BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan hasil analisis dan pengembangan tipe data abstrak map dan program simulasi yang telah selesai dilakukan, maka dapat diambil beberapa kesimpulan.

1. Struktur data *right-threaded AVL tree* telah berhasil dikembangkan beserta dengan program simulasi yang berbasis GUI. Dengan berhasilnya pengembangan struktur *right-threaded AVL tree* ini maka dapat memudahkan dalam proses pembuatan tipe data abstrak map selanjutnya.
2. Tipe data abstrak *map* berhasil dikembangkan beserta program simulasi yang berbasis GUI.
3. Penggunaan *right-thread* pada struktur AVL *tree* bisa memudahkan implementasi fungsi *iterator()* pada kelas `IntKeyTreeMap`
4. Program pengetesan kebutuhan memori dan kecepatan waktu untuk mengukur performansi dari tipe data abstrak *map* berhasil dikembangkan. Dengan adanya program tersebut dapat memudahkan dalam menganalisis performansi tipe data abstrak tersebut berdasarkan hasil yang didapat.
5. Dari hasil pengujian sebanyak 10000 data, `IntKeyTreeMap` memerlukan hanya 13% memori dari jumlah memori pada kelas `TreeMap` atau memori yang dibutuhkan oleh `IntKeyTreeMap` 87% lebih hemat dibandingkan dengan kelas `TreeMap` pada Java.
6. Kecepatan waktu eksekusi yang diuji coba menggunakan 10000 data pada fungsi *insert()*, menunjukkan bahwa `IntKeyTreeMap` membutuhkan kecepatan waktu eksekusi rata-rata 3% lebih cepat dari kelas `TreeMap` pada Java.
7. Kecepatan waktu eksekusi yang diuji coba menggunakan 10000 data pada fungsi *remove()*, menunjukkan bahwa kelas `TreeMap` pada Java

membutuhkan kecepatan waktu eksekusi rata-rata 33% lebih cepat dari kelas `IntKeyTreeMap`.

8. Kecepatan waktu eksekusi yang diuji coba menggunakan 10000 data pada fungsi *get()*, menunjukkan bahwa kelas `IntKeyTreeMap` membutuhkan kecepatan waktu eksekusi rata-rata 75% lebih cepat dari kelas `TreeMap` pada Java.
9. Jika terdapat aplikasi yang dibangun dengan bahasa pemograman Java memerlukan sebuah tipe *map* yang menghasilkan *key* secara terurut dan mengutamakan kecepatan dalam proses *insertion* dan *deletion*, maka `TreeMap` merupakan pilihan yang tepat.
10. Jika terdapat aplikasi yang dibangun dengan bahasa pemograman Java memerlukan sebuah tipe *map* yang menghasilkan *key* secara terurut dan mengutamakan kecepatan dalam proses *retrieving* dan penggunaan memori yang lebih sedikit, maka `IntKeyTreeMap` merupakan pilihan yang tepat.
11. Kekurangan dari kelas `IntKeyTreeMap` adalah keterbatasan *key* yang hanya bisa digunakan dengan tipe *integer* dan kelas tersebut akan menunjukkan hasil yang maksimal apabila *key* disusun secara *sequence* atau *reverse*.

6.2 Saran

Saran yang diberikan untuk pengembangan lebih lanjut dari tipe data abstrak ini antara lain.

1. Mengganti tipe data yang menjadi prefix pada `NodeEntry` di dalam kelas `IntKeyTreeMap` menjadi tipe `long`, sehingga jumlah maksimal penyimpanan *key-value* dalam satu node menjadi 64 pasang.
2. Membandingkan performansi seluruh fungsi-fungsi selain *put()*, *get()* dan *remove()*, serta mengoptimalkan kecepatan waktu proses pada fungsi-fungsi lainnya selain tiga fungsi yang disebutkan tadi.
3. Memperluas batasan *key* dengan tipe data selain *integer*.

4. Membuat sebuah konsep *map* dapat menyimpan satu atau lebih *key* yang bernilai sama, sehingga jika terdapat *key* yang sama, maka *value* dari *key* sebelumnya tidak ditimpa atau diganti dengan *value* dari *key* yang baru.