

Analisis Sistem Pencarian Aplikasi BunniesBase dengan Pendekatan Iteratif dan Rekursif

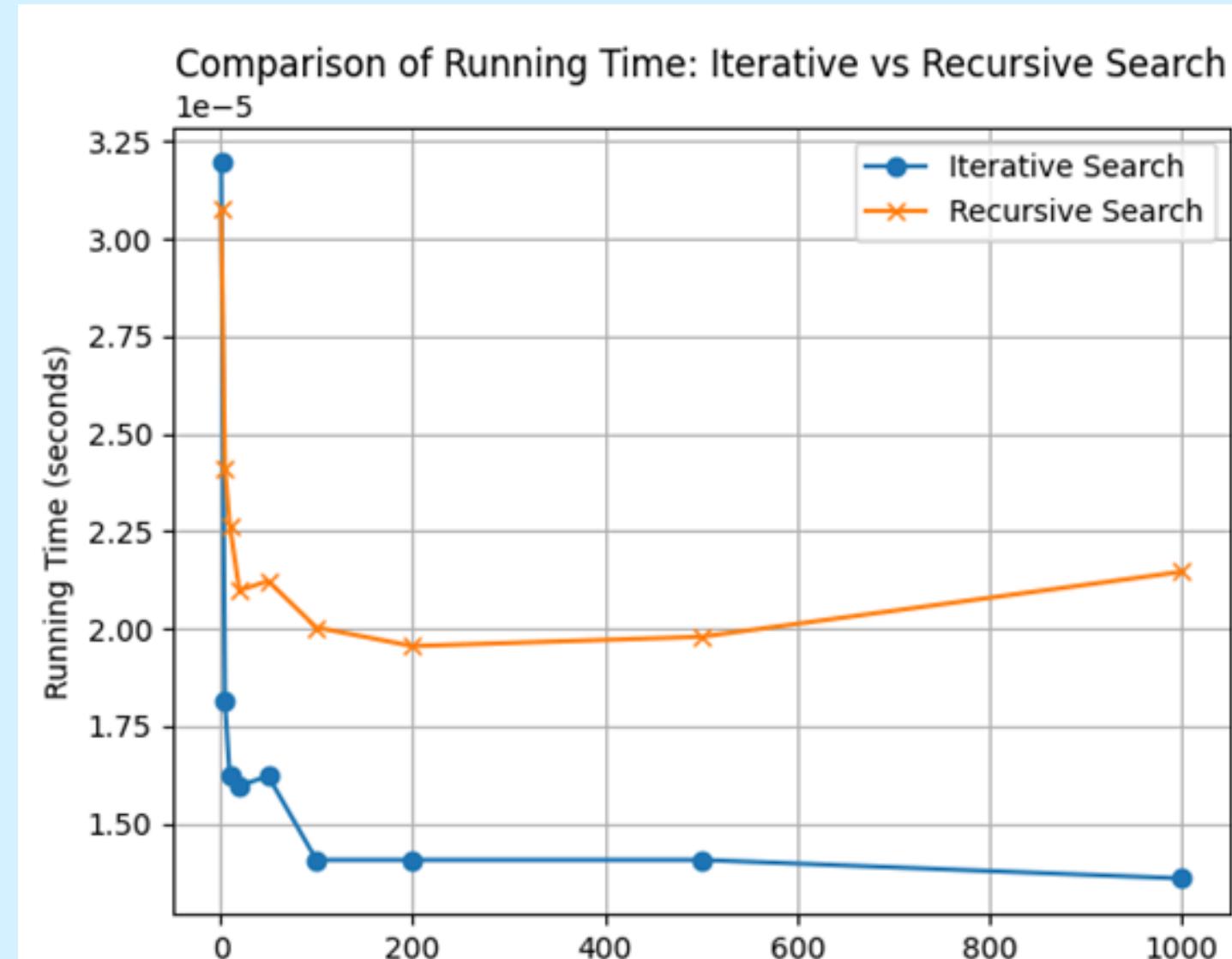
Studi Kasus Algoritma Pencarian di Platform Musik Digital

Deskripsi

Aplikasi BunniesBase adalah platform yang didedikasikan bagi para penggemar girlgroup asal Korea Selatan, New Jeans. Dalam aplikasi ini, pengguna dapat dengan mudah menemukan berbagai lagu dari New Jeans, baik melalui pencarian berdasarkan judul lagu maupun nama album. Salah satu fitur utama yang membedakan aplikasi ini adalah analisis mendalam terkait algoritma pencarian yang digunakan. Akan dilakukan pencarian menggunakan dua pendekatan utama: **iteratif** dan **rekursif**.

Pendekatan iteratif menawarkan kecepatan dan efisiensi dalam pencarian, sementara pendekatan rekursif memberikan fleksibilitas dan solusi yang elegan untuk kasus yang lebih kompleks.

Melalui BunniesBase, penggemar tidak hanya bisa menikmati musik favorit mereka, tetapi juga mendapatkan wawasan tambahan tentang cara kerja algoritma pencarian di balik layar. Aplikasi ini dirancang untuk memberikan pengalaman yang kaya dan informatif, menjadikannya lebih dari sekedar platform streaming musik biasa, tetapi juga alat pembelajaran bagi mereka yang tertarik dengan teknologi di balik pencarian musik digital.



Grafik Perbandingan running time



Analisis Perbandingan Kedua Algoritma

Algoritma Pencarian Iteratif

- Ukuran input:** n , di mana n adalah jumlah elemen dalam `self.songs`.
- Operasi dasar:** Perbandingan string dan iterasi pada setiap elemen dalam koleksi `self.songs`. (`search_term.lower() in song.lower()` or `search_term.lower() in album.lower()`)
- Kasus Terbaik:** $O(1)$ - Jika pencarian menemukan hasil pada iterasi pertama.
- Kasus Terburuk:** Kata kunci pencarian tidak ditemukan, atau ditemukan pada elemen terakhir.

$$C_{worst}(n) = \sum_{i=1}^n 1 = n \in O(n)$$

Kasus Rata-rata:

$$C_{worst}(n) = \frac{1}{n} \sum_{i=1}^n i = \frac{1}{n} \left(\frac{n(n+1)}{2} \right) = \frac{n(n+1)}{2n} = \frac{n+1}{2} \in O(n)$$

Jadi, Kompleksitas waktu dari algoritma ini adalah $O(n)$.

Algoritma Pencarian Rekursif

- Ukuran input:** n , di mana n adalah jumlah elemen dalam `self.songs`.
- Operasi dasar:** Perbandingan string dan iterasi pada setiap elemen dalam koleksi `self.songs`. (`search_term.lower() in song.lower()` or `search_term.lower() in album.lower()`)

Fungsi `search_recursive` memiliki dua kasus dasar:

- Kasus dasar:** Jika index sudah mencapai panjang `song_list`, kembalikan `result`.
- Kasus rekursif:** Jika index lebih kecil dari panjang `song_list`, periksa elemen saat ini dan panggil fungsi untuk indeks berikutnya.

Recurrence relation untuk $T(n)$ adalah:

$$T(n) = \begin{cases} 0, & \text{Jika } n = 0 \\ T(n-1) + C, & \text{Jika } n > 0 \end{cases}$$

Di mana C adalah konstanta yang mewakili jumlah operasi dasar yang dilakukan dalam satu iterasi (misalnya, perbandingan string dan penambahan elemen ke dalam `result`).

$$\begin{aligned} T(n) &= T(n-1) + C \\ &= (T(n-2) + C) + C \\ &= T(n-2) + 2C \\ &= (T(n-3) + C) + 2C \\ &= T(n-3) + 3C \\ &= T(n-i) + iC \\ &= T(0) + nC \\ &= nC \end{aligned}$$

Fungsi rekursif $M(n)$ memiliki kompleksitas waktu $O(n)$, di mana n adalah ukuran input. Ini berarti waktu eksekusi dari fungsi tumbuh secara linier dengan n .

Avriel Fitria Rachma Suryantari (2311102036)

Widari Dwi Hayati (2311102060)

Algoritma yang Dipilih untuk Menyelesaikan Permasalahan

Algoritma Pencarian Iteratif

- Deskripsi:** Algoritma pencarian iteratif adalah pendekatan di mana pencarian dilakukan dengan mengiterasi elemen-elemen dalam koleksi data satu per satu hingga ditemukan kecocokan dengan kata kunci pencarian atau hingga semua elemen telah diperiksa. Metode ini menggunakan loop (seperti `for` atau `while`) untuk melakukan iterasi.
- Implementasi:** Dalam aplikasi BunniesBase, pencarian iteratif diimplementasikan dalam metode `search_iterative` yang menggunakan loop `for` untuk mengiterasi setiap pasangan (lagu, album) dalam koleksi `self.songs`.

```
def search_iterative(self, search_term):  
    result = []  
    for song, album in self.songs.items():  
        if search_term.lower() in song.lower() or search_term.lower() in album.lower():  
            result.append((song, album))  
    return result
```

- Proses:** Inisialisasi daftar `result` kosong untuk menyimpan hasil pencarian. Gunakan loop `for` untuk mengiterasi setiap pasangan (lagu, album) dalam `self.songs`. Periksa apakah kata kunci pencarian (`search_term`) ditemukan dalam judul lagu atau nama album (dengan mengabaikan huruf besar/kecil). Jika ditemukan kecocokan, tambahkan pasangan (lagu, album) ke dalam daftar `result`. Kembalikan daftar `result` setelah iterasi selesai. Kelebihan: Sederhana dan mudah diimplementasikan. Efisien untuk koleksi data dengan jumlah elemen yang relatif kecil. Kekurangan: Kurang efisien untuk koleksi data dengan jumlah elemen yang besar karena harus memeriksa setiap elemen satu per satu.

Algoritma Pencarian Rekursif

- Deskripsi:** Algoritma pencarian rekursif adalah metode di mana pencarian dilakukan dengan menggunakan pemanggilan fungsi secara rekursif untuk memeriksa setiap elemen dalam koleksi data. Setiap pemanggilan fungsi rekursif memeriksa satu elemen dan memanggil dirinya sendiri dengan elemen berikutnya hingga ditemukan kecocokan atau hingga semua elemen telah diperiksa.
- Implementasi dalam Kode:**

```
def search_recursive(self, search_term, song_list=None, index=0, result=None):  
    if result is None:  
        result = []  
    if song_list is None:  
        song_list = list(self.songs.items())  
  
    if index >= len(song_list):  
        return result  
  
    song, album = song_list[index]  
    if search_term.lower() in song.lower() or search_term.lower() in album.lower():  
        result.append((song, album))  
  
    return self.search_recursive(search_term, song_list, index + 1, result)
```

- Proses:** Jika `result` belum diinisialisasi, buat daftar kosong untuk `result`. Jika `song_list` belum diinisialisasi, buat daftar pasangan (lagu, album) dari `self.songs`. Jika index sudah mencapai panjang `song_list`, kembalikan `result` sebagai hasil pencarian. Ambil pasangan (lagu, album) pada indeks saat ini dari `song_list`. Periksa apakah kata kunci pencarian (`search_term`) ditemukan dalam judul lagu atau nama album, dengan mengabaikan huruf besar/kecil. Jika ditemukan kecocokan, tambahkan pasangan (lagu, album) ke dalam `result`. Panggil kembali `search_recursive` dengan index ditambah 1 hingga mencapai akhir `song_list`. Kelebihan: Lebih elegan dan mudah dipahami untuk masalah yang dapat dipecah menjadi submasalah yang lebih kecil. Dapat lebih fleksibel dalam menangani struktur data yang kompleks. Kekurangan: Lebih lambat dibandingkan dengan pendekatan iteratif karena overhead pemanggilan fungsi rekursif. Dapat menyebabkan stack overflow jika koleksi data sangat besar.



Referensi

- Khazraei, A., Kötzting, T., & Seidel, K. (2021). Towards a Map for Incremental Learning in the Limit from Positive and Negative Information. , 273-284. https://doi.org/10.1007/978-3-030-80049-9_25.
- Dijkstra, E. (2022). Recursive Programming. Edsger Wybe Dijkstra. <https://doi.org/10.1145/3544585.3544601>.