

**LAPORAN PRAKTIKUM
ALGORITMA PEMROGRAMAN 2**

MODUL X

PENCARIAN NILAI EKSTRIM PADA HIMPUNAN DATA



Oleh:

RIZKULLOH ALPRIYANSAH

2311102142

IF-11-08

PROGRAM STUDI S1 INFORMATIKA

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM PURWOKERTO

2024

I. DASAR TEORI

Pencarian nilai ekstrem pada himpunan data merupakan proses untuk menemukan nilai maksimum (terbesar) dan minimum (terkecil) dalam suatu koleksi data. Operasi ini penting dalam analisis data untuk mendapatkan wawasan awal tentang batas-batas nilai dalam dataset.

Konsep Nilai Ekstrem

1. Nilai Maksimum (Max): Nilai terbesar dalam himpunan data.
2. Nilai Minimum (Min): Nilai terkecil dalam himpunan data.

Pendekatan Algoritmik

Pencarian nilai ekstrem dapat dilakukan dengan cara iterasi sederhana:

1. Inisialisasi nilai maksimum dengan nilai terkecil (biasanya elemen pertama dari himpunan data).
2. Inisialisasi nilai minimum dengan nilai terbesar (juga elemen pertama dari himpunan data).
3. Iterasi melalui elemen-elemen himpunan data.
 - Perbarui nilai maksimum jika ditemukan elemen lebih besar.
 - Perbarui nilai minimum jika ditemukan elemen lebih kecil.

Implementasi di Golang

Golang menyediakan kontrol loop dan tipe data array atau slice yang memudahkan pencarian nilai ekstrem. Berikut adalah contoh sederhana implementasinya:

```
// Rizkulloh Alpriyansah
// 2311102142
package main

import (
    "fmt"
    "math"
)

// Function to find min and max values in a dataset
func findExtremes(data []int) (int, int) {
    if len(data) == 0 {
```

```

        panic("Dataset cannot be empty")
    }

    minVal := math.MaxInt64 // Initialize with the largest integer
    maxVal := math.MinInt64 // Initialize with the smallest integer

    for _, value := range data {
        if value > maxVal {
            maxVal = value
        }
        if value < minVal {
            minVal = value
        }
    }

    return minVal, maxVal
}

func main() {
    dataset := []int{23, 1, 45, -10, 77, 3, 0, -22}
    min, max := findExtremes(dataset)

    fmt.Printf("Minimum value: %d\n", min)
    fmt.Printf("Maximum value: %d\n", max)
}

```

1. InisialisasiNilai:

Variabel `minVal` diinisialisasi dengan nilai terbesar yang dapat direpresentasikan oleh tipe data integer (`math.MaxInt64`), sementara `maxVal` diinisialisasi dengan nilai terkecil (`math.MinInt64`). Hal ini dilakukan untuk memastikan nilai awal tidak membatasi proses pencarian nilai ekstrem dalam dataset.

2. Iterasi dalam Loop:

Setiap elemen dalam himpunan data diperiksa melalui iterasi. Jika elemen saat ini lebih besar daripada `maxVal`, maka `maxVal` diperbarui dengan elemen tersebut. Demikian pula, jika elemen saat ini lebih kecil daripada `minVal`, maka `minVal` diperbarui. Proses ini berlanjut hingga seluruh elemen dataset selesai diperiksa.

3. Penanganan Error dengan Panic:

Fungsi memvalidasi bahwa dataset tidak kosong sebelum memulai proses. Apabila dataset kosong, fungsi akan memicu mekanisme panic untuk

mencegah error runtime akibat akses elemen pada himpunan data yang tidak valid.

Kompleksitas Algoritma

1. Kompleksitas Waktu(TimeComplexity):

Algoritma memiliki kompleksitas waktu sebesar $O(n)$, di mana n adalah jumlah elemen dalam dataset. Hal ini disebabkan oleh kebutuhan untuk mengunjungi setiap elemen dataset satu kali.

2. Kompleksitas Ruang(SpaceComplexity):

Kompleksitas ruang algoritma adalah **$O(1)$** karena penggunaan memori tambahan untuk variabel bersifat konstan dan tidak tergantung pada ukuran dataset.

Kesimpulan

Pendekatan ini memungkinkan pencarian nilai maksimum dan minimum dalam himpunan data dengan efisien. Algoritma bekerja dalam kompleksitas waktu linear dan memanfaatkan sumber daya memori minimal, sehingga cocok digunakan untuk berbagai kebutuhan analisis data.

I. GUIDED

Guided1

Source Code

```
// Rizkulloh Alpriansah
// 2311102142

package main

import "fmt"

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

type arrMhs [2023]mahasiswa

func IPK_1(T arrMhs, n int) float64 {
    var terkecil float64 = T[0].ipk
    var j int = 1

    for j < n {
        if terkecil > T[j].ipk {
            terkecil = T[j].ipk
        }
        j = j + 1
    }
    return terkecil
}

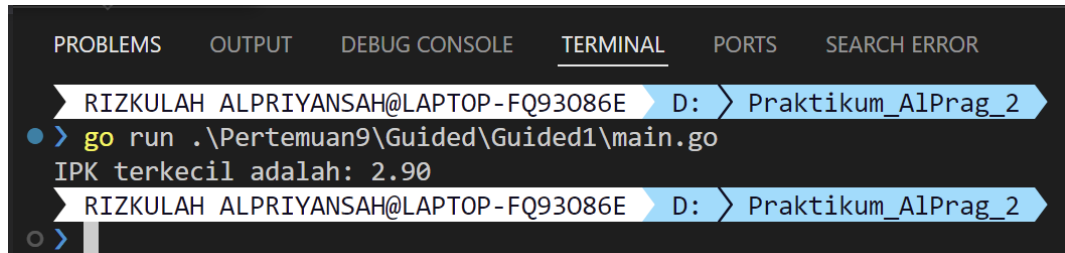
func main() {
    var mahasiswaArr arrMhs
    mahasiswaArr[0] = mahasiswa{"Alice", "2211102123", "IF1", "Teknik
Informatika", 3.2}
    mahasiswaArr[1] = mahasiswa{"Bob", "2211102124", "IF-2", "Teknik
Informatika", 3.8}
    mahasiswaArr[2] = mahasiswa{"Charlie", "22111021235", "IF-1",
    "Teknik Informatika", 2.9}
    mahasiswaArr[3] = mahasiswa{"Diana", "22111022189", "IF3", "Teknik
Informatika", 3.0}

    n := 4

    ipkTerkecil := IPK_1(mahasiswaArr, n)
```

```
    fmt.Printf("IPK terkecil adalah: %.2f\n", ipkTerkecil)
}
```

ScreenShot Output

A screenshot of a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected), 'PORTS', and 'SEARCH ERROR'. The terminal shows a command prompt for 'RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E' in the directory 'D: > Praktikum_AlPrag_2'. The user enters the command 'go run .\Pertemuan9\Guided\Guided1\main.go'. The output of the program is 'IPK terkecil adalah: 2.90'. The prompt returns to 'RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E' in the same directory.

```
RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E D: > Praktikum_AlPrag_2
> go run .\Pertemuan9\Guided\Guided1\main.go
IPK terkecil adalah: 2.90
RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E D: > Praktikum_AlPrag_2
>
```

Deskripsi

Program di atas dibuat untuk mencari nilai IPK terkecil dari sekumpulan data mahasiswa. Program ini menggunakan struct mahasiswa yang berisi atribut seperti nama, nim, kelas, jurusan, dan ipk, serta sebuah array bernama `arrMhs` dengan kapasitas 2023 untuk menyimpan data mahasiswa. Fungsi `IPK_1` digunakan untuk menerima array data mahasiswa dan jumlah data yang dimiliki, lalu mengembalikan nilai IPK terkecil melalui proses iterasi dengan membandingkan setiap elemen dalam array. Pada bagian `main`, program mendeklarasikan data mahasiswa contoh, memanggil fungsi `IPK_1`, dan mencetak hasil IPK terkecil dengan format dua angka di belakang koma. Program ini memanfaatkan konsep array, loop, dan fungsi untuk pengolahan data secara efisien.

II. UNGUIDED

Unguided1

Source Code

```
// Rizkulloh Alpriansah
// 2311102142

package main

import "fmt"

func main() {
    var beratAnak_142 [1000]float64

    var jumlahAnak_142 int
    fmt.Print("Masukkan jumlah anak kelinci: ")
    fmt.Scan(&jumlahAnak_142)

    fmt.Println("Masukkan berat masing-masing anak kelinci:")
    for i := 0; i < jumlahAnak_142; i++ {
        fmt.Scan(&beratAnak_142[i])
    }

    beratTerkecil_142 := beratAnak_142[0]
    beratTerbesar_142 := beratAnak_142[0]

    for i := 1; i < jumlahAnak_142; i++ {
        if beratAnak_142[i] < beratTerkecil_142 {
            beratTerkecil_142 = beratAnak_142[i]
        }
        if beratAnak_142[i] > beratTerbesar_142 {
            beratTerbesar_142 = beratAnak_142[i]
        }
    }

    fmt.Printf("Berat terkecil: %.2f\n", beratTerkecil_142)
    fmt.Printf("Berat terbesar: %.2f\n", beratTerbesar_142)
}
```

Screenshot

```
RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E D: > Praktikum_AlPrag_2
● > go run .\Pertemuan9\Unguided\Unguided1\main.go
Masukkan jumlah anak kelinci: 7
Masukkan berat masing-masing anak kelinci:
1
2
3
4
5
6
7
Berat terkecil: 1.00
Berat terbesar: 7.00
RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E D: > Praktikum_AlPrag_2
```

Deskripsi

Program ini digunakan untuk menentukan berat terkecil dan terbesar dari sejumlah anak kelinci berdasarkan data berat yang dimasukkan oleh pengguna. Program diawali dengan deklarasi array `beratAnak_142` untuk menyimpan berat anak kelinci hingga maksimal 1000 data, serta variabel `jumlahAnak_142` untuk menyimpan jumlah anak kelinci yang akan diinput oleh pengguna.

Pengguna diminta memasukkan jumlah anak kelinci terlebih dahulu, lalu berat masing-masing anak kelinci disimpan dalam array. Selanjutnya, berat terkecil dan terbesar diinisialisasi dengan nilai berat anak kelinci pertama.

Proses pencarian berat ekstrem dilakukan dengan iterasi melalui array, dimulai dari elemen kedua hingga elemen terakhir. Dalam setiap iterasi, berat anak kelinci dibandingkan dengan berat terkecil dan terbesar yang telah ditemukan sebelumnya. Jika ditemukan berat yang lebih kecil dari berat terkecil, maka berat terkecil diperbarui. Demikian pula jika ditemukan berat yang lebih besar dari berat terbesar, berat terbesar diperbarui.

Setelah iterasi selesai, program mencetak hasil berupa berat terkecil dan terbesar dengan format desimal dua angka di belakang koma. Program ini memastikan hasilnya dapat diinterpretasikan dengan mudah dan sesuai kebutuhan analisis data berat anak kelinci.

Unguided2

Source Code

```
package main

import "fmt"

func main() {
    var totalFish_142, fishPerContainer_142 int
    fmt.Println("Masukkan jumlah ikan (x) dan jumlah ikan per wadah (y): ")
    fmt.Scan(&totalFish_142, &fishPerContainer_142)

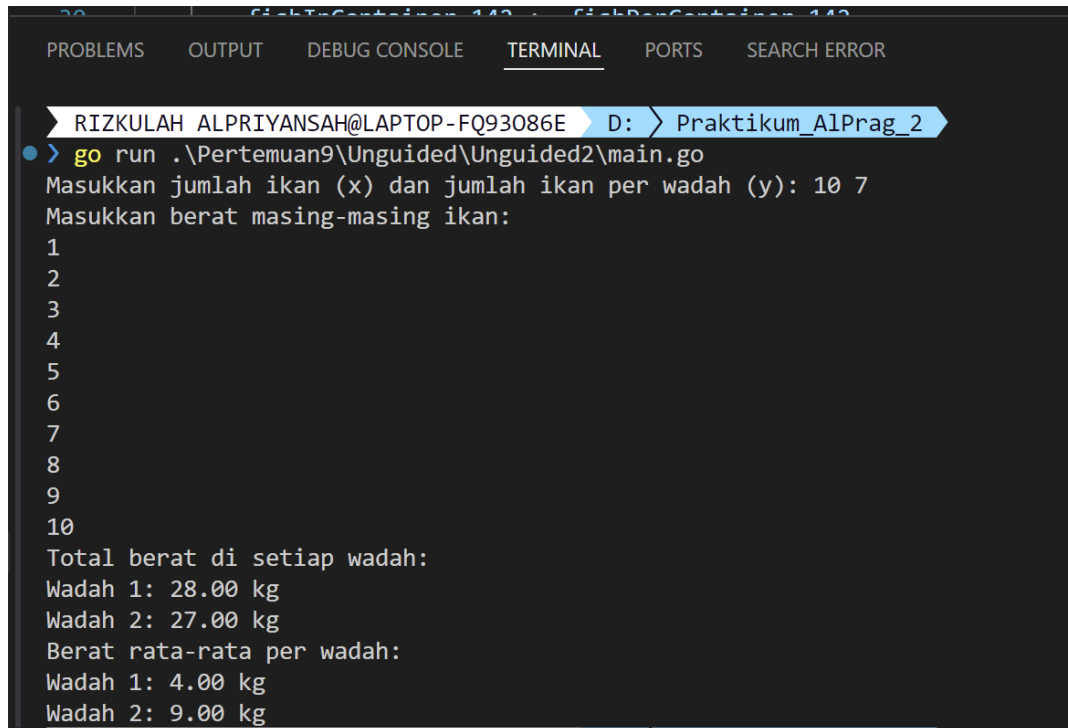
    fishWeights_142 := make([]float64, totalFish_142)
    fmt.Println("Masukkan berat masing-masing ikan:")
    for i := 0; i < totalFish_142; i++ {
        fmt.Scan(&fishWeights_142[i])
    }

    totalContainers_142 := (totalFish_142 + fishPerContainer_142 - 1) /
fishPerContainer_142
    containerWeights_142 := make([]float64, totalContainers_142)
    for i := 0; i < totalFish_142; i++ {
        containerIdx_142 := i / fishPerContainer_142
        containerWeights_142[containerIdx_142] += fishWeights_142[i]
    }

    fmt.Println("Total berat di setiap wadah:")
    for i := 0; i < totalContainers_142; i++ {
        fmt.Printf("Wadah %d: %.2f kg\n", i+1, containerWeights_142[i])
    }

    fmt.Println("Berat rata-rata per wadah:")
    for i := 0; i < totalContainers_142; i++ {
        fishInContainer_142 := fishPerContainer_142
        if i == totalContainers_142-1 && totalFish_142%fishPerContainer_142
!= 0 {
            fishInContainer_142 = totalFish_142 % fishPerContainer_142
        }
        averageWeight_142 := containerWeights_142[i] /
float64(fishInContainer_142)
        fmt.Printf("Wadah %d: %.2f kg\n", i+1, averageWeight_142)
    }
}
```

Screenshot



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR
RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E D: > Praktikum_AlPrag_2
> go run .\Pertemuan9\Unguided\Unguided2\main.go
Masukkan jumlah ikan (x) dan jumlah ikan per wadah (y): 10 7
Masukkan berat masing-masing ikan:
1
2
3
4
5
6
7
8
9
10
Total berat di setiap wadah:
Wadah 1: 28.00 kg
Wadah 2: 27.00 kg
Berat rata-rata per wadah:
Wadah 1: 4.00 kg
Wadah 2: 9.00 kg
```

Deskripsi

di atas bertujuan menghitung total berat ikan di setiap wadah dan berat rata-rata per wadah. Program dimulai dengan meminta input dari pengguna berupa jumlah total ikan, jumlah ikan per wadah, dan berat masing-masing ikan. Data berat ikan disimpan dalam slice bernama `fishWeights_142`.

Jumlah total wadah dihitung dengan rumus pembulatan ke atas yaitu dengan menambahkan jumlah ikan per wadah dikurangi satu ke jumlah total ikan, lalu dibagi jumlah ikan per wadah. Kemudian, slice `containerWeights_142` dibuat untuk menyimpan total berat ikan pada setiap wadah. Proses pengisian berat setiap wadah dilakukan dengan mengelompokkan berat ikan berdasarkan indeksinya menggunakan perhitungan indeks wadah yang diperoleh dari pembagian indeks ikan dengan jumlah ikan per wadah.

Setelah itu, program mencetak total berat ikan untuk masing-masing wadah. Program juga menghitung berat rata-rata ikan per wadah dengan membagi total berat ikan pada wadah tersebut dengan jumlah ikan di wadah itu. Jika wadah terakhir memiliki jumlah ikan kurang dari jumlah ikan per wadah, maka jumlah ikan di wadah terakhir dihitung secara khusus. Hasil berupa total berat dan rata-rata berat per wadah ditampilkan dalam format yang rapi.

Unguided3

Source Code

```
package main

import (
    "fmt"
)

func calculateMinMax_142(weights_142 []float64, minWeight_142,
maxWeight_142 *float64) {
    *minWeight_142 = weights_142[0]
    *maxWeight_142 = weights_142[0]
    for _, weight_142 := range weights_142 {
        if weight_142 < *minWeight_142 {
            *minWeight_142 = weight_142
        }
        if weight_142 > *maxWeight_142 {
            *maxWeight_142 = weight_142
        }
    }
}

func averageWeight_142(weights_142 []float64) float64 {
    totalWeight_142 := 0.0
    for _, weight_142 := range weights_142 {
        totalWeight_142 += weight_142
    }
    return totalWeight_142 / float64(len(weights_142))
}

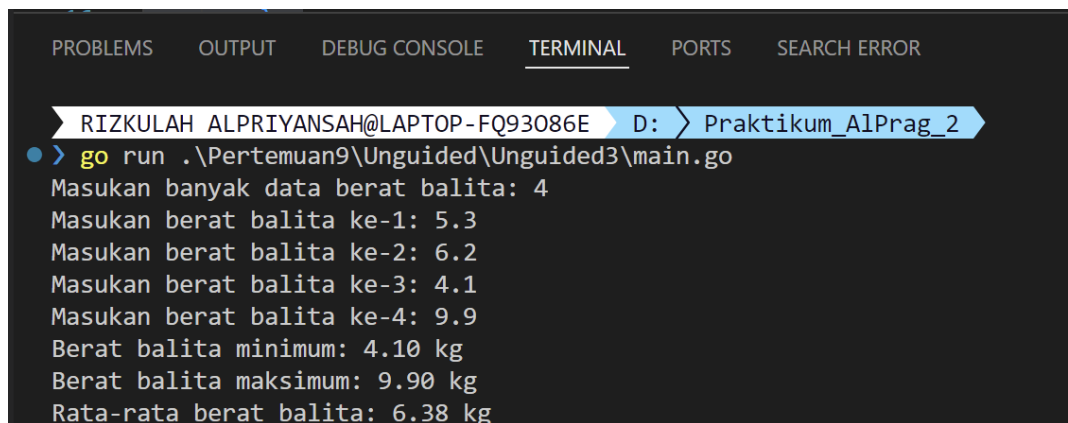
func main() {
    var totalData_142 int
    fmt.Print("Masukan banyak data berat balita: ")
    fmt.Scan(&totalData_142)

    weights_142 := make([]float64, totalData_142)
    for i_142 := 0; i_142 < totalData_142; i_142++ {
        fmt.Printf("Masukan berat balita ke-%d: ", i_142+1)
        fmt.Scan(&weights_142[i_142])
    }
}
```

```
var minWeight_142, maxWeight_142 float64
calculateMinMax_142(weights_142, &minWeight_142, &maxWeight_142)

avgWeight_142 := averageWeight_142(weights_142)
fmt.Printf("Berat balita minimum: %.2f kg\n", minWeight_142)
fmt.Printf("Berat balita maksimum: %.2f kg\n", maxWeight_142)
fmt.Printf("Rata-rata berat balita: %.2f kg\n", avgWeight_142)
}
```

Screenshot

A screenshot of a terminal window showing the execution of a Go program. The terminal title bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), PORTS, and SEARCH ERROR. The prompt shows the user is RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E, and the current directory is D:\Praktikum_AlPrag_2. The command executed is 'go run .\Pertemuan9\Unguided\Unguided3\main.go'. The program prompts the user to enter the number of baby weight data (4), then asks for each weight (5.3, 6.2, 4.1, 9.9). It then displays the calculated minimum (4.10 kg), maximum (9.90 kg), and average (6.38 kg) weights.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR
RIZKULAH ALPRIYANSAH@LAPTOP-FQ93086E D: > Praktikum_AlPrag_2
> go run .\Pertemuan9\Unguided\Unguided3\main.go
Masukan banyak data berat balita: 4
Masukan berat balita ke-1: 5.3
Masukan berat balita ke-2: 6.2
Masukan berat balita ke-3: 4.1
Masukan berat balita ke-4: 9.9
Berat balita minimum: 4.10 kg
Berat balita maksimum: 9.90 kg
Rata-rata berat balita: 6.38 kg
```

Deskripsi

Program di atas menghitung berat minimum, maksimum, dan rata-rata dari sejumlah data berat balita yang dimasukkan pengguna. Proses utamanya dibagi menjadi tiga bagian: mencari nilai ekstrem (minimum dan maksimum), menghitung rata-rata, dan menampilkan hasilnya.

Pertama, fungsi `calculateMinMax_142` menerima slice berat balita dan dua pointer (`minWeight_142` dan `maxWeight_142`) untuk menyimpan nilai minimum dan maksimum. Fungsi ini melakukan iterasi pada semua elemen dalam slice dan memperbarui nilai minimum atau maksimum jika ditemukan berat yang lebih kecil atau lebih besar dari nilai sebelumnya.

Kedua, fungsi `averageWeight_142` menghitung rata-rata berat balita dengan menjumlahkan semua elemen dalam slice dan membaginya dengan jumlah data. Total berat diakumulasi menggunakan loop, kemudian hasilnya dibagi dengan panjang slice yang dikonversi ke tipe `float64` untuk memastikan hasilnya memiliki presisi desimal.

Di bagian main, program meminta input jumlah data berat balita dan berat masing-masing balita, yang disimpan dalam slice `weights_142`. Setelah data dimasukkan, program memanggil `calculateMinMax_142` untuk menentukan berat minimum dan maksimum serta `averageWeight_142` untuk menghitung rata-rata. Hasil perhitungan ditampilkan dengan format desimal dua angka di belakang koma untuk mempermudah pembacaan.

Program ini fleksibel karena menggunakan slice, yang memungkinkan jumlah data bervariasi sesuai input pengguna, serta terstruktur dengan pemisahan fungsi untuk tugas-tugas tertentu.

III. DAFTAR PUSTAKA

- 1) Asisten praktikum, Akmelia Zahara dan Kyla Azzahra Kinan “Modul X
PENCARIAN NILAI EKSTRIM PADA HIMPUNAN DATA” Learning
Management System, 2024