

**LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA**

**MODUL VIII
ALGORITMA SEARCHING**



Disusun Oleh :
DIO GILBRAN PRAMANA
NIM : 2311102062

Dosen
WAHYU ANDI SAPUTRA, S.Pd., M.Eng.

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

A. DASAR TEORI

Pencarian (Searching) yaitu proses menemukan suatu nilai tertentu pada kumpulan data. Hasil pencarian adalah salah satu dari tiga keadaan ini: data ditemukan, data ditemukan lebih dari satu, atau data tidak ditemukan. Searching juga dapat dianggap sebagai proses pencarian suatu data di dalam sebuah array dengan cara mengecek satu persatu pada setiap index baris atau setiap index kolomnya dengan menggunakan teknik perulangan untuk melakukan pencarian data. Terdapat 2 metode pada algoritma Searching, yaitu:

a. Sequential Search

Sequential Search merupakan salah satu algoritma pencarian data yang biasa digunakan untuk data yang berpola acak atau belum terurut. Sequential search juga merupakan teknik pencarian data dari array yang paling mudah, dimana data dalam array dibaca satu demi satu dan diurutkan dari index terkecil ke index terbesar, maupun sebaliknya.

Konsep Sequential Search yaitu:

1. Membandingkan setiap elemen pada array satu per satu secara berurut.
2. Proses pencarian dimulai dari indeks pertama hingga indeks terakhir.
3. Proses pencarian akan berhenti apabila data ditemukan. Jika hingga akhir array data masih juga tidak ditemukan, maka proses pencarian tetap akan dihentikan.
4. Proses perulangan pada pencarian akan terjadi sebanyak jumlah N elemen pada array.

Algoritma pencarian berurutan dapat dituliskan sebagai berikut :

1. $i \leftarrow 0$
2. $ketemu \leftarrow false$
3. Selama (tidak $ketemu$) dan $(i \leq N)$ kerjakan baris 4
4. Jika $(Data[i] = x)$ maka $ketemu \leftarrow true$, jika tidak $i \leftarrow i + 1$
5. Jika ($ketemu$) maka i adalah indeks dari data yang dicari, jika tidak data tidak ditemukan

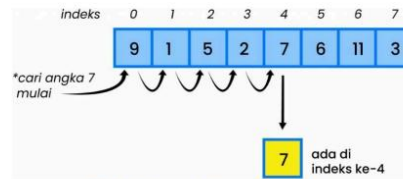
Di bawah ini merupakan fungsi untuk mencari data menggunakan pencarian sekuensial

```
int SequentialSearch (int x)
{
    int i = 0;
    bool ketemu = false;
    while ((!ketemu) && (i < Max)){
        if (Data[i] == x)
            ketemu = true;
        else
            i++;
    }
    if (ketemu)
        return i;
    else
        return -1;
}
```

Fungsi diatas akan mengembalikan indeks dari data yang dicari. Apabila data tidak ditemukan maka fungsi diatas akan mengembalikan nilai -1.

Contoh dari Sequential Search, yaitu:

Int A[8] = {9,1,5,2,7,6,11,3}



Gambar 1. Ilustrasi Sequential Search

Misalkan, dari data di atas angka yang akan dicari adalah angka 7 dalam array A, maka proses yang akan terjadi yaitu:

1. Pencarian dimulai pada index ke-0 yaitu angka 9, kemudian dicocokkan dengan angka yang akan dicari, jika tidak sama maka pencarian akan dilanjutkan ke index selanjutnya.
2. Pada index ke-1, yaitu angka 1, juga bukan angka yang dicari, maka pencarian akan dilanjutkan pada index selanjutnya.
3. Pada index ke-2 dan index ke-3 yaitu angka 5 dan 2, juga bukan angka yang dicari, sehingga pencarian dilanjutkan pada index selanjutnya.
4. Pada index ke-4 yaitu angka 7 dan ternyata angka 7 merupakan angka yang dicari, sehingga pencarian akan dihentikan dan proses selesai.

b. Binary Search

Binary Search termasuk ke dalam interval search, dimana algoritma ini merupakan algoritma pencarian pada array/list dengan elemen terurut. Pada metode ini, data harus diurutkan terlebih dahulu dengan cara data dibagi menjadi dua bagian (secara logika), untuk setiap tahap pencarian. Dalam penerapannya algoritma ini sering digabungkan dengan algoritma sorting karena data yang akan digunakan harus sudah terurut terlebih dahulu.

Konsep Binary Search:

1. Data diambil dari posisi 1 sampai posisi akhir N.
2. Kemudian data akan dibagi menjadi dua untuk mendapatkan posisi data tengah. Selanjutnya data yang dicari akan dibandingkan dengan data yang berada di posisi tengah, apakah lebih besar atau lebih kecil.
3. Apabila data yang dicari lebih besar dari data tengah, maka dapat dipastikan bahwa data yang dicari kemungkinan berada di sebelah kanan dari data tengah. Proses pencarian selanjutnya akan dilakukan pembagian data menjadi dua bagian pada bagian kanan dengan acuan posisi data tengah akan menjadi posisi awal untuk pembagian tersebut.
4. Apabila data yang dicari lebih kecil dari data tengah, maka dapat dipastikan bahwa data yang dicari kemungkinan berada di sebelah kiri dari data tengah. Proses pencarian selanjutnya akan dilakukan pembagian data menjadi dua bagian pada bagian kiri. Dengan acuan posisi data tengah akan menjadi posisi akhir untuk pembagian selanjutnya.
5. Apabila data belum ditemukan, maka pencarian akan dilanjutkan dengan

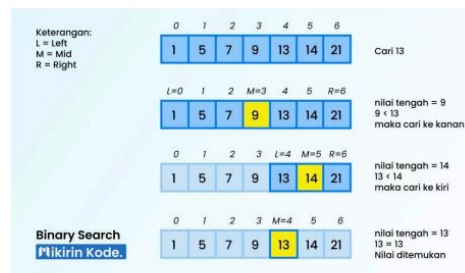
kembali membagi data menjadi dua.

6. Namun apabila data bernilai sama, maka data yang dicari langsung ditemukan dan pencarian dihentikan.

Algoritma pencarian biner dapat dituliskan sebagai berikut :

- 1) $L \leftarrow 0$
- 2) $R \leftarrow N - 1$
- 3) $ketemu \leftarrow false$
- 4) Selama $(L \leq R)$ dan (tidak ketemu) kerjakan baris 5 sampai dengan 8
- 5) $m \leftarrow (L + R) / 2$
- 6) Jika $(Data[m] = x)$ maka $ketemu \leftarrow true$
- 7) Jika $(x < Data[m])$ maka $R \leftarrow m - 1$
- 8) Jika $(x > Data[m])$ maka $L \leftarrow m + 1$
- 9) Jika (ketemu) maka m adalah indeks dari data yang dicari, jika tidak data tidak ditemukan

Contoh dari Binary Search, yaitu :



Gambar 2. Ilustrasi Binary Search

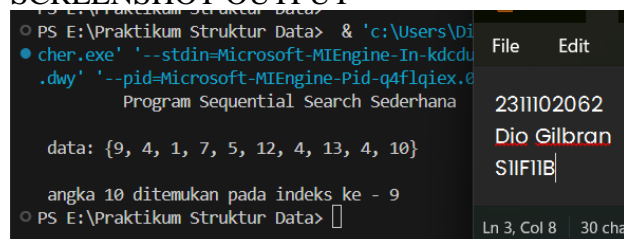
- 1) Terdapat sebuah array yang menampung 7 elemen seperti ilustrasi di atas. Nilai yang akan dicari pada array tersebut adalah 13.
- 2) Jadi karena konsep dari binary search ini adalah membagi array menjadi dua bagian, maka pertama tama kita cari nilai tengahnya dulu, total elemen dibagi 2 yaitu $7/2 = 4.5$ dan kita bulatkan jadi 4.
- 3) Maka elemen ke empat pada array adalah nilai tengahnya, yaitu angka 9 pada indeks ke 3.
- 4) Kemudian kita cek apakah $13 > 9$ atau $13 < 9$?
- 5) 13 lebih besar dari 9, maka kemungkinan besar angka 13 berada setelah 9 atau di sebelah kanan. Selanjutnya kita cari ke kanan dan kita dapat mengabaikan elemen yang ada di kiri.
- 6) Setelah itu kita cari lagi nilai tengahnya, didapatkan angka 14 sebagai nilai tengah. Lalu, kita bandingkan apakah $13 > 14$ atau $13 < 14$?
- 7) Ternyata 13 lebih kecil dari 14, maka selanjutnya kita cari ke kiri.
- 8) Karna tersisa 1 elemen saja, maka elemen tersebut adalah nilai tengahnya. Setelah dicek ternyata elemen pada indeks ke-4 adalah elemen yang dicari, maka telah selesai proses pencariannya.

B. GUIDED

GUIDED 1

```
#include <iostream>
using namespace std;
int main()
{
    int n = 10;
    int data[n] = {9, 4, 1, 7, 5, 12, 4, 13, 4, 10};
    int cari = 10;
    bool ketemu = false;
    int i;
    // algoritma Sequential Search
    for (i = 0; i < n; i++)
    {
        if (data[i] == cari)
        {
            ketemu = true;
            break;
        }
    }
    cout << "\t Program Sequential Search Sederhana\n " << endl;
    cout << " data: {9, 4, 1, 7, 5, 12, 4, 13, 4, 10}" << endl;
    if (ketemu)
    {
        cout << "\n angka " << cari << " ditemukan pada indeks ke - " << i << endl;
    }
    else
    {
        cout << cari << " tidak dapat ditemukan pada data." << endl;
    }
    return 0;
}
```

SCREENSHOT OUTPUT



DESKRIPSI

Program di atas adalah implementasi algoritma Sequential Search dalam bahasa C++ untuk mencari sebuah nilai dalam array. Program ini mendeklarasikan array data berisi sepuluh angka dan mencari nilai cari yang telah ditentukan, yaitu 10. Dengan menggunakan loop for, program memeriksa setiap elemen dalam array mulai dari indeks pertama hingga menemukan nilai yang dicari. Jika nilai ditemukan, variabel ketemu diatur menjadi true dan loop berhenti. Program kemudian mencetak pesan yang menunjukkan apakah nilai tersebut ditemukan atau tidak dan, jika ditemukan, indeks tempat nilai tersebut berada. Desain ini memungkinkan pencarian sederhana dan cepat untuk array kecil atau data yang tidak terstruktur.

GUIDED 2

```
#include <iostream>
using namespace std;
#include <conio.h>
#include <iomanip>
int data[7] = {1, 8, 2, 5, 4, 9, 7};
int cari;
void selection_sort()
{
    int temp, min, i, j;
    for (i = 0; i < 7; i++)
    {
        min = i;
        for (j = i + 1; j < 7; j++)
        {
            if (data[j] < data[min])
            {
                min = j;
            }
        }
        temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
}
void binarysearch()
{
    // searching
    int awal, akhir, tengah, b_flag = 0;
    awal = 0;
    akhir = 7;
    while (b_flag == 0 && awal <= akhir)
    {
```

```

        tengah = (awal + akhir) / 2;
        if (data[tengah] == cari)
        {
            b_flag = 1;
            break;
        }
        else if (data[tengah] < cari)
            awal = tengah + 1;
        else
            akhir = tengah - 1;
    }
    if (b_flag == 1)
        cout << "\n Data ditemukan pada index ke- " << tengah <<
endl;
    else
        cout << "\n Data tidak ditemukan\n";
}

int main()
{
    cout << "\t BINARY SEARCH " << endl;
    cout << "\n Data : ";
    // tampilkan data awal
    for (int x = 0; x < 7; x++)
        cout << setw(3) << data[x];
    cout << endl;
    cout << "\n Masukkan data yang ingin Anda cari :";
    cin >> cari;
    cout << "\n Data diurutkan : ";
    // urutkan data dengan selection sort
    selection_sort();
    // tampilkan data setelah diurutkan
    for (int x = 0; x < 7; x++)
        cout << setw(3) << data[x];
    cout << endl;
    binarysearch();
    _getche();
    return EXIT_SUCCESS;
}

```

SCREENSHOT OUTPUT

```

PS E:\Praktikum Struktur Data>
PS E:\Praktikum Struktur Data> g++ & 'c:\Users\
cher.exe' '-std=c++11' '-std=Microsoft-MIEngine-In-15:
.xll' '-pid=Microsoft-MIEngine-Pid-21wvfr'
BINARY SEARCH

Data :  1  8  2  5  4  9  7

Masukkan data yang ingin Anda cari :
Data diurutkan :  1  2  4  5  7  8  9

Data ditemukan pada index ke- 6
PS E:\Praktikum Struktur Data>

```

DESKRIPSI

Program di atas mengimplementasikan Binary Search untuk mencari nilai dalam array setelah mengurutkannya menggunakan Selection Sort dalam bahasa C++. Array data berisi tujuh angka yang diurutkan terlebih dahulu dengan `selection_sort()`, di mana elemen terkecil dipindahkan ke posisi yang sesuai secara berurutan. Setelah pengurutan, pengguna diminta memasukkan nilai yang akan dicari. Fungsi `binarysearch()` kemudian digunakan untuk mencari nilai tersebut dalam array yang telah diurutkan dengan menggunakan teknik pencarian biner, yang memecah array menjadi dua bagian hingga nilai yang dicari ditemukan atau dipastikan tidak ada. Jika nilai ditemukan, program mencetak indeks tempat nilai tersebut berada; jika tidak, program mencetak pesan bahwa nilai tidak ditemukan.

C. UNGUIDED

UNGUIDED 1

```
#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;

void selection_sort(char data[], int n)
{
    int min, i, j;
    char temp;
    for (i = 0; i < n - 1; i++)
    {
        min = i;
        for (j = i + 1; j < n; j++)
        {
            if (data[j] < data[min])
            {
                min = j;
            }
        }
        temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
}

void binary_search(char data[], int n, char cari)
{
    int awal = 0, akhir = n - 1, tengah;
    bool found = false;
    while (awal <= akhir)
```



```

    {
        tengah = (awal + akhir) / 2;
        if (data[tengah] == cari)
        {
            found = true;
            break;
        }
        else if (data[tengah] < cari)
        {
            awal = tengah + 1;
        }
        else
        {
            akhir = tengah - 1;
        }
    }
    if (found)
    {
        cout << "\nData ditemukan pada index ke- " << tengah <<
endl;
    }
    else
    {
        cout << "\nData tidak ditemukan\n";
    }
}

int main()
{
    const int MAX_LENGTH = 100;
    char kalimat[MAX_LENGTH];
    char cari;

    cout << "\tPENCARIAN BINARY SEARCH" << endl;
    cout << "\nMasukkan sebuah kalimat: ";
    cin.getline(kalimat, MAX_LENGTH);

    int n = strlen(kalimat);

    cout << "\nData diurutkan: ";
    selection_sort(kalimat, n);

    // Tampilkan data setelah diurutkan
    for (int x = 0; x < n; x++)
    {
        cout << setw(3) << kalimat[x];
    }
}

```

```

    cout << endl;

    cout << "\nMasukkan karakter yang ingin Anda cari: ";
    cin >> cari;

    binary_search(kalimat, n, cari);

    return 0;
}

```

SCREENSHOT OUTPUT

```

PS E:\Praktikum Struktur Data> & 'c:\Users\Dio Gilbran\.vscode\extensions\ms-vscode.cpptools-1.24
n-jlmdizvy.cox' '--stdout=Microsoft-MIEngine-Out-xgltwc0h.rq2' '--stderr=Microsoft-MIEngine-Error-
odeBlocks\MinGW\bin\gdb.exe' '--interpreter=mi'
PENCARIAN BINARY SEARCH

Masukkan sebuah kalimat: bapak presiden baik sekali

Data diurutkan:      a a a a b b d e e e i i i k k k l n p p r s s

Masukkan karakter yang ingin Anda cari: n

Data ditemukan pada index ke- 20
PS E:\Praktikum Struktur Data>

```

DESKRIPSI

Program ini adalah aplikasi pencarian karakter menggunakan metode binary search dalam bahasa C++. Pertama, pengguna diminta untuk memasukkan sebuah kalimat. Setelah kalimat dimasukkan, program mengurutkan karakter-karakter dalam kalimat tersebut menggunakan selection sort. Kalimat yang telah diurutkan kemudian ditampilkan ke pengguna. Selanjutnya, pengguna diminta untuk memasukkan satu karakter yang ingin dicari. Program melakukan pencarian karakter tersebut menggunakan binary search pada kalimat yang sudah diurutkan dan menampilkan hasil apakah karakter tersebut ditemukan beserta indeksnya, atau tidak ditemukan.

UNGUIDED 2

```

#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;

void selection_sort(char data[], int n)
{
    int min, i, j;
    char temp;
    for (i = 0; i < n - 1; i++)
    {
        min = i;
        for (j = i + 1; j < n; j++)
        {

```

```

        if (data[j] < data[min])
        {
            min = j;
        }
    }
    temp = data[i];
    data[i] = data[min];
    data[min] = temp;
}

int hitung_vokal(const char kalimat[], int n)
{
    int count = 0;
    for (int i = 0; i < n; ++i)
    {
        if (kalimat[i] == 'a' || kalimat[i] == 'e' ||
            kalimat[i] == 'i' || kalimat[i] == 'o' || kalimat[i]
== 'u' ||
            kalimat[i] == 'A' || kalimat[i] == 'E' ||
            kalimat[i] == 'I' || kalimat[i] == 'O' || kalimat[i]
== 'U')
        {
            ++count;
        }
    }
    return count;
}

int main()
{
    const int MAX_LENGTH = 100;
    char kalimat[MAX_LENGTH];

    cout << "\tPROGRAM MENGHITUNG HURUF VOKAL" << endl;
    cout << "\nMasukkan sebuah kalimat: ";
    cin.getline(kalimat, MAX_LENGTH);

    int n = strlen(kalimat);

    cout << "\nData diurutkan: ";
    selection_sort(kalimat, n);

    // Tampilkan data setelah diurutkan
    for (int x = 0; x < n; x++)
    {
        cout << setw(3) << kalimat[x];
    }
}

```

```

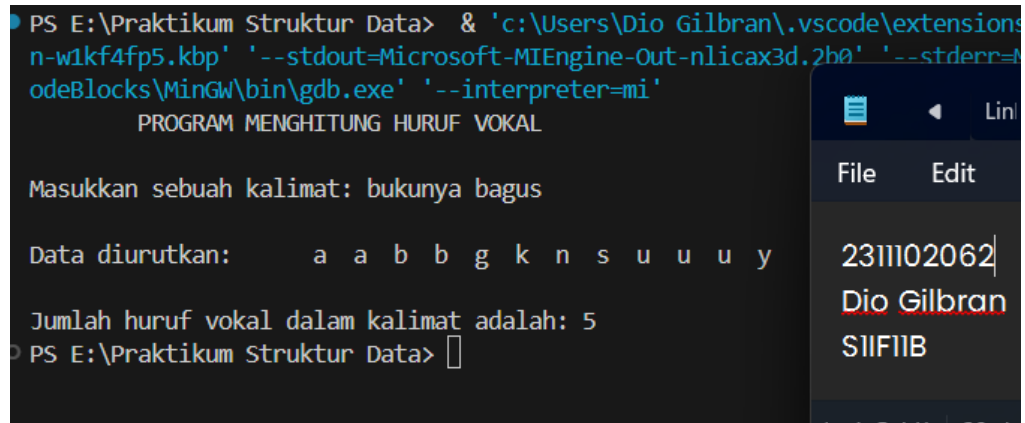
    }
    cout << endl;

    int jumlah_vokal = hitung_vokal(kalimat, n);
    cout << "\nJumlah huruf vokal dalam kalimat adalah: " <<
    jumlah_vokal << endl;

    return 0;
}

```

SCREENSHOT OUTPUT



```

PS E:\Praktikum Struktur Data> & 'c:\Users\Dio Gilbran\.vscode\extensions\n-w1kf4fp5.kbp' '--stdout=Microsoft-MIEngine-Out-nlicax3d.2b0' '--stderr=todeBlocks\MinGW\bin\gdb.exe' '--interpreter=mi'
PROGRAM MENGHITUNG HURUF VOKAL

Masukkan sebuah kalimat: bukunya bagus

Data diurutkan:      a a b b g k n s u u u y

Jumlah huruf vokal dalam kalimat adalah: 5
PS E:\Praktikum Struktur Data>

```

DESKRIPSI

Program ini adalah aplikasi untuk menghitung jumlah huruf vokal dalam sebuah kalimat menggunakan bahasa C++. Pertama, pengguna diminta untuk memasukkan sebuah kalimat. Setelah itu, program mengurutkan karakter-karakter dalam kalimat tersebut menggunakan metode selection sort dan menampilkannya. Kemudian, program menghitung jumlah huruf vokal (a, e, i, o, u, baik huruf kecil maupun huruf besar) dalam kalimat yang sudah diurutkan tersebut dan menampilkan hasilnya kepada pengguna.

D. KESIMPULAN

Pada praktikum ini, dua metode pencarian data dalam array dipelajari: Sequential Search dan Binary Search. Sequential Search mencari elemen satu per satu dari awal hingga akhir, cocok untuk data yang tidak terurut dan berukuran kecil. Binary Search, yang lebih efisien untuk array terurut, membagi data menjadi dua dan fokus pada bagian yang relevan berdasarkan perbandingan elemen tengah, mengurangi jumlah perbandingan yang diperlukan. Binary Search lebih cepat untuk dataset besar yang terurut, namun membutuhkan data yang sudah terurut terlebih dahulu.

E. REFERENSI

Asisten Praktikum, "Modul Algoritma Searching", 2024.

Karumanchi, N. (2016). Data Structures and algorithms made easy: Concepts, problems, Interview Questions. CareerMonk Publications.

Budi Raharjo. 2015. Pemrograman C++. Bandung: Informatika