INTISARI

MODIFIKASI STRUKTUR DATA *TREE LIST* MENJADI POHON PENCARIAN BINER

Oleh

Abdul Malik Nurrokhman
19/440065/PA/19054

Pada tahun 2019, Sibin James, dkk. mengembangkan sebuah struktur data yang bernama *Tree List*. *Tree List* merupakan struktur data dinamis yang menyimpan data dalam bentuk pohon biner yang selalu seimbang setiap terjadi perubahan data. Akan tetapi, perubahan data pada *Tree List* hanya dapat dikenakan pada indeks terakhir dan data yang tersimpan tidak selalu terurut. Sementara itu, ada struktur data Pohon Pencarian Biner yang dapat dikenakan perubahan data pada posisi yang sesuai untuk menjaga keterurutan data yang tersimpan. Akan tetapi setelah terjadi perubahan data, bentuk pohon biner bisa jadi tidak seimbang sehingga memerlukan algoritma penyeimbangan seperti pada *WAVL Tree* yang dikembangkan oleh Haeupler, dkk. pada tahun 2015.

Penelitian ini mengembangkan sebuah struktur data yang diberi nama *DPS Tree (Domain Partition Search Tree)* yang merupakan Pohon Pencarian Biner hasil modifikasi dari *Tree List* yang selalu seimbang setiap terjadi perubahan data. Untuk menjaga keseimbangan pohon biner tersebut, *DPS Tree* membatasi data yang dapat disimpan hanya berupa bilangan bulat dalam rentang tertentu. Setiap *node* dalam *DPS Tree* mempunyai dua buah *children* dan sebuah rentang bilangan bulat yang akan dibagi dua lalu masing-masing bagian menjadi rentang dari salah satu *children*. *DPS Tree* dapat dikenakan operasi *insert*, *delete*, dan *search*.

Dalam penelitian ini, rentang yang digunakan yaitu dari -2^{63} sampai $2^{63} - 1$. Kedalaman maksimal dari *DPS Tree* adalah $O(1)$ ketika banyaknya data mencapai 65 dan $O(n)$ ketika kurang dari itu. Kompleksitas memori dari *DPS Tree* adalah $O(n)$ dan kompleksitas waktunya sebanding dengan kedalamannya. Kompleksitas kedalaman pada kasus terburuk adalah $O(1)$ asimtot terhadap nilai konstan 65 dan kasus terbaik maupun rata-ratanya adalah $\Omega(\log n) = \Theta(\log n)$. Jika dibandingkan dengan *WAVL Tree*, kedalaman rata-rata dari *DPS Tree* lebih banyak meskipun sama-sama $\Theta(\log n)$. Adapun dari sisi implementasi, banyaknya baris kode dari *DPS Tree* untuk operasi *insert* dan *delete* lebih sedikit 56% dan 54% dibandingkan dengan *WAVL Tree* serta memiliki kode yang sama persis untuk operasi *search*.

Kata kunci: struktur data, pohon biner, partisi domain, kompleksitas, implementasi

ABSTRACT

MODIFICATION OF TREE LIST DATA STRUCTURE INTO BINARY SEARCH TREE

By

Abdul Malik Nurrokhman
19/440065/PA/19054

In 2019, Sibin James, et al. develop a data structure named Tree List. Tree List is a dynamic data structure that stores data in the form of a binary tree which is always balanced whenever data modification occurs. However, data modification in Tree List can only be applied to the last index, and the stored data is not always sorted. Meanwhile, there is a Binary Search Tree data structure that can be applied with data modification at the appropriate position to maintain the order of the stored data. However, after a data modification occurs, the form of the binary tree may become unbalanced, requiring a balancing algorithm such as WAVL Tree developed by Haeupler, et al. in 2015.

This research develops a data structure named DPS Tree (Domain Partition Search Tree) which is a modified Binary Search Tree from Tree List that is always balanced whenever data modification occurs. To maintain the balance of the binary tree, DPS Tree limits the data that can be stored to only integers within a certain range. Each node in the DPS Tree has two children and an integer range that will be halved and then each part becomes a range of one of the children. DPS Tree can be applied with insert, delete, and search operations.

In this research, the range used is from -2^{63} to $2^{63}-1$. The maximum depth of DPS Tree is $O(1)$ when the number of data reaches 65 and $O(n)$ when less than that. The memory complexity of DPS Tree is $O(n)$ and the time complexity is proportional to the depth. The depth complexity in the worst case is $O(1)$ asymptote to the constant value of 65, and both the best and average case is $\Omega(\log n) = \Theta(\log n)$. When compared to WAVL Tree, the average depth of DPS Tree is greater even though both of them are $\Theta(\log n)$. In the implementation, the number of lines of code in DPS Tree for insert and delete operations are 56% and 54% less than WAVL Tree and have the same code for search operation.

Keywords: data structure, binary tree, domain partition, complexity, implementation