LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2

MODUL 11

PENCARIAN NILAI EKSTIM PADA HIMPUNAN DATA

Disusun Oleh :

Ariiq Radhitya Pradana / 2311102260

IF 11 06

Dosen Pengampu :

ABEDNEGO DWI SEPTIADI

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. DASAR TEORI

Pencarian nilai ekstrem pada himpunan data, yaitu nilai maksimum dan minimum, merupakan konsep penting dalam analisis data. Dalam konteks ini, nilai maksimum adalah titik di mana fungsi mencapai nilai tertinggi, sedangkan nilai minimum adalah titik di mana fungsi mencapai nilai terendah. Untuk menemukan nilai ekstrem, kita dapat menggunakan metode analisis titik kritis, di mana kita menghitung turunan pertama dari fungsi dan menyamakannya dengan nol untuk menemukan kandidat titik ekstrem. Selain itu, evaluasi batas interval juga penting untuk memastikan bahwa kita tidak melewatkan nilai ekstrem yang mungkin terjadi di tepi interval tersebut. Dalam implementasinya menggunakan bahasa pemrograman Go (Golang), kita dapat membuat fungsi yang menerima himpunan data berupa slice dari float64 dan mengembalikan nilai maksimum dan minimum. Contohnya, dengan menggunakan loop untuk membandingkan setiap elemen dalam himpunan data, kita dapat menentukan nilai maksimum dan minimum secara efisien. Dengan demikian, pemahaman tentang dasar teori pencarian nilai ekstrem dan penerapannya dalam kode memberikan alat yang berguna untuk melakukan analisis mendalam pada data yang tersedia.

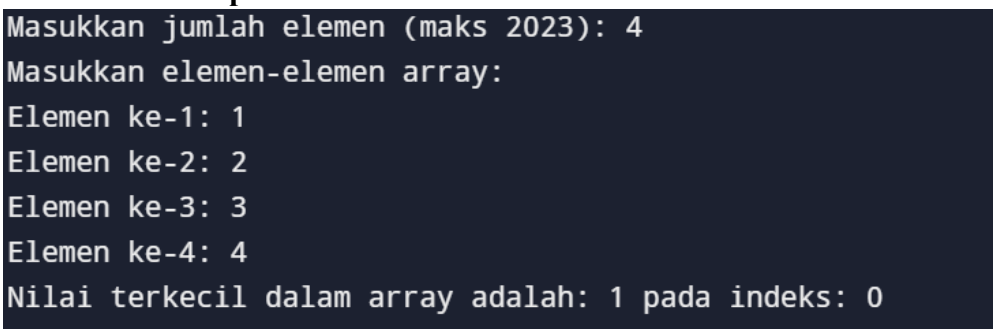
II. GUIDED

Guided 1

```
package main

import "fmt"

// Mendeklarasikan tipe data array arrInt dengan panjang
2023
type arrInt [2023]int

// Fungsi untuk mencari indeks elemen terkecil dalam
array
func terkecil(tabInt arrInt, n int) int {
    var idx int = 0 // idx menyimpan indeks elemen
terkecil
    var j int = 1
    for j < n {
        if tabInt[idx] > tabInt[j] {
            idx = j // Simpan indeks j jika elemen di
indeks j lebih kecil
        }
        j = j + 1
    }
    return idx
}

// Fungsi main untuk menguji fungsi terkecil
func main() {
    var n int
    var tab arrInt

    // Meminta input jumlah elemen array
    fmt.Print("Masukkan jumlah elemen (maks 2023): ")
    fmt.Scan(&n)

    // Validasi input jumlah elemen
    if n < 1 || n > 2023 {
        fmt.Println("Jumlah elemen harus antara 1 dan
2023.")
        return
    }

    // Memasukkan elemen-elemen array
    fmt.Println("Masukkan elemen-elemen array:")
    for i := 0; i < n; i++ {
        fmt.Print("Elemen ke-", i+1, ": ")
        fmt.Scan(&tab[i])
    }

    // Memanggil fungsi terkecil untuk menemukan indeks
elemen terkecil
```

```
    idxMin := terkecil(tab, n)

    // Menampilkan nilai dan indeks terkecil
    fmt.Println("Nilai terkecil dalam array adalah:",
tab[idxMin], "pada indeks:", idxMin)
}
```

Screenshoot Output



Deskripsi Program

Kode tersebut merupakan program dalam bahasa Go yang bertujuan untuk mencari nilai terkecil dalam sebuah array dan menampilkan indeksnya. Program dimulai dengan mendeklarasikan tipe data array bernama arrInt yang memiliki panjang tetap sebanyak 2023 elemen bertipe integer. Kemudian, terdapat fungsi bernama terkecil yang menerima dua parameter, yaitu array bertipe arrInt dan sebuah integer n yang menunjukkan jumlah elemen efektif dalam array. Fungsi ini menggunakan perulangan untuk membandingkan setiap elemen array, menyimpan indeks elemen terkecil dalam variabel idx, dan mengembalikannya sebagai hasil. Di dalam fungsi main, program meminta pengguna untuk memasukkan jumlah elemen array yang akan diproses, dengan batas validasi antara 1 hingga 2023. Jika jumlah elemen valid, pengguna diminta untuk menginput nilai setiap elemen array. Setelah itu, fungsi terkecil dipanggil dengan parameter array dan jumlah elemen untuk mencari indeks elemen terkecil. Program akhirnya menampilkan nilai terkecil beserta indeksnya. Validasi input serta pesan yang jelas membantu pengguna memahami langkah-langkah yang harus dilakukan saat menjalankan program.

Guided 2

```
package main

import "fmt"

// Definisi struct mahasiswa dengan atribut nama, nim,
kelas, jurusan, dan ipk
type mahasiswa struct {
    nama, nim, kelas, jurusan string
    ipk                       float64
}

// Definisi tipe data array mahasiswa dengan kapasitas
maksimal 2023
type arrMhs [2023]mahasiswa

// Fungsi untuk mencari IPK tertinggi dalam array
mahasiswa
func ipk(T arrMhs, n int) float64 {
    var tertinggi float64 = T[0].ipk
    var j int = 1
    for j < n {
        if tertinggi < T[j].ipk {
            tertinggi = T[j].ipk
        }
        j = j + 1
    }
    return tertinggi
}

// Fungsi main untuk mengisi data mahasiswa dan mencari
IPK tertinggi
func main() {
    var n int
    var dataMhs arrMhs

    // Meminta input jumlah mahasiswa
    fmt.Print("Masukkan jumlah mahasiswa (maks 2023): ")
    fmt.Scan(&n)

    // Validasi jumlah mahasiswa yang dimasukkan
    if n < 1 || n > 2023 {
        fmt.Println("Jumlah mahasiswa harus antara 1 dan
2023.")
        return
    }

    // Mengisi data mahasiswa
    for i := 0; i < n; i++ {
        fmt.Printf("\nMasukkan data mahasiswa ke-%d\n",
i+1)
```

```
        fmt.Print("Nama: ")
        fmt.Scan(&dataMhs[i].nama)
        fmt.Print("NIM: ")
        fmt.Scan(&dataMhs[i].nim)
        fmt.Print("Kelas: ")
        fmt.Scan(&dataMhs[i].kelas)
        fmt.Print("Jurusan: ")
        fmt.Scan(&dataMhs[i].jurusan)
        fmt.Print("IPK: ")
        fmt.Scan(&dataMhs[i].ipk)
    }

    // Mencari dan menampilkan IPK tertinggi
    tertinggi := ipk(dataMhs, n)
    fmt.Printf("\nIPK tertinggi dari %d mahasiswa
adalah: %.2f\n", n, tertinggi)

}
```

Screenshoot Output

```
Masukkan jumlah mahasiswa (maks 2023): 2

Masukkan data mahasiswa ke-1
Nama: vin
NIM: 2311102260
Kelas: 6
Jurusan: informatika
IPK: 4

Masukkan data mahasiswa ke-2
Nama: 4
NIM: ris
Kelas: 2311102250
Jurusan: rpl
IPK: 4

IPK tertinggi dari 2 mahasiswa adalah: 4.00
```

Deskripsi Program

Kode di atas adalah program dalam bahasa Go yang dirancang untuk mencatat data sejumlah mahasiswa, kemudian menentukan IPK (Indeks Prestasi Kumulatif) tertinggi dari data yang dimasukkan. Program dimulai dengan mendefinisikan sebuah struct bernama mahasiswa, yang memiliki atribut seperti nama, nim, kelas, jurusan, dan ipk. Kemudian, didefinisikan pula tipe data array bernama arrMhs dengan kapasitas maksimal 2023 elemen, di mana setiap elemen bertipe mahasiswa.

Program memiliki fungsi bernama ipk yang bertugas mencari IPK tertinggi dari array mahasiswa. Fungsi ini menerima array mahasiswa dan jumlah elemen efektifnya sebagai parameter. Proses pencarian dilakukan dengan membandingkan setiap elemen array dalam perulangan, dan hasil tertinggi disimpan dalam variabel tertinggi.

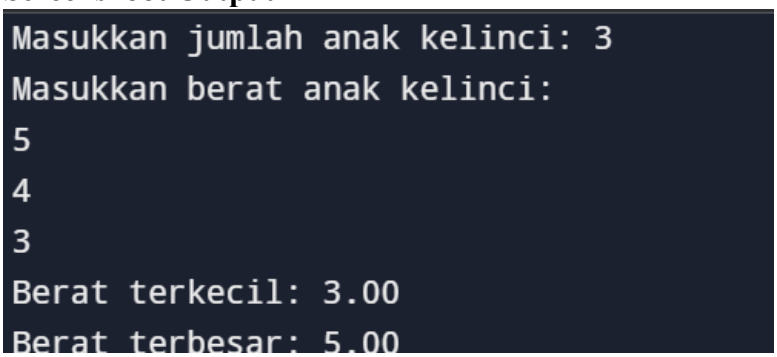
Di dalam fungsi main, program meminta input dari pengguna berupa jumlah mahasiswa yang akan diolah. Jumlah ini divalidasi agar tidak melebihi kapasitas maksimal (2023) dan tidak kurang dari 1. Selanjutnya, untuk setiap mahasiswa, pengguna diminta memasukkan data seperti nama, NIM, kelas, jurusan, dan IPK. Setelah data semua mahasiswa dimasukkan, fungsi ipk dipanggil untuk menghitung IPK tertinggi. Hasilnya ditampilkan dalam format desimal dengan dua angka di belakang koma. Program ini berguna untuk mengelola data mahasiswa dan menganalisis performa akademik mereka berdasarkan IPK.

III. UNGUIDED

Unguided 1

```
package main

import (
    "fmt"
)

func main() {
    var n int

    // Memasukkan jumlah anak kelinci
    fmt.Print("Masukkan jumlah anak kelinci: ")
    fmt.Scan(&n)

    // Validasi jumlah anak kelinci
    if n <= 0 || n > 1000 {
        fmt.Println("Jumlah anak kelinci harus di antara
1 dan 1000.")
        return
    }

    weights := make([]float64, n)

    // Memasukkan berat anak kelinci
    fmt.Println("Masukkan berat anak kelinci:")
    for i := 0; i < n; i++ {
        fmt.Scan(&weights[i])
    }

    // Mencari berat terkecil dan terbesar
    minWeight := weights[0]
    maxWeight := weights[0]

    for _, weight := range weights {
        if weight < minWeight {
            minWeight = weight
        }
        if weight > maxWeight {
            maxWeight = weight
        }
    }

    // Menampilkan hasil
    fmt.Printf("Berat terkecil: %.2f\n", minWeight)
    fmt.Printf("Berat terbesar: %.2f\n", maxWeight)
}
```

Screenshoot Output



Deskripsi Program

Kode di atas adalah program dalam bahasa Go yang digunakan untuk menentukan berat terkecil dan terbesar dari sejumlah anak kelinci. Program dimulai dengan meminta pengguna memasukkan jumlah anak kelinci yang akan diolah. Jumlah ini divalidasi agar berada dalam rentang 1 hingga 1000. Jika jumlah anak kelinci tidak valid, program akan berhenti dan memberikan pesan kesalahan.

Setelah jumlah anak kelinci valid dimasukkan, program menggunakan slice bertipe float64 untuk menyimpan berat masing-masing anak kelinci. Pengguna diminta menginput berat setiap anak kelinci satu per satu. Setelah semua data berat dimasukkan, program melakukan pencarian berat terkecil dan terbesar menggunakan perulangan for. Setiap berat dibandingkan dengan nilai terkecil (minWeight) dan terbesar (maxWeight) yang ditemukan sejauh ini, dan nilai-nilai tersebut diperbarui jika ditemukan berat yang lebih kecil atau lebih besar.

Hasil akhir berupa berat terkecil dan terbesar ditampilkan dalam format desimal dengan dua angka di belakang koma. Program ini dirancang untuk memproses data secara dinamis menggunakan slice, sehingga lebih fleksibel dibandingkan penggunaan array dengan panjang tetap.

Unguided 2

```
package main

import (
    "fmt"
)

func main() {
    // Step 1: Penjelasan Program
    fmt.Println("Program ini mendistribusikan berat ikan
ke dalam wadah.")
    fmt.Println("Masukkan jumlah ikan (x) dan jumlah
wadah (y), diikuti berat ikan masing-masing.")

    // Step 2: Input
    var x, y int
    fmt.Print("Masukkan jumlah ikan (x) dan jumlah wadah
(y): ")
    fmt.Scan(&x, &y)

    // Membuat slice untuk menyimpan berat ikan
    fishWeights := make([]float64, x)
    fmt.Printf("Masukkan %d berat ikan (pisahkan dengan
spasi): ", x)
    for i := 0; i < x; i++ {
        fmt.Scan(&fishWeights[i])
    }

    // Step 3: Distribusi ikan ke dalam wadah
    buckets := make([][]float64, y)
    for i, weight := range fishWeights {
        bucketIndex := i % y // Distribusi secara
bergilir (round-robin)
        buckets[bucketIndex] =
append(buckets[bucketIndex], weight)
    }

    // Step 4: Perhitungan total dan rata-rata berat di
setiap wadah
    totalWeights := make([]float64, y)
    averageWeights := make([]float64, y)

    for i := 0; i < y; i++ {
        var totalWeight float64
        for _, weight := range buckets[i] {
            totalWeight += weight
        }
        totalWeights[i] = totalWeight
        if len(buckets[i]) > 0 {
            averageWeights[i] = totalWeight /
float64(len(buckets[i]))
```

```
            }
        }

        // Step 5: Output Hasil
        fmt.Println("\nHasil Distribusi:")
        fmt.Println("Total berat di setiap wadah:")
        for i, total := range totalWeights {
            fmt.Printf("Wadah %d: %.2f\n", i+1, total)
        }

        fmt.Println("\nRata-rata berat ikan di setiap
    wadah:")
        for i, avg := range averageWeights {
            fmt.Printf("Wadah %d: %.2f\n", i+1, avg)
        }
    }
```

Screenshoot Output

```
Program ini mendistribusikan berat ikan ke dalam wadah.
Masukkan jumlah ikan (x) dan jumlah wadah (y), diikuti berat ikan
    masing-masing.
Masukkan jumlah ikan (x) dan jumlah wadah (y): 6 3
Masukkan 6 berat ikan (pisahkan dengan spasi): 6 5 4 3 2 1

Hasil Distribusi:
Total berat di setiap wadah:
Wadah 1: 9.00
Wadah 2: 7.00
Wadah 3: 5.00

Rata-rata berat ikan di setiap wadah:
Wadah 1: 4.50
Wadah 2: 3.50
Wadah 3: 2.50
```

]

Deskripsi Program

Kode di atas adalah program dalam bahasa Go yang bertujuan untuk mendistribusikan berat sejumlah ikan ke dalam wadah-wadah secara bergilir (round-robin). Program dimulai dengan menjelaskan tujuan kepada pengguna, yaitu mengelompokkan berat ikan ke dalam wadah dan menghitung total serta rata-rata berat ikan di setiap wadah. Pengguna diminta untuk memasukkan jumlah ikan (x) dan jumlah wadah (y), kemudian memasukkan berat masing-masing ikan.

Program menyimpan berat ikan dalam sebuah slice fishWeights. Selanjutnya, proses distribusi dilakukan menggunakan algoritma round-robin, di mana setiap berat ikan dimasukkan ke wadah secara bergantian berdasarkan indeks modulus (i % y). Data wadah disimpan dalam slice dua dimensi buckets.

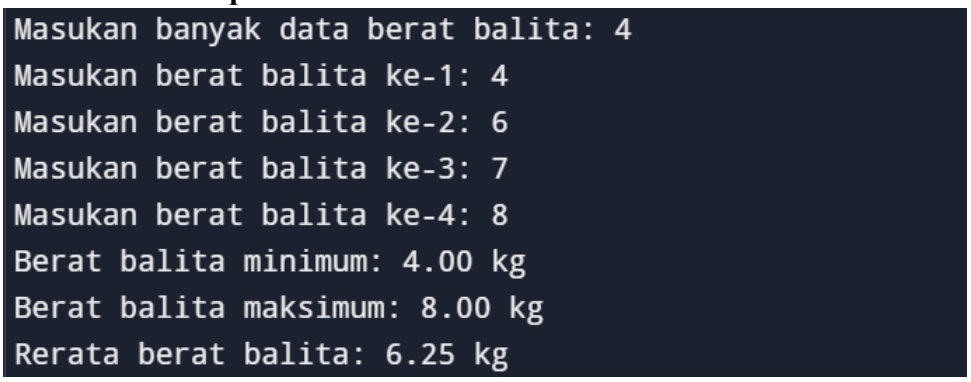
Setelah distribusi selesai, program menghitung total berat ikan di setiap wadah dengan menjumlahkan semua berat dalam masing-masing wadah. Rata-rata berat ikan di setiap wadah juga dihitung dengan membagi total berat dengan jumlah ikan dalam wadah tersebut, jika wadah tidak kosong. Hasil distribusi ditampilkan kepada pengguna, termasuk total berat ikan dan rata-rata berat ikan di setiap wadah. Informasi ini disajikan dalam format yang rapi, dengan setiap wadah diberi nomor dan nilai total serta rata-ratanya ditampilkan hingga dua angka di belakang koma. Program ini membantu mengelola distribusi berat ikan dengan efisien, terutama dalam kasus dengan jumlah ikan dan wadah yang bervariasi.

Unguided 3

```
package main

import (
    "fmt"
)

// Mendefinisikan tipe data arrBalita sebagai array
float64 dengan ukuran maksimum 100
type arrBalita [100]float64

// Fungsi untuk menghitung berat minimum dan maksimum
func hitungMinMax(arrBerat []float64, bMin, bMax
*float64) {
    *bMin = arrBerat[0]
    *bMax = arrBerat[0]

    for _, berat := range arrBerat {
        if berat < *bMin {
            *bMin = berat
        }
        if berat > *bMax {
            *bMax = berat
        }
    }
}

// Fungsi untuk menghitung rata-rata berat
func hitungRerata(arrBerat []float64) float64 {
    var total float64
    for _, berat := range arrBerat {
        total += berat
    }
    return total / float64(len(arrBerat))
}

func main() {
    var n int
    var berat arrBalita

    // Input: jumlah data berat balita
    fmt.Print("Masukan banyak data berat balita: ")
    fmt.Scanln(&n)

    // Input: berat masing-masing balita
    for i := 0; i < n; i++ {
        fmt.Printf("Masukan berat balita ke-%d: ", i+1)
        fmt.Scanln(&berat[i])
    }

    // Membuat slice untuk data berat yang dimasukkan
    arrBerat := berat[:n]
```

```
    // Menghitung berat minimum, maksimum, dan rata-rata
    var bMin, bMax float64
    hitungMinMax(arrBerat, &bMin, &bMax)
    rerata := hitungRerata(arrBerat)

    // Output hasil
    fmt.Printf("Berat balita minimum: %.2f kg\n", bMin)
    fmt.Printf("Berat balita maksimum: %.2f kg\n", bMax)
    fmt.Printf("Rerata berat balita: %.2f kg\n", rerata)
}
```

Screenshoot Output



Deskripsi Program

Kode di atas adalah program dalam bahasa Go yang digunakan untuk menganalisis data berat badan balita. Program ini meminta pengguna untuk memasukkan jumlah balita dan berat badan masing-masing balita. Berat badan balita disimpan dalam array statis bernama arrBalita dengan ukuran maksimum 100 elemen. Data yang valid kemudian diolah dalam bentuk slice agar sesuai dengan jumlah balita yang dimasukkan.

Program memiliki dua fungsi utama: hitungMinMax dan hitungRerata. Fungsi hitungMinMax digunakan untuk menentukan berat badan minimum dan maksimum balita dalam array, dengan memanfaatkan parameter pointer untuk memperbarui nilai minimum dan maksimum. Fungsi hitungRerata menghitung rata-rata berat badan balita dengan menjumlahkan semua berat badan dan membaginya dengan jumlah balita.

Di dalam fungsi main, program meminta input dari pengguna berupa jumlah data balita (n) dan berat masing-masing balita. Setelah data terkumpul, fungsi hitungMinMax dan hitungRerata dipanggil untuk menghitung nilai berat minimum, maksimum, dan rata-rata. Hasil analisis kemudian ditampilkan dalam format yang rapi dengan dua angka desimal, mencakup berat badan minimum, maksimum, dan rata-rata.

Program ini dirancang untuk memproses data balita dengan batasan ukuran array yang jelas, membuatnya efisien untuk dataset kecil hingga sedang, sambil memberikan informasi yang relevan tentang distribusi berat badan balita.