Jurnal Penelitian Nusantara
Volume 1 ; Nomor 6 ; Juni 2025 ; Page 871-877
Doi : https://doi.org/10.59435/menulis.v1i6.417
Website : https://padangjurnal.web.id/index.php/menulis
E-ISSN : 3088-988X

Implementasi Struktur Data Array Dan Linked List Dalam Pengelolaan Data Mahasiswa

Hanie Ardina Sofianti[1], Yesica Vioretti Manullang[2], Nopita Agustina Tampubolon[3], Livia Helen Naibaho[4], Indra Gunawan[5]

[1] Teknik Informatika, Stikom Tunas Bangsa
[1]haniardinasofianti11@gmail.com, [2]yesicamanullang363@gmail.com, [3]nopitaagustinat@gmail.com, liviahelen023@gmail.com,

Abstrak

Pengelolaan data mahasiswa merupakan aspek penting dalam sistem akademik. Dua struktur data utama yang sering digunakan adalah **Array** dan **Linked List.** Array memiliki keunggulan dalam akses data secara langsung melalui indeks, sedangkan Linked List lebih fleksibel dalam penyisipan dan penghapusan data tanpa perlu menggeser elemen lain. Penelitian ini bertujuan untuk mengimplementasikan kedua struktur data dalam pengelolaan informasi mahasiwa serta membandingkan efisiensinya berdasarkan kompleksitas waktu operasi dasar seperti penambahan, penghapusan, pencarian data. Eksperimen dilakukan menggunakan Bahasa pemrograman python dengan mengukur efisiensi kedua struktur data dari segi kecepatan akses dan penggunaan memori. Hasil analisis menunjukkan bahwa Array lebih unggul dalam akses langsung (0)(1)), sedangkan linked list lebih fleksibel dalam perubahan data karena ukurannya dinamis. Dengan demikian, pemilihan struktur data yang tepat bergantung pada kebutuhan sistem Array lebih sesuai untuk data statis, sementara Linked List lebih cocok untuk data yang sering mengalami perubahan.

Kata Kunci: Struktur Data, Array, Linked List, Pengelolaan Data, Mahasiswa.

PENDAHULUAN

Dalam dunia akademik, sistem pengelolaan data mahasiswa memliki peran yang sangat penting. Data seperti **Nama, Nim, Program Studi,** dan **Indeks Prestasi Kumulatif (IPK)** harus disimpan, diakses, dan dikelola dengan efisien. Struktur data yang digunakan dalam implementasi sistem ini akan memengaruhi peforma dalam penyimpanan, pencarian, penambahan, dan penghapusan data.

Dua struktur data yang umum digunakan dalam pengelolaan data mahasiswa adalah Array dan Linked List. Array merupakan struktur data yang menyimpan elemen dalam satu blok memori dengan akses cepat menggunakan indeks. Namun, Array memiliki kelemahan dalam hal fleksibilitas, terutama saat menambah atau menghapus elemen ditengah daftar. Sementara itu, Linked List merupakan struktur data yang lebih dinamis, memungkinkan penambahan dan penghapusan elemen tanpa harus menggeser elemen laun. Namun, akses ke elemen tertentu dalam Linked List lebih lambat dibandingkan Array karena harus dilakukan secara sekuensial. Dalam system pengelolaan data mahasiswa, pemilihan struktur data yang tepat menjadi factor penting untuk meninggkatkan efisiensi operasional. Misalnya, dalam system akademik universitas yang menangani ribuan data mahasiswa, penggunaan struktur data yang salah dapat menyebabkan kinerja system menjadi lambat dan boros memori. Oleh karena itu, penelitian ini akan membandingkan Array dan Linked List untuk menentukan struktur data yang lebih sesuai dalam pengelolaan data mahasiswa berdasarkan kompleksitas waktu operasi dasar dan penggunaan memori.

METODE

Tahapan Penelitian

A. Implementasi Hasil

Bagian ini menjelaskan bagaimana struktur data Array dan Linked List diimplementasikan dalam pengelolaan data mahasiswa. Implementasi dilakukan menggunakan Bahasa pemrograman python dengan operasi dasar seperti penambahan, pencarian, dan penghapusan data mahasiswa.

1. Implementasi dengan Array

Array dalam python diimplementasikan menggunakan tipe data list, yang memungkinkan penyimpanan data dalam satu blok memori dengan akses cepat menggunakan indeks. Dalam implementasi ini, setiap mahasiswa disimpan dalam sebuah list python, di mana setiap elemen list merupakan dictionary yang berisi informasi mahasiswa, seperti Nim, Nama, Program studi, dan IPK. Berikut program implementasi kelas MahasiswaArray :

```python
class MahasiswaArray:
    def _init_(self):
        self.data = []  # List kosong untuk menyimpan data mahasiswa

    def tambah_mahasiswa(self, nim, nama, prodi, ipk): """Menambahkan mahasiswa ke dalam array"""
        self.data.append({"NIM": nim, "Nama": nama, "Prodi": prodi, "IPK": ipk})

    def hapus_mahasiswa(self, nim):
        """Menghapus mahasiswa berdasarkan NIM"""
        self.data = [mhs for mhs in self.data if mhs["NIM"] != nim]

    def cari_mahasiswa(self, nim):
        """Mencari mahasiswa berdasarkan NIM"""
        for mhs in self.data:
            if mhs["NIM"] == nim:
                return mhs
    return None
```

2. Analisis Implementasi dengan Array

➢ **Penambahan Mahasiswa**

Menggunakan metode .append() pada list, yang memiliki kompleksitas waktu 0(1) karena elemen baru langsung ditambahkan di akhir array tanpa perlu pergeseran data lain

➢ **Pencarian Mahasiswa**

Dilakukan dengan iterasi melalui semua elemn dalam list, memiliki komleksitas 0(n) dalam kasus terburuk, karena harus mencari satu per satu.

➢ **Penghapusan Mahasiswa**

Memerlukan pencarian mahasiswa yang sesuai, kemudian membuat ulang list tanpa elemen tersebut, sehingga memiliki kompleksitas 0(n).

3. Implementasi dengan Linked List

Dalam bagian ini, implementasi Linked List diperluas dengan beberapa fitur tambahan, yaitu :

1. Menampilkan seluruh data mahasiswa.
2. Memperbarui data mahasiswa berdasarjan NIM.
3. Mengurutkan data berdassarkan IPK.
4. Menghitung jumlah mahasiswa dalam daftar.

Berikut adalah implementasi lengkapnya :

```python
class Node:
    def _init_(self, nim, nama, prodi, ipk):
        """Membuat node baru dengan informasi mahasiswa"""
        self.nim = nim  # Nomor Induk Mahasiswa
        self.nama = nama  # Nama mahasiswa
        self.prodi = prodi  # Program studi
        self.ipk = ipk  # IPK mahasiswa
        self.next = None  # Pointer ke node berikutnya

class MahasiswaLinkedList:
    def _init_(self):
        """Inisialisasi Linked List dengan kepala (head) kosong"""
        self.head = None

    def tambah_mahasiswa(self, nim, nama, prodi, ipk):
```

```
    """Menambahkan mahasiswa ke dalam Linked List di posisi terakhir"""
    new_node = Node(nim, nama, prodi, ipk)
    if not self.head:  # Jika Linked List masih kosong
        self.head = new_node
    else:
        temp = self.head
        while temp.next:  # Iterasi hingga node terakhir
            temp = temp.next
        temp.next = new_node  # Tambahkan node baru di akhir

def hapus_mahasiswa(self, nim):
    """Menghapus mahasiswa berdasarkan NIM"""
    temp = self.head
    if temp and temp.nim == nim:  # Jika mahasiswa yang dihapus adalah node pertama
        self.head = temp.next
        return
    prev = None
    while temp and temp.nim != nim:
        prev = temp
        temp = temp.next
    if temp:  # Jika mahasiswa ditemukan
        prev.next = temp.next

def cari_mahasiswa(self, nim):
    """Mencari mahasiswa berdasarkan NIM dan mengembalikan datanya"""
    temp = self.head
    while temp:
        if temp.nim == nim:
            return vars(temp)  # Mengembalikan data dalam bentuk dictionary
        temp = temp.next
    return None  # Jika tidak ditemukan

def tampilkan_mahasiswa(self):
    """Menampilkan seluruh data mahasiswa dalam Linked List"""
    temp = self.head
    if not temp:
        print("Daftar mahasiswa kosong.")
        return
    while temp:
        print(f"NIM: {temp.nim}, Nama: {temp.nama}, Prodi: {temp.prodi}, IPK: {temp.ipk}")
        temp = temp.next

def perbarui_mahasiswa(self, nim, nama=None, prodi=None, ipk=None):
    """Memperbarui data mahasiswa berdasarkan NIM"""
    temp = self.head
    while temp:
        if temp.nim == nim:
            if nama:
                temp.nama = nama
            if prodi:
                temp.prodi = prodi
            if ipk:
                temp.ipk = ipk
            return True  # Menandakan pembaruan berhasil
        temp = temp.next
    return False  # Jika mahasiswa tidak ditemukan

def urutkan_berdasarkan_ipk(self):
    """Mengurutkan mahasiswa berdasarkan IPK (dari terbesar ke terkecil) menggunakan metode Bubble Sort"""
```

```
    if not self.head or not self.head.next:
        return  # Jika Linked List kosong atau hanya satu elemen, tidak perlu diurutkan
    sorted = False
    while not sorted:
        sorted = True
        temp = self.head
        prev = None
        while temp and temp.next:
            if temp.ipk < temp.next.ipk:  # Jika urutan salah, tukar data
                sorted = False
                if prev:
                    prev.next = temp.next
                else:
                    self.head = temp.next
                temp.next, temp.next.next, temp = temp.next.next, temp, temp.next
            prev = temp
            temp = temp.next

def hitung_mahasiswa(self):
    """Menghitung jumlah mahasiswa dalam Linked List"""
    count = 0
    temp = self.head
    while temp:
        count += 1
        temp = temp.next
    return count  # Mengembalikan jumlah mahasiswa dalam daftar
```

5. Analisis Perbandingan

Hasil pengujian mennjukkan bahwa Array dan Linked List memiliki kelebihan dan kekurangan masing-masing dalam pengelolaan data mahasiswa. Berikut adalah perbandingan kedua stuktur data berdasarkan aspek utama

Aspek	Array	Linked list
Akses Data	0(1) (langsung dengan indeks)	0(n) (harus menelusuri daftar)
Penambahan Data	0(1) jika di akhir, 0(n) jika di tengah	0(1) jika di awal, 0(n) jika di akhir
Penghapusan Data	0(n) (harus geser elemen lain)	0(1) jika di awal, 0(n) jika di tengah/ akhir
Pencarian Data	0(n) (linear search)	0(n) (harus menelusuri daftar)
Penggunaan Memori	Tetap (tergantung ukuran array)	Dinamis (bisa bertambah sesuai kebutuhan)
Efisiensi	Cocok untuk data statis	Cocok untuk data dinamis

Dari tabel diatas dapat disimpulkan bahwa:

- **Array lebih cocok** jika membutuhkan akses cepat ke data, tetapi kurang fleksibel dalam hal perubahan ukuran.
- **Linked List lebih fleksibel** karena ukurannya bisa bertambah tanpa harus mengalokasikan ulang memori, tetapi akses data lebih lambat dibandingkan Array.

HASIL DAN PEMBAHASAN

Penelitian ini dilakukan melalui tahapan implementasi dan analisis kinerja dua jenis struktur data utama, yaitu *Array* (menggunakan list Python) dan *Linked List*, dalam pengelolaan data mahasiswa. Pengujian dilakukan dengan mengukur efektivitas masing-masing struktur data terhadap beberapa operasi dasar, seperti penambahan, pencarian, penghapusan, dan pengurutan data mahasiswa.

Langkah-langkah yang ditempuh dalam penelitian ini meliputi:

a. Mendesain struktur data mahasiswa yang terdiri dari atribut: NIM, Nama, Prodi, dan IPK.
   1. Untuk *Array*, digunakan list Python dengan elemen berupa dictionary.
   2. Untuk *Linked List*, dibuat node yang merepresentasikan objek mahasiswa.

b. Melakukan implementasi operasi dasar pada kedua struktur data, yaitu:
   1. Penambahan data mahasiswa.

2. Penghapusan data berdasarkan NIM.
3. Pencarian data berdasarkan NIM.
4. Pengurutan data berdasarkan IPK (khusus pada Linked List).
5. Penghitungan jumlah mahasiswa dalam struktur.

c. Membandingkan hasil pengujian terhadap aspek kinerja dan efisiensi memori.

a. Implementasi dengan Array

Struktur *Array* diimplementasikan menggunakan list Python. Berikut adalah cuplikan kode implementasi:

```
class MahasiswaArray:
    def __init__(self):
        self.data = []

    def tambah_mahasiswa(self, nim, nama, prodi, ipk):
        self.data.append({"NIM": nim, "Nama": nama, "Prodi": prodi, "IPK": ipk})

    def hapus_mahasiswa(self, nim):
        self.data = [mhs for mhs in self.data if mhs["NIM"] != nim]

    def cari_mahasiswa(self, nim):
        for mhs in self.data:
            if mhs["NIM"] == nim:
                return mhs
        return None
```

b. Implementasi dengan Linked List

Struktur *Linked List* menggunakan node dinamis untuk menyimpan dan mengelola data mahasiswa. Berikut contoh kode dasarnya:

```
class Node:
    def __init__(self, nim, nama, prodi, ipk):
        self.nim = nim
        self.nama = nama
        self.prodi = prodi
        self.ipk = ipk
        self.next = None
```

Hasil Pengujian

Pengujian dilakukan dengan data mahasiswa fiktif dan pengukuran waktu untuk operasi-operasi berikut:

a. Penambahan Data
   1. *Array* membutuhkan waktu konstan (O(1)) untuk menambah di akhir list.
   2. *Linked List* memerlukan waktu O(n) untuk traversing hingga akhir sebelum menambah node baru.

b. Penghapusan Data
   1. Pada *Array*, seluruh elemen harus disalin ulang tanpa elemen yang dihapus → O(n).
   2. Pada *Linked List*, jika elemen berada di awal → O(1), jika di tengah/akhir → O(n).

c. Pencarian Data
   Baik *Array* maupun *Linked List* membutuhkan iterasi linear → O(n).

d. Pengurutan Berdasarkan IPK
   Implementasi pengurutan dilakukan hanya pada *Linked List* menggunakan *Bubble Sort*.

Pembahasan

Berdasarkan hasil implementasi dan pengujian, diperoleh kesimpulan berikut:

a. Kelebihan Array
   1. Akses data cepat (O(1)), cocok untuk sistem di mana indeks diketahui.
   2. Struktur sederhana dan implementasi mudah.

b. Kekurangan Array
1. Tidak efisien dalam penghapusan dan penyisipan elemen di tengah.
2. Ukuran array harus diatur ulang jika kapasitas perlu diubah (tidak fleksibel).

c. Kelebihan Linked List
1. Ukuran dinamis, cocok untuk data yang sering berubah.
2. Efisien dalam penambahan dan penghapusan di awal.

d. Kekurangan Linked List
1. Akses data lambat karena harus traversing (O(n)).
2. Memerlukan memori tambahan untuk pointer setiap node.

4.5 Tabel Perbandingan

Aspek	Array	Linked List
Akses Data	O(1) (langsung dengan indeks)	O(n) (traversal node)
Penambahan Data	O(1) (di akhir)	O(n) (di akhir), O(1) (di awal)
Penghapusan Data	O(n) (salin ulang)	O(1) – O(n) tergantung posisi
Pencarian Data	O(n)	O(n)
Penggunaan Memori	Tetap	Dinamis
Efisiensi	Baik untuk data statis	Baik untuk data dinamis

4.6 Gambar Diagram Struktur Data

Gambar 4.1 – Struktur Array Mahasiswa:

```
[ {"NIM": "123", ...}, {"NIM": "124", ...}, {"NIM": "125", ...} ]
```

Gambar 4.2 – Struktur Linked List Mahasiswa:

```
Head --> [Node: NIM 123] --> [Node: NIM 124] --> [Node: NIM 125] --> None
```

KESIMPULAN

Array lebih efiisien untuk akses langsung karena dapat langsung mengakses data berdasarkan indeks, tetapi memiliki keterbatasan dalam penambahan dan penghapusan data karena harus menggeser elemen lainnya sedangkan **Linked list** lebih fleksibel dalam penambahan dan penghapusan data tanpa harus menggeser elemen lain, tetapi pencarian data lebih lambat karena harus menelusuri seluruh daftar. Pemilihan struktur data bergantung pada kebutuhan sistem misalnya, jika data bersifat statis (jarang berubah). Array lebih baik sedangkan jika data sering berubah (sering ditambah atau dihapus), Linked List lebih sesuai. Penelitian ini hanya membandingkan Array dan Linked List. Untuk pengelolaan data yang lebih kompleks, studi lanjutan dapat membandingkan struktur data lain seperti, Hash Table untuk pencarian lebih cepat dan Binary Search Tree (BST) untuk pengurutan data yang lebih efisien.

UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas segala rahmat, karunia, dan petunjuk-Nya sehingga penulis dapat menyelesaikan laporan/skripsi dengan judul **"Implementasi Struktur Data Array dan Linked List dalam Pengelolaan Data Mahasiswa"** ini dengan baik.

Penulis menyadari bahwa tersusunnya laporan ini tidak lepas dari bantuan, bimbingan, dan dukungan dari berbagai pihak. Oleh karena itu, dengan segala kerendahan hati, penulis menyampaikan ucapan terima kasih kepada:

a. **Kepala Program Studi** dan seluruh **dosen pengajar** yang telah memberikan ilmu, arahan, serta motivasi selama proses studi.
b. **Dosen Pembimbing**, yang telah dengan sabar membimbing, memberikan masukan, dan arahan yang sangat berarti dalam penyusunan laporan ini.
c. **Orang tua dan keluarga tercinta**, atas doa, dukungan moral maupun materi, serta semangat yang tidak pernah putus selama proses penyelesaian tugas ini.
d. **Teman-teman seperjuangan** yang selalu memberikan dukungan, diskusi, serta kebersamaan selama menjalani proses perkuliahan dan penyusunan laporan ini.
e. Serta semua pihak yang tidak dapat disebutkan satu per satu, namun telah memberikan kontribusi yang berarti dalam proses penyusunan laporan ini.

Penulis menyadari bahwa laporan ini masih jauh dari sempurna. Oleh karena itu, segala kritik dan saran yang membangun sangat penulis harapkan untuk perbaikan di masa yang akan datang.

Semoga laporan ini dapat memberikan manfaat bagi semua pihak yang membacanya.

DAFTAR PUSTAKA

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
2. Goodrich, M. T., & Tamassia, R. (2014). *Data Structures and Algorithms in Python*. Wiley.
3. Sedgewick, R., & Wayne, K. (2011). *Algorithms*. Addison-Wesley.