LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2

MODUL 11
PENCARIAN NILAI ACAK PADA HIMPUNAN DATA

Oleh:

NAMA : Muhamad faza fahri aziz NIM : 103112400072 KELAS : 12-IF-01

S1 TEKNIK INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

I. DASAR TEORI

Metode Pencarian Data Acak dalam Bahasa Go

Pencarian data dalam himpunan nilai acak merupakan operasi penting dalam pemrograman, khususnya dalam bahasa Go yang menyediakan array dan slice sebagai struktur data utama. Metode pencarian yang umum digunakan adalah sekuensial search dan binary search.

Sekuensial Search memeriksa data satu per satu secara berurutan tanpa memerlukan data terurut. Metode ini mudah diimplementasikan dan sangat berguna untuk dataset kecil atau yang tidak terstruktur, namun kurang efisien untuk data besar karena waktu pencarian bertambah seiring jumlah data.

Binary Search memerlukan data yang sudah terurut dan menggunakan prinsip pembagian ruang pencarian menjadi dua bagian pada setiap langkah. Dengan cara ini, binary search mampu mengurangi jumlah perbandingan secara signifikan, sehingga sangat efisien untuk dataset besar.

Selain itu, dalam bahasa Go, array bertipe data struct memungkinkan penyimpanan data dengan banyak atribut. **Pencarian pada array struct** dilakukan dengan membandingkan atribut tertentu. Jika data terurut berdasarkan atribut tersebut, binary search dapat digunakan, jika tidak, sekuensial search menjadi pilihan utama.

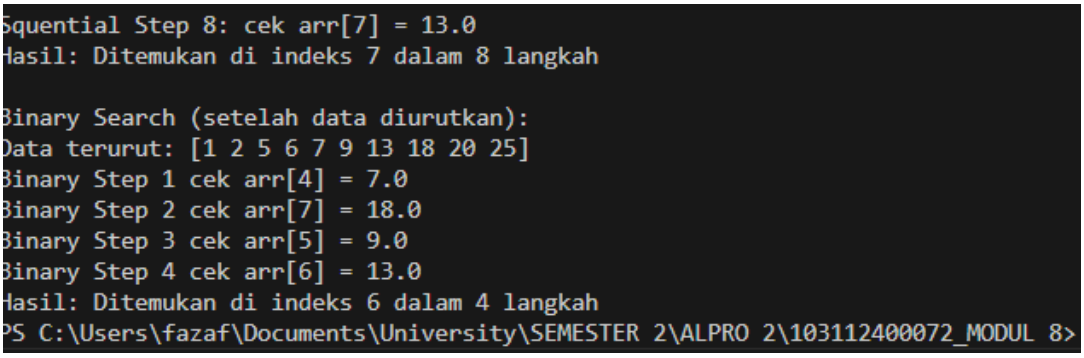
Implementasi kedua metode ini dalam Go sangat membantu dalam pengelolaan data seperti pencarian nilai dalam daftar siswa, pencarian produk dalam katalog, dan lain-lain, sehingga meningkatkan efisiensi program secara signifikan.

II. GUIDED

Contoh 1

```go
package main

import (
    "fmt"
    "sort"
)

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

func binarySearch(arr []float64, target float64) (int, int) {
    iterations := 0
    low := 0
    high := len(arr) - 1

    for low <= high {
        iterations++
        mid := (low + high) / 2
        fmt.Printf("Binary Step %d cek arr[%d] = %.1f\n", iterations, mid, arr[mid])

        if arr[mid] == target {
            return mid, iterations
        } else if target < arr[mid] {
            high = mid - 1
        } else {
            low = mid + 1
        }
    }
    return -1, iterations
}

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

    fmt.Println("Squential 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 setelah %d langkah\n\n", iterSeq)
    }

    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 setelah %d langkah\n", iterBin)
    }
}
```

Output:



Penjelasan:

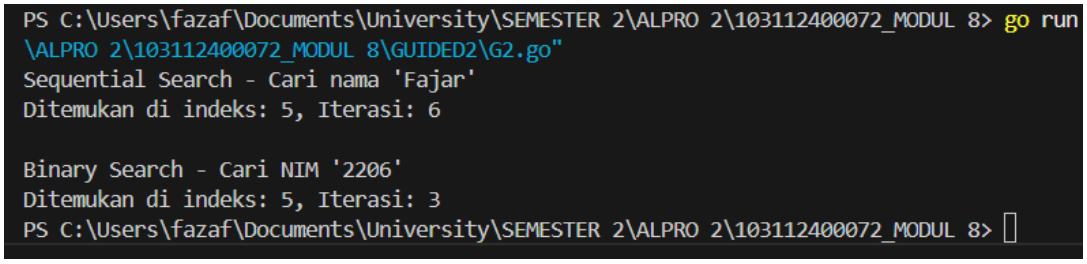
Program ini ditulis dalam bahasa Go dan dirancang untuk mencari sebuah nilai dalam array berisi angka desimal menggunakan dua metode: pencarian sekuensial pada data acak dan pencarian biner pada data yang sudah diurutkan. Dalam metode pencarian sekuensial, program memeriksa elemen satu per satu sampai nilai yang dicari ditemukan atau seluruh data telah diperiksa. Setelah data diurutkan, pencarian biner dilakukan dengan cara membagi ruang pencarian menjadi dua secara berulang, sehingga prosesnya menjadi lebih cepat dan efisien.

Setiap tahapan pencarian ditampilkan di layar untuk membantu pengguna memahami cara kerja masing-masing metode. Hasil akhir dari pencarian berupa indeks tempat nilai ditemukan dan jumlah langkah yang diperlukan, yang menunjukkan bahwa pencarian biner lebih efisien dibandingkan pencarian sekuensial pada data yang sudah terurut.

Contoh Soal 2

```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 mod int
    var kr int = 0
    var kn int = n - 1
    var iterasi int = 0

    for kr <= kn && found == -1 {
        iterasi++
        mod = (kr + kn) / 2
        if X < T[mod].nim {
            kn = mod - 1
        } else if X > T[mod].nim {
            kr = mod + 1
        } else {
            found = mod
        }
    }
    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)
    }
}
```

Output:



Penjelasan

Program ini ditulis menggunakan bahasa Go untuk mencari data mahasiswa dalam array yang bertipe struct. Pencarian dilakukan dengan dua metode berbeda: metode pertama adalah **sequential search** yang mencari nama mahasiswa secara berurutan dari awal hingga akhir array. Metode kedua adalah **binary search**, yang digunakan untuk mencari NIM mahasiswa setelah data terlebih dahulu diurutkan berdasarkan NIM.

Sequential search efektif digunakan pada data yang belum diurutkan, sementara binary search menawarkan kecepatan lebih tinggi namun mensyaratkan data dalam keadaan terurut. Program ini juga menampilkan indeks hasil pencarian dan jumlah langkah yang dilakukan, sehingga pengguna dapat membandingkan efisiensi kedua metode dengan lebih jelas.

.

III. UNGUIDED

Soal 1

```go
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])
        }
    }
}
```

Output:

```
Masukkan suara pemilih (akhiri dengan angka 0):
7 19 3 2 78 3 1 -3 18 19 0
Suara masuk: 10
Suara sah: 8

1: 1
2: 1
3: 2
7: 1
18: 1
19: 2
PS C:\Users\fazaf\Documents\University\SEMESTER 2\ALPRO 2\103112400072_MODUL 8>
```

Penjelasan

Program ini merupakan implementasi sederhana dalam bahasa Go yang digunakan untuk menghitung hasil suara dalam sebuah pemilihan ketua. Berikut ringkasan fungsionalitasnya:

- **Input Suara**: Program menerima masukan suara dari pengguna secara berurutan hingga pengguna memasukkan angka 0, yang menjadi penanda bahwa proses pemungutan suara telah selesai.
- **Validasi Suara**: Untuk memastikan hanya calon yang sah yang menerima suara, program menggunakan dua metode pencarian: **Sequential Search** apabila jumlah calon 10 atau kurang, dan **Binary Search** jika jumlah calon lebih dari 10.
- **Penyimpanan Suara**: Suara yang valid dicatat dalam struktur data **map** bernama perolehanSuara, yang digunakan untuk menyimpan dan menghitung jumlah suara yang diperoleh oleh masing-masing calon.
- **Output Akhir**: Setelah semua suara diproses, program menampilkan informasi berupa total suara yang masuk, jumlah suara sah, serta rincian jumlah suara yang diperoleh setiap calon yang mendapatkan suara.

Program ini menunjukkan penggunaan logika pencarian yang adaptif dan penyimpanan data yang efisien untuk keperluan pemrosesan suara dalam konteks pemilihan sederhana.

Soal 2

```go
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.")
    }
}
```

Output:

```
PS C:\Users\fazaf\Documents\University\SEMESTER 2\ALPRO 2\103112400072_MODUL 8> go run "c:\Us
Masukkan suara pemilih (akhiri dengan angka 0):
7 19 3 2 78 3 1 -3 18 19 0
Suara masuk: 10
Suara sah: 8

Ketua RT: 3
Wakil Ketua RT: 19
PS C:\Users\fazaf\Documents\University\SEMESTER 2\ALPRO 2\103112400072_MODUL 8>
```

Penjelasan

Program ini memanfaatkan metode **binary search** untuk memverifikasi keabsahan setiap suara sebelum dilakukan penghitungan jumlah suara untuk masing-masing calon. Data suara dibaca sekaligus dalam satu baris, lalu setiap suara diperiksa apakah termasuk dalam daftar calon sah (nomor 1 hingga 20) yang telah diurutkan sebelumnya.

Setelah proses validasi selesai, suara yang dianggap sah akan dihitung dan disimpan menggunakan struktur data **map** untuk mencatat perolehan suara tiap calon. Kemudian, hasil perolehan suara akan diurutkan berdasarkan jumlah suara secara **menurun**, dan jika terdapat calon dengan jumlah suara yang sama, maka diurutkan berdasarkan **nomor calon yang lebih kecil**. Dengan proses ini, program dapat dengan mudah menentukan siapa yang menjadi ketua dan wakil ketua berdasarkan jumlah suara terbanyak

Soal 3

```go
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
}
```

Output:

```
> go run "c:\Users\fazaf\Documents\University
12 534
1 3 8 16 32 123 323 534 543 823 999
8
7
PS C:\Users\fazaf\Documents\University\SEMESTER 2\ALPRO 2\103112400072_MODUL 8>
```

Penjelasan:

Program ini menerapkan metode **pencarian sekuensial (linear search)**, yaitu dengan memeriksa setiap elemen dalam array secara berurutan, dimulai dari indeks 0 hingga indeks ke-(n-1). Input program terdiri dari tiga bagian: jumlah elemen array (n), nilai yang ingin dicari (k), serta data array yang telah **terurut secara menaik**.

Fungsi isiArray bertugas untuk mengisi array dengan data yang dibaca dari input. Sedangkan fungsi posisi melakukan proses pencarian linear dengan membandingkan setiap elemen dalam array dengan nilai k. Jika nilai tersebut ditemukan, fungsi akan mengembalikan **indeks** dari elemen tersebut. Namun jika tidak ditemukan, fungsi akan mengembalikan nilai **-1** sebagai penanda bahwa elemen tidak ada dalam array.

IV. KESIMPULAN

Pencarian sekuensial dan binary search merupakan dua metode utama untuk mencari data dalam kumpulan nilai acak. Pencarian sekuensial mudah digunakan karena tidak memerlukan data yang terurut, namun kurang efisien terutama jika data berukuran besar karena harus memeriksa tiap elemen satu per satu. Sebaliknya, binary search membutuhkan data yang sudah terurut, tetapi memberikan kecepatan pencarian yang jauh lebih tinggi dengan cara membagi ruang pencarian secara berulang secara sistematis. Oleh karena itu, binary search lebih tepat digunakan untuk data yang terurut dan berukuran besar, sedangkan pencarian sekuensial lebih cocok untuk data yang kecil atau tidak terurut. Kedua metode ini saling melengkapi, dan pemilihannya harus disesuaikan dengan karakteristik data serta kebutuhan efisiensi pencarian.

REFERENSI

Modul 11 Algoritma Pemrograman Pencarian Nilai Acak pada Himpunan Data