Optimisasi Penghitungan Total Waktu Layar dalam Rentang Tertentu pada Aplikasi TikTok dengan Menggunakan Segment Tree

Brian Kheng - 13521049
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13521049@std.stei.itb.ac.id

***Abstrak*—Makalah ini mengimplementasikan *Segment Tree* untuk mengoptimalkan penghitungan total waktu layar dalam rentang tertentu pada aplikasi TikTok. *Segment Tree* digunakan untuk mempercepat dan efisien mengakses data dalam penghitungan tersebut. Metode *Divide and Conquer* pada *Segment Tree* membagi persoalan menjadi upa-persoalan yang lebih kecil, menyelesaikan masing-masing upa-persoalan, dan menggabungkan solusi untuk mencapai solusi akhir. Penelitian ini memberikan panduan dan kontribusi dalam meningkatkan performa dan fitur aplikasi TikTok melalui penggunaan *Segment Tree* dalam Bahasa C++.**

Kata Kunci—Segment Tree, Divide and Conquer, TikTok, C++

I. PENDAHULUAN

A. Latar Belakang

Aplikasi TikTok telah menjadi fenomena global dalam beberapa tahun terakhir. Aplikasi ini memungkinkan pengguna untuk membuat dan berbagi video pendek dengan beragam konten kreatif. Dalam penggunaan sehari-hari, pengguna TikTok dapat menghabiskan banyak waktu untuk menonton video dalam rentang waktu tertentu.

Namun, dalam konteks pengembangan dan pengoptimalan aplikasi TikTok, perlu ada perhatian khusus terhadap penghitungan total waktu layar dalam rentang tertentu. Penghitungan ini penting untuk memahami dan menganalisis perilaku pengguna, serta untuk melakukan peningkatan dan penyesuaian pada fitur dan konten yang disajikan.

B. Permasalahan

Penghitungan total waktu layar pada rentang tertentu pada aplikasi TikTok bisa menjadi tugas yang kompleks dan memakan waktu jika dilakukan secara langsung pada dataset yang besar. Proses penghitungan tersebut memerlukan akses dan pencarian data yang efisien, terutama ketika dataset menjadi semakin besar dan semakin kompleks.

Oleh karena itu, diperlukan sebuah metode atau teknik yang efisien untuk mengoptimalkan penghitungan total waktu layar dalam rentang tertentu pada aplikasi TikTok. Dalam hal ini, penggunaan *Segment Tree* dapat menjadi solusi yang efektif dan efisien.

C. Tujuan Penelitian

Penelitian ini bertujuan untuk mengimplementasikan *Segment Tree* dalam penghitungan total waktu layar dalam rentang tertentu pada penggunaan aplikasi TikTok. Dengan menggunakan *Segment Tree*, diharapkan dapat meningkatkan efisiensi dan kecepatan penghitungan total waktu layar, sehingga memungkinkan pengembang aplikasi TikTok untuk melakukan analisis data yang lebih cepat dan akurat.

D. Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

- Memberikan pemahaman yang lebih baik tentang penggunaan aplikasi TikTok dan pola interaksi pengguna.
- Mengoptimalkan proses penghitungan total waktu layar dalam rentang tertentu.
- Meningkatkan efisiensi pengolahan data dalam aplikasi TikTok.
- Memberikan panduan bagi pengembang aplikasi TikTok dalam meningkatkan performa dan fitur aplikasi.

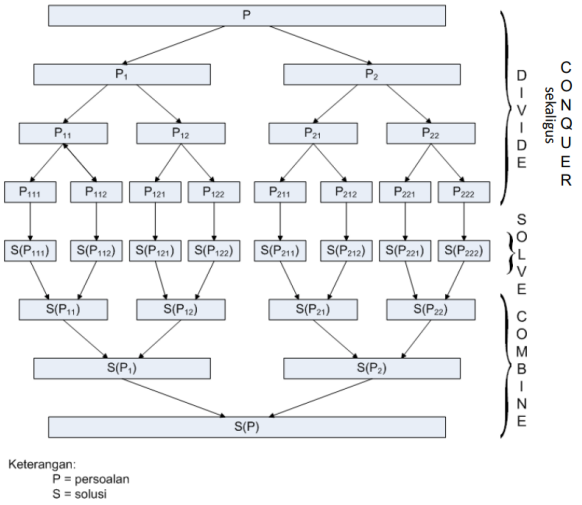
E. Ruang Lingkup Penelitian

Penelitian ini akan fokus pada implementasi *Segment Tree* untuk mengoptimalkan penghitungan total waktu layar dalam rentang tertentu pada penggunaan aplikasi TikTok. Penelitian ini tidak akan membahas implementasi detil aplikasi TikTok secara keseluruhan, melainkan hanya pada aspek penghitungan waktu layar.

II. LANDASAN TEORI

A. *Divide and Conquer*

Metode pemecahan masalah dengan pendekatan *Divide and Conquer* terdiri dari tiga langkah utama. Pertama, langkah *Divide* membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula, namun berukuran lebih kecil, diharapkan idealnya hampir memiliki ukuran yang sama. Kedua, langkah *Conquer* melibatkan penyelesaian masing-masing upa-persoalan, baik secara langsung jika ukurannya sudah cukup kecil, maupun secara rekursif jika masih memiliki ukuran yang besar. Terakhir, langkah *Combine* melibatkan penggabungan solusi dari masing-masing upa-persoalan sehingga membentuk solusi untuk persoalan semula.



Gambar 1. *Divide and Conquer*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf)

Metode *Divide and Conquer* telah banyak diterapkan dalam berbagai bidang, termasuk pemrosesan data dan optimisasi. Salah satu contoh penerapan *Divide and Conquer* adalah dalam pengolahan citra dan grafik. Penelitian oleh Liu et al. (2018) mengenai pemrosesan citra menggunakan pendekatan *Divide and Conquer* menunjukkan peningkatan signifikan dalam kecepatan dan efisiensi pemrosesan citra kompleks. Dalam studi ini, pemecahan masalah citra menjadi upa-persoalan yang lebih kecil memungkinkan pemrosesan paralel yang efisien, menghasilkan hasil yang akurat dalam waktu yang lebih singkat.

Selain itu, *Divide and Conquer* juga telah diterapkan dalam algoritma pemetaan dan pencarian. Penelitian oleh Zhang et al. (2019) dalam pemetaan genom menggunakan pendekatan *Divide and Conquer* menunjukkan efektivitasnya dalam menangani kompleksitas data genom yang besar. Melalui pemecahan masalah menjadi upa-persoalan yang lebih kecil, pengolahan dan analisis genom dapat dilakukan dengan lebih efisien, menghasilkan solusi yang akurat dan cepat.

Kompleksitas algoritma *Divide and Conquer*:

$$T(n) = g(n),\ n \le n_0$$
$$T(n) = T(n_1) + T(n_2) \ldots + T(n_r) + f(n),\ n > n_0$$

- $T(n)$: Kompleksitas waktu penyelesaian persoalan p yang berukuran n.
- $g(n)$: Kompleksitas waktu untuk *Solve* jika n sudah berukuran kecil.
- $T(n_1) + T(n_2) \ldots + T(n_r)$: Kompleksitas waktu untuk memproses setiap upa-persoalan.
- $f(n)$: Kompleksitas waktu untuk *Combine* solusi dari masing-masing upa-persoalan.

B. *Segment Tree*

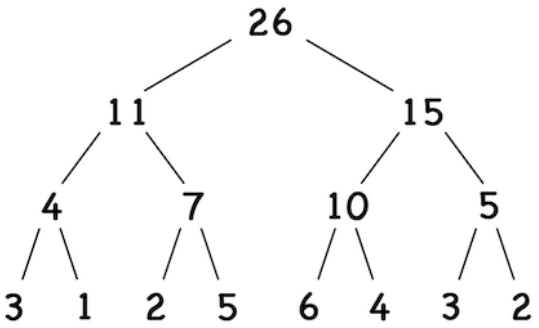
Segment Tree adalah struktur data yang mendukung dua operasi yakni, pemrosesan pertanyaan rentang dan pembaruan nilai dalam sebuah larik. *Segment tree* dapat mendukung pertanyaan penjumlahan, minimum, dan maksimum, serta banyak pertanyaan lainnya. Kedua operasi tersebut dapat dilakukan dalam waktu $\mathcal{O}(\log n)$.

Struktur dari *segment tree*:

1. Misalkan terdapat suatu larik sebagai berikut:

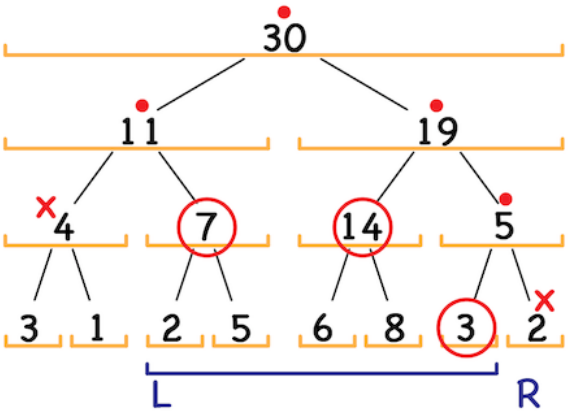
Gambar 2. Suatu larik
(Sumber: https://espresso.codeforces.com/62a4ed9368574e5a3a879804287c51434b5c29a3.png)

2. Dari larik di atas, dapat dikonstruksi suatu pohon biner, dengan daunnya sebagai elemen dari larik asal, dan masing-masing simpul internal sebagai penjumlahan dari anak-anaknya.



Gambar 3. Hasil konstruksi *segment tree* dari larik
(Sumber: https://espresso.codeforces.com/35bc9067266d25646e8a8a5e4ee10159df770d48.png)

3. Dari *segment tree* di atas, kita dapat mencari jumlah dari rentang tertentu $[a, b]$. Sebagai contoh, untuk mendapatkan jumlah dari rentang [2, 6] pada larik tersebut adalah sebagai berikut:



Gambar 4. Pencarian jumlah dari rentang [2, 6]

(Sumber: https://espresso.codeforces.com/0349d47f2df242db7d9e04098f39fa6bdcacddb3.png)

III. ANALISIS

A. *Data Penggunaan Aplikasi TikTok 30 hari terakhir*

Tanggal	Waktu Layar (menit)
19/04/2023	135
20/04/2023	96
21/04/2023	341
22/04/2023	302
23/04/2023	239
24/04/2023	287
25/04/2023	251
26/04/2023	192
27/04/2023	208
28/04/2023	277
29/04/2023	70
30/04/2023	202
01/05/2023	276
02/05/2023	244
03/05/2023	342
04/05/2023	188
05/05/2023	121
06/05/2023	68
07/05/2023	112
08/05/2023	477
09/05/2023	316
10/05/2023	202
11/05/2023	265
12/05/2023	236
13/05/2023	330
14/05/2023	154
15/05/2023	180
16/05/2023	135
17/05/2023	88
18/05/2023	60

Tabel 1. Data waktu layar penggunaan aplikasi TikTok 30 hari terakhir
(Sumber: Dokumen pribadi)

B. *Implementasi Segment Tree*

```cpp
#include <bits/stdc++.h>

using namespace std;

int seg[1 << 21];

int get(int idx, int l, int r,
        int x, int y)
{
    if(l > y || r < x)
        return 0; // NEUTRAL_ELEMENT
    if(l >= x && r <= y)
        return seg[idx];

    int mid = (l + r) / 2;
    int lc = 2 * idx;
    int rc = 2 * idx + 1;

    return get(lc, l, mid, x, y) +
           get(rc, mid + 1, r, x, y);
}

void upd(int idx, int l, int r,
         int x, int val)
{
    if(l > x || r < x) return;
    if(l == x && r == x){
        seg[idx] = val;
        return;
    }

    int mid = (l + r) / 2;
    int lc = 2 * idx;
    int rc = 2 * idx + 1;

    upd(lc, l, mid, x, val);
    upd(rc, mid + 1, r, x, val);

    seg[idx] = seg[lc] + seg[rc];
}

int main()
```

```
{
    ios::sync_with_stdio(0); cin.tie(0);

    int waktu_layar[30] =
    {
        135, 96, 341, 302, 239, 287, 251,
        192, 208, 277, 70, 202, 276, 244,
        342, 188, 121, 68, 112, 477, 316,
        202, 265, 236, 330, 154, 180, 135,
        88, 60
    };

    // Segment tree initiation
    for(int i = 0; i < 30; i++){
        upd(1, 0, (1 << 20) - 1, i,
            waktu_layar[i]);
    }

    // Get sum on segment [0..29]
    cout << get(1, 0, (1 << 20) - 1, 0, 29)
         << " menit\n";

    return 0;
}
```

C. Analisis Kode Program

1. Dilakukan inisiasi *header library* dan *variable* yang akan digunakan menggunakan struktur data yang efektif.

```
#include <bits/stdc++.h>

using namespace std;

int seg[1 << 21];
```

2. Mendefinisikan fungsi ***get*** yang menerima input bertipe integer untuk parameter idx: posisi simpul, l: batas kiri rentang idx, r: batas kanan rentang idx, x: batas kiri rentang pencarian, dan y: batas kanan rentang pencarian dan mengeluarkan output hasil penjumlahan dari rentang pencarian [x, y] yang bertipe integer. Fungsi ini menerapkan algoritma *Divide and Conquer*, Basisnya meliputi: jika rentang pencarian berada di kanan atau kiri dari batas rentang idx maka akan me-return 0, jika rentang idx berada di dalam rentang pencarian maka akan me-return seg[idx] yang merupakan jumlah total dari rentang idx. Jika tidak memenuhi keduanya, maka akan dilakukan *DIVIDE* rentang idx menjadi 2 bagian sama rata dan memanggil kembali fungsi *get* untuk masing-masing bagian. Kemudian, dilakukan *COMBINE* hasil dari 2 bagian yang telah dibagi sebelumnya sebagai hasil return-nya.

```
int get(int idx, int l, int r,
        int x, int y)
{
    if(l > y || r < x)
        return 0; // NEUTRAL_ELEMENT
    if(l >= x && r <= y)
        return seg[idx];

    int mid = (l + r) / 2;
    int lc = 2 * idx;
    int rc = 2 * idx + 1;

    return get(lc, l, mid, x, y) +
           get(rc, mid + 1, r, x, y);
}
```

3. Mendefinisikan fungsi ***upd*** yang menerima input bertipe integer untuk parameter idx: posisi simpul, l: batas kiri dari idx, r: batas kanan dari idx, x: index pada larik yang ingin diupdate, dan val: nilai yang diinginkan. Fungsi ini menerapkan algoritma *Divide and Conquer*, Basisnya meliputi: jika rentang idx bukan merupakan rentang pencarian [x, x] maka akan di-return, jika rentang idx merupakan rentang pencarian [x, x] maka nilai dari seg[idx] akan digantikan dengan val. Jika tidak memenuhi keduanya, maka akan dilakukan *DIVIDE* rentang idx menjadi 2 bagian sama rata dan memanggil kembali fungsi *upd* untuk masing-masing bagian. Kemudian, dilakukan *COMBINE* hasil perubahannya dengan seg[idx] = seg[lc] + seg[rc].

```
void upd(int idx, int l, int r,
         int x, int val)
{
    if(l > x || r < x) return;
    if(l == x && r == x){
        seg[idx] = val;
        return;
    }

    int mid = (l + r) / 2;
    int lc = 2 * idx;
    int rc = 2 * idx + 1;

    upd(lc, l, mid, x, val);
```

```
    upd(rc, mid + 1, r, x, val);

    seg[idx] = seg[lc] + seg[rc];
}
```

4. Mendefinisikan fungsi ***main*** untuk menjalankan program dan mencari jumlah total waktu layar aplikasi TikTok selama 30 hari terakhir. Dilakukan inisiasi awal untuk mengkonstruksi *segment tree* dengan memanfaatkan fungsi *upd* untuk masing-masing waktu layar. Dilanjutkan dengan menggunakan fungsi *get* untuk mendapatkan rentang pencarian [0, 29]. Didapatkan hasil output: **6394 menit**.

```
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    int waktu_layar[30] =
    {
        135, 96, 341, 302, 239,
        287, 251, 192, 208, 277,
        70, 202, 276, 244, 342,
        188, 121, 68, 112, 477,
        316, 202, 265, 236, 330,
        154, 180, 135, 88, 60
    };

    // Segment tree initiation
    for(int i = 0; i < 30; i++){
        upd(1, 0, (1 << 20) - 1, i,
            waktu_layar[i]);
    }

    // Get sum on segment [0..29]
    cout << get(1, 0, (1 << 20) - 1,
                0, 29)
         << " menit\n";

    return 0;
}
```

D. Perbandingan Waktu Komputasi Segment Tree dan Brute Force

Untuk N yang cukup besar seperti 1.000.000.000, diperlukan waktu yang cukup lama ketika dikomputasi menggunakan *brute force* yakni sekitar 3.9 sekon, sedangkan dengan menggunakan *segment tree* hanya diperlukan 0.7 sekon saja.

```
N = 1000000000
Time taken using segment tree: 734 ms
Time taken using bruteforce: 3921 ms
```

Gambar 5. Waktu komputasi menggunakan algoritma *segment tree* dan *brute force* untuk N besar
(Sumber: Dokumen pribadi)

IV. KESIMPULAN

Berdasarkan penelitian ini, dapat disimpulkan bahwa *Segment Tree* merupakan solusi yang efektif untuk mengoptimalkan penghitungan total waktu layar dalam rentang tertentu pada aplikasi TikTok. Implementasi *Segment Tree* dapat menghasilkan peningkatan kecepatan komputasi hingga 5 kali lipat dibandingkan dengan pendekatan *brute force*, terutama ketika jumlah data yang besar dan sering diakses pada rentang tertentu. Analisis yang dilakukan pada data penggunaan TikTok selama 30 hari terakhir menunjukkan bahwa total waktu layar yang digunakan adalah 6394 menit atau setara dengan 107 jam. Penggunaan *Segment Tree* dapat memberikan kontribusi penting dalam meningkatkan efisiensi dan kinerja aplikasi TikTok dalam menghitung waktu layar pengguna pada rentang waktu yang ditentukan.

UCAPAN TERIMA KASIH

Terima kasih kepada semua pihak yang telah membantu dalam pembuatan makalah ini. Peneliti menghargai semua masukan dan dukungan yang telah diberikan selama proses penyusunan makalah ini. Terima kasih juga kepada Bapak Dr. Ir. Rinaldi, M.T. selaku dosen pengampu mata kuliah IF2211. Peneliti juga mengucapkan terima kasih kepada semua yang telah membaca makalah ini dan memberikan saran dan masukan yang berguna. Tanpa bantuan dan dukungan dari Anda semua, makalah ini tidak akan dapat terselesaikan dengan baik. Terima kasih atas semua bantuan dan dukungan Anda.

REFERENSI

[1] Laaksonen, Antti. (2019). *Competitive Programmer's Handbook*.

[2] Liu, Z., Wang, L., & Xu, Y. (2018). *An Image Segmentation Algorithm Based on Divide and Conquer Strategy. In Proceedings of the 2018 4th International Conference on Information Management Systems*.

[3] Mavrin, Pavel. (2020). *Segment Tree, part 1*. https://codeforces.com/edu/course/2/lesson/4/1. Diakses pada 18 Mei 2023, pukul 12.07 WIB.

[4] Munir, Rinaldi. (2021). *Algoritma Divide and Conquer (Bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf. Diakses pada 18 Mei 2021, pukul 10.46 WIB.

[5] Zhang, D., Liu, S., Yang, X., & Lu, C. (2019). *Divide and Conquer: A Fast Mapping Algorithm for Large-Scale Genomic Sequencing Data. Frontiers in Genetics*.

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, 22 Mei 2023

Brian Kheng - 13521049