LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2

MODUL 12

PENCARIAN NILAI ACAK PADA HIMPUNAN DATA

Oleh:

NAMA: ICHYA ULUMIDDIIN

NIM: 103112400076

KELAS: 12IF-01-A

S1 TEKNIK INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

I. DASAR TEORI

1. Sequential Search (atau Linear Search) adalah metode pencarian data dalam sebuah struktur data (biasanya array atau list) dengan memeriksa elemen satu per satu dari awal hingga akhir sampai elemen yang dicari ditemukan atau semua elemen telah diperiksa.

Langkah-langkah:

a) Mulai dari indeks pertama (biasanya indeks 0).
b) Bandingkan setiap elemen array dengan elemen yang dicari.
c) Jika ditemukan, kembalikan posisi (indeks) elemen tersebut.
d) Jika tidak ditemukan sampai akhir, berarti elemen tersebut tidak ada.

2. Binary Search adalah metode pencarian data dalam array yang sudah terurut. Metode ini membagi dua rentang pencarian secara berulang hingga data yang dicari ditemukan atau rentang pencarian menjadi kosong.

Langkah-langkah:

a) Tentukan indeks batas bawah (low) dan batas atas (high).
b) Hitung indeks tengah (mid = (low + high) / 2).
c) Bandingkan elemen tengah dengan nilai yang dicari (k):
d) Jika sama, kembalikan mid.
e) Jika lebih kecil, cari ke sebelah kanan (low = mid + 1).
f) Jika lebih besar, cari ke sebelah kiri (high = mid - 1).
g) Ulangi proses hingga ditemukan atau low > high.

II. GUIDED

Source Code Guided 1

```
package main
import (
            "fmt"
            "sort"
)
func sequentialSeacrh(arr []float64, target float64)
(int, int) {
      iterations := 0
      for i, val := range arr {
            iterations ++
            fmt.Printf("Sequential 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("Sequential Search (data tidak perlu
urut):")
	idxSeq, iterSeq := sequentialSeacrh(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\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\n", idxBin, iterBin )
	} else {
		fmt.Printf("Hasil: Tidak Ditemukan setelah
indeks %d langkah\n", iterBin )
	}
}
```

Output

```
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

Program ini menunjukkan perbandingan antara sequential search dan binary search dalam mencari sebuah nilai dalam array. Sequential search mencari data dengan memeriksa setiap elemen satu per satu dari awal hingga akhir, tanpa memerlukan data dalam keadaan terurut. Metode ini sederhana namun menjadi tidak efisien jika data berjumlah besar atau nilai yang dicari berada di akhir. Sebaliknya, binary search hanya dapat digunakan pada data yang sudah terurut, dan bekerja dengan membagi dua rentang pencarian setiap langkahnya, sehingga jauh lebih cepat dan efisien. Program mencetak proses pencarian langkah demi langkah untuk memperlihatkan bahwa binary search membutuhkan lebih sedikit iterasi dibandingkan sequential search, terutama saat menangani data besar yang terurut.

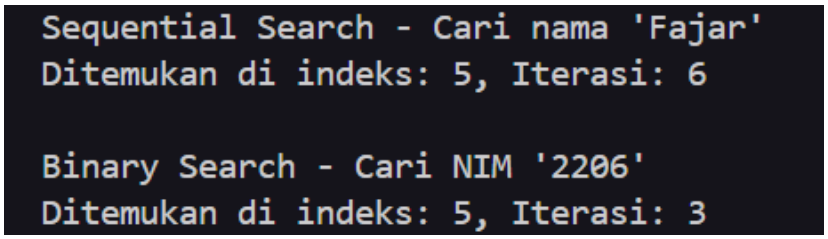
Source Code Guided 2

```
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
	temp := [10]mahasiswa{
		{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},
	}
	for i := 0; i < n; i++ {
		data[i] = temp[i]
	}
	// 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 menyimpan data mahasiswa dan melakukan pencarian menggunakan dua metode: sequential search dan binary search. Data mahasiswa dimuat dalam array bertipe struct berisi nama, NIM, kelas, jurusan, dan IPK. Pencarian nama dilakukan dengan sequential search, yaitu memeriksa elemen satu per satu hingga ditemukan. Sementara itu, pencarian berdasarkan NIM menggunakan binary search, yang lebih efisien karena membagi ruang pencarian secara bertahap, namun memerlukan data yang telah terurut. Program mencetak hasil pencarian termasuk indeks dan jumlah iterasi. Tujuan utamanya adalah membandingkan efisiensi kedua metode, di mana sequential search cocok untuk data kecil dan tidak terurut, sedangkan binary search lebih cepat untuk data besar yang telah diurutkan.

III. UNGUIDED

Source Code Unguided 1

```
package main

import (
	"fmt"
	"sort"
)
func binarySearch(arr []int, target int) bool {
	low := 0
	high := len(arr) - 1
	for low <= high {
		mid := (low + high) / 2
		if arr[mid] == target {
			return true
		} else if arr[mid] < target {
			low = mid + 1
		} else {
			high = mid - 1
		}
	}
	return false
}
func main() {
	var input int
	validCandidates := make([]int, 20)
	for i := 0; i < 20; i++ {
		validCandidates[i] = i + 1
	}
	sort.Ints(validCandidates)

	voteCounts := make([]int, 21)
	totalVotes := 0
	validVotes := 0
```

```
fmt.Println("Masukkan suara satu per satu (akhiri
dengan 0):")
	for {
		_, err := fmt.Scan(&input)
		if err != nil {
			break
		}
		if input == 0 {
			break
		}
		totalVotes++
		if binarySearch(validCandidates, input) {
			voteCounts[input]++
			validVotes++
		}
	}

	fmt.Printf("Suara masuk: %d\n", totalVotes)
	fmt.Printf("Suara sah: %d\n", validVotes)

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

Output

```
Masukkan suara satu per satu (akhiri dengan 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
```

Penjelasan

Program ini digunakan untuk menghitung hasil suara dalam pemilihan ketua RT, dengan validasi suara menggunakan binary search. Pemilih memasukkan nomor calon (1–20) satu per satu melalui input, dan proses dihentikan saat angka 0 dimasukkan. Program menyimpan daftar kandidat yang valid dalam slice `validCandidates`, yang telah diurutkan agar bisa divalidasi secara efisien menggunakan binary search. Jika input merupakan suara yang sah, maka suara tersebut dicatat dalam array `voteCounts`, dan jumlah suara sah serta total suara masuk diperbarui. Setelah seluruh suara selesai dimasukkan, program mencetak jumlah total suara yang masuk, jumlah suara sah, dan rekapitulasi jumlah suara untuk masing-masing calon yang mendapat suara. Hanya calon dengan minimal satu suara yang ditampilkan. P

Source Code Unguided 2

```
package main

import (
	"fmt"
	"sort"
)

func binarySearch(arr []int, target int) bool {
	low := 0
	high := len(arr) - 1
	for low <= high {
		mid := (low + high) / 2
		if arr[mid] == target {
			return true
		} else if arr[mid] < target {
			low = mid + 1
		} else {
			high = mid - 1
		}
	}
	return false
}
type Calon struct {
	Nomor int
	Suara int
}
func main() {
	var input int
	validCandidates := make([]int, 20)
	for i := 0; i < 20; i++ {
		validCandidates[i] = i + 1
	}
	sort.Ints(validCandidates)

	voteCounts := make([]int, 21)
	totalVotes := 0
	validVotes := 0
```

```
fmt.Println("Masukkan suara satu per satu (akhiri dengan
0):")
	for {
		_, err := fmt.Scan(&input)
		if err != nil {
			break
		}
		if input == 0 {
			break
		}
		totalVotes++
		if binarySearch(validCandidates, input) {
			voteCounts[input]++
			validVotes++
		}
	}
	var hasil []Calon
	for i := 1; i <= 20; i++ {
		if voteCounts[i] > 0 {
			hasil = append(hasil, Calon{Nomor: i,
Suara: voteCounts[i]})
		}
	}
	sort.Slice(hasil, func(i, j int) bool {
		if hasil[i].Suara == hasil[j].Suara {
			return hasil[i].Nomor < hasil[j].Nomor
		}
		return hasil[i].Suara > hasil[j].Suara
	})

	fmt.Printf("Suara masuk: %d\n", totalVotes)
	fmt.Printf("Suara sah: %d\n", validVotes)

	if len(hasil) == 0 {
		fmt.Println("Tidak ada suara sah, tidak ada
ketua dan wakil.")
	} else if len(hasil) == 1 {
		fmt.Printf("Ketua RT: Calon nomor %d\n",
hasil[0].Nomor)
		fmt.Println("Wakil RT: Tidak tersedia (hanya
satu calon sah).")
	} else {
		fmt.Printf("Ketua RT: Calon nomor %d\n",
hasil[0].Nomor)
		fmt.Printf("Wakil RT: Calon nomor %d\n",
hasil[1].Nomor)
	}
}
```

Output

```
Masukkan suara satu per satu (akhiri dengan 0):
7 19 3 2 78 3 1 -3 18 19 0
Suara masuk: 10
Suara sah: 8
Ketua RT: Calon nomor 3
Wakil RT: Calon nomor 19
```

Penjelasan

Program ini digunakan untuk menghitung hasil suara pemilihan ketua dan wakil ketua RT. Pemilih memasukkan suara berupa nomor calon (1–20), dan input dihentikan dengan angka 0. Untuk memastikan keabsahan suara, program menggunakan binary search pada daftar kandidat yang valid. Suara sah dicatat dalam array `voteCounts`, dan setelah input selesai, data disusun dalam struct `Calon` yang memuat nomor calon dan jumlah suara. Data kemudian diurutkan berdasarkan suara terbanyak, dan jika jumlah suara sama, calon dengan nomor lebih kecil diprioritaskan. Program menampilkan total suara masuk, suara sah, serta calon yang terpilih sebagai ketua dan wakil ketua.

Source Code Unguided 3

```
package main

import "fmt"

const NMAX = 1000000
var data [NMAX]int

func main() {
	var n, k int
	fmt.Scan(&n, &k)
	isiArray(n)
	idx := posisi(n, k)

	if idx == -1 {
		fmt.Println("TIDAK ADA")
	} else {
		fmt.Println(idx)
	}
}

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

func posisi(n, k int) int {
	low := 0
	high := n - 1

	for low <= high {
		mid := (low + high) / 2
		if data[mid] == k {
			return mid
		} else if data[mid] < k {
			low = mid + 1
		} else {
			high = mid - 1
		}
	}
	return -1
}
```

Output

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

Penjelasan

Program ini digunakan untuk mencari posisi sebuah bilangan `k` dalam array `data` yang telah terurut membesar menggunakan metode binary search. Array `data` memiliki kapasitas maksimum 1.000.000 elemen dan diisi dengan `n` buah bilangan positif melalui prosedur `isiArray(n)`. Setelah data diisi, program memanggil fungsi `posisi(n, k)` untuk melakukan pencarian bilangan `k`.

Fungsi `posisi` menggunakan binary search, yaitu membandingkan elemen di posisi tengah array dengan nilai `k`. Jika sama, maka indeks tengah dikembalikan. Jika nilai tengah lebih kecil, pencarian dilanjutkan ke bagian kanan; jika lebih besar, pencarian dilanjutkan ke kiri. Jika pencarian selesai tanpa menemukan nilai `k`, fungsi akan mengembalikan `-1`.

Di fungsi `main`, jika hasil pencarian adalah `-1`, program mencetak `"TIDAK ADA"`, dan jika ditemukan, mencetak indeks (dengan posisi dimulai dari nol).

IV. KESIMPULAN

Secara keseluruhan, sequential search dan binary search merupakan dua algoritma pencarian yang sering digunakan dalam pemrograman, masing-masing dengan keunggulan dan keterbatasannya. Sequential search cocok digunakan untuk data yang tidak terurut karena prosesnya yang sederhana dan tidak memerlukan prasyarat khusus. Namun, dari segi efisiensi, sequential search kurang optimal karena harus memeriksa setiap elemen satu per satu, yang menyebabkan waktu pencarian menjadi lama terutama pada data berukuran besar.

Di sisi lain, binary search menawarkan efisiensi tinggi dengan waktu pencarian yang jauh lebih cepat karena membagi ruang pencarian menjadi dua secara bertahap. Namun, metode ini hanya bisa diterapkan jika data sudah dalam keadaan terurut. Oleh karena itu, dalam penerapannya, pemilihan metode pencarian harus disesuaikan dengan kondisi data dan kebutuhan program: gunakan sequential search untuk data kecil atau tidak terurut, dan gunakan binary search saat menangani data besar yang sudah terurut untuk hasil yang lebih optimal.

REFERENSI

Afrizalmy. (2020, September 20). *Belajar Golang - Implementasi Algoritma Sequential Search*. Retrieved from afrizalmy.com: https://afrizalmy.com/belajar-golang-implementasi-algoritma-sequential-search

PC, P. (2023, September 20). *Exploring Sorting and Searching Algorithms With Golang — Binary Search*. Retrieved from Medium: https://pcpratheesh.medium.com/exploring-sorting-and-searching-algorithms-with-golang-binary-search-6679716bcd3a

Tenteromano, P. (2022, Juny 25). *Concurrency in Golang with a Binary Search Problem*. Retrieved from dev.to: https://dev.to/vetswhocode/concurrency-in-golang-with-a-binary-search-problem-2j5b