**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**

1. **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. IterasidalamLoop:  
   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. PenangananErrordenganPanic**:**  
   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. KompleksitasWaktu(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. KompleksitasRuang(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.

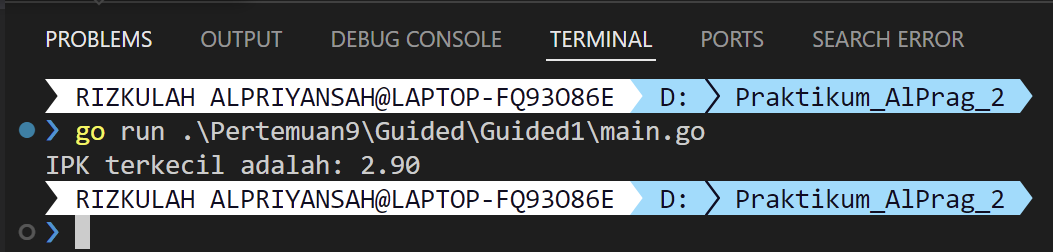
1. **GUIDED**

**Guided1**

**Source Code**

|  |
| --- |
| // Rizkulloh Alpriyansah  // 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**



**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.

1. **UNGUIDED**

**Unguided1**

**Source Code**

|  |
| --- |
| // Rizkulloh Alpriyansah  // 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**

**A screen shot of a computer

Description automatically generated**

**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.

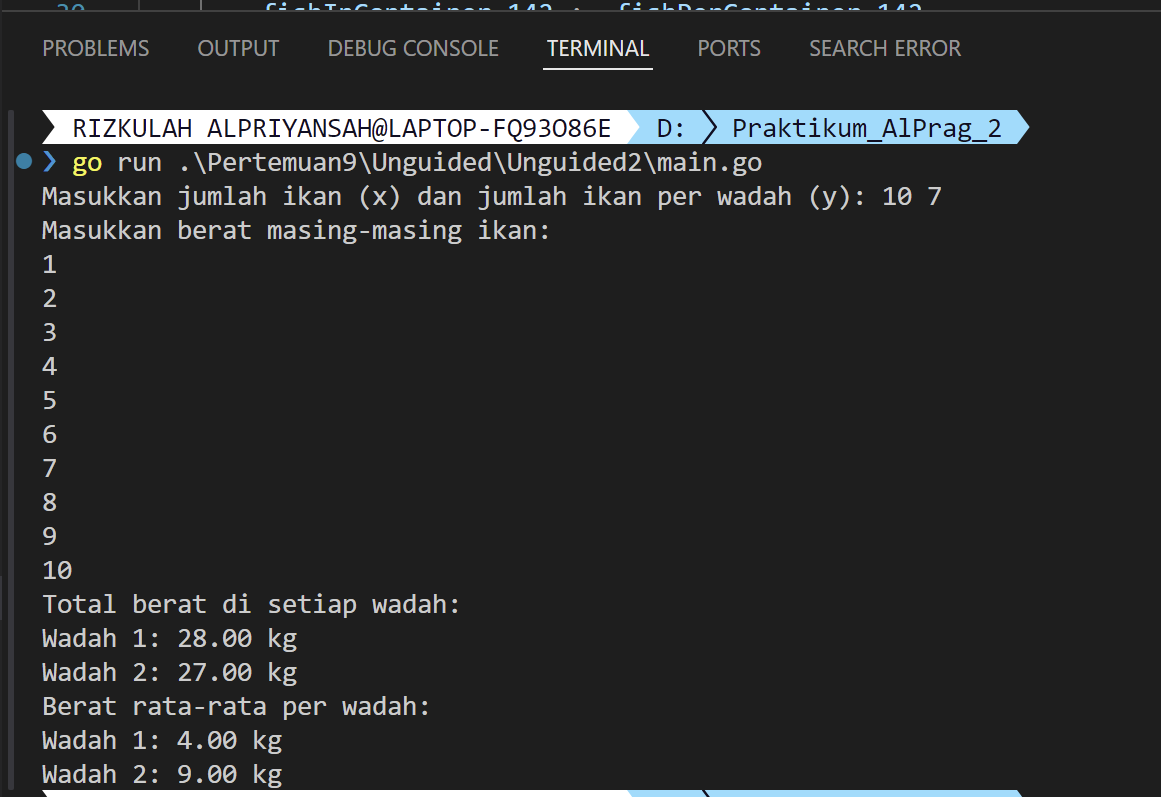
Setelah iterasi selesai, program mencetak hasil berupa berat terkecil dan terbesar dengan format desimal dua angka di belakang koma. Program ini memastikan hasilnya dapat diinterpretasikan dengan mudah dan sesuai kebutuhan analisis data berat anak kelinci.

**Unguided2**

**Source Code**

|  |
| --- |
| package main  import "fmt"  func main() {      var totalFish\_142, fishPerContainer\_142 int      fmt.Print("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**

****

**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 indeksnya menggunakan perhitungan indeks wadah yang diperoleh dari pembagian indeks ikan dengan jumlah ikan per wadah.

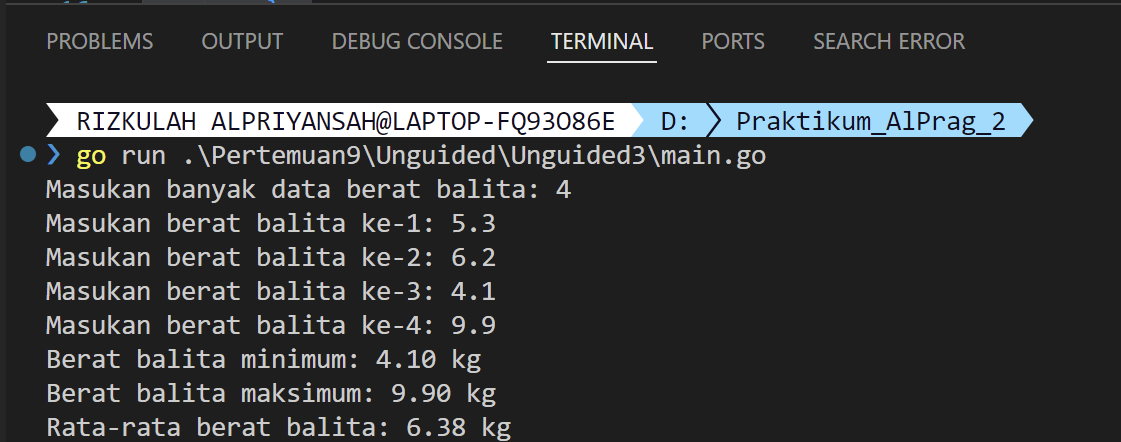
Setelah itu, program mencetak total berat ikan untuk masing-masing wadah. Program juga menghitung berat rata-rata ikan per wadah dengan membagi total berat ikan pada wadah tersebut dengan jumlah ikan di wadah itu. Jika wadah terakhir memiliki jumlah ikan kurang dari jumlah ikan per wadah, maka jumlah 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**



**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 diakumulasikan 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.

1. **DAFTAR PUSTAKA**

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