LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2

MODUL 11

MATERI

Oleh:

MUHAMMAD ZAKY MUBAROK

103112400073

KELAS:

IF-12-01

S1 TEKNIK INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

I. DASAR TEORI

Modul 11 tentang **Pencarian Nilai Acak pada Himpunan Data** membahas berbagai teknik pencarian dalam struktur data, terutama **Sequential Search** dan **Binary Search**. Berikut adalah dasar teori dari konsep pencarian dalam himpunan data:

Dasar Teori Pencarian dalam Himpunan Data

1. **Definisi Pencarian Data**
   Pencarian adalah proses menemukan suatu nilai dalam kumpulan data yang telah diberikan. Nilai yang dicari bisa berupa angka, teks, atau objek lain dalam struktur data.
2. **Jenis Pencarian**
   - **Sequential Search (Pencarian Berurutan)**
     - Memeriksa setiap elemen satu per satu dari awal hingga akhir.
     - Berhenti jika elemen ditemukan atau seluruh data telah diperiksa.
     - Kompleksitas waktu: **O(N)** (linear).
     - Cocok untuk data **tidak terurut** atau jumlah data kecil.
   - **Binary Search (Pencarian Biner)**
     - Hanya bisa digunakan pada **data yang sudah terurut**.
     - Membagi data menjadi dua bagian dan mencari di bagian yang sesuai.
     - Kompleksitas waktu: **O(log N)** (lebih cepat dibanding Sequential Search).
     - Cocok untuk **jumlah data besar** yang sudah terurut.
3. **Implementasi dalam Pemrograman**
   - **Sequential Search** digunakan untuk mencari data dalam array atau list tanpa perlu sorting.
   - **Binary Search** lebih efisien tetapi membutuhkan data yang sudah terurut sebelumnya.

II. GUIDED

```
package main
import (
    "fmt"
    "sort"

)

func sequentialSearch(arr []float64, target float64) (int, int) {
    itterations := 0
    for i, val := range arr {
        itterations++
        fmt.Printf("Sequential Step %d: cek arr[%d] = %.1f\n",
itterations, i, val)
        if val == target {
            return i, itterations
        }
    }
    return -1, itterations
}

func binarySearch(arr []float64, target float64) (int, int) {
    itterations := 0
    low := 0
    high := len(arr) - 1
    for low <= high {
        itterations++
        mid := (low + high) / 2
        fmt.Printf("Binary Step %d: cek arr[%d] = %.1f\n",
itterations, mid, arr[mid])
        if arr[mid] == target {
            return mid, itterations
        } else if target < arr[mid] {
            high = mid - 1
        } else {
            low = mid + 1
        }
            }
            return -1, itterations
    }
func main() {

    //Array awal
```

```
    data := []float64{2, 7, 9, 1, 5, 6, 18, 13,25,20}
    target := 13.0

    fmt.Println("Sequential Search (data tidak perlu urut):")
    idxSeq, iterSeq := sequentialSearch(data, target)
    if idxSeq != -1 {
        fmt.Printf("Hasil : Ditemukan di indeks %d dalam %d
langkah\n\n", idxSeq, iterSeq)
    } else {
        fmt.Printf("Hasil : Tidak ditemukan dalam %d langkah\n",
iterSeq)
    }

    //Binary search perlu array diurutkan
    sort.Float64s(data)
    fmt.Println("Binary Search (setelah data diurutkan):")
    fmt.Println("Data terurut:", data)

    idxBin, iterBin := binarySearch(data, target)
    if idxBin != -1 {
        fmt.Printf("Hasil : Ditemukan di indeks %d dalam %d
langkah\n", idxBin, iterBin)
    } else {
        fmt.Printf("Hasil : Tidak ditemukan dalam %d langkah\n",
iterBin)
    }


}
```

```
Sequential Search (data tidak perlu urut):
Sequential Step 1: cek arr[0] = 2.0
Sequential Step 2: cek arr[1] = 7.0
Sequential Step 3: cek arr[2] = 9.0
Sequential Step 4: cek arr[3] = 1.0
Sequential Step 5: cek arr[4] = 5.0
Sequential Step 6: cek arr[5] = 6.0
Sequential Step 7: cek arr[6] = 18.0
Sequential Step 8: cek arr[7] = 13.0
Hasil : Ditemukan di indeks 7 dalam 8 langkah

Binary Search (setelah data diurutkan):
Data terurut: [1 2 5 6 7 9 13 18 20 25]
Binary Step 1: cek arr[4] = 7.0
Binary Step 2: cek arr[7] = 18.0
Binary Step 3: cek arr[5] = 9.0
Binary Step 4: cek arr[6] = 13.0
Hasil : Ditemukan di indeks 6 dalam 4 langkah
```

Penjelasan :

Kode di atas adalah implementasi **pencarian berurutan (Sequential Search) dan pencarian biner (Binary Search)** dalam bahasa Go.

Penjelasan Singkat:

1. **Sequential Search (sequentialSearch)**
    - Memeriksa setiap elemen satu per satu dari awal hingga akhir.
    - Jika ditemukan, mengembalikan indeks elemen dan jumlah iterasi.
    - Kompleksitas waktu: **O(N)** (linear).
2. **Binary Search (binarySearch)**
    - Hanya bisa digunakan pada **array yang sudah terurut**.
    - Membagi data menjadi dua dan mencari di bagian yang sesuai.
    - Jika ditemukan, mengembalikan indeks dan jumlah iterasi.
    - Kompleksitas waktu: **O(log N)** (lebih cepat dibanding Sequential Search).
3. **Fungsi main()**
    - Mengisi array data dengan angka acak.
    - Mencari nilai target dengan **Sequential Search** tanpa perlu sorting.
    - Mengurutkan array menggunakan sort.Float64s(data).
    - Mencari nilai target dengan **Binary Search** setelah array diurutkan.

III. GUIDED

```go
package main

import (
    "fmt"
    "sort"
)

type mahasiswa struct {
    nama, nim, kelas, jurusan string
    ipk                       float64
}

type arrMhs [2023]mahasiswa

// Sequential Search berdasarkan nama
func SeqSearch_3(T arrMhs, n int, X string) (int, int) {
    var found int = -1
    var j int = 0
    var iterasi int = 0

    for j < n && found == -1 {
        iterasi++
        if T[j].nama == X {
            found = j
        }
        j++
    }
    return found, iterasi
}

// Binary Search berdasarkan NIM (data harus sudah terurut
berdasarkan nim)
func BinarySearch_3(T arrMhs, n int, X string) (int, int) {
    var found int = -1
    var med int
    var kr int = 0
    var kn int = n - 1
    var iterasi int = 0

    for kr <= kn && found == -1 {
        iterasi++
        med = (kr + kn) / 2
```

```
        if X < T[med].nim {
            kn = med - 1
        } else if X > T[med].nim {
            kr = med + 1
        } else {
            found = med
        }
    }
    return found, iterasi
}

func main() {
    var data arrMhs
    n := 10

    // Mengisi data secara manual
    data = arrMhs{
        {nama: "Ari", nim: "2201", kelas: "A", jurusan:
"Informatika", ipk: 3.4},
        {nama: "Budi", nim: "2203", kelas: "A", jurusan:
"Informatika", ipk: 3.6},
        {nama: "Cici", nim: "2202", kelas: "B", jurusan: "Sistem
Informasi", ipk: 3.5},
        {nama: "Dina", nim: "2205", kelas: "A", jurusan:
"Informatika", ipk: 3.3},
        {nama: "Eko", nim: "2204", kelas: "B", jurusan: "Sistem
Informasi", ipk: 3.7},
        {nama: "Fajar", nim: "2206", kelas: "C", jurusan:
"Informatika", ipk: 3.1},
        {nama: "Gita", nim: "2209", kelas: "C", jurusan:
"Informatika", ipk: 3.8},
        {nama: "Hana", nim: "2208", kelas: "B", jurusan: "Sistem
Informasi", ipk: 3.2},
        {nama: "Iwan", nim: "2207", kelas: "C", jurusan:
"Informatika", ipk: 3.0},
        {nama: "Joko", nim: "2210", kelas: "A", jurusan:
"Informatika", ipk: 3.9},
    }

    // Pencarian Sequential Search berdasarkan nama
    namaDicari := "Fajar"
    idxSeq, iterSeq := SeqSearch_3(data, n, namaDicari)

    fmt.Printf("Sequential Search - Cari nama '%s'\n", namaDicari)
    if idxSeq != -1 {
```

```
        fmt.Printf("Ditemukan di indeks: %d, Iterasi: %d\n", idxSeq,
iterSeq)
    } else {
        fmt.Printf("Tidak ditemukan, Iterasi: %d\n", iterSeq)
    }

    // Urutkan data berdasarkan NIM untuk binary search
    sort.Slice(data[:n], func(i, j int) bool {
        return data[i].nim < data[j].nim
    })

    // Pencarian Binary Search berdasarkan NIM
    nimDicari := "2206"
    idxBin, iterBin := BinarySearch_3(data, n, nimDicari)

    fmt.Printf("\nBinary Search - Cari NIM '%s'\n", nimDicari)
    if idxBin != -1 {
        fmt.Printf("Ditemukan di indeks: %d, Iterasi: %d\n", idxBin,
iterBin)
    } else {
        fmt.Printf("Tidak ditemukan, Iterasi: %d\n", iterBin)
    }
}
```

```
Sequential Search - Cari nama 'Fajar'
Ditemukan di indeks: 5, Iterasi: 6

Binary Search - Cari NIM '2206'
Ditemukan di indeks: 5, Iterasi: 3
```

Penjelasan :

Kode di atas adalah implementasi pencarian Sequential dan Binary Search dalam array mahasiswa yang berisi nama, NIM, kelas, jurusan, dan IPK.

Penjelasan Singkat

1. Struct mahasiswa
    - Digunakan untuk menyimpan data mahasiswa seperti nama, nim, kelas, jurusan, dan ipk.

    - Disimpan dalam array arrMhs.
2. Pencarian Sequential (SeqSearch_3)
    - Mencari nama mahasiswa dalam array secara berurutan.
    - Jika ditemukan, mengembalikan indeksnya dan jumlah iterasi.
    - Kompleksitas waktu: O(N) (linear).
3. Pencarian Binary (BinarySearch_3)
    - Mencari NIM mahasiswa, tetapi hanya bekerja jika data sudah terurut berdasarkan NIM.
    - Membagi array menjadi dua bagian untuk pencarian lebih cepat.
    - Jika ditemukan, mengembalikan indeksnya dan jumlah iterasi.
    - Kompleksitas waktu: O(log N) (lebih cepat dari Sequential Search).
4. Fungsi main()
    - Mengisi data mahasiswa secara manual.
    - Mencari nama "Fajar" dengan Sequential Search dan mencetak hasilnya.
    - Mengurutkan array berdasarkan NIM untuk Binary Search.
    - Mencari NIM "2206" dengan Binary Search dan mencetak hasilnya.

IV. UNGUIDED I

```go
//Muhammad Zaky Mubarok
package main

import (
    "fmt"
    "sort"
)


func seqSearch(arr []int, target int) bool {
    for _, nilai := range arr {
        if nilai == target {
            return true
        }
    }
    return false
}


func binSearch(arr []int, target int) bool {
    bawah := 0
    atas := len(arr) - 1
    for bawah <= atas {
        tengah := (bawah + atas) / 2
        if arr[tengah] == target {
            return true
        } else if target < arr[tengah] {
            atas = tengah - 1
        } else {
            bawah = tengah + 1
        }
    }
    return false
}

func main() {

    calonKetua := []int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20}
    var dataSuara []int
    var suara int
```

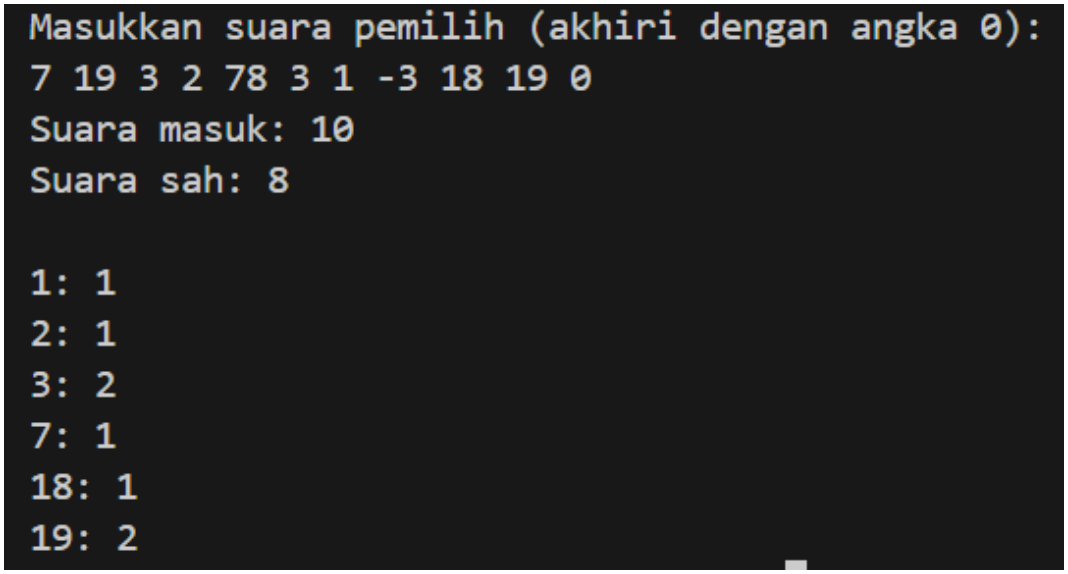
```
    fmt.Println("Masukkan suara pemilih (akhiri dengan angka 0):")
    for {
        fmt.Scan(&suara)
        if suara == 0 {
            break
        }
        dataSuara = append(dataSuara, suara)
    }

    jumlahSuara := len(dataSuara)
    jumlahSuaraSah := 0
    perolehanSuara := make(map[int]int)


    sort.Ints(calonKetua)
    for _, suara := range dataSuara {
        if len(calonKetua) <= 10 {
            if seqSearch(calonKetua, suara) {
                jumlahSuaraSah++
                perolehanSuara[suara]++
            }
        } else {
            if binSearch(calonKetua, suara) {
                jumlahSuaraSah++
                perolehanSuara[suara]++
            }
        }
    }


    fmt.Println("Suara masuk:", jumlahSuara)
    fmt.Println("Suara sah:", jumlahSuaraSah)
    fmt.Println()


    for calon := 1; calon <= 20; calon++ {
        if perolehanSuara[calon] > 0 {
            fmt.Printf("%d: %d\n", calon, perolehanSuara[calon])
        }
    }
}
```



Penjelasan :

Kode ini adalah program sederhana dalam Go untuk menghitung perolehan suara dalam sebuah pemilihan ketua. Berikut penjelasan singkat:

- **Data Pemilih**: Program menerima input suara dari pengguna hingga angka **0** dimasukkan, yang menandai akhir proses pemungutan suara.
- **Metode Pencarian**: Untuk memastikan suara hanya diberikan kepada calon yang valid, program menggunakan **Sequential Search** jika daftar calon berjumlah 10 atau kurang, dan **Binary Search** jika lebih dari 10.
- **Penyimpanan dan Perhitungan**: Suara yang valid dihitung dan disimpan dalam sebuah map `perolehanSuara` untuk mengetahui jumlah suara tiap calon.
- **Hasil Akhir**: Setelah semua suara diproses, program menampilkan total suara yang masuk dan jumlah suara sah, serta daftar calon yang menerima suara beserta jumlahnya.

V. UNGUIDED II

```go
//Muhammad Zaky Mubarok
package main

import (
    "fmt"
    "sort"
)


func binSearch(arr []int, target int) bool {
    bawah := 0
    atas := len(arr) - 1
    for bawah <= atas {
        tengah := (bawah + atas) / 2
        if arr[tengah] == target {
            return true
        } else if target < arr[tengah] {
            atas = tengah - 1
        } else {
            bawah = tengah + 1
        }
    }
    return false
}

func main() {

    calonKetua := []int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20}

    var dataSuara []int
    var suara int


    fmt.Println("Masukkan suara pemilih (akhiri dengan angka 0):")
    for {
        fmt.Scan(&suara)
        if suara == 0 {
            break
        }
        dataSuara = append(dataSuara, suara)
    }
```

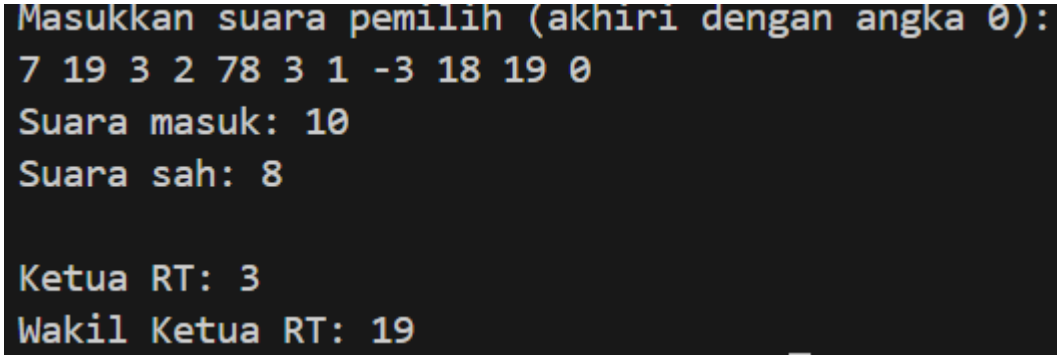
```
    jumlahSuara := len(dataSuara)
    jumlahSuaraSah := 0
    perolehanSuara := make(map[int]int)


    sort.Ints(calonKetua)
    for _, suara := range dataSuara {
        if binSearch(calonKetua, suara) {
            jumlahSuaraSah++
            perolehanSuara[suara]++
        }
    }


    var daftarCalon []int
    for calon := range perolehanSuara {
        daftarCalon = append(daftarCalon, calon)
    }
    sort.Slice(daftarCalon, func(i, j int) bool {
        return perolehanSuara[daftarCalon[i]] >
perolehanSuara[daftarCalon[j]] ||
            (perolehanSuara[daftarCalon[i]] ==
perolehanSuara[daftarCalon[j]] && daftarCalon[i] < daftarCalon[j])
    })


    fmt.Println("Suara masuk:", jumlahSuara)
    fmt.Println("Suara sah:", jumlahSuaraSah)


    if len(daftarCalon) > 0 {
        fmt.Println("\nKetua RT:", daftarCalon[0])
        if len(daftarCalon) > 1 {
            fmt.Println("Wakil Ketua RT:", daftarCalon[1])
        } else {
            fmt.Println("Wakil Ketua RT: Belum ada cukup suara")
        }
    } else {
        fmt.Println("\nBelum ada suara yang sah untuk menentukan
Ketua RT.")
    }
}
```



Penjelasan :

- **Input Data**: Pengguna memasukkan suara pemilih satu per satu hingga angka **0** dimasukkan sebagai tanda berhenti.

- **Pencarian Kandidat**: Program menggunakan **Binary Search** untuk memastikan bahwa suara hanya diberikan kepada calon yang valid.
- **Perhitungan Suara**: Suara sah dicatat dalam sebuah map `perolehanSuara`, yang merekam jumlah suara yang diterima setiap calon.
- **Menentukan Ketua & Wakil**: Program mengurutkan calon berdasarkan jumlah suara yang diperoleh. Kandidat dengan suara terbanyak ditetapkan sebagai **Ketua RT**, dan posisi **Wakil Ketua RT** diberikan kepada peraih suara terbanyak kedua.
- **Hasil Akhir**: Program mencetak jumlah total suara masuk, jumlah suara sah, serta nama Ketua dan Wakil Ketua RT—atau menampilkan pesan jika belum ada cukup suara untuk menentukan pimpinan.

VI. UNGUIDED III

```go
//Muhammad Zaky Mubarok
package main

import "fmt"

const NMAX = 1000000
var data [NMAX]int

func main() {

    var n, k int
    fmt.Scan(&n, &k)


    isiArray(n)


    pos := posisi(n, k)
    if pos != -1 {
        fmt.Println(pos)
    } else {
        fmt.Println("TIDAK ADA")
    }
}


func isiArray(n int) {
    for i := 0; i < n; i++ {
        fmt.Scan(&data[i])
    }
}


func posisi(n, k int) int {
    bawah, atas := 0, n-1
    for bawah <= atas {
        tengah := (bawah + atas) / 2
        if data[tengah] == k {
            return tengah
        } else if k < data[tengah] {
            atas = tengah - 1
        } else {
            bawah = tengah + 1
        }
```

```
    }
    return -1
}
```

```
12 534
1 3 8 16 32 123 323 323 534 543 823 999
8
```

Penjelasan :

Kode ini adalah implementasi **Binary Search** dalam Go untuk mencari posisi suatu elemen dalam array. Berikut ringkasannya:

- **Deklarasi & Input Data**:
  - `data[NMAX]` menyimpan elemen hingga **1 juta** angka.
  - Program meminta jumlah elemen (`n`) dan angka yang dicari (`k`).
  - Array diisi melalui fungsi `isiArray(n)`, membaca input satu per satu.
- **Proses Pencarian**:

- Fungsi `posisi(n, k)` menerapkan **Binary Search** untuk menemukan indeks angka `k`.
- Jika angka ditemukan, indeksnya dicetak.
- Jika tidak ditemukan, program mencetak `"TIDAK ADA"`.

VII. KESIMPULAN

Modul ini membahas dua metode pencarian dalam himpunan data: **Sequential Search** dan **Binary Search**, serta bagaimana memilih algoritma yang paling sesuai dengan kebutuhan.

Kesimpulan:

1. **Sequential Search (Pencarian Berurutan)**:
    - Dilakukan dengan memeriksa elemen satu per satu dari awal hingga akhir.
    - Cocok untuk **data yang tidak terurut** atau jumlah data yang kecil.
    - Kompleksitas waktu **O(n)**, karena perlu mengecek setiap elemen satu per satu.
2. **Binary Search (Pencarian Biner)**:
    - Memanfaatkan keterurutan data, dengan membagi rentang pencarian menjadi dua secara rekursif atau iteratif.
    - Cocok untuk **data yang telah terurut secara ascending atau descending**.
    - Kompleksitas waktu **O(log n)**, jauh lebih efisien dibandingkan pencarian sekuensial.
3. **Penerapan dalam Kasus Nyata**:
    - Pencarian dalam database mahasiswa (menggunakan nama/NIM).
    - Pencarian alamat rumah dalam daftar terurut.
    - Mencari indeks data dalam array.
4. **Perbedaan utama**:

- **Sequential Search** lebih fleksibel karena tidak membutuhkan keterurutan data.
- **Binary Search** jauh lebih cepat tetapi hanya bekerja pada data yang sudah terurut.

Pemilihan algoritma pencarian bergantung pada kondisi dataset dan kebutuhan efisiensi. Untuk dataset kecil atau tidak terurut, **Sequential Search** cukup digunakan. Namun, jika dataset besar dan terurut, **Binary Search** jauh lebih optimal.

REFERENSI

MODUL 11. PENCARIAN NILAI ACAK PADA HIMPUNAN DATA