dengan tipe 3, masukan hanya berupa parameter y yang merupakan nilai posisi indeks yang dicari pada konfigurasi

rope saat ini. Pada operasi dengan tipe 1 dan 2, baris masukan berupa nilai x dan y yang merupakan posisi indeks dari *string* pada *rope*. Setiap masukan dengan tipe 3 akan ditampilkan sebagai jawaban akhir dari permasalahan. Implementasi fungsi Main dapat dilihat pada Kode Sumber 4.3.

```
int main() {
  char st[N];
  scanf("%s", st);
  Node* root = 0;
  int Q, type;
  root = insert(root, st);
  scanf("%d", &Q);
  while ( Q-- ) {
    int x, y;
    scanf("%d", &type);
    if (type == 3) {
      scanf("%d", &y);
      printf("%c\n", print(root, y));
    } else {
      scanf("%d%d", &x, &y);
      if(type == 1) {
        root = mutable_begin(root, x, y - x + 1);
      } else {
        root = mutable_end(root, x, y - x + 1);
      }
    }
  }
  return 0;
}
```

Kode Sumber 4.3 Fungsi Main

4.3.4 Implementasi Struct Node

Fungsi Struct Node berisi atribut yang dimiliki *node* pada *tree*. Implementasi dari fungsi Struct Node dapat dilihat pada Kode Sumber 4.4.

```
1  struct Node {
2    Node *left, *right;
3    int size;
4    char value;
5    Node(char v) {
6      left = right = 0;
7      size = 1;
8      value = v;
9    }
10
11   Node* update() {
12     size = 1;
13     if ( left ) size += left->size;
14     if ( right ) size += right->size;
15     return this;
16   }
17 };
```

Kode Sumber 4.4 Fungsi Struct Node

4.3.5 Implementasi Fungsi Newnode

Fungsi Newnode digunakan untuk membentuk sebuah *node* yang berisikan karakter yang diberikan dan relasi terhadap karakter pada posisi yang bersebelahan. Sehingga membentuk sebuah *tree* seperti pada Gambar 2.4. Implementasi fungsi newnode dapat dilihat pada Kode Sumber 4.5.

```
1  Node* newnode(char c, Node* left, Node* right) {
2    Node* r = new Node(c);
3    r->left = left;
4    r->right = right;
5    r->update();
6    return r;
7  }
```

Kode Sumber 4.5 Fungsi Newnode

4.3.6 Implementasi Fungsi Build

Fungsi Build membangun struktur *tree* dari *rope* yang dilakukan dari karakter yang berada pada posisi indeks di tengah sampai pada karakter paling awal maupun akhir. Setiap *node* akan memiliki anak kiri dan anak kanan jika dan hanya jika masih terdapat *string* yang tersisa. Ilustrasinya dapat dilihat pada Gambar 2.6. Implementasi dari Fungsi Build dapat dilihat pada Kode Sumber 4.6.

```
1  Node* build(char* start, char* end) {
2    if ( start == end ) return NULL;
3    char* mid = start + (end - start)/2;
4    return newnode(*mid, build(start, mid),\
5     build(mid+1, end));
6  }
```

Kode Sumber 4.6 Fungsi Build

4.3.7 Implementasi Fungsi Getsize

Fungsi Getsize digunakan untuk mendapatkan berat *node* yang diperlukan. Implementasi dari fungsi Getsize dapat dilihat pada Kode Sumber 4.7.

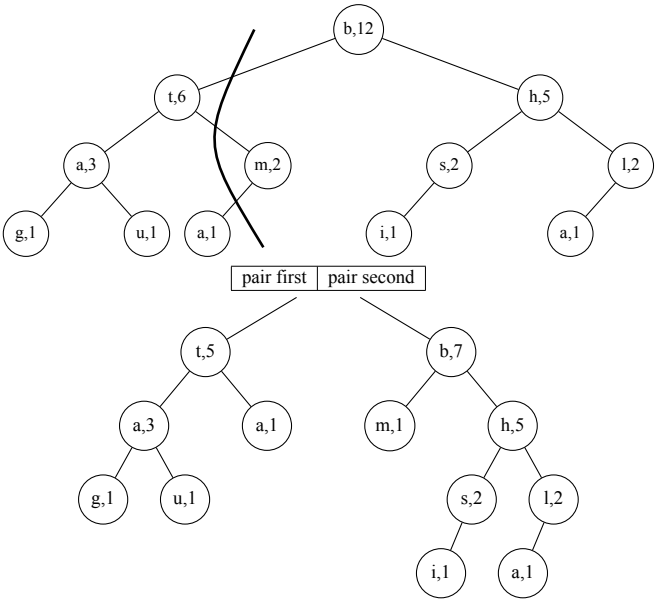
```
1  int getSize(Node* o) {
2    return o ? o->size : 0;
3  }
```

Kode Sumber 4.7 Fungsi Getsize

4.3.8 Implementasi Fungsi Split

Fungsi Split memanfaatkan struktur data Pair yang berupa pasangan dari dua *pointer* menuju *node*. *Pointer node* pertama akan berisi semua *node* yang bernilai lebih kecil dari posisi indeks parameter *split*, sedangkan *pointer node* kedua berisi semua *node* yang bernilai

lebih besar sama dengan posisi indeks parameter *split*.
Misalkan dilakukan *split* pada indeks ke-5 dari *rope* pada Gambar 4.1, *pointer node* pertama akan menyimpan semua *node* dari indeks ke-0 sampai dengan 4. Sedangkan *pointer node* kedua menyimpan *node* dari indeks ke-5 sampai akhir. Implementasi dari fungsi *split* dapat dilihat pada Kode Sumber 4.8 dan 4.9.



Gambar 4.1 Ilustrasi Penyimpanan Hasil Operasi *Split Rope* pada Struktur Data Pair

```
1  pair<Node*, Node*> split(Node* r, int pos) {
2    Node *R1 = 0, *R2 = 0;
3    if ( !r ) return make_pair(R1, R2);
4    int idx = getSize(r->left);
```

Kode Sumber 4.8 Fungsi Split(1)

```
1    if ( idx < pos ) {
2      pair<Node*, Node*> temp =\
3       split(r->right, pos - idx - 1);
4      r->right = temp.first;
5      R2 = temp.second;
6      R1 = r;
7    } else {
8      pair<Node*, Node*> temp = split(r->left, pos);
9      R1 = temp.first;
10     r->left = temp.second;
11     R2 = r;
12   }
13   r->update();
14   return make_pair(R1, R2);
15 }
```

Kode Sumber 4.9 Fungsi Split(2)

4.3.9 Implementasi Fungsi Random

Fungsi Random digunakan untuk menentukan posisi *node* pada saat penggabungan dua buah *rope*. Digunakan fungsi Random agar posisi *node* tidak konstan dan berat suatu *rope* tidak selalu sama. Implementasi fungsi Random ditunjukkan pada Kode Sumber 4.10.

```
1  bool random(int a, int b) {
2    return rand() % ( a + b ) < a;
3  }
```

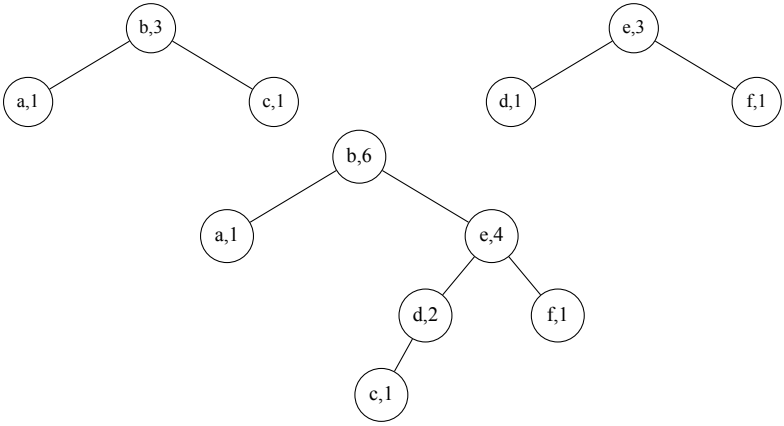
Kode Sumber 4.10 Fungsi Random

4.3.10 Implementasi Fungsi Concate

Fungsi Concate menggabungkan dua buah *rope* menjadi sebuah *rope* utuh. Untuk setiap *node* yang memiliki hasil modular nilai acak lebih kecil dari berat *node root rope* pertama, akan menjadi

root dari *rope* yang baru terbentuk. Untuk *rope* kedua akan menjadi anak kanan atau anak kirinya, tergantung pada urutan BST yang seharusnya dibuat.

Misalkan terdapat dua buah *rope* R_1 yang berisi *string* "abc" dan R_2 yang berisi *string* "def" seperti pada Gambar 4.2. Apabila akan digabungkan dengan R_1 berada di posisi depan sehingga membentuk *string* "$abcdef$", maka semua *node* pada R_2 berada di posisi kanan rope R_1. Jika nilai hasil modular R_1 lebih kecil dari berat *node root* R_1, *node* tersebut akan menjadi *root* dari *rope* yang baru terbentuk. *Node root* R_2 akan menjadi anak kanannya. Jika *root* R_1 memiliki anak kanan, akan menjadi anak kiri dari *node* R_1 yang memiliki indeks 0. Implementasi Fungsi Concate ditunjukkan pada Kode Sumber 4.11 dan 4.12.



Gambar 4.2 Ilustrasi Operasi Concate. Setiap *Node* Berisi Sebuah Karakter dan Nilai Berat Masing-Masing *Node*

```
1  Node* concate(Node* R1, Node* R2) {
2    if ( !R1 || !R2) return R1 ? R1 : R2;
```

Kode Sumber 4.11 Fungsi Concate(1)

```
1    if (random(R1->size, R2->size)) {
2      R1->right = concate(R1->right, R2);
3      return R1->update();
4    } else {
5      R2->left = concate(R1, R2->left);
6      return R2->update();
7    }
8  }
```

Kode Sumber 4.12 Fungsi Concate(2)

4.3.11 Implementasi Fungsi Insert

Fungsi Insert adalah implementasi dari desain algoritma pada Gambar 3.7. Fungsi ini berguna untuk memasukkan data *string* ke dalam *rope*. Setiap *string* akan dimasukkan ke dalam suatu *node*. Setiap *node* hanya berisi oleh satu karakter pecahan dari *string* masukan. Implementasi dari Fungsi Insert dapat dilihat pada Kode Sumber 4.13.

```
1  Node* insert(Node* r, char s[]) {
2    Node* x = build(s, s + strlen(s));
3    return concate(r, x);
4  }
```

Kode Sumber 4.13 Fungsi Insert

4.3.12 Implementasi Fungsi Mutable Begin

Fungsi Mutable Begin adalah implementasi dari desain algoritma pada Gambar 3.9. Operasi ini dilakukan untuk menjawab permasalahan pada subbab 2.1.1 dengan memanfaatkan dua buah operasi Split dan dua buah operasi Concate. Misal dilakukan operasi 1 5 9 pada *rope* di Gambar 4.3. Maka *rope* akan dipotong pada posisi indeks ke-5 menghasilkan *rope* R_1 dan R_2. Kemudian

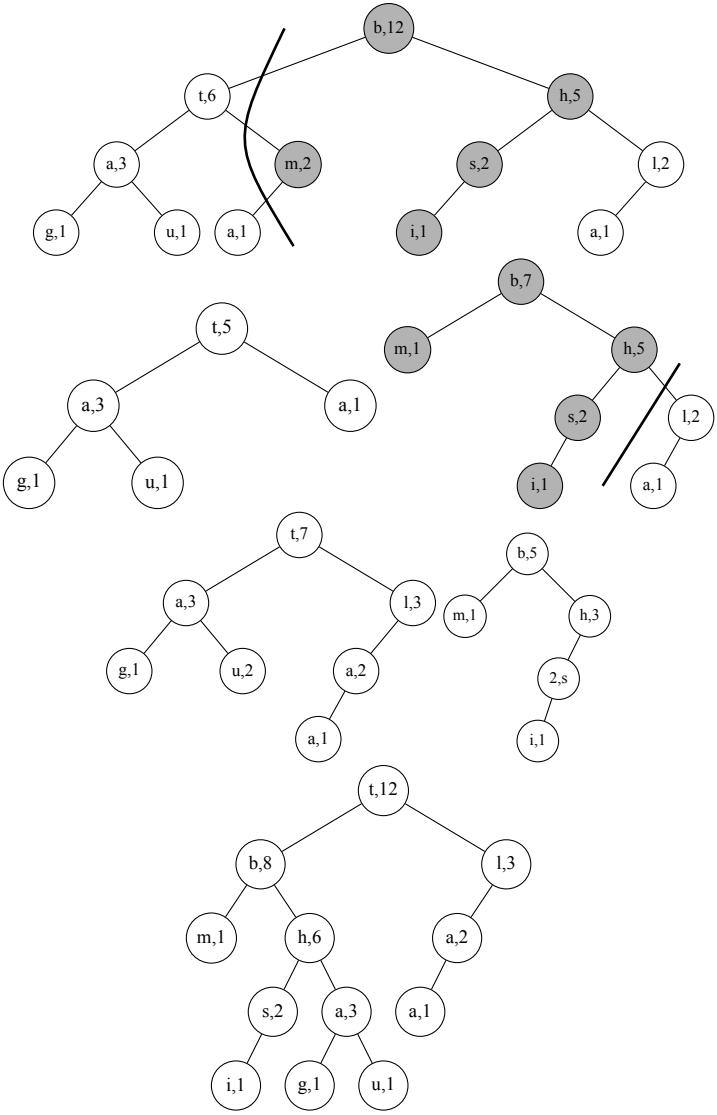
dilakukan *split* kedua kali pada indeks $y - x + 1$ pada *rope* R_2 menghasilkan *rope* R_{21} dan R_{22}. Sehingga menghasilkan tiga buah *rope* yang disimpan dalam struktur data Pair. Langkah selanjutnya dilakukan operasi Concate pada *rope* R_1 dengan R_{22}. Hasil penggabungan *rope* R_1 dengan R_{22} akan digabungkan dengan *rope* R_{21}. Pada operasi ini, *rope* R_{21} akan berada di posisi paling kiri dari keseluruhan *rope*. Dan menghasilkan sebuah *rope* utuh dengan urutan posisi *rope* saat ini adalah R_{21}, R_1 dan R_{22}. Implementasi Fungsi Mutable Begin dapat dilihat pada Kode Sumber 4.14.

```
1  Node* mutable_begin(Node* r, int x, int len) {
2    pair<Node*, Node*> fir = split(r, x);
3    pair<Node*, Node*> sec = split(fir.second, len);
4    return concate(sec.first, concate(fir.first,\
5     sec.second));
6  }
```

Kode Sumber 4.14 Fungsi Mutable Begin

4.3.13 Implementasi Fungsi Mutable End

Fungsi Mutable End dilakukan untuk menjawab permasalahan pada subbab 2.1.2. Implementasinya dilakukan dengan memanfaatkan dua buah operasi Split dan dua buah operasi Concate. Misal dilakukan operasi 2 5 9 pada rope di Gambar 4.4. Maka *rope* akan dipotong pada posisi indeks ke-5 menghasilkan rope R_1 dan R_2. Kemudian dilakukan *split* kedua kali pada indeks $y - x + 1$ pada *rope* R_2 menghasilkan *rope* R_{21} dan R_{22}. Sehingga menghasilkan tiga buah *rope* yang disimpan dalam struktur data Pair. Langkah selanjutnya dilakukan operasi Concate pada *rope* R_1 dengan R_{22}. Hasil penggabungan *rope* R_1 dengan R_{22} akan digabungkan dengan *rope* R_{21}. Pada operasi ini, *rope* R_{21} akan berada di posisi paling kanan dari keseluruhan *rope*. Dan menghasilkan sebuah *rope* utuh dengan urutan posisi *rope* saat ini adalah R_1, R_{22} dan R_{21}. Implementasi Fungsi Mutable End dapat dilihat pada Kode Sumber



Gambar 4.3 Ilustrasi Operasi Mutable Begin. Setiap *Node* Berisi Sebuah Karakter dan Nilai Prioritas Masing-Masing *Node*

4.15.

```
1  Node* mutable_end(Node* r, int x, int len) {
2    pair<Node*, Node*> fir = split(r, x);
3    pair<Node*, Node*> sec = split(fir.second, len);
4    return concate(concate(fir.first, sec.second),\
5     sec.first);
6  }
```

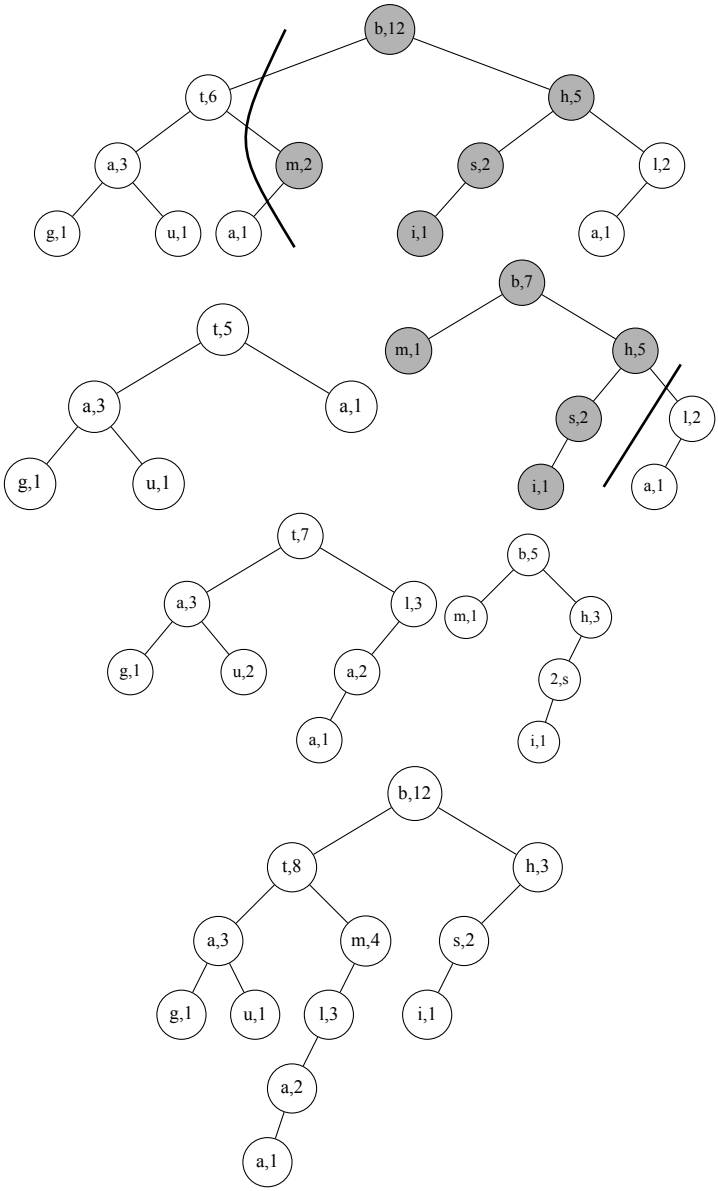
Kode Sumber 4.15 Fungsi Mutable End

4.3.14 Implementasi Fungsi Print

Fungsi Print digunakan untuk menjawab permasalahan klasik *Alphabetic Rope* sesuai dengan permasalahan pada subbab 2.1.3. Implementasi fungsi ini berdasarkan dari desain algorithma pada Gambar 3.8. Implementasi dari Fungsi Print dapat dilihat pada Kode Sumber 4.16.

```
1  char print(Node* r, int pos) {
2    int idx = getSize(r->left);
3    if ( idx == pos ) return r->value;
4    if ( pos <= idx ) return print(r->left, pos);
5    else return print(r->right, pos - idx - 1);
6  }
```

Kode Sumber 4.16 Fungsi Print



Gambar 4.4 Ilustrasi Operasi Mutable End. Setiap *Node* Berisi Sebuah Karakter dan Nilai Prioritas Masing-Masing *Node*

BAB V
UJI COBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan pada Tugas Akhir ini.

5.1 Lingkungan Uji Coba

Linkungan uji coba yang digunakan adalah salah satu sistem yang digunakan situs penilaian daring SPOJ, yaitu kluster *Cube* dengan spesifikasi sebagai berikut:

1. Perangkat Keras:
    - *Processor* Intel(R) Pentium G860 CPU @ 3GHz.
    - *Memory* 1536 MB.
2. Perangkat Lunak:
    - *Compiler* g++ versi 4.3.2.

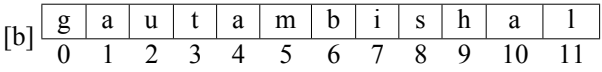
5.2 Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan analisis penyelesaian sebuah contoh kasus menggunakan pendekatan penyelesaian yang telah dijelaskan pada subbab 2.3 serta pengumpulan berkas kode sumber hasil implementasi ke dalam situs penilaian daring SPOJ.
Kasus yang akan digunakan sebagai bahan uji kebenaran dalam analisis penyelesaian permasalahan klasik *Alphabetic Rope* menggunakan contoh kasus pada Gambar 5.1.
Mula-mula dalam setiap permasalahan adalah membangun struktur *rope* yang sesuai dengan *string* masukan. Operasi dasar pada sebuah *rope* adalah fungsi Insert, dimana dibentuk sebuah *tree* yang berfungsi untuk menyimpan potongan karakter pada *rope*.

```
gautambishal
3
3 1
2 0 5
3 1
```

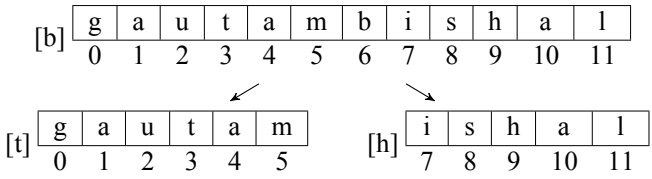
Gambar 5.1 Contoh kasus uji permasalahan Alphabetic Rope

Selanjutnya *rope* akan dibangun pada *node root* yang dilakukan berdasarkan posisi tengah dari panjang *string*. Pembentukan ini dapat dilihat pada Gambar 5.2. Nilai pada tanda kurung siku menyatakan isi karakter pada *node* tersebut. Dikarenakan masih terdapat *string* yang tersisa maka proses pembentukan dilanjutkan untuk membentuk anak kiri dan anak kanan *root*. Rentang *string*

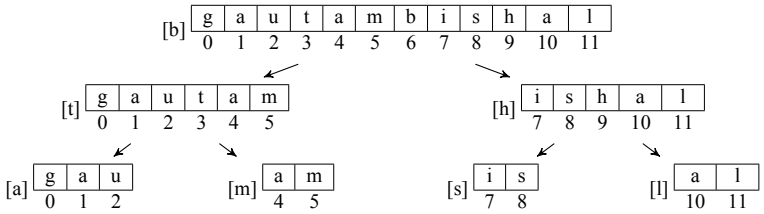


Gambar 5.2 Pembentukan *Root Rope*

pada anak kiri diperoleh dari indeks 0 hingga nilai tengah-1 sedangkan rentang kanan diperoleh dari nilai tengah$+1$ hingga panjang *string*. Pembentukan anak dari *root* ditunjukkan pada Gambar 5.3. Pada setiap anak kiri dan anak kanan, posisi karakter yang berada ditengah-tengah *string* anak menjadi nilai dari *node* tersebut. Yang ditunjukkan oleh nilai di dalam tanda kurung siku.
Dikarenakan masih terdapat *string* yang tersisa, proses pembentukan dilanjutkan untuk tingkat ke-3 yang ditunjukkan pada Gambar 5.4.
Untuk tahap selanjutnya dilakukan pembentukan *rope* pada tingkat ke-4 yang ditunjukkan pada Gambar 5.5.
Query pertama memiliki parameter $type = 3$, $y = 0$. Tahapan pertama dengan mencari karakter pada indeks ke-y. Untuk



Gambar 5.3 Pembentukan *Rope* pada Tingkat ke-2



Gambar 5.4 Pembentukan *Rope* pada Tingkat ke-3

memperoleh jawaban dilakukan penelusuran sesuai dengan algoritma yang dijelaskan pada subbab 2.3.4.

Tahapan pencarian pada *rope* adalah sebagai berikut,

$y = 1$, $left.weight = 6$, $idx = left.weight$; $idx <= y$; dilanjutkan ke anak kiri;

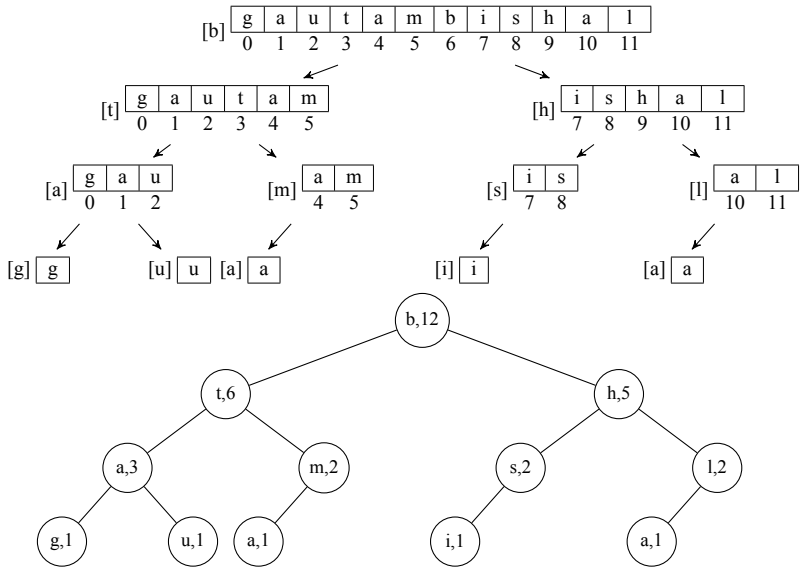
$y = 1$, $left.weight = 3$, $idx = left.weight$; $idx <= y$; dilanjutkan ke anak kiri;

$y = 1$, $left.weight = 1$, $idx = left.weight$; $idx = y$; kembalikan nilai *node* saat ini;

Maka jawaban untuk *query* ini adalah a.

Query kedua memiliki parameter $type = 2$, $x = 0$, $y = 5$. Proses perubahan pada *rope* adalah sebagai berikut,

$x = 0$, $y = 5$, $left.weight = 6$, $idx = left.weight$; $idx <= y$;



Gambar 5.5 Struktur *Rope* yang Terbentuk

dilanjutkan ke anak kiri;

$x = 0, y = 5, left.weight = 3, idx = left.weight; idx <= y;$ dilanjutkan ke anak kiri;

$x = 0, y = 5, left.weight = 1, idx = left.weight; idx <= y;$ dilanjutkan ke anak kiri;

$x = 0, y = 5, left.weight = 0, idx = left.weight; idx = y;$ kembalikan nilai *node* saat ini;

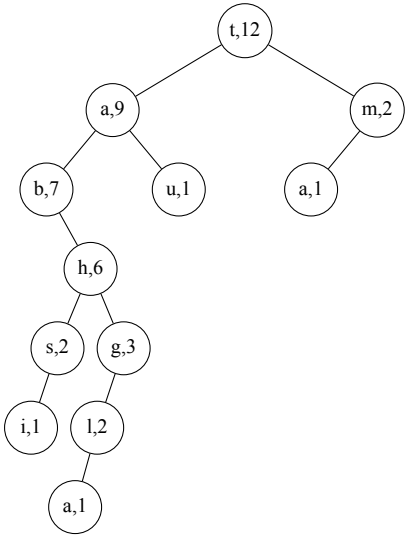
Dikarenakan $x = 0$ operasi Split menghasilkan *rope* R_1 yang berisi NULL dan R_2 sisa dari potongan *rope* yang berarti keseluruhan *rope* saat ini.

Proses perubahan selanjutnya adalah sebagai berikut,

$x = 0$, $y = 5$, $left.weight = 1$, $len = y - x + 1$, $idx = left.weight$; $idx = len$; kembalikan nilai *node* saat ini;

Setelah menemukan posisi *node* yang akan dipotong, hapus sambungan dari seluruh *node* yang memiliki nilai indeks kurang dari nilai indeks saat ini. Hasil potongan dari *rope* akan disimpan dengan menggunakan struktur data Pair yang bertipe *pointer node*. Sehingga menghasilkan dua buah *rope* baru R_{21} yang berisi *string* "$gautam$" dan R_{22} yang berisi *string* "$bishal$".

Potongan *rope* yang baru terbentuk akan digabungkan berdasarkan $type = 2$, dimana R_{21} berada diposisi belakang dari keseluruhan *rope*. Konfigurasi *rope* yang baru terbentuk berisi *string* "$bishalgautam$" yang ditunjukkan pada Gambar 5.6.



Gambar 5.6 Konfigurasi *Rope* Setelah Operasi 2 0 5

Query ketiga memiliki parameter $type = 3$, $y = 0$. Proses perubahan dan pencarian pada *rope* adalah sebagai berikut,

$y = 0$, $left.weight = 9$, $idx = left.weight$; $idx <= y$; dilanjutkan ke anak kiri;

$y = 0$, $left.weight = 7$, $idx = left.weight$; $idx <= y$; dilanjutkan ke anak kiri;

$y = 0$, $left.weight = 0$, $idx = left.weight$; $idx = y$; kembalikan nilai *node* saat ini;

Maka jawaban untuk *query* ini adalah b.

Secara terurut jawaban dari kedua *query* tersebut adalah a dan b. Kemudian sistem penyelesaian dijalankan dan diberi masukan sesuai kasus uji dari analisis sebelumnya dan hasil luaran sistem adalah a dan b seperti terlihat pada Gambar 5.7.

```
gautambishal
3
3 1
a
2 0 5
3 0
b
```

Gambar 5.7 Hasil Luaran Program pada Contoh Kasus Uji Alphabetic Rope

Selanjutnya dilakukan juga uji coba kebenaran dengan mengirimkan kode sumber program ke dalam situs penilaian daring SPOJ. Permasalahan yang diselesaikan adalah permasalahan klasik *Alphabetic Rope*. Hasil uji kebenaran dan waktu eksekusi program pada saat pengumpulan kasus uji pada situs SPOJ ditunjukkan pada Gambar 5.8.

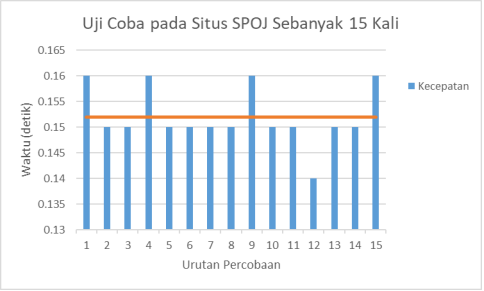
20966212	■	2018-01-12 16:18:01	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2

Gambar 5.8 Hasil Uji Coba pada Situs Penilaian SPOJ

Tabel 5.1 Kecepatan Maksimal, Minimal dan Rata-Rata dari Hasil Uji Coba Sebanyak 15 Kali pada Situs Pengujian SPOJ

Waktu Maksimal	0.16 detik
Waktu Minimal	0.14 detik
Waktu Rata-Rata	0.152 detik
Memori	5.0 MB

Hal ini membuktikan bahwa implementasi yang dilakukan telah berhasil menyelesaikan permasalahan klasik *Alphabetic Rope* dengan batasan-batasan yang telah ditetapkan. Setelah itu dilakukan pengiriman kode sumber implementasi sebanyak 15 kali untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba sebanyak 15 kali dapat dilihat pada Gambar A.1. Grafik hasil uji coba sebanyak 15 kali ditunjukkan pada Gambar 5.9.



Gambar 5.9 Hasil Uji Coba pada Situs Penilaian SPOJ

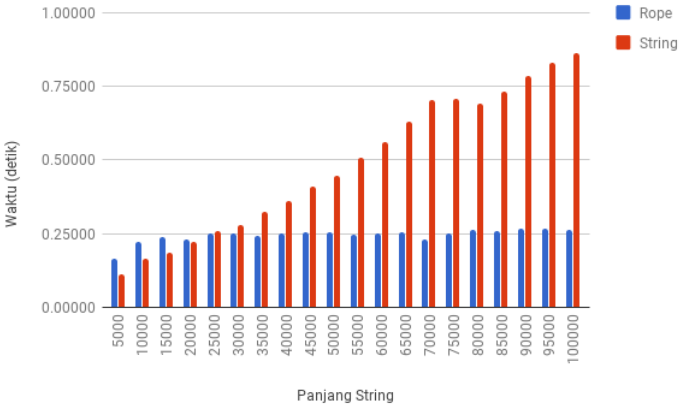
Berdasarkan Tabel 5.1 dari percobaan pengujian yang dilakukan, didapat waktu rata-rata program yaitu 0.152 detik dan penggunaan memori yang dibutuhkan program yaitu 5.0 MB. Hasil ini masih dibawah dari batas maksimal waktu dan memori pada permasalahan klasik *Alphabetic Rope*, yaitu 1 detik dan 1536 MB.

5.3 Uji Coba Kinerja

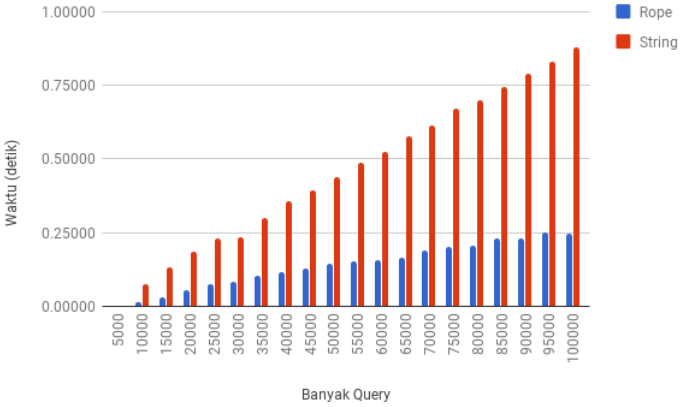
Uji coba kinerja penyelesaian permasalahan klasik *Alphabetic Rope* dilakukan dengan cara membandingkan waktu yang dibutuhkan untuk penyelesaian menggunakan *string* dengan struktur data Rope yang dibuat.

5.3.1 Operasi 1 Menggabungkan *Rope* pada Posisi Awal

Pada Gambar 5.10 terlihat bahwa untuk $N = 5000$ sampai $N = 10^5$ dengan banyak $Q = 10^5$, struktur Rope yang dibuat untuk penyelesaian permasalahan klasik *Alphabetic Rope* menunjukkan hasil yang sangat signifikan dimana pertumbuhan waktu datar dibandingkan dengan algoritma String yang memiliki kompleksitas $O(N)$. Sedangkan untuk $N = 10^5$ dengan $Q = 5000$ sampai $Q = 10^5$, pertumbuhan waktu secara logaritmik dan lebih kecil dibandingkan dengan algoritma String yang ditunjukkan pada Gambar 5.11. Untuk tabel performa antara algoritma String dengan struktur data Rope yang telah dibuat dapat dilihat pada Tabel B.1 dan B.2.



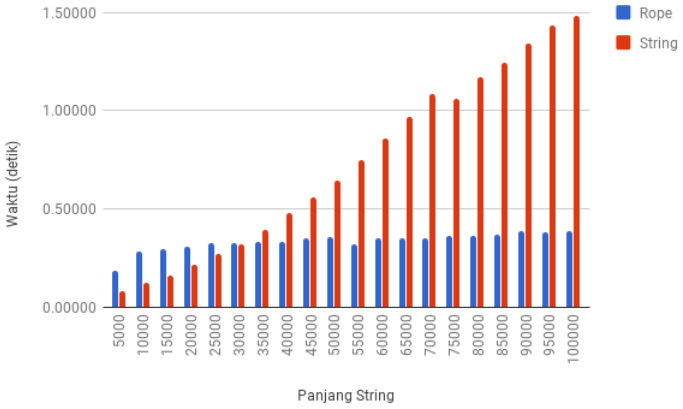
Gambar 5.10 Hasil Uji Coba pada Operasi 1 dengan Jumlah *Query* Tetap dan Panjang *String* Bertambah



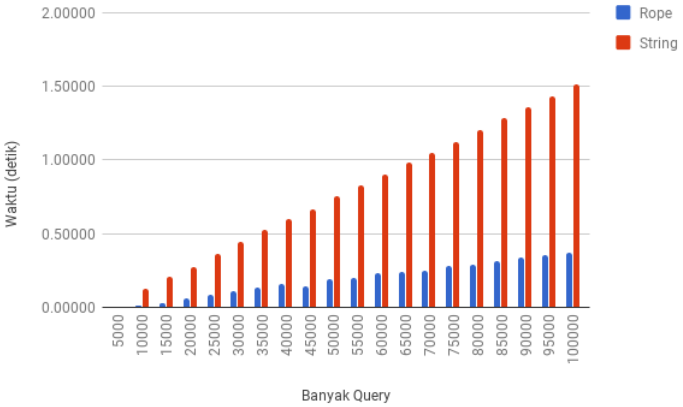
Gambar 5.11 Hasil Uji Coba pada Operasi 1 dengan Jumlah *String* Tetap dan Jumlah *Query* Bertambah

5.3.2 Operasi 2 Menggabungkan *Rope* pada Posisi Akhir

Pada Gambar 5.12 terlihat bahwa untuk $N = 5000$ sampai $N = 10^5$ dengan banyak $Q = 10^5$, struktur data Rope yang dibuat untuk penyelesaian permasalahan klasik *Alphabetic Rope* menunjukkan hasil yang sangat signifikan dimana pertumbuhan waktu datar dibandingkan dengan algoritma String yang memiliki kompleksitas $O(N)$. Sedangkan untuk $N = 10^5$ dengan $Q = 5000$ sampai $Q = 10^5$, pertumbuhan waktu secara logaritmik dan lebih kecil dibandingkan dengan algoritma String yang ditunjukkan pada Gambar 5.13. Untuk tabel performa antara algoritma String dengan struktur data Rope yang telah dibuat dapat dilihat pada Tabel C.1 dan C.2.



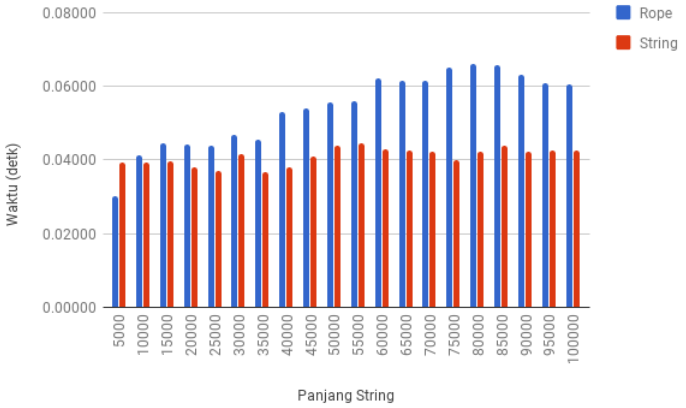
Gambar 5.12 Hasil Uji Coba pada Operasi 2 dengan Jumlah *Query* Tetap dan Jumlah *String* Bertambah



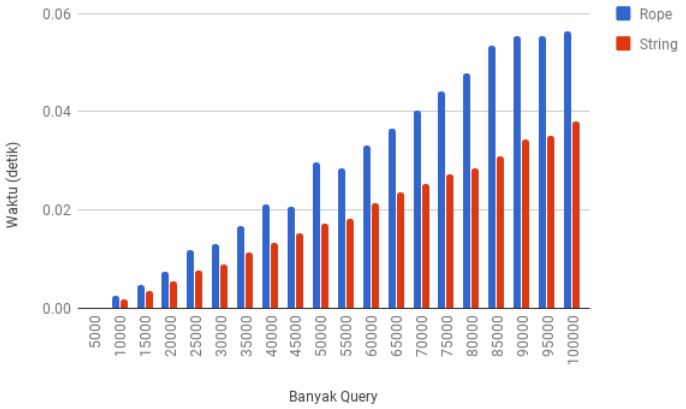
Gambar 5.13 Hasil Uji Coba pada Operasi 2 dengan Panjang *String* Tetap dan Jumlah *Query* Bertambah

5.3.3 Operasi 3 Mencetak Karakter pada Indeks ke-Y

Pada Gambar 5.14 terlihat bahwa untuk $N = 5000$ sampai $N = 10^5$ dengan banyak $Q = 10^5$, struktur data Rope yang dibuat untuk penyelesaian permasalahan klasik *Alphabetic Rope* menunjukkan pertumbuhan waktu secara logaritmik dibandingkan dengan algoritma String yang memiliki pertumbuhan datar dimana kompleksitas sebesar $O(1)$. Sedangkan untuk $N = 10^5$ dengan $Q = 5000$ sampai $Q = 10^5$, pertumbuhan waktu secara logaritmik dibandingkan dengan algoritma String yang memiliki pertumbuhan datar yang ditunjukkan pada Gambar 5.15. Untuk tabel performa antara algoritma String dengan struktur data Rope yang telah dibuat dapat dilihat pada Tabel D.1 dan D.2.



Gambar 5.14 Hasil Uji Coba pada Operasi 3 dengan Jumlah *Query* Tetap dan Panjang *String* Bertambah



Gambar 5.15 Hasil Uji Coba pada Operasi 3 dengan Jumlah *String* Tetap dan Jumlah *Query* Bertambah

5.4 Analisis Hasil Uji Coba

Pada subbab ini akan dibahas mengenai analisis hasil uji coba yang dikerjakan pada subbab 5.2. Berdasarkan ketiga skenario yang telah dilakukan, masing-masing membuktikan kebenaran dan efisiensi hasil implementasi Tugas Akhir ini.

Dari ketiga hasil uji coba yang telah dilakukan, dapat dilihat bahwa hasil implementasi algoritma penyelesaian permasalahan klasik *Alphabetic Rope* pada Tugas Akhir ini mengeluarkan hasil keluaran yang sama dengan jalur yang ditelusuri secara manual. Hasil uji coba kebenaran pada subbab 5.3 dapat digunakan sebagai acuan bahwa hasil keluaran program akan benar untuk segala jenis operasi pertanyaan dan perubahan.

Selanjutnya berdasarkan uji coba performa yang ditunjukkan pada Gambar A.1, dapat dilihat bahwa memori dan waktu yang dibutuhkan untuk mengeksekusi program dapat dikategorikan efisien menurut situs SPOJ. Kemudian pada subbab 5.3 dapat dilihat bahwa kecepatan waktu eksekusi program dipengaruhi oleh panjang *string*. Semakin bertambah panjang *string*, maka semakin lama waktu yang dibutuhkan untuk mengeksekusi program. Kompleksitas waktu untuk algoritma komputasi *string* pada Tugas Akhir ini adalah $O(\log N)$, dimana N merupakan panjang *string*.

Pada hasil perbandingan antara algoritma String dengan struktur Rope yang digunakan pada Tugas Akhir ini, ditunjukkan perbedaan yang signifikan pada kecepatan eksekusi program. Hal ini menunjukkan bahwa algoritma pada Tugas Akhir ini lebih efisien daripada algoritma yang telah digunakan sebelumnya.

Halaman ini sengaja dikosongkan

BAB VI
KESIMPULAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap perancangan dan implementasi algoritma untuk menyelesaikan permasalahan klasik *Alphabetic Rope* dapat diambil kesimpulan sebagai berikut:

1. Implementasi algoritma dengan menggunakan struktur data Rope dapat menyelesaikan permasalahan klasik *Alphabetic Rope* dengan benar dan cepat.
2. Kompleksitas waktu sebesar $O(\log N)$ dapat menyelesaikan permasalahan klasik *Alphabetic Rope*.
3. Waktu yang dibutuhkan program untuk menyelesaikan permasalahan klasik *Alphabetic Rope* minimum 0.14 detik, maksimum 0.16 detik dan rata-rata 0.152 detik. Memori yang dibutuhkan sebesar 5.0 MB.
4. Struktur data Rope yang dibuat sangat baik diaplikasikan untuk melakukan komputasi *string* yang panjang.

6.2 Saran

Pada Tugas Akhir kali ini tentunya terdapat kekurangan serta nilai-nilai yang dapat penulis ambil. Berikut adalah saran-saran yang dapat diambil melalui Tugas Akhir ini:

1. Struktur data Rope adalah pendekatan yang sesuai untuk menyelesaikan permasalahan *string* yang sangat panjang.

2. Struktur data Rope dapat diimplementasikan pada bahasa pemrograman lain.

DAFTAR PUSTAKA

[1] **AROPE** - Alphabetic Rope [Online]. Available: http://www.spoj.com/problems/AROPE/. [Accessed 24-October-2017].

[2] **Advance Programming with C++**. Available: http://www.acsu.buffalo.edu/ fineberg/mfc158/week10lecture.htm. [Accessed 24-October-2017].

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, **Introduction To Algorithm,** 2nd ed. Cambridge, Massachusetts London, England: The MIT Press, 2001.

[4] Cecilia R. Aragon, **Randomize Search Trees**, USA: Foundation of Computer Science, 1989.

[5] Hans-J. Boehm, Russ Atkinson and Michael Pass, **Ropes: an Alternative to String**, California: Xeroc PARC, 1994.

Halaman ini sengaja dikosongkan

LAMPIRAN A
HASIL UJI COBA KEBENARAN PADA SITUS SPOJ

20966263	☐	2018-01-11 16:22:03	Alphabetic Rope	**accepted** edit ideone it	0.16	5.0M	C++ 4.3.2
20966258	☐	2018-01-11 16:21:46	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966253	☐	2018-01-11 16:21:28	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966248	☐	2018-01-11 16:21:08	Alphabetic Rope	**accepted** edit ideone it	0.16	5.0M	C++ 4.3.2
20966241	☐	2018-01-11 16:20:52	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966240	☐	2018-01-11 16:20:35	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966238	☐	2018-01-11 16:20:18	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966236	☐	2018-01-11 16:20:01	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966231	☐	2018-01-11 16:19:44	Alphabetic Rope	**accepted** edit ideone it	0.16	5.0M	C++ 4.3.2
20966227	☐	2018-01-11 16:19:26	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966224	☐	2018-01-11 16:19:08	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966220	☐	2018-01-11 16:18:45	Alphabetic Rope	**accepted** edit ideone it	0.14	5.0M	C++ 4.3.2
20966218	☐	2018-01-11 16:18:25	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966212	☐	2018-01-11 16:18:03	Alphabetic Rope	**accepted** edit ideone it	0.15	5.0M	C++ 4.3.2
20966209	☐	2018-01-11 16:17:45	Alphabetic Rope	**accepted** edit ideone it	0.16	5.0M	C++ 4.3.2

Gambar A.1 Hasil Pengujian Sebanyak 15 Kali pada Situs Penilaian Daring SPOJ

Halaman ini sengaja dikosongkan

LAMPIRAN B

HASIL UJI PERFORMA PENYELESAIAN OPERASI 1 MENGGUNAKAN STRUKTUR DATA ROPE DAN ALGORITHMA STRING

Tabel B.1 Tabel Hasil Uji Coba pada Operasi 1 dengan Jumlah *Query* tetap dan Panjang *String* Bertambah

Panjang String	Rope Waktu (detik)	String Waktu (detik)
5000	0.22297	0.16508
10000	0.23635	0.18649
15000	0.23204	0.22387
20000	0.24874	0.25948
25000	0.24952	0.27861
30000	0.24111	0.32174
35000	0.24973	0.35983
40000	0.25347	0.40759
45000	0.25554	0.44766
50000	0.24770	0.50785
55000	0.24874	0.56113
60000	0.25410	0.62867
65000	0.22816	0.70070
70000	0.25131	0.70681
75000	0.26446	0.69239
80000	0.25786	0.73221
85000	0.26865	0.78513
90000	0.26531	0.83095
95000	0.26228	0.86237
100000	0.26590	0.91585

Tabel B.2 Tabel Hasil Uji Coba pada Operasi 1 dengan Panjang *String* tetap dan Jumlah *Query* Bertambah

Banyak Query	Rope Waktu (detik)	String Waktu (detik)
5000	0.01367	0.07475
10000	0.03069	0.13296
15000	0.05544	0.18666
20000	0.07516	0.23203
25000	0.08385	0.23522
30000	0.10193	0.29990
35000	0.11545	0.35637
40000	0.12880	0.39486
45000	0.14614	0.43901
50000	0.15392	0.48696
55000	0.15846	0.52213
60000	0.16648	0.57634
65000	0.18748	0.61121
70000	0.20337	0.66826
75000	0.20561	0.70057
80000	0.22864	0.74330
85000	0.22859	0.78668
90000	0.24984	0.82813
95000	0.24628	0.87657
100000	0.26590	0.91585

LAMPIRAN C

HASIL UJI PERFORMA PENYELESAIAN OPERASI 2 MENGGUNAKAN STRUKTUR DATA ROPE DAN STRING

Tabel C.1 Tabel Hasil Uji Coba pada Operasi 2 dengan Jumlah *Query* tetap dan Panjang *String* Bertambah

Panjang String	Rope Waktu (detik)	String Waktu (detik)
5000	0.28377	0.12619
10000	0.29701	0.16408
15000	0.30636	0.21687
20000	0.32590	0.27467
25000	0.32970	0.32248
30000	0.33190	0.39326
35000	0.33405	0.47976
40000	0.34860	0.55986
45000	0.35496	0.64512
50000	0.32062	0.74809
55000	0.34872	0.85830
60000	0.35328	0.96547
65000	0.35061	1.08336
70000	0.36411	1.06059
75000	0.36469	1.17042
80000	0.37024	1.24225
85000	0.38693	1.34229
90000	0.37870	1.43055
95000	0.38947	1.48104
100000	0.38388	1.59755

Tabel C.2 Tabel Hasil Uji Coba pada Operasi 2 dengan Panjang String tetap dan Jumlah Query Bertambah

Banyak Query	Rope Waktu (detik)	String Waktu (detik)
5000	0.01140	0.12558
10000	0.02674	0.20645
15000	0.06116	0.26995
20000	0.08433	0.36020
25000	0.10924	0.44044
30000	0.13237	0.52400
35000	0.15651	0.59788
40000	0.14079	0.66545
45000	0.18880	0.75032
50000	0.19879	0.82605
55000	0.23276	0.89987
60000	0.23973	0.97796
65000	0.24871	1.04965
70000	0.27870	1.11826
75000	0.29287	1.20108
80000	0.31036	1.28571
85000	0.33650	1.35636
90000	0.35154	1.42740
95000	0.36962	1.51285
100000	0.38388	1.59755

LAMPIRAN D

HASIL UJI PERFORMA PENYELESAIAN OPERASI 3 MENGGUNAKAN STRUKTUR DATA ROPE DAN STRING

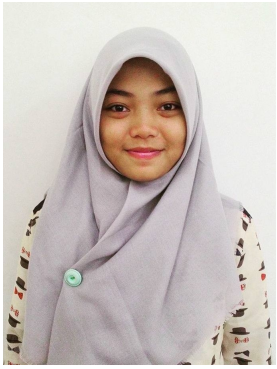
Tabel D.1 Tabel Hasil Uji Coba pada Operasi 3 dengan Jumlah *Query* tetap dan Panjang *String* Bertambah

Panjang String	Rope Waktu (detik)	String Waktu (detik)
5000	0.04487	0.05499
10000	0.05078	0.05819
15000	0.05559	0.06057
20000	0.05350	0.06055
25000	0.05431	0.05590
30000	0.05741	0.05428
35000	0.05760	0.05122
40000	0.06101	0.05246
45000	0.06243	0.05184
50000	0.06065	0.05343
55000	0.06470	0.05183
60000	0.06242	0.05171
65000	0.06509	0.05166
70000	0.06531	0.04990
75000	0.06700	0.04985
80000	0.07001	0.04822
85000	0.06970	0.04795
90000	0.07264	0.04603
95000	0.07143	0.05049
100000	0.06792	0.05197

Tabel D.2 Tabel Hasil Uji Coba pada Operasi 3 dengan Panjang String tetap dan Jumlah Query Bertambah

Banyak Query	Rope Waktu (detik)	String Waktu (detik)
5000	0.00316	0.00183
10000	0.00541	0.00352
15000	0.00777	0.00556
20000	0.01129	0.00776
25000	0.01530	0.01007
30000	0.01860	0.01201
35000	0.01927	0.01516
40000	0.02438	0.01903
45000	0.02705	0.02074
50000	0.02978	0.02290
55000	0.03049	0.02688
60000	0.04021	0.02632
65000	0.03843	0.03110
70000	0.04672	0.03292
75000	0.04813	0.03941
80000	0.05481	0.03636
85000	0.05957	0.04345
90000	0.06257	0.04330
95000	0.06671	0.04974
100000	0.06792	0.05197

BIODATA PENULIS



Desy Nurbaiti Rahmi, lahir di Banyuwangi tanggal 9 Desember 1995. Penulis merupakan anak ketiga dari 3 bersaudara. Penulis telah menempuh pendidikan formal TK Aisyiyah I Banyuwangi, SD Negeri 1 Kebalenan (2002-2008), SMP Negeri 1 Banyuwangi (2008-2011) dan SMA Negeri 1 Glagah (2011-2014). Penulis melanjutkan studi kuliah program sarjana di Jurusan Teknik Informatika ITS.

Selama kuliah di Teknik Informatika ITS, penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Sistem Operasi (2016 dan 2017), Jaringan Komputer(2016). Selama menempuh perkuliahan penulis juga aktif di kegiatan organisasi dan kepanitiaan diantaranya menjadi staff Departemen Pengembangan Profesi HMTC ITS, wakil dua Bendahara Schematics 2015, Bendahara Schematics 2015 dan panitia Pemusatan Latihan Nasional 2 TOKI 2017 di ITS. Penulis dapat dihubungi melalui surel di
desyrahmi09@gmail.com.