Implementasi Struktur Data Rope pada Studi Kasus Permasalahan SPOJ Alphabetic Rope

Desy Nurbaiti Rahmi, Rully Soelaiman dan Abdul Munif
Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember (ITS)
e-mail: munif@if.its.ac.id

***Abstrak*—Permasalahan *alphabetic rope* merupakan sebuah permasalahan yang melibatkan sebuah rentang pencarian/*query*. 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, dibutuhkan struktur data yang mampu mendukung operasi-operasi tersebut dengan efisien. Pada penelitian ini 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 penelitian ini digunakan struktur data Rope untuk menyelesaikan tipe-tipe operasi yang diberikan. Algoritma yang dirancang dapat menyelesaikan permasalahan yang diberikan dengan benar dan memiliki pertumbuhan waktu secara logaritmik dengan kompleksitas waktu sebesar *O*(log *N*) per query.**

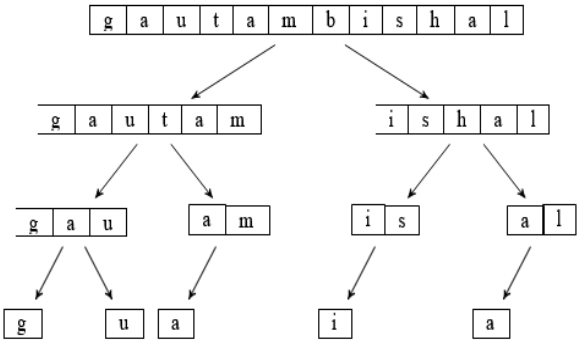
***Kata Kunci*—*concatenation*, *rope*, *split*, *string*.**

I. PENDAHULUAN

ALPHABETIC Rope [1] merupakan rangkaian karakter berupa huruf kecil yang terlihat seperti serangkaian *string*. Rangkaian karakter tersebut akan dilakukan beberapa operasi sesuai dengan tipenya. Diberikan sejumlah data berupa *string*, permasalahan *alphabetic rope* digunakan untuk mencari sebuah huruf pada indeks tertentu dalam rentang. Diberikan sebuah urutan *string* $R = \{S_0, S_1, ..., S_N\}$ dan beberapa operasi atas urutan tersebut. Penelitian ini akan membahas tiga rangkaian operasi, dua operasi tersebut merupakan operasi pertanyaan dan perubahan, sedangkan operasi terakhir adalah operasi pencarian.

A. Bagian 1

Diberikan tiga buah parameter berupa *type*, *x*, dan *y*. Dilakukan operasi perubahan pada indeks ke-*x* sampai *y* pada urutan *string* yang dimiliki. Urutan *string* pada indeks ke-*x* sampai *y* akan dipotong dan digabungkan pada bagian depan *rope*. Jika terdapat *query type* = 1, *x* = 4 dan *y* = 7 pada urutan *string* *R = {g, a, u, t, a, m, b, i, s, h, a, l}*, maka karakter pada indeks ke-4 sampai 7 akan dipotong dan digabungkan pada bagian awal dari urutan *string* yang dimiliki. Sehingga urutan *string* saat ini menjadi *R = {a, m, b, i, g, a, u, t, s, h, a, l}*.



Gambar 1. Struktur *rope* berdasarkan *string "gautambishal"*.

B. Bagian 2.

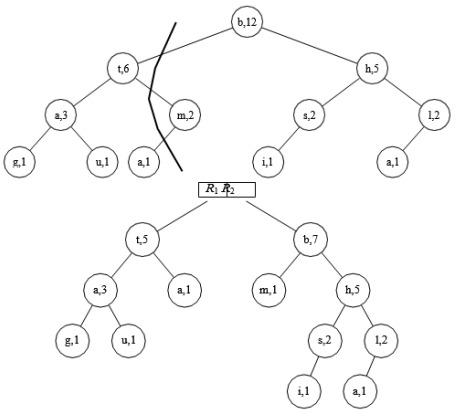
Diberikan tiga buah parameter berupa *type*, *x*, dan dilakukan operasi perubahan pada indeks ke-*x* sampai *y* pada urutan *string* yang dimiliki. Urutan *string* pada indeks ke-*x* sampai *y* akan dipotong dan digabungkan pada bagian belakang *rope*. Jika terdapat *query type*=2, *x* = 3 dan *y* = 6 pada urutan *string* *R = {g, a, u, t, a, m, b, i, s, h, a, l}*, maka karakter pada indeks ke-3 sampai 6 akan dipotong dan digabungkan pada bagian akhir dari urutan *string* yang dimiliki. Sehingga urutan *string* saat ini menjadi *R = {g, a, u, i, s, h, a, l, t, a, m, b}*.

C. Bagian 3

Diberikan dua buah parameter berupa *type* dan *y*. Operasi ini berbeda dari bagian 1 dan bagian 2. Dari kedua parameter ini, akan dilakukan pencarian berupa karakter pada indeks ke-*y* dari urutan *string* yang dimiliki. Jika terdapat *query type* = 3 dan *y* = 5 pada urutan *string* *R = {g, a, u, t, a, m, b, i, s, h, a, l}*, maka akan dicari karakter pada indeks ke-5. Maka hasil pencarian pada indeks ke-5 adalah *ans* = *m*. Solusi yang ada untuk permasalahan ini biasanya menggunakan struktur data String. Pencarian rentang indeks dilakukan dengan sangat cepat, memerlukan kompleksitas sekitar $O(1)$. Akan tetapi, pada operasi pemotongan dan penggabungan karakter memerlukan kompleksitas waktu yang sangat besar untuk setiap operasinya. Dengan kompleksitas waktu sebesar $O(n)$, dapat menyebabkan penyelesaian pekerjaan menjadi tidak efisien dan memberikan dampak yang signifikan terhadap kinerjanya. Saat ini, sebuah struktur data baru yang disebut Rope diusulkan untuk mendukung operasi penggabungan karakter dengan efisiensi waktu dan memori [2]. Karena penggabungan tidak menyalin argumennya, alternatif alaminya adalah dengan mewakili string sebagai sebuah *ordered tree* dimana setiap *node*-nya menyimpan karakter-karakternya. Dengan demikian, rangkaian *string* yang

```
SPLIT(r, pos)
1  if (!r) return make_pair(R1, R2)
2  idx ← GETSIZE(r.left)
3  if idx < pos
4     then
5          temp = SPLIT(r.right, pos − idx − 1)
6          r.right = temp.first
7          R2 = temp.second
8          R1 = r
9     else
10         temp = SPLIT(r.left, pos)
11         R1 = temp.first r.left
12         t = temp.second R2
13         = r
14 r.UPDATE()
15 return make_pair(R1, R2)
```

Gambar 2. *Pseudocode* Fungsi Split.



Gambar 3. Ilustrasi penyimpanan hasil operasi *split rope* pada struktur data pair.

diwakilkan dalam sebuah *tree* adalah gabungan urutan dari anak kiri sampai kanannya. Pembuatan sebuah *rope* berdasarkan sebuah urutan *string* sepanjang *N* memiliki kompleksitas waktu sebesar *O*(log *N*). Tujuan penelitian ini adalah untuk menyelesaikan permasalahan *alphabetic rope* secara optimal dan menguji performa dari struktur data Rope yang telah dibangun.

II. METODE PENYELESAIAN

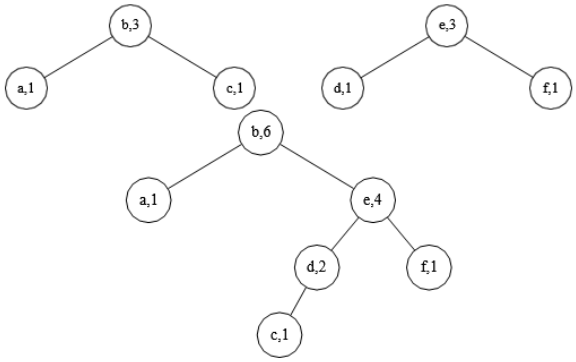
A. Struktur Data Rope

Struktur data Rope adalah sebuah struktur berbentuk *binary tree* yang secara rekursif membagi dua panjang urutan *string* yang diberikan. Setiap *node* berisi sebuah karakter dengan jumlah berat masing-masingnya. Setiap *parent node* akan menyimpan berat anak kanan dan anak kirinya serta beratnya sendiri. Untuk membuat struktur ini, dimulai dari *root node* yang menyimpan karakter pada posisi tengah dari panjang indeks *string* yang diberikan. Ketika berlanjut ke partisi anak kiri dan kanannya, *string* akan terbagi menjadi dua himpunan bagian dan berulang sampai tidak ada lagi *string* yang tersisa. Dengan cara ini, akan didapatkan sebuah struktur *complete binary tree* yang memiliki kedalaman *O*(log *N*).

Berdasarkan string berikut, *R = {g, a, u, t, a, m, b, i, s, h, a, l}* akan dibangun sebuah *rope*. *Root* dari *rope* yang terbentuk akan berisi karakter *b*. Seperti yang dijelaskan sebelumnya, panjang *string* akan dibagi menjadi dua (hampir) sama rata. Hal ini diperlukan untuk mendapatkan *rope* dengan ketinggian *O*(log *N*). Kemudian membangun anak kiri dan anak kanannya berdasarkan potongan pertama dan kedua dari masing-masing himpunan bagian. Anak kirinya *L* akan berisi karakter pada indeks tengah dari *string {g, a, u, t, a, m}* dan anak kanannya *R* berisi karakter pada indeks tengah dari *string {i, s, h, a, l}*. Proses ini terus berlanjut sampai tidak ada lagi *string* yang tersisa. Gambar 1 menunjukkan struktur *rope* berdasarkan *string R*.

```
CONCATE(R1, R2)
1  if (!R1||!R2) return R1?R1 : R2
2  if RANDOM(R1.size, R2.size)
3     then
4          R1.right = CONCATE(R1.right, R2)
5          return R1.UPDATE()
6     else
7          R2.left = CONCATE(R1, R2.left)
8          return R2.UPDATE()
```

Gambar 4. *Pseudocode* Fungsi Concate.



Gambar 5. Ilustrasi operasi concate. Setiap *node* berisi sebuah karakter dan nilai berat masing-masing *node.*

```
INDEX(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 6. *Pseudocode* Fungsi Index.

B. Split

Untuk menjawab pertanyaan dari Bagian 1 dan Bagian 2, diperlukan sebuah operasi pemotongan segmen *rope* yang dimiliki saat ini. Operasi ini disebut operasi Split. Operasi Split (*R, i*) akan membagi *rope R* menjadi dua buah *rope* baru R_1 dan R_2, dimana $R_1 = S_0, ..., S_{i-1} dan R_2 = S_i, ..., S_N$ yang akan disimpan menggunakan sebuah struktur data Pair menuju *node*.

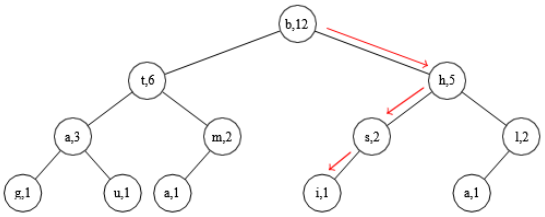
Pseudocode fungsi Split ditunjukkan pada Gambar 2. Gambar 3 menunjukkan pemotongan *rope* pada *i* = 5 yang kemudian disimpan ke dalam sebuah struktur data Pair menuju *node*.

C. Concatenate

Operasi lain yang dapat membantu menjawab pertanyaan dari Bagian 1 dan Bagian 2 adalah sebuah operasi penggabungan dua buah *rope* menjadi sebuah *rope* utuh. Operasi ini disebut operasi Concate. Operasi Concate (R_1, R_2) akan menggabungkan kedua *rope*. Penggabungan dilakukan dengan menghubungkan *root* pada masing-masing *rope*. Hal ini dilakukan berdasarkan hasil modular nilai acak yang didapatkan dari kedua *root* dari *rope*. Apabila hasil modular nilai acak lebih kecil dari berat *root* R_1, maka *root* R_1 akan menjadi *root* dari *rope* yang baru terbentuk. Sebaliknya jika mendapat hasil modular nilai acak lebih besar dari berat *root* R_1 maka *root* R_2 yang menjadi *root* dari *rope* yang baru terbentuk. *Pseudocode* fungsi Concate ditunjukkan pada Gambar 4. Gambar 5 menunjukkan penggabungan dua buah *rope* R_1 = *{a, b, c}* dan R_2 = *{d, e, f}* menjadi sebuah *rope* utuh.

Tabel 1.
Kecepatan maksimal, minimal dan rata-rata dari hasil uji coba pengumpulan 15 kali pada situs pengujian daring SPOJ

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



Gambar 7. Mencari karakter pada indeks ke-*i*=7.

D. Index

Permasalahan pada Bagian 3 memerlukan sebuah pencarian karakter dalam rentang *rope*. Dengan memanfaatkan operasi Index, didapatkan sebuah karakter dari *rope* pada indeks tertentu. Operasi Index (*R, i*) akan menghasilkan karakter pada indeks ke-*i* dari *rope R*.

Pseudocode fungsi Index ditunjukkan pada Gambar 6. Gambar 7 menunjukkan hasil pencarian pada indeks ke-*i* = 7 dari *rope R = {g, a, u, t, a, m, b, i, s, h, a, l}*.

III. UJI COBA DAN ANALISIS

A. Uji coba kebenaran

Uji coba kebenaran dilakukan dengan mengumpulkan berkas kode sumber hasil implementasi ke situs sistem penilaian daring SPOJ kali. Permasalahan yang diselesaikan adalah Alphabetic Rope dengan kode AROPE. Hasil uji kebenaran dan waktu eksekusi program saat pengumpulan kasus uji pada situs SPOJ ditunjukkan pada Gambar 8.

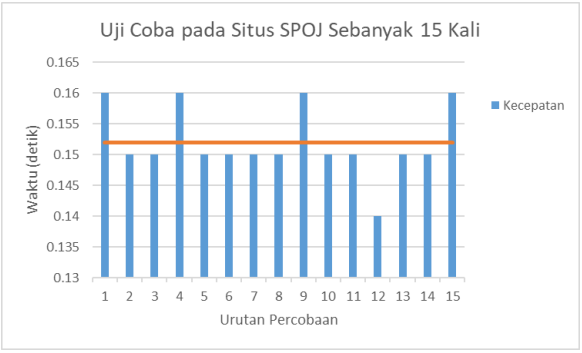
Berikutnya adalah pengujian performa dari algoritma yang dirancang dan diimplementasi dengan melakukan uji *submission* dengan mengumpulkan berkas kode implementasi dari algoritma yang dibangun sebanyak 15 kali ke situs penilaian daring SPOJ dengan mencatat waktu eksekusi serta memori yang dibutuhkan. Hasil dari pengujian dapat dilihat pada grafik pada Gambar 9 dan Tabel 1.

B. Uji Coba Kinerja

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

1. Operasi 1 Menggabungkan Rope pada Posisi Awal Berdasarkan Gambar 10 untuk *query* sejumlah N = 5000 sampai $N = 10^5$ dengan banyak *string* konstan pada $Q = 10^5$, implementasi struktur data Rope menunjukkan waktu yang lebih cepat dibandingkan dengan String yang memiliki kompleksitas sebesar $O(N)$ [3]. Sedangkan untuk *string* konstan pada $N = 10^5$ dengan banyak *query* Q = 5000 sampai $Q = 10^5$, pertumbuhan waktu sebesar $O(\log N)$ dan lebih kecil dibandingkan dengan menggunakan algoritma String, yang ditunjukkan pada Gambar 11.

Gambar 8. Hasil uji kebenaran dengan melakukan *submission* ke situs penilaian daring SPOJ.



Gambar 9. Hasil uji coba *submission* ke situs penilaian daring SPOJ sebanyak 15 kali.

2. Operasi 2 Menggabungkan Rope pada Posisi Akhir Berdasarkan Gambar 12 untuk *query* sejumlah N = 5000 sampai $N = 10^5$ dengan banyak *string* konstan pada $Q = 10^5$, implementasi struktur data Rope menunjukkan waktu yang lebih cepat dibandingkan dengan String yang memiliki kompleksitas sebesar $O(N)$ [3]. Sedangkan untuk *string* konstan pada $N = 10^5$ dengan banyak *query* Q = 5000 sampai $Q = 10^5$, pertumbuhan waktu sebesar $O(\log N)$ dan lebih kecil dibandingkan dengan menggunakan algoritma String, yang ditunjukkan pada Gambar 13.
3. Operasi 3 Mencetak Karakter pada Indeks ke-Y Berdasarkan Gambar 14 untuk *query* sejumlah N = 5000 sampai $N = 10^5$ dengan banyak *string* konstan pada $Q = 10^5$, implementasi struktur data String menunjukkan waktu yang lebih cepat dibandingkan dengan Rope yang telah dibuat. Hal ini dikarenaka String memiliki kompleksitas sebesar $O(1)$ [3]. Dan berlaku ketika menjalankan operasi untuk $N = 10^5$ dengan banyak *query* Q = 5000 sampai $Q = 10^5$, yang ditunjukkan pada Gambar 15.

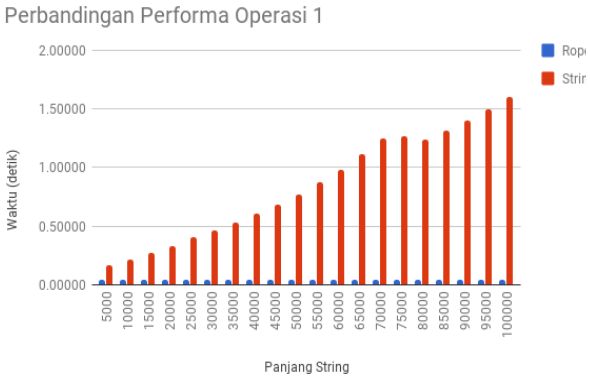
C. Analisis Kompleksitas

Berdasarkan ketiga skenario yang telah dilakukan, masing-masing membuktikan kebenaran dan efisiensi implementasi struktur data Rope yang telah dibuat. Kecuali pada proses pencarian suatu karakter dalam rentang indeks, algoritma String lebih unggul dengan kompleksitas sebesar $O(1)$. Proses spli, concate dan index pada rope mengacu pada algoritma pencarian pada tree sehingga dapat disimpulkan bahwa kompleksitas waktu yang diperlukan sebesar $O(\log N)$, dimana N merupakan panjang *string*. Algoritma dengan kompleksitas waktu sebesar $O(\log N)$ dapat menyelesaikan permasalahan yang diberikan dibandingkan dengan menggunakan algoritma String.

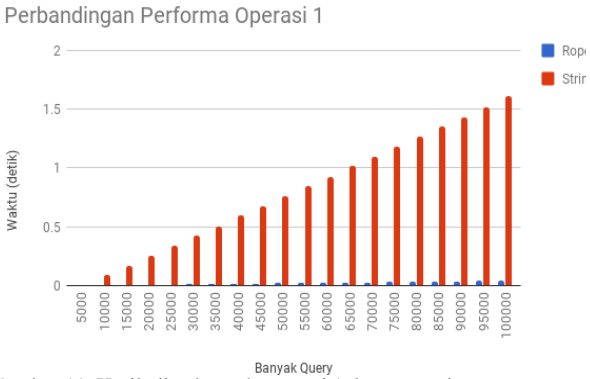
IV. KESIMPULAN

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

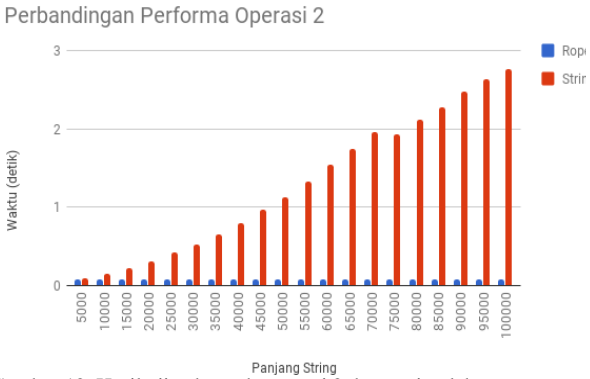
1. Implementasi algoritma dengan menggunakan struktur data Rope dapat menyelesaikan permasalahan klasik Alphabetic Rope dengan benar.



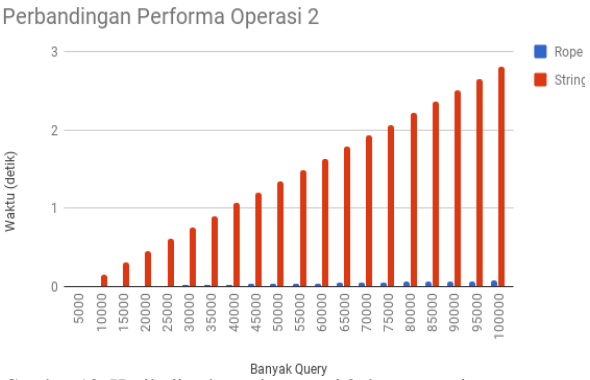
Gambar 10. Hasil uji coba pada operasi 1 dengan jumlah *query* tetap dan panjang *string* bertambah.



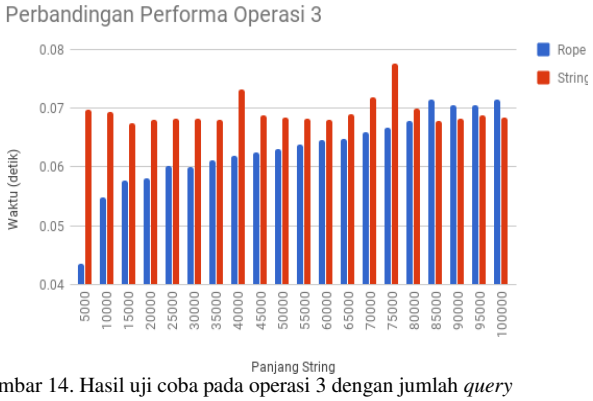
Gambar 11. Hasil uji coba pada operasi 1 dengan panjang *string* tetap dan jumlah *query* bertambah.



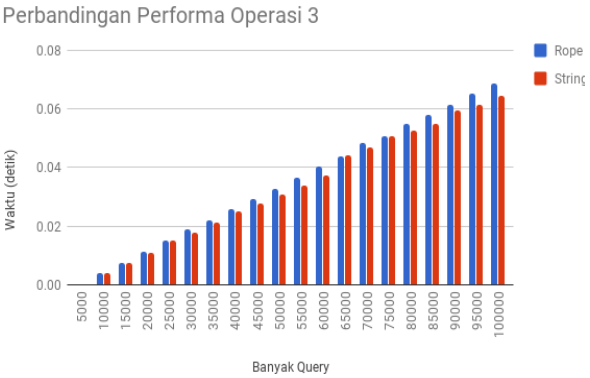
Gambar 12. Hasil uji coba pada operasi 2 dengan jumlah *query* tetap dan panjang *string* bertambah.



Gambar 13. Hasil uji coba pada operasi 2 dengan panjang *string* tetap dan jumlah *query* bertambah.



Gambar 14. Hasil uji coba pada operasi 3 dengan jumlah *query* tetap dan panjang *string* bertambah.



Gambar 15. Hasil uji coba pada operasi 3 dengan panjang *string* tetap dan jumlah *query* bertambah.

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 adalah sebesar 5.0 MB.
4. Struktur data Rope yang dibuat sangat baik diaplikasikan untuk melakukan komputasi *string* yang panjang.

DAFTAR PUSTAKA

[1] Spoj.com, "AROPE - Alphabetic Rope." [Online]. Available: http://www.spoj.com/problems/AROPE/. [Accessed: 24-Oct-2017].

[2] H.-J. Boehm, R. Atkinson, and M. Pass, *Ropes: an Alternative to String*. California: Xeroc PARC, 1994.

[3] H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithm*. Massachusetts: MIT Press, 2001.