Aplikasi Segment Tree dalam Pendataan Peminjaman Buku Perpustakaan

Irfan Sidiq Permana - 13522007[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522007@std.stei.itb.ac.id

***Abstrak*—Perpustakaan menyimpan ratusan hingga jutaan buku sekaligus di satu tempat yang melayani peminjaman dan pengembalian buku oleh pengunjung. Masing-masing buku tersebut dimasukkan kedalam *database* yang berfungsi untuk melacak informasi terkini mengenai buku tersebut, seperti jumlah buku yang tersedia dan jumlah buku yang sedang dipinjam. Buku perpustakaan juga dikelompokkan sesuai topik/*genre*-nya, tiap kelompok memiliki *range* nomor entri tertentu. Dihadapkan dengan ukuran data yang tergolong besar akibat jumlah buku yang banyak, maka diperlukan suatu struktur data yang dapat melakukan pengubahan (*update*) data serta *query* berjarak (*range query*) dengan cepat. Makalah ini membahas mengenai salah satu struktur data yang dapat melakukan dua hal tersebut dengan efisien, yaitu Pohon Segmen atau *Segment Tree*.**

***Keywords*—Perpustakaan, *Segment Tree*, *Range Sum Query* (RSQ), Kompleksitas Algoritma**

I. Pendahuluan

Perpustakaan adalah suatu tempat yang menyimpan koleksi buku, dimana pengunjung yang telah terdaftar dapat meminjam dan mengembalikan buku sesuai kuota yang telah ditentukan. Selain untuk meminjam dan mengembalikan buku, perpustakaan kerap dijadikan tempat bagi masyarakat untuk belajar, kerja kelompok, atau mengerjakan tugas pribadi. Terdapat berbagai macam buku yang tersedia di perpustakaan, mulai dari buku fiksi hingga buku non-fiksi. Tiap buku di perpustakaan umumnya dikelompokkan sesuai dengan topik/*genre* buku, serta diberi label entri yang berupa sebuah angka sesuai dengan kelompok buku tersebut. Hal ini bertujuan untuk melacak dan mendata tiap buku yang ada di perpustakaan, memasukkan data tiap buku ke dalam *database* komputer dengan nomor entri buku sebagai ID untuk melacak letak buku pada perpustakaan, jumlah buku yang tersedia, jumlah buku yang sedang dipinjam, dan sebagainya.

Suatu perpustakaan dapat menyimpan ratusan, ribuan, hingga jutaan buku sekaligus dalam satu tempat. *Library of Congress*, perpustakaan terbesar di dunia yang terletak di Washington D.C., Amerika Serikat, bahkan menyimpan lebih dari 175 juta koleksi buku. Jumlah buku yang sangat signifikan ini tentu membuat pendataan sirkulasi buku menjadi sulit dan lama tanpa adanya sistem yang efisien. Suatu sistem yang efisien haruslah berjalan dengan tepat dan cepat dalam menjalankan berbagai fungsi dan operasinya walaupun dihadapkan dengan ukuran data yang besar. Oleh karena itu, diperlukan suatu struktur data yang tepat untuk menyimpan pendataan buku dan algoritma yang efisien untuk menjalankan operasi yang dibutuhkan.

Pada makalah ini, penulis membahas mengenai salah satu pendekatan struktur data yang dapat digunakan untuk menangani persoalan diatas, yaitu Pohon Segmen atau *Segment Tree*. Penulis juga menganalisis efisiensi struktur data tersebut serta perbandingan waktu yang diperlukan bila menggunakan struktur data yang paling sederhana yaitu larik/*array*, melalui percobaan dengan program yang telah dirancang.

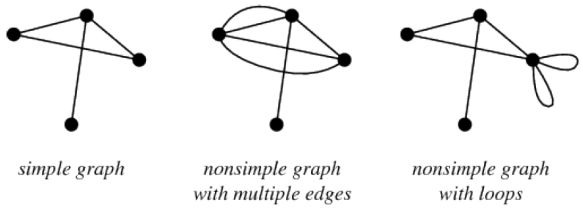
II. Landasan Teori

A. Graf

Graf adalah suatu struktur data yang terdiri dari himpunan simpul dan himpunan busur yang menghubungkan simpul-simpul tersebut. Graf digunakan untuk merepresentasikan objek-objek diskrit (yang direpresentasikan sebagai simpul) serta hubungan antara objek-objek tersebut (yang direpresentasikan sebagai busur).

Berdasarkan ada tidaknya gelang dan sisi ganda, graf dibedakan menjadi dua yaitu:

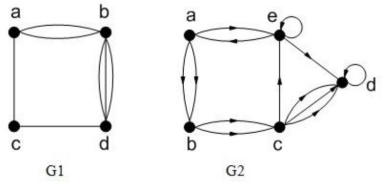
1) Graf sederhana, yaitu graf yang tidak memiliki gelang (sisi yang menghubungkan suatu simpul dengan simpul itu sendiri) maupun sisi ganda (dua atau lebih busur yang menghubungkan dua buah simpul yang sama).
2) Graf tak-sederhana, yaitu graf yang memiliki gelang dan/atau sisi ganda.



Gambar 1. Ilustrasi graf sederhana dan graf tidak sederhana
Sumber: [1]

Sedangkan berdasarkan orientasi arah pada sisi, graf dibedakan kembali menjadi dua yaitu:

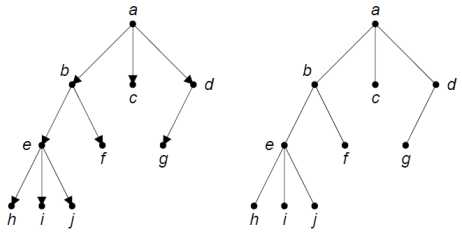
1) Graf berarah, yakni graf yang setiap sisinya diberikan orientasi arah.
2) Graf tak-berarah, yakni graf yang semua sisinya tidak memiliki orientasi arah.



Gambar 2. Ilustrasi graf tak berarah (G1) dan graf berarah (G2)
Sumber: [1]

B. Pohon

Pohon adalah graf tak-berarah yang tidak mengandung sirkuit, yakni suatu lintasan dari suatu simpul awal (v_0) menuju simpul itu sendiri. Sedangkan pohon berakar adalah pohon yang salah satu simpulnya diperlakukan sebagai akar dan sisi-sisinya diberikan arah yang secara langsung maupun tidak langsung menghubungkan simpul akar ke semua simpul lainnya.



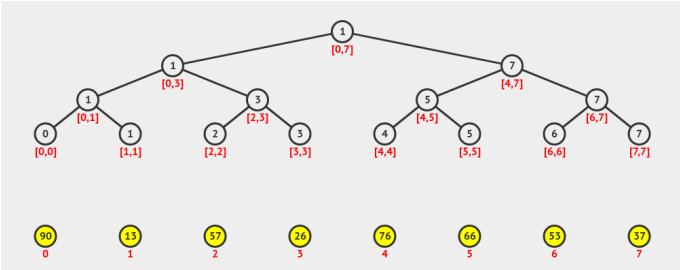
Gambar 3. Contoh ilustrasi pohon berakar
Sumber: [2]

Beberapa terminologi yang terdapat dalam pohon berakar:

1) Anak (*children*) dan orangtua (*parent*)
   Sebuah simpul v_1 dikatakan anak dari simpul v_2 jika terdapat busur berarah yang menghubungkan v_2 menuju v_1. Simpul asal yaitu v_2 dikatakan sebagai orangtua dari v_1.
2) Leluhur (*ancestor*)
   Sebuah simpul v_1 dikatakan leluhur dari simpul v_2 jika terdapat lintasan yang berawal dari v_1 menuju v_2. Dengan kata lain, leluhur dari suatu simpul v_2 adalah semua simpul yang ada pada lintasan simpul akar menuju simpul v_2.
3) Lintasan (*path*)
   Lintasan adalah suatu rangkaian urutan simpul dan busur dari suatu simpul awal menuju simpul tujuan.
4) Saudara kandung (*sibling*)
   Sebuah simpul v_1 dikatakan saudara kandung dari simpul v_2 jika v_1 dan v_2 memiliki orangtua yang sama.
5) Upapohon (*subtree*)
   Upapohon adalah suatu pohon yang akarnya merupakan anak dari simpul lain pada pohon utama, atau dapat dibilang upapohon adalah subset dari pohon utama.
6) Derajat (*degree*)
   Derajat dari suatu simpul merupakan jumlah anak yang dimiliki oleh simpul tersebut. Derajat yang dimaksud pada pohon ialah derajat keluar pada graf berarah.
7) Daun (*leaf*)
   Daun ialah simpul yang tidak memiliki anak, yaitu berderajat nol.
8) Simpul dalam (*internal nodes*)
   Simpul dalam ialah simpul yang memiliki anak.
9) Aras (*level*) atau tingkat
   Aras atau tingkat dari suatu simpul ialah jarak antara akar dengan simpul tersebut.
10) Tinggi (*height*)
    Tinggi dari sebuah pohon ialah jarak maksimum antara akar dengan daun pohon.

C. Pohon Segmen

Pohon segmen atau *segment tree* adalah pohon biner yang digunakan untuk menyimpan informasi terkait interval, atau segmen, dari suatu data. Setiap simpul dari pohon segmen merepresentasikan data dari suatu interval.



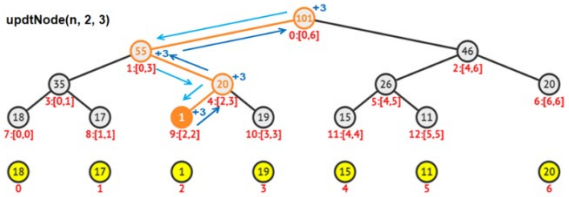
Gambar 4. Ilustrasi pohon segmen yang bersesuaian dengan datanya
Sumber: [3]

Beberapa ciri dari pohon segmen adalah sebagai berikut :

1) Tiap simpul merepresentasikan suatu interval $[i : j]$, $0 \leq i \leq j < N$.
2) Semua simpul pada kedalaman (*depth*) yang sama memiliki *range* interval $(j - i)$ yang sama.
3) Tiap simpul pada suatu kedalaman k+1 memiliki *range* interval setengah dari *range* interval simpul pada kedalaman k.
4) Akar (*root*) dari pohon segmen mereprentasikan seluruh data ($[0 : N]$).
5) Tiap daun (*leaf*) dari pohon segmen merepresentasikan satu elemen data, misalnya $A[i]$.
6) Merupakan pohon lengkap, yaitu tiap kedalaman pohon memiliki jumlah simpul yang penuh kecuali pada kedalaman terakhir yang dapat penuh atau tidak penuh.
7) Memerlukan memori sebesar $O(N \log N)$, dengan tinggi pohon adalah $\log(N)$.

Pohon segmen digunakan untuk memproses dua operasi khusus, yaitu:

1) *Update*
   Operasi *update* yang dimaksud adalah mengubah nilai suatu elemen dari data awal. Mengubah nilai elemen pada pohon segmen berarti mengubah daun pohon yang bersesuaian dengan elemen yang diubah, lalu menerapkan perubahan tersebut ke semua leluhur dari daun tersebut agar seluruh simpul tetap memiliki data yang valid.



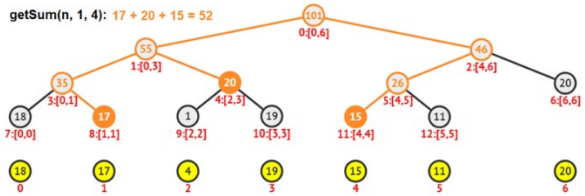
Gambar 5. Ilustrasi contoh operasi *update* pada pohon segmen
Sumber: [4]

Pada contoh gambar diatas, operasi *update* dilakukan pada data dengan indeks 2, dengan skema *increment* yaitu menjumlahkan nilai awal data k menjadi k+3. Maka selain mengubah nilai daun yang berkoresponden dengan elemen

data berindeks 2, semua leluhur dari daun tersebut juga harus di-*increment* dengan nilai 3.

2) *Query*

Operasi *query* yang dimaksud adalah meminta nilai keseluruhan dari suatu interval yang diinginkan pada data. *Query* ini memiliki fungsi yang berbeda-beda tergantung pada informasi yang disimpan pada pohon segmen. Misalnya, bila tiap simpul dalam menyimpan informasi nilai minimum dari masing-masing interval, maka *query* yang dimaksud ialah *Range Minimum Query* (RMQ) yaitu mencari nilai minimum data dari suatu interval [*i* : *j*]. Bila tiap simpul dalam menyimpan informasi jumlah nilai dari masing-masing interval, maka *query* yang dimaksud adalah *Range Sum Query* (RSQ), dan sebagainya.



Gambar 6. Ilustrasi contoh operasi *query* pada pohon segmen
Sumber: [5]

Pada contoh gambar diatas, operasi *query* RSQ dilakukan pada interval [1 : 4], sehingga nilai yang dihitung adalah penjumlahan nilai dari simpul 8 yang berisi data elemen berindeks 1, simpul 4 yang berisi data elemen berindeks 2 hingga 3, dan simpul 11 yang berisi data elemen berindeks 4.

C. Kompleksitas Algoritma

Kompleksitas dari suatu algoritma digunakan untuk mengukur seberapa mangkus dan sangkil suatu algoritma, serta untuk membandingkan satu algoritma dengan algoritma yang lain demi menentukan mana yang lebih baik. Kompleksitas algoritma berfungsi untuk mengukur performa keseluruhan algoritma seiring membesarnya ukuran data menuju tak hingga.

Terdapat dua macam kompleksitas algoritma, yaitu:

1) Kompleksitas waktu, $T(n)$

Merupakan jumlah tahapan yang dilakukan dalam algoritma sebagai fungsi dengan ukuran masukan n. Jumlah tahapan ini berkoresponden dengan waktu yang diperlukan suatu algoritma untuk menyelesaikan prosesnya, semakin banyak tahapan yang dilakukan maka waktu yang diperlukan semakin banyak.

2) Kompleksitas ruang, $S(n)$

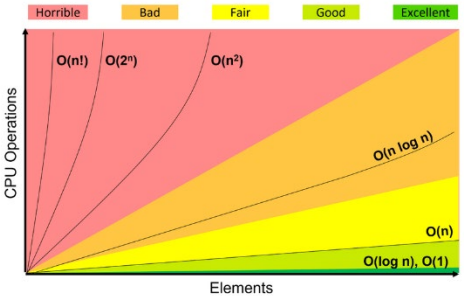
Merupakan jumlah memori total yang digunakan selama algoritma berjalan sebagai fungsi dengan ukuran masukan n. Kompleksitas ruang berbeda dengan ruang tambahan (*auxiliary space*), yang menyatakan jumlah memori tambahan yang diperlukan selama algoritma berjalan. Memori tambahan ini berguna sebagai memori sementara yang menyimpan hasil kalkulasi sementara program, dan tidak digunakan lagi setelah algoritma selesai berjalan.

Suatu algoritma yang ideal tentunya diharapkan memiliki kompleksitas waktu dan kompleksitas memori yang sekecil mungkin, sehingga waktu yang diperlukan sesedikit mungkin dan memori yang dibutuhkan sekecil mungkin. Hal ini dapat dicapai dengan memilih struktur data yang tepat serta algoritma yang mangkus dan sangkil.

Seringkali nilai kompleksitas yang presisi untuk suatu ukuran n tertentu tidak terlalu penting dibandingkan dengan melihat kebutuhan waktu dan/atau memori seiring meningkatnya ukuran data. Oleh karena itu, digunakan beberapa notasi yang berfungsi menyatakan kompleksitas algoritma asimptotik seiring membesarnya ukuran data n, yaitu:

1) Notasi O-Besar (*Big-O*)

Notasi Big-O merupakan notasi yang paling sering digunakan dalam dunia praktik, dikarenakan fungsinya sebagai batas atas (*upper bound*) dari suatu algoritma. Kompleksitas $T(n)$ dinyatakan sama dengan $O(f(n))$ bila terdapat suatu konstanta C dan n_0 sehingga $T(n) \leq Cf(n)$, untuk $n > n_0$. Dengan menggunakan notasi *Big-O*, kita dapat menyatakan bahwa kompleksitas dari algoritma kita tidak akan melebihi $Cf(n)$ seiring membesarnya nilai n.



Gambar 6. Ilustrasi contoh beberapa fungsi $O(f(n))$
Sumber: [6]

Berikut merupakan beberapa fungsi $O(f(n))$ yang dijumpai pada algoritma-algoritma umum:

Tabel 1. Beberapa fungsi $O(f(n))$ dengan contoh algoritmanya

$O(f(n))$	Algoritma
$O(1)$	Perbandingan, pertukaran elemen
$O(\log n)$	*Binary search*
$O(n)$	*Linear search*, mencari nilai maksimum/minimum, rata-rata, dsb.
$O(n \log n)$	*Divide and conquer*
$O(n^2)$	*Selection sort*, *bubble sort*, *insertion sort*
$O(n^3)$	Perkalian matriks
$O(2^n)$	*Knapsack*, *sum of subset*
$O(n!)$	*Travelling Salesperson Problem*

2) Notasi *Big-Omega*

Notasi *Big*-Ω berfungsi sebagai batas bawah (*lower bound*) dari suatu algoritma. Kompleksitas $T(n)$ dinyatakan sama dengan $\Omega(f(n))$ bila terdapat suatu konstanta C dan n_0 sehingga $T(n) \geq \Omega(f(n))$, untuk $n > n_0$. Dengan menggunakan notasi *Big*-Ω, kita dapat menyatakan bahwa kompleksitas dari algoritma kita tidak akan kurang dari $Cf(n)$ seiring membesarnya nilai n.

3) Notasi *Big-Theta*

Suatu kompleksitas $T(n)$ dinyatakan sama dengan $\theta(f(n))$ apabila $T(n)$ sama dengan $O(f(n))$ dan $T(n)$ sama dengan $\Omega(f(n))$.

III. METODOLOGI

A. Deskripsi Masalah

Dalam suatu perpustakaan bernama Perpustakaan Matdis, terdapat 100000 buku yang tersedia untuk dipinjam oleh pengunjung. Setiap buku memiliki judul yang berbeda-beda, dan masing-masing buku diberi nomor entri yang berbeda berupa angka dari 1-100000. Data berupa judul tiap buku dan jumlah buku yang dipinjam dari tiap buku disimpan dalam file bernana 'data.txt' yang akan dimuat program saat program dijalankan. Contoh isi data.txt yaitu sebagai berikut:

Gambar 7. Cuplikan daftar buku yang tersedia di Perpustakaan Matdis beserta jumlah buku yang sedang dipinjam
Sumber: Dokumen Penulis

Masing-masing buku diatas telah dikelompokkan berdasarkan topik/*genre*-nya, yaitu:

Tabel 2. Nomor entri buku beserta kelompok topik/*genre*-nya

Nomor Entri	Topik/*Genre*
1-1799	Antropologi
1800-2999	Seni dan Desain
3000-4199	Biografi & Memoir
4200-5099	Bisnis
5100-6399	Ekonomi
6400-8199	Fantasi
…	…
97500-100000	Sains & Ilmu Pengetahuan Alam

Untuk melakukan pendataan peminjaman buku pelanggan, program *database* Perpustakaan Matdis memiliki dua fungsionalitas yaitu: 1) *Update,* yaitu mengubah nilai jumlah buku yang dipinjam untuk masing-masing buku, dan 2) *Range Sum Query* (RSQ), yaitu mencari tahu jumlah buku yang sedang dipinjam pada tiap kelompok topik/*genre* maupun beberapa kelompok topik/*genre* sekaligus.

Namun dari dulu Perpustakaan Matdis selalu menggunakan *array* untuk menyimpan data-data tersebut, dan implementasi struktur data pohon segmen baru digunakan beberapa tahun silam karena jumlah buku yang semakin membludak berkat populernya Perpustakaan Matdis akhir-akhir ini. Oleh karena itu, untuk membandingkan efisiensi *array* yang selama ini mereka gunakan dengan pohon segmen pihak pengurus Perpustakaan Matdis melakukan uji coba untuk mengukur waktu yang diperlukan bagi kedua struktur data untuk melakukan fungsi-fungsi diatas.

B. Batasan Masalah

Dalam mengimplementasikan program untuk melakukan uji coba, terdapat beberapa batasan yang digunakan untuk menyederhanakan masalah pendataan peminjaman buku perpustakaan. Batasan tersebut diantaranya adalah:

1) Fitur yang diimplementasikan pada program berjumlah dua, yaitu "pinjam buku" dan "data peminjaman buku". Hal ini dikarenakan pohon segmen hanya memiliki dua operasi utama yaitu *update* dan *query*, sehingga dua fitur diatas dirasa sudah cukup untuk merepresentasikan kedua operasi tersebut.
2) Agar implementasi program menjadi lebih sederhana, semua buku yang tersedia dianggap berjumlah tak berhingga sehingga buku selalu dapat dipinjam (jumlah buku yang dipinjam untuk setiap buku tidak dibatasi).
3) Dalam file 'data.txt', tidak semua 100000 buku yang tertera memiliki judul yang berbeda. Angka 100000 sebagai jumlah buku dipilih untuk memperbesar selisih perbedaan waktu antara implementasi *segment tree* dengan *array* dalam melakukan kedua operasi diatas.

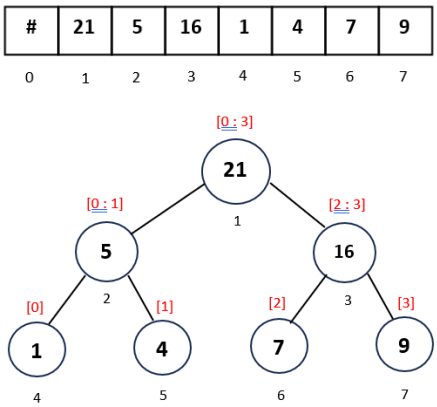
C. Implementasi Program

Untuk kode lengkap program dapat dilihat di:

https://github.com/IrfanSidiq/matdis-segtree

Program diimplementasikan dengan bahasa C, terdiri dari dua file *source code* utama yaitu 'segtree.c' dan 'main.c'. File 'segtree.c' berisi fungsi primitif yang dimiliki oleh struktur data pohon segmen, sedangkan 'main.c' berfungsi sebagai program utama yang menjalankan semua fungsi program.

Pohon segmen yang diimplementasikan oleh penulis ialah pohon segmen dengan representasi *array*, tidak menggunakan struktur data pohon pada umumnya yang menggunakan *pointer*. Pohon segmen menyimpan nilai penjumlahan tiap elemen dari tiap interval untuk menjawab persoalan RSQ.



Gambar 8. Ilustrasi Pohon Segmen dengan representasi *array*
Sumber: Dokumen Penulis

Terdapat tiga fungsi utama yang ada pada implementasi pohon segmen ini, yaitu:

1) Build

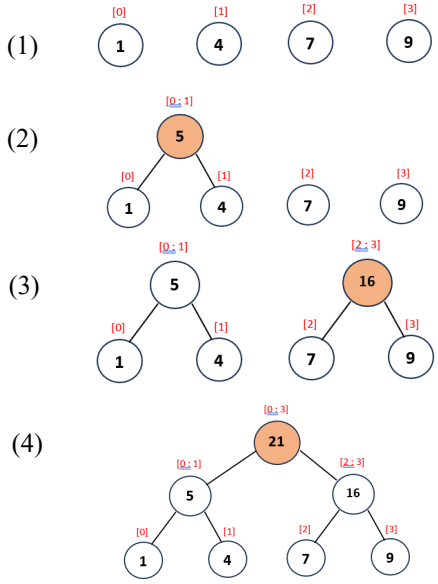
Fungsi *build* digunakan untuk membangun pohon segmen sehingga pohon segmen siap digunakan. Sebelum memanggil fungsi *build*, data awal dimasukkan terlebih

dahulu pada indeks [$n/2 \ldots n$] dari *array*. Kemudian, fungsi *build* dapat dipanggil dengan implementasi sebagai berikut:

```
/* ********** Konstruktor ********** */
void BuildSegTree(SegTree *t) {
/* I.S. sembarang */
/* F.S. Segment tree yaitu t berhasil terbangun, dengan
    IdxType i;
    for(i = NEFF(*t) - 1; i > 0; --i) {
        ELMT(*t, i) = ELMT(*t, i*2) + ELMT(*t, i*2 + 1);
    }
}
```

Gambar 9. *Source code* dari fungsi *build*
Sumber: Dokumen Penulis

Fungsi tersebut membangun pohon dari bawah ke atas, yakni mulai dari daun terus membangun simpul *parent* yang berisi penjumlahan dua anaknya, hingga sampai di simpul akar pada indeks *T*[1].



Gambar 10. Alur kerja dari fungsi *build*
Sumber: Dokumen Penulis

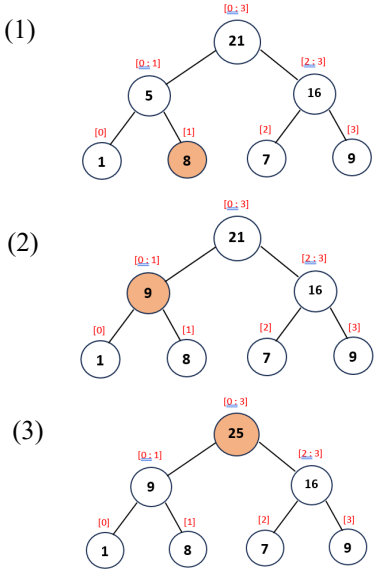
2) Modify

Fungsi *modify* digunakan untuk mengubah salah satu nilai daun pohon segmen yang menyimpan nilai tiap elemen pada data awal. Berikut implementasi dari fungsi *modify*:

```
/* ********** Primitif Lain  ********** */
void modify(SegTree *t, IdxType pos, ElType val) {
/* I.S. t terdefinisi */
/* F.S. Melakukan increment atau decrement pada nilai data
    ELMT(*t, pos + NEFF(*t)) += val;

    IdxType i;
    for(i = pos/2; i > 0; i /= 2) {
        ELMT(*t, i) = ELMT(*t, i*2) + ELMT(*t, i*2 + 1);
    }
}
```

Gambar 11. *Source code* dari fungsi *modify*
Sumber: Dokumen Penulis

Fungsi tersebut bekerja dengan cara mengubah nilai daun terlebih dahulu, kemudian naik ke atas mengubah nilai semua simpul leluhurnya hingga sampai ke simpul akar.



Gambar 12. Alur kerja dari fungsi *modify*
Sumber: Dokumen Penulis

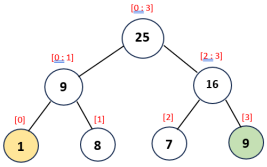
1) Query

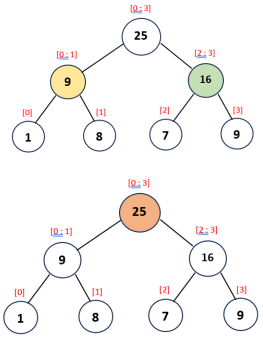
Fungsi *query* digunakan untuk mencari jumlah nilai tiap elemen pada interval [l : r], $0 \leq l \leq r <$ n. Berikut implementasi dari fungsi *query*:

```
ElType query(SegTree t, IdxType l, IdxType r) {
/* I.S. t terdefinisi */
/* F.S. Mengembalikan jumlah nilai elemen-elemen
    l += NEFF(t); r += NEFF(t) + 1;

    ElType result = 0;
    for(; l < r; l /= 2, r /= 2) {
        if (l % 2 == 1) {
            result += ELMT(t, l);
            l++;
        }
        if (r % 2 == 1) {
            r--;
            result += ELMT(t, r);
        }
    }

    return result;
}
```

Gambar 13. *Source code* dari fungsi *query*
Sumber: Dokumen Penulis

Fungsi tersebut menggunakan teknik dua *pointer*, *l* dan *r*, untuk membatasi interval pencarian. Tiap iterasi, *l* dan *r* akan naik satu level. Bila pada suatu waktu *l* menunjuk simpul kanan, maka *l* akan menjumlahkan nilai simpul yang ditunjuknya lalu pindah ke subpohon kanan. Sedangkan bila *r* menunjuk simpul kanan, maka *r* akan pindah ke subpohon kiri terlebih dahulu sebelum kemudian menjumlahkan nilai simpul yang ditunjuknya.





Gambar 14. Alur kerja dari fungsi *query*
Sumber: Dokumen Penulis

IV. HASIL DAN PEMBAHASAN

A. Hasil Percobaan

Hasil perhitungan waktu dari pohon segmen dan *array* untuk tiap fitur (*update* dan *query*) adalah sebagai berikut:

1) *Update*

```
Selamat datang di program simulasi kecil Perpustakaan Matdis!
Pilih aksi yang diinginkan:
 ____________________________________
| 1. Pinjam Buku                     |
| 2. Data Peminjaman Buku            |
 ____________________________________


Pilihan menu: 1

Silahkan pilih struktur data yang diinginkan:
 ______________________________
| 1. Segment Tree              |
| 2. Array                     |
 ______________________ ______

Pilihan menu: 1

Masukkan nama buku yang ingin dipinjam: Computer Programming 4
Anda telah berhasil meminjam buku Computer Programming 4!
Jumlah buku Computer Programming 4 yang dipinjam: 1
Waktu eksekusi: 1 microseconds
```

Gambar 15. Hasil eksekusi fungsi *update* dengan pohon segmen
Sumber: Dokumen Penulis

```
Selamat datang di program simulasi kecil Perpustakaan Matdis!
Pilih aksi yang diinginkan:
 ____________________________________
| 1. Pinjam Buku                     |
| 2. Data Peminjaman Buku            |
 ____________________________________


Pilihan menu: 1

Silahkan pilih struktur data yang diinginkan:
 ______________________________
| 1. Segment Tree              |
| 2. Array                     |
 ______________________ ______


Pilihan menu: 2

Masukkan nama buku yang ingin dipinjam: Computer Programming 4
Anda telah berhasil meminjam buku Computer Programming 4!
Jumlah buku Computer Programming 4 yang dipinjam: 1
Waktu eksekusi: 0 microseconds
```

Gambar 16. Hasil eksekusi fungsi *update* dengan *array*
Sumber: Dokumen Penulis

Tabel 3. Waktu eksekusi fitur *update*

Struktur Data	Waktu
Pohon Segmen	1 microseconds
Array	0 microseconds

2) *Query*

```
Selamat datang di program simulasi kecil Perpustakaan Matdis!
Pilih aksi yang diinginkan:
 ____________________________________
| 1. Pinjam Buku                     |
| 2. Data Peminjaman Buku            |
 ____________________________________


Pilihan menu: 2

Silahkan pilih struktur data yang diinginkan:
 ______________________________
| 1. Segment Tree              |
| 2. Array                     |
 ______________________ ______


Pilihan menu: 1

Masukkan range entri buku yang ingin dilihat:
Entri awal: 1
Entri akhir: 100000
Jumlah buku yang sedang dipinjam dari entri 1-100000: 208
Waktu eksekusi: 129 microseconds
```

Gambar 17. Hasil eksekusi fungsi *query* dengan pohon segmen
Sumber: Dokumen Penulis

```
Selamat datang di program simulasi kecil Perpustakaan Matdis!
Pilih aksi yang diinginkan:
 ____________________________________
| 1. Pinjam Buku                     |
| 2. Data Peminjaman Buku            |
 ____________________________________


Pilihan menu: 2

Silahkan pilih struktur data yang diinginkan:
 ______________________________
| 1. Segment Tree              |
| 2. Array                     |
 ______________________ ______


Pilihan menu: 2

Masukkan range entri buku yang ingin dilihat:
Entri awal: 1
Entri akhir: 100000
Jumlah buku yang sedang dipinjam dari entri 1-100000: 209
Waktu eksekusi: 183 microseconds
```

Gambar 17. Hasil eksekusi fungsi *query* dengan *array*
Sumber: Dokumen Penulis

Tabel 4. Waktu eksekusi fitur *query*

Struktur Data	Waktu
Pohon Segmen	129 microseconds
Array	183 microseconds

B. Analisis Kompleksitas Algoritma

Pada fitur *update*, *array* memerlukan waktu yang lebih sedikit dari pohon segmen (0 microseconds vs 1 microseconds). Hal ini sangat wajar, karena akses pada *array* hanya memerlukan 1 langkah (O(1)), sedangkan walaupun akses pada pohon segmen dengan representasi *array* juga 1 langkah (O(1)), selain mengubah nilai daun pohon segmen juga harus mengubah semua simpul leluhur dari daun tersebut. Karena pohon segmen memiliki tinggi pohon O(*log n*), maka kompleksitas waktu dari fitur update pada pohon segmen adalah O(*log n*).

Sedangkan pada fitur *query*, *array* memerlukan waktu yang lebih banyak dari pohon segmen (183 microseconds vs 129 microseconds). Hal ini juga wajar, karena untuk mencari penjumlahan nilai dari suatu interval di *array* harus menjumlahkan satu per satu tiap elemen yang ada di interval tersebut, sehingga kasus terburuknya memerlukan waktu O(n). Di sisi lain, pohon segmen hanya perlu menjumlahkan beberapa simpul yang intervalnya termasuk dalam interval query, dan karena selisih range simpul pada kedalaman *k* dan *k*+1 adalah ½, maka penjumlahan maksimal yang dilakukan adalah *log n*. Jadi, kompleksitas algoritma untuk fitur query pada pohon segmen adalah O(log n).

Selain itu, dilihat dari sisi kompleksitas memorinya pohon

segmen membutuhkan memori yang lebih besar dari array, yaitu maksimal 2*n untuk masukan data berukuran n.

V. Kesimpulan

Pohon Segmen merupakan salah satu struktur data yang dapat menjalankan dua buah fitur dengan cepat, yaitu *update* dan *query*, keduanya dalam waktu O($log\ n$). Untuk fitur *update* pohon segmen memang lebih lambat dibandingkan array yang memiliki kompleksitas waktu O(1), namun untuk fitur *query* pohon segmen lebih cepat dari array yang memerlukan waktu O(n). Dalam percobaan ini, untuk data masukan N = 100000 diperoleh pohon segmen menjalankan update dengan waktu 1 microseconds, sedangkan array memerlukan waktu 0 microseconds. Namun untuk fitur range query, pohon segmen memerlukan waktu 129 microseconds dibandingkan array yang memerlukan waktu 183 microseconds. Selisih waktu ini tentunya akan jauh lebih besar lagi untuk masukan data N yang lebih besar. Oleh karena itu, pohon segmen dapat diterapkan pada permasalahan pendataan peminjaman buku perpustakaan yang memerlukan operasi *range query*.

VI. Ucapan Terima Kasih

Penulis menyampaikan ucapan terima kasih kepada:

1. Tuhan Yang Maha Esa atas berkatnya sehingga penulis dapat menyelesaikan makalah ini dengan baik,
2. Orangtua penulis yang senantiasa selalu mendukung penulis,
3. Bapak Dr. Ir. Rinaldi Munir, M.T., Ibu Fariska Zakhralativa Ruskanda S.T.,M.T, dan Ibu Dr. Nur Ulfa Maulidevi S.T., M.Sc selaku dosen mata kuliah IF2120 Matematika Diskrit yang telah membimbing penulis dalam menulis jurnal dan mengajarkan segala ilmu yang relevan dengan topik penulis.

Akhir kata, penulis berharap makalah ini dapat bermanfaat tidak hanya bagi penulis namun juga bagi pihak yang membacanya.

Referensi

[1] Munir, Rinaldi. (2023). “Graf: Bagian 1”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf (diakses tanggal 10 Desember 2023)

[2] Munir, Rinaldi. (2023). “Pohon: Bagian 2”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/23-Pohon-Bag2-2023.pdf (diakses tanggal 10 Desember 2023)

[3] Aggarwal, Yash. (2020). “Segment Tree”. https://iq.opengenus.org/segment-tree/ (diakses tanggal 10 Desember 2023)

[4] J, Keshav. (2021). “Node and range updates in Segment Tree using C++”. https://www.codespeedy.com/node-range-updates-in-segment-tree-using-cpp/ (diakses pada 11 Desember 2023)

[5] J. Keshav. (2021). “Sum of range in Segment Tree using C++”. https://www.codespeedy.com/sum-of-range-in-segment-tree-in-cpp/ (diakses pada 11 Desember 2023)

[6] Masood, Abdur. (2017). “Understanding Time Complexity and its Importance in Technology”. https://medium.com/@abdurrafeymasood/understanding-time-complexity-and-its-importance-in-technology-8279f72d1c6a (diakses pada 11 Desember 2023)

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, 11 Desember 2023

Irfan Sidiq Permana 13522007