Implementasi Struktur Data *Rope* menggunakan *Binary Tree* dan Aplikasinya dalam Pengolahan Teks Sangat Panjang

Edwin Rachman (NIM 13515042)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13515042@std.stei.itb.ac.id

***Abstrak*—Pengolahan data teks yang sangat panjang yang disimpan sebagai *string* biasa kemungkinan besar akan memakan waktu yang sangat lama. Salah satu alternatif dari *string* adalah struktur data *rope* yang menggunakan sebuah *binary tree* sebagai basisnya. *Rope* memiliki beberapa operasi dasar seperti *index, concatenate, split, insert, delete,* dan *report* yang semuanya memiliki kompleksitas waktu logaritmik yang lebih cepat dibandingkan kompleksitas waktu liniernya *string*. *Rope* dapat diaplikasikan dalam sebuah *text editor* dimana operasi-operasi *insert, delete,* dan *concatenate* sering sekali digunakan.**

***Kata Kunci*—Balanced Binary Tree, Pengolahan Teks, Rope, Struktur Data**

I. PENDAHULUAN

Data teks di bidang komputer dan informatika lazimnya direpresentasikan dalam bentuk sebuah struktur data yang disebut *string*, yang sebetulnya hanya sebuah array karakter biasa. Untuk *string-string* berukuran pendek representasi array ini dapat dikatakan cukup baik meskipun operasi *string* menyisipkan, menghapus, ataupun menyambung membutuhkan kompleksitas waktu linier. Akan tetapi ketika *string* yang diolah berukuran sangat panjang, misalnya sebesar 10 MB ataupun 1 GB, operasi-operasi tersebut menjadi sangat lambat.

Terdapat beberapa alternatif struktur data *string* untuk menyimpan data teks secara digital, salah satunya adalah struktur data *rope*. Struktur data *rope* berbeda dengan *string* dimana *rope* tidak menggunakan sebuah array, tetapi menggunakan sebuah *binary tree* sebagai basisnya. Di daun-daun *binary tree* tersebut terdapat *string-string* yang kecil sehingga jika *binary tree* tersebut di-*traverse* secara *in-order* maka akan didapatkan string besar yang utuh.

Karena *rope* menggunakan *binary tree*, maka beberapa operasi-operasi yang dapat digunakan seperti menyisipkan, menghapus, ataupun menyambung membutuhkan kompleksitas waktu logaritmik yang lebih cepat disbandingkan kompleksitas waktu liniernya *array*. Akan tetapi, jumlah memori yang dibutuhkan oleh *rope* lebih besar dibandingkan array dan juga ada beberapa operasi dimana *rope* lebih lambat dibandingkan *array*.

II. TEORI DASAR

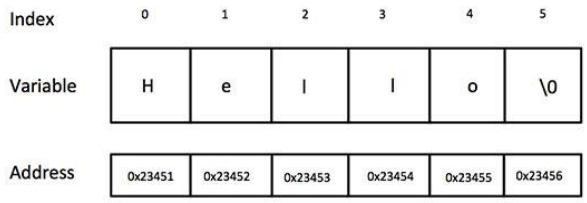
1. String

String adalah sebuah struktur data yang digunakan untuk merepresentasikan data teks sebagai sebuah deretan simbol karakter. *String* lazimnya diimplementasikan sebagai sebuah *array* statik tipe data *char* (berukuran 1 byte) jika menggunakan *encoding* ANSI, atau tipe data yang lebih besar jika menggunakan *encoding* teks yang membutuhkan ukuran per karakter lebih dari 1 byte. Ukuran dari sebuah *string* dapat disimpan secara eksplisit di byte pertama *string* tersebut (jika menggunakan gaya *string* Pascal), atau secara implisit dengan cara menambahkan sebuah karakter NUL (‘\0’) setelah karakter terakhir sebagai menanda akhir dari *string* tersebut (jika menggunakan gaya *string* C).

Keuntungan dari menggunakan struktur data *string* sebagai representasi data teks adalah kesederhanaannya dan kemudahan implementasinya. Selain itu, karena diimplementasikan sebuah *array* yang kontigu maka memori yang digunakan hanya sebesar jumlah karakternya dan operasi *indexing* atau *random access* dapat dilakukan dengan kompleksitas waktu konstan.

Kelemahannya adalah beberapa operasi-operasi pengolahan teks dasar seperi *insert*, *delete*, *concatentate*, dan *split*, memerlukan kompleksitas waktu linear. Hal ini membuat struktur data *string* tidak cocok untuk digunakan dalam mengoperasikan data teks yang sangat panjang (misalnya teks 10 MB atau lebih) dan sering menggunakan operasi-operasi tersebut.

Jenis Operasi	Kompleksitas Waktu
Index	O(1)
Insert	O(n)
Delete	O(n)
Concatenate	O(n)
Split	O(n)
Report	O(n)



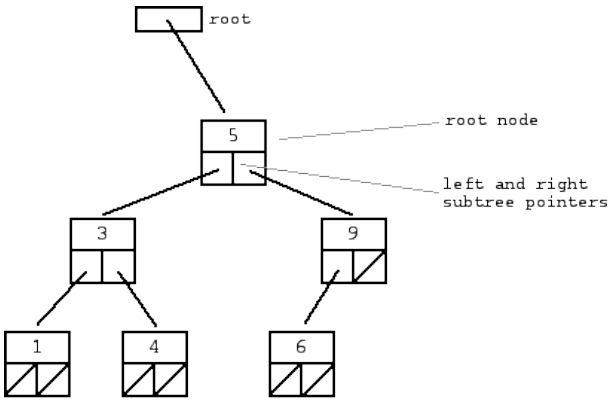
Gambar 1.1 – NUL-terminated string gaya C
(Sumber: https://www.tutorialspoint.com/cprogramming/c_strings.htm)

2. Binary Tree

Struktur data *tree* merupakan struktur data tak linear yang menggambarkan hubungan hirakis antara sebuah *node* dan anak-anaknya yang juga berupa sebuah *tree*. *Tree* biasanya digunakan untuk melambangkan hubungan-hubungan hirakis antar satu data dengan data yang lain. *Node* tertinggi sebuah *tree* disebut sebagai akar dari *tree* tersebut. *Node* tanpa anak kiri ataupun anak kanan disebut sebagai daun-daun tree tersebut. Ketinggian sebuah *tree* adalah beberapa tingkat antar akar dan daun *tree* tersebut. Contoh penggunaannya adalah sebagai diagram organigram anggota-anggota organisasi, pohon keluarga, dll.

Struktur data *binary tree* merupakan variasi dari struktur data *tree* dimana anak-anak setiap *node-nya* hanya terdapat dua, yaitu anak kiri dan anak kanan. *Binary tree* terdapat banyak aplikasi terutama di dunia computer dan informatika dimana *binary tree* dapat digunakan sebagai penampung data yang memudahkan proses pencarian dengan menggunakan algoritma *binary searching* dimana dengan algoritma ini sebuah bilangan dalam sebuah deret bilangan terurut dapat dicari hanya dengan kompleksitas waktu logaritmik.

Struktur data *balanced binary tree* merupakan variasi dari struktur data *tree* dimana perbedaan tinggi *tree* anak kiri dan *tree* kanan tidak lebih dari satu. Hal ini diperlukan karena beberapa algoritma yang dioperasikan pada *tree* memerlukan kondisi ini untuk alasan efisiensi, dll. Contohnya adalah *binary search tree*.



Gambar 1.2 – Sebuah binary tree
(Sumber:http://cslibrary.stanford.edu/110/BinaryTrees.html)

3. Rope

Rope adalah struktur data yang juga digunakan untuk merepresentasikan data teks seperi halnya dengan *string*. Akan tetapi, berbeda dengan *string* yang diimplementasikan sebagai sebuah array yang kontigu, *rope* diimplementasikan sebagai sebuah *binary tree* dimana daun-daunnya mengandung *string-string* kecil sehingga keseluruhan dari *binary tree* tersebut membentuk *string* yang besar.

Binary tree digunakan untuk struktur data *rope* supaya kompleksitas waktu logaritmik dari algoritma *binary searching* dapat dipergunakan bukan untuk deretan angka terurut, tetapi untuk deretan karakter. Selain itu, dengan partisi *string* yang besar menjadi *string-string* yang kecil di setiap daunnya maka operasi *split*, *concatenate*, dan *delete* juga lebih cepat dan menggunakan kompleksitas waktu logaritmik.

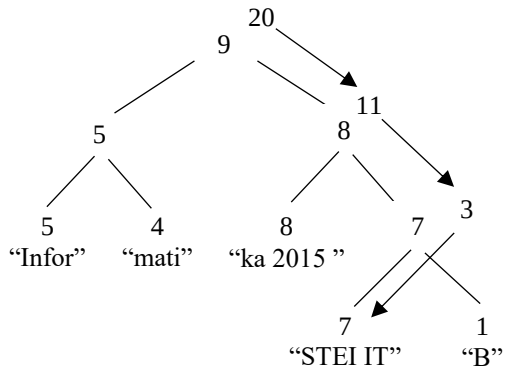
Kelemahan dari *rope* adalah kerumitan dari implementasinya sehingga lebih sulit untuk diimplementasikan dan rawan error. Juga terdapat beberapa operasi *string* yang lebih cepat dibandingkan *rope* seperti *index*. Selain itu, kebutuhan memori untuk *rope* juga lebih besar dibandingkan *string*.

Jenis Operasi	Kompleksitas Waktu
Index	O(log n)
Insert	O(log n)
Delete	O(log n)
Concatenate	O(log n)
Split	O(log n)
Report	O(log n)

III. OPERASI-OPERASI DASAR

1. Index

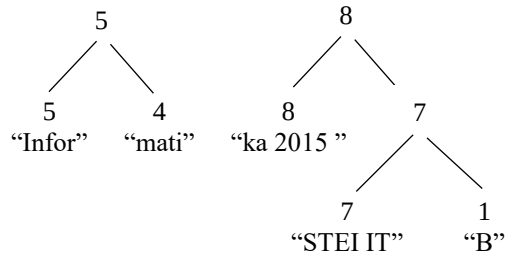
Index adalah operasi mendapatkan karakter ke-i sebuah *string*. Operasi *index* untuk struktur data *rope* dapat dilakukan dengan melakukan traversal seperti binary search tree. [1]



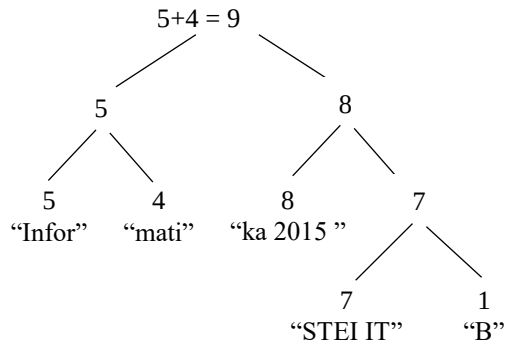
Gambar 3.1: Proses traversal pencarian karakter ke-20

2. Concatenate

Concatenate adalah operasi menyambungkan dua buah string menjadi satu dimana awal dari string kedua tepat setelah akhir string pertama. Operasi *concatenate* untuk struktur data *rope* dapat dilakukan hanya dengan membuat sebuah *node* baru dimana anak kirinya merupakan *node rope* pertama dan anak kanannya merupakan *node rope* kedua. [1]



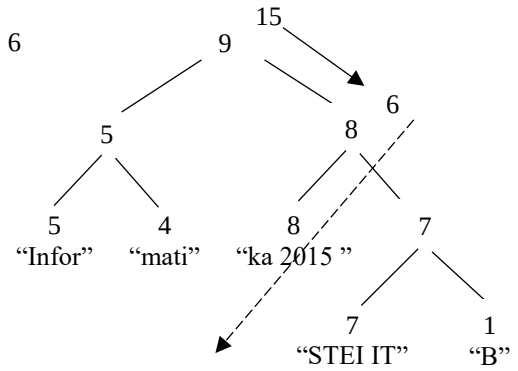
Gambar 3.2a: Dua *rope* yang akan di-*concatenate*



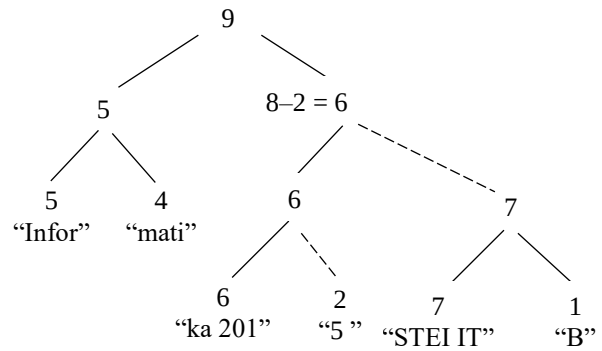
Gambar 3.2a: *Rope* hasil operasi *concatenate*

3. Split

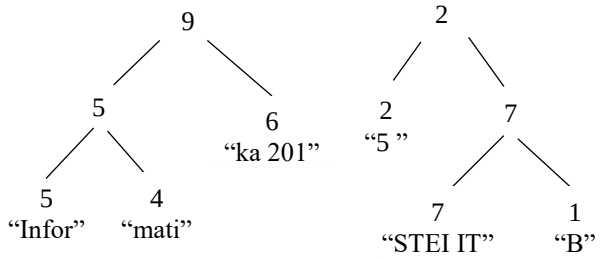
Split adalah operasi memisahkan sebuah *string* menjadi dua buah *string* setelah karakter i tertentu. Operasi *split* untuk struktur data *rope* menggunakan *index* untuk mencari titik pemisahannya kemudian memisahkan string di node tersebut menjadi dua. Kemudian semua node-node di kanan node tersebut juga dipisahkan dari *rope* utamanya. Setelah itu, semua node yang terpisah dari digabung dengan *concat* menjadi *rope* yang baru. [1]



Gambar 3.3a: Mencari titik potong *rope*.



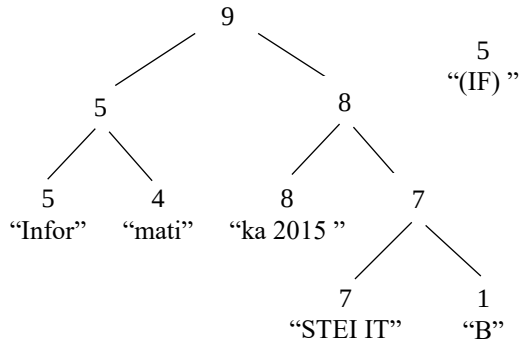
Gambar 3.3b: Pemisahan *rope*



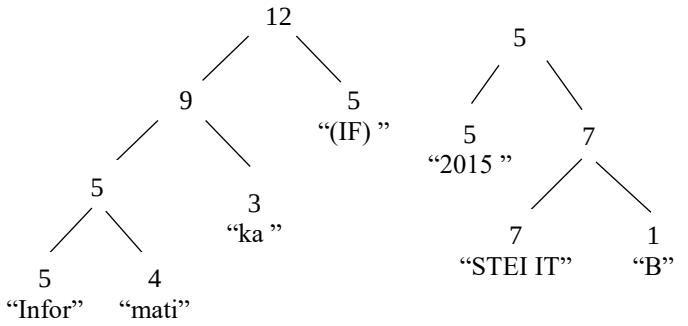
Gambar 3.3c: Penyederhanaan *rope* lama dan penggabungan *rope* baru.

4. Insert

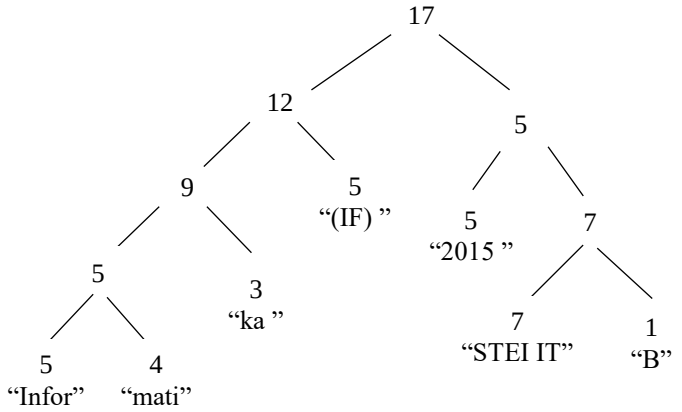
Insert adalah operasi menyisipkan sebuah string ke antara posisi tertentu string lain. Operasi ini dilakukan dengan melakukan *split* di posisi i, kemudian *concat rope* pertama dan *rope* yang disisipkan, kemudian *concat rope* yang baru terbentuk dengan *rope* kedua. [1]



Gambar 3.4a: Rope utama dan rope yang ingin disisipkan



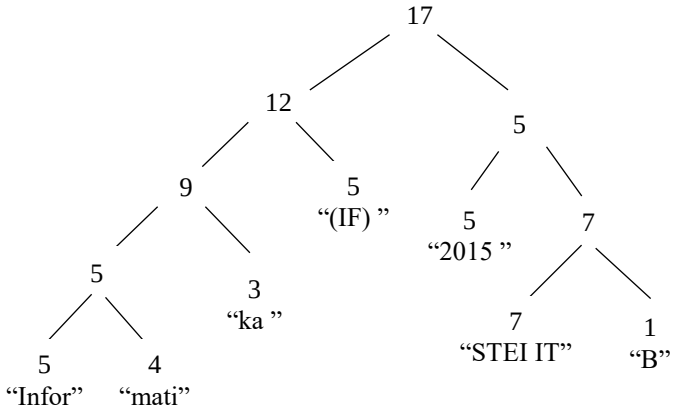
Gambar 3.4b: Rope setelah di-*split* kemudian rope pertama di-*concat* dengan yang disisipkan.

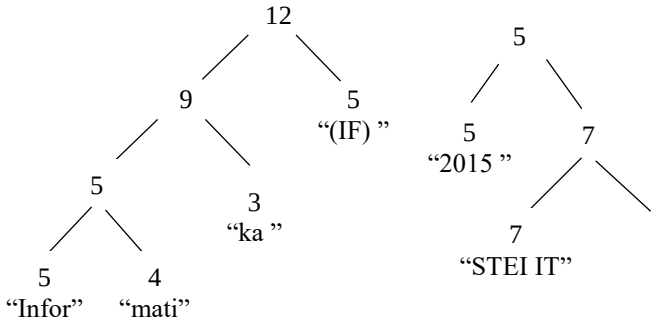


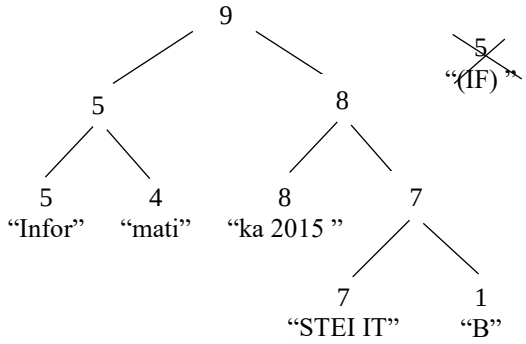
Gambar 3.4c: Rope hasil concat kedua.

5. Delete

Delete adalah operasi menghapuskan karakter sebuah *string* dari posisi i hingga posisi j. [1]

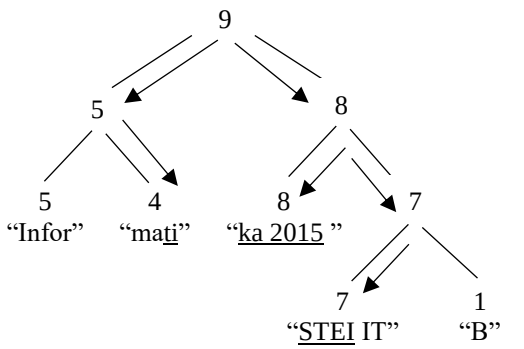






6. Report

Delete adalah operasi menghapuskan karakter sebuah *string* dari posisi i hingga posisi j.



IV. APLIKASI

Struktur data *rope* memiliki banyak aplikasi di dunia komputer dan informatika. Aplikasi utamanya adalah dalam pengolahan teks yang sangat panjang dan juga untuk software teks editor. Akan tetapi, lazimnya teks editor tidak menggunakan *rope* melainkan menggunakan variasi dari *array* yang disebut *gap buffer*. Ini dikarenakan keuntungan dari struktur data *rope* tidak terlalu terlihat jika data teks yang diolah tidak besar sekali.

V. KESIMPULAN

Pengolahan data teks yang sangat panjang yang disimpan sebagai *string* biasa kemungkinan besar akan memakan waktu yang sangat lama. Salah satu alternatif dari *string* adalah struktur data *rope* yang menggunakan sebuah *binary tree* sebagai basisnya. *Rope* memiliki beberapa operasi dasar seperti *index*, *concatenate*, *split*, *insert*, *delete*, dan *report* yang semuanya memiliki kompleksitas waktu logaritmik yang lebih cepat dibandingkan kompleksitas waktu liniernya *string*. *Rope* dapat diaplikasikan dalam sebuah *text editor* dimana operasi-operasi *insert*, *delete*, dan *concatenate* sering sekali digunakan.

VI. UCAPAN TERIMA KASIH

Saya mengucapkan terima kasih kepada Pak Rinaldi Munir sebagai dosen Matematika Diskrit saya selama semester ini, dan juga teman-teman dan kelurga saya yang membantu saya selama semester ini.

REFERENCES

[1] Boehm, Hans-J; Atkinson, Russ; Plass, Michael *(December 1995).* *"Ropes: an Alternative to Strings"* (diakses 8/12/2016 dalam bentuk PDF)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2016

Edwin Rachman (NIM 13515042)