LAPORAN TUGAS BESAR ANALISIS KOMPLEKSITAS ALGORITMA

Analisis Perbandingan Kompleksitas Algoritma String Search: Implementasi Iteratif vs Rekursif

Disusun Oleh :

Ryan Maulana Bagus Putra – 103012430029

Azmi Hanif Fauzil Islami – 103012430018

Muhammad Rafiul Izzah – 103012430004

S-1 INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY BANDUNG

2025

1. Deskripsi Studi Kasus Permasalahan

Pencarian pola (*pattern matching*) di dalam teks merupakan salah satu persoalan fundamental dalam ilmu komputer. Masalah ini sering ditemukan dalam berbagai aplikasi, mulai dari fitur "Find" pada aplikasi pengolah kata, pencarian *keyword* pada mesin pencari, hingga analisis bioinformatika untuk mencocokkan urutan DNA.

Dalam tugas besar ini, permasalahan yang diangkat adalah pencarian keberadaan sebuah string pendek (*pattern*) di dalam string yang lebih panjang (*text*). Tujuannya adalah menemukan indeks kemunculan *pattern* tersebut di dalam *text*.

Studi kasus ini secara spesifik membandingkan dua pendekatan implementasi untuk algoritma Naive String Search (Pencarian Naif/Brute Force):

a. Pendekatan Iteratif: Menggunakan struktur perulangan (*looping*) bersarang untuk menggeser pola terhadap teks.
b. Pendekatan Rekursif: Menggunakan pemanggilan fungsi terhadap diri sendiri (*self-reference*) untuk menelusuri indeks teks tanpa menggunakan struktur *loop* eksplisit untuk pergeseran indeks utama.

Perbandingan ini dilakukan untuk menganalisis efisiensi *running time* dan penggunaan sumber daya (memori) antara kedua pendekatan tersebut ketika menangani ukuran masukan (*input size*) yang semakin membesar.

2. Deskripsi Dua Algoritma

a. Algoritma Naive Search Iteratif
   Algoritma ini menggunakan pendekatan *sliding window* dengan dua lapis perulangan (nested loop):
   i. Mekanisme: *Outer loop* menggeser posisi awal pengecekan dari indeks ke-0 hingga $n - m$ (dimana n adalah panjang teks dan m adalah panjang pola). *Inner loop* membandingkan karakter demi karakter antara *pattern* dan *text*.
   ii. Karakteristik: Algoritma ini bersifat *in-place* dan hanya membutuhkan memori tambahan konstan $O(1)$ untuk menyimpan variabel indeks.

Implementasi Kode (Python)

def naive_search_iterative(text, pattern):
  n = len(text)
  m = len(pattern)
  results = []
  for i in range(n - m + 1):
  match = True
  for j in range(m):
  if text[i + j] != pattern[j]:
  match = False
  break
  if match:
  results.append(i)
  return results

b. Algoritma Naive Search Rekursif

Algoritma ini menggantikan *outer loop* dengan pemanggilan fungsi secara rekursif:

i. Mekanisme: Fungsi menerima parameter `index` yang merepresentasikan posisi saat ini di teks.
   - *Base Case:* Jika index melebihi batas panjang teks yang mungkin $(n - m)$, rekursi berhenti.
   - *Recursive Step:* Program memeriksa kecocokan di `index` saat ini (logika sama dengan *inner loop*). Setelah selesai, fungsi memanggil dirinya sendiri dengan $index + 1$.

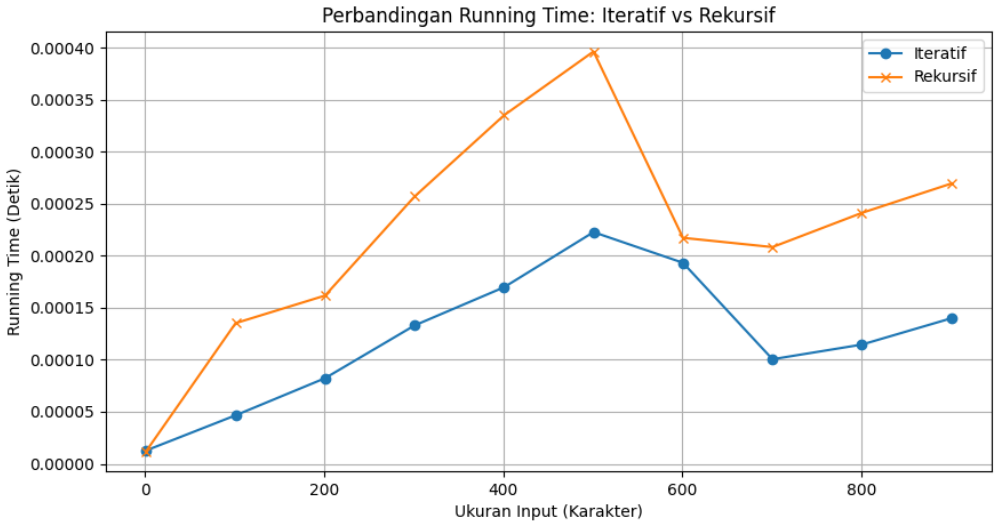
ii. Karakteristik: Setiap pemanggilan fungsi membuat *stack frame* baru di memori. Hal ini menyebabkan penggunaan memori bertambah secara linear $O(n)$ seiring kedalaman rekursi, yang berisiko menyebabkan *Stack Overflow* (RecursionError) pada input yang sangat besar jika limit rekursi tidak dinaikkan.

Implementasi Kode (Python)

```
def naive_search_recursive(text, pattern, index=0, results=None):
    if results is None: results = []
    n = len(text); m = len(pattern)

    # Base Case
    if index > n - m: return results

    # Logic Pengecekan (Sama seperti inner loop)
    match = True
    for j in range(m):
        if text[index + j] != pattern[j]:
            match = False; break
    if match: results.append(index)

    # Recursive Step
    return naive_search_recursive(text, pattern, index + 1, results)
```

3. Grafik Perbandingan Running Time

Berikut adalah hasil visualisasi eksperimen perbandingan *running time* antara implementasi iteratif dan rekursif dengan ukuran input bervariasi dari 100 hingga 4.500 karakter.



Grafik menunjukkan pertumbuhan linear $O(N)$ untuk kedua algoritma, namun kurva rekursif memiliki gradien yang lebih curam karena adanya overhead pemanggilan fungsi.

Keterangan Eksperimen:

- Sumbu X (Input): Ukuran teks (jumlah karakter).
- Sumbu Y (Time): Waktu eksekusi dalam detik.
- Garis Biru: Algoritma Iteratif.
- Garis Oranye: Algoritma Rekursif.
- Perangkat: Pengujian dilakukan menggunakan Python 3.11 pada lingkungan Windows.

4. Analisis Perbandingan Kedua Algoritma

Berdasarkan data grafik dan landasan teori, berikut adalah analisis perbandingan kedua algoritma:

a. Analisis Kompleksitas Waktu (Time Complexity)
   i. Secara teoritis (Big-O Notation), kedua algoritma memiliki kelas kompleksitas yang sama:
   ii. Worst Case: $O\big((N - M + 1) \times M\big)$ atau disederhanakan menjadi $O(N \times M)$.
   iii. Best Case: $O(N)$ (jika ketidakcocokan selalu ditemukan pada karakter pertama pola).

   Namun, secara empiris (berdasarkan grafik), terlihat perbedaan performa yang signifikan. Algoritma Iteratif (Garis Biru) konsisten berjalan lebih cepat dibandingkan Rekursif (Garis Oranye). Selisih waktu semakin melebar seiring bertambahnya ukuran input.

b. Analisis Overhead dan Memori
   i. Penyebab utama algoritma rekursif lebih lambat adalah Overhead Pemanggilan Fungsi.
   ii. Iteratif: Pergeseran indeks hanya melibatkan operasi aritmatika sederhana dan instruksi JUMP pada CPU, yang sangat cepat.
   iii. Rekursif: Setiap kali fungsi memanggil dirinya sendiri, Python harus mengalokasikan memori untuk *stack frame* baru (menyimpan variabel lokal, alamat kembali, dll). Proses alokasi dan dealokasi ini memakan waktu CPU tambahan (*overhead*).
   iv. Selain itu, algoritma rekursif memiliki Kompleksitas Ruang (Space Complexity) $O(N)$ karena tumpukan rekursi, sedangkan algoritma iteratif hanya $O(1)$. Dalam eksperimen, kode rekursif memerlukan pengaturan `sys.setrecursionlimit(10000)` agar dapat berjalan pada input di atas 1.000 karakter, yang membuktikan ketidakefisienan memori pada pendekatan rekursif untuk kasus ini.

c. Kesimpulan Akhir Untuk kasus *String Search* sederhana pada bahasa pemrograman Python:

i. Algoritma Iteratif lebih disarankan karena lebih cepat, hemat memori, dan stabil untuk input yang sangat besar.
ii. Algoritma Rekursif kurang efisien karena *overhead* fungsi yang tinggi dan batasan *recursion depth* pada Python, meskipun secara logika memberikan hasil yang sama benarnya.

5. Daftar Pustaka

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Data structures and algorithms in Python*. Wiley.

Levitin, A. (2012). *Introduction to the design and analysis of algorithms* (3rd ed.). Pearson.