**LAPORAN PRAKTIKUM**

**STRUKTUR DATA**

****

**DISUSUN OLEH :**

**MUHAMMAD NURIL ANWARI**

**362458302106**

**TEKNOLOGI REKAYASA PERANGKAT LUNAK**

**POLITEKNIK NEGERI BANYUWANGI**

**2025**

**TUJUAN PRAKTIKUM**

1. Memahami konsep dasar algoritma pencarian (searching) dalam struktur data.
2. Mampu mengimplementasikan algoritma pencarian:

o Sequential Search (Pencarian Berurutan).

o Binary Search (Pencarian Biner).

1. Menganalisis perbedaan kelebihan dan kekurangan kedua algoritma berdasarkan kompleksitas waktu.
2. Mengaplikasikan algoritma pencarian dalam program menggunakan bahasa C/C++.

**TEORI SINGKAT**

1. **Konsep Pencarian (Searching)**

• Proses menemukan data atau elemen tertentu dalam kumpulan data (array, list, dll).

• Dua metode utama: Sequential Search dan Binary Search.

1. **Sequential Search**

• Prinsip: Memeriksa setiap elemen satu per satu dari awal hingga data ditemukan atau sampai akhir koleksi.

• Kompleksitas Waktu:

Kasus Terbaik: O(1) (data ditemukan di elemen pertama).

Kasus Terburuk: O(n) (data tidak ada atau di elemen terakhir).

• Kelebihan: Sederhana dan bisa digunakan pada data tidak terurut.

• Kekurangan: Lambat untuk data besar karena pemeriksaan linear.

1. **Binary Search**

• Prinsip: Membagi data menjadi dua bagian secara rekursif berdasarkan perbandingan nilai dengan elemen tengah.

• Syarat: Data harus terurut (ascending/descending).

• Kompleksitas Waktu: O(log n) (lebih efisien untuk data besar).

• Kelebihan: Cepat karena mengurangi wilayah pencarian secara eksponensial.

• Kekurangan: Hanya bekerja pada data terurut.

**TUGAS PENDAHULUAN**

1. **Buatlah flowchart untuk operasi pencarian bilangan bulat dengan metode sequential search dan binary search**

**Flowchart Sequential Search**

Langkah-Langkah:

a) Mulai.

b) Input array data dan bilangan target yang dicari.

c) Inisialisasi indeks pencarian (i = 0).

d) Bandingkan target dengan elemen array di indeks i:

* + Jika sama, tampilkan posisi i dan lanjut ke langkah 6 (data ditemukan).
  + Jika tidak sama, increment indeks (i++).

e) Ulangi langkah 4 hingga elemen terakhir atau data ditemukan.

f) Jika tidak ditemukan setelah seluruh array diperiksa, tampilkan "Tidak ada".

g) Selesai.

**Flowchart Binary Search**

Langkah-Langkah:

a) Mulai.

b) Input array terurut dan bilangan target.

c) Inisialisasi: a. low = 0 (indeks awal). b. high = panjang array - 1 (indeks akhir).

d) Hitung mid = (low + high) / 2 (indeks tengah).

e) Bandingkan target dengan array[mid]:

* + Jika sama, tampilkan mid dan selesai.
  + Jika target < array[mid], geser high = mid - 1.
  + Jika \*\*target > array[mid], geser low = mid + 1`.

f) Ulangi langkah 4–5 selama low <= high.

g) Jika tidak ditemukan, tampilkan "Tidak ada".

h) Selesai.

1. **Buatlah flowchart untuk operasi penyisipan sebelum dan sesudah serta operasi penghapusan data kunci**

***Operasi Penyisipan***

**Menyisipkan *sebelum* data kunci**

Langkah langkah :

1. Mulai
2. Input:
3. Array/Linked List
4. Data baru (new\_data)
5. Data kunci (key)
6. Cari posisi key dalam struktur data
7. Jika key tidak ditemukan → tampilkan "Key tidak ada" → Selesai
8. Jika ditemukan:
9. Geser semua elemen setelah posisi key ke kanan (untuk array)
10. Atau sesuaikan pointer (untuk linked list)
11. Sisipkan new\_data sebelum key
12. Selesai

**Menyisipkan *sesudah* data kunci**

Langkah langkah :

* + - 1. Mulai
      2. Input: - Array/Linked List - Data baru (new\_data) - Data kunci (key)
      3. Inisialisasi: - posisi = -1 - i = 0
      4. Cek kondisi: - Jika i < panjang struktur data → lanjut
      5. - Jika tidak → g) e) Bandingkan: - Jika elemen ke-i == key:
      6. posisi = i
      7. lanjut f) - Jika tidak:
      8. i++
      9. kembali ke d)
      10. Jika posisi != -1 → h)
      11. Jika tidak → g)
      12. Tampilkan "Key tidak ada" → l)
      13. Jika array: Geser elemen dari posisi ke kanan - array[posisi] = new\_data
      14. Jika linked list: - Buat node baru - Atur pointer:
      15. node baru.next = node key
      16. node prev.next = node baru
      17. Tampilkan "Data berhasil disisipkan" r) Update struktur data
      18. Selesai

**Operasi Penghapusan Data Kunci**

Langkah langkah :

* + - 1. Mulai
      2. Input: ♣ Array/Linked List ♣ Data kunci (key) yang akan dihapus
      3. Cari posisi key
      4. Jika tidak ditemukan → tampilkan "Key tidak ada" → Selesai
      5. Jika ditemukan:

♣ Untuk array: geser elemen setelah key ke kiri

♣ Untuk linked list: lewati node yang berisi key

* + - 1. Hapus data key
      2. Selesai

1. **Buatlah deklarasi data pegawai dan flowchart fungsi untuk pencarian data dengan metode sequential search dan binary search**

**Deklarasi Data Pegawai**

Struct Pegawai {

NIP : Integer

Nama : String

Alamat : String

Golongan : Char

}

**Sequential Search (Pencarian Berurutan)**

Langkah-Langkah**:**

1. Mulai
2. Input:
3. Array data pegawai
4. Kunci pencarian (NIP atau Nama)
5. Bandingkan kunci dengan setiap data secara berurutan dari indeks ke-0:
6. Jika NIP/Nama cocok: Tampilkan data pegawai.
7. Jika tidak cocok: Lanjut ke data berikutnya.
8. Jika semua data sudah diperiksa dan tidak ditemukan: Tampilkan "Data tidak
9. ditemukan".
10. Selesai.

**Binary Search (Pencarian Biner)**

Langkah-Langkah:

Mulai

1. Syarat: Data harus terurut (contoh: NIP ascending).
2. Input:
   1. Array data pegawai (terurut)
   2. Kunci pencarian (NIP)
3. Inisialisasi:
   1. low = 0 (indeks awal)
   2. high = jumlah\_data - 1 (indeks akhir)
4. Selama low <= high:
   1. Hitung mid = (low + high) / 2
   2. Jika data[mid].NIP == kunci: Tampilkan data, selesai.
   3. Jika kunci > data[mid].NIP: low = mid + 1
   4. Jika kunci < data[mid].NIP: high = mid - 1
5. Jika low > high: Tampilkan "Data tidak ditemukan".
6. Selesai.

**PERCOBAAN**

1. **Implementasi pencarian dengan metode Linear search dengan Teknik iterative**

**Code :**

int sequentialSearch(List<int> arr, int x) {

int N = arr.length;

for (int i = 0; i < N; i++) {

if (arr[i] == x) return i;

}

return -1;

}

void main() {

List<int> arr = [2, 3, 4, 10, 40];

int x = 10;

int result = sequentialSearch(arr, x);

if (result == -1) {

print("Elemen tidak ada dalam array");

} else {

print("Elemen ditemukan pada indeks ke: $result");

}

}

**Output :**

**C:\Users\user\Pictures\Screenshots\Screenshot (264).png**

1. **Implementasi pencarian dengan metode Linear search dengan Teknik rekursif**

**Code :**

int linearSearch(List<int> arr, int size, int key) {

if (size == 0) {

return -1;

} else if (arr[size - 1] == key) {

// Return the index of found key.

return size - 1;

} else {

return linearSearch(arr, size - 1, key);

}

}

void main() {

List<int> arr = [2, 3, 4, 10, 40];

int x = 10;

int result = linearSearch(arr, arr.length, x);

if (result == -1) {

print("Elemen tidak ada dalam array");

} else {

print("Elemen ditemukan pada indeks ke: $result");

}

}

**Output :**

**C:\Users\user\Pictures\Screenshots\Screenshot (265).png**

1. **Implementasi pencarian dengan metode binary search dengan Iterative**

**Code:**

void binarySearch(List<int> arr, int l, int r, int x) {

while (r - l > 1) {

int mid = (l + r) ~/ 2;

if (arr[mid] < x) {

l = mid + 1;

} else {

r = mid;

}

}

if (arr[l] == x) {

print("Found At Index $l");

} else if (arr[r] == x) {

print("Found At Index $r");

} else {

print("Not Found");

}

}

void main() {

List<int> arr = [2, 3, 4, 10, 40];

int x = 10;

int size = arr.length;

binarySearch(arr, 0, size - 1, x);

}**Output :**

**C:\Users\user\Pictures\Screenshots\Screenshot (266).png**

1. **Implementasi pencarian dengan metode binary search dengan Rekursif**

**Code:**

int binarySearch(List<int> arr, int l, int r, int x) {

if (r >= l) {

int mid = l + (r - l) ~/ 2;

if (arr[mid] == x) return mid;

if (arr[mid] > x) return binarySearch(arr, l, mid - 1, x);

return binarySearch(arr, mid + 1, r, x);

}

return -1;

}

void main() {

List<int> arr = [2, 3, 4, 10, 40];

int x = 10;

int size = arr.length;

int result = binarySearch(arr, 0, size - 1, x);

if (result == -1) {

print("Elemen tidak ada dalam array");

} else {

print("Elemen ditemukan pada indeks ke: $result");

}

}

**Output :**

**C:\Users\user\Pictures\Screenshots\Screenshot (267).png**

**LATIHAN**

1. **Buatlah program untuk menjalankan Algoritma Binary Search, namun diterapkan pada kumpulan data yang acak / tidak urut! Output program adalah indeks berdasarkan pada indeks pada data asli sebelum diurutkan!**

int binarySearch(List<MapEntry<int, int>> arr, int target) {

  int left = 0;

  int right = arr.length - 1;

  while (left <= right) {

    int mid = (left + right) ~/ 2;

    if (arr[mid].key == target) {

      return arr[mid].value;  // Mengembalikan indeks asli

    } else if (arr[mid].key < target) {

      left = mid + 1;

    } else {

      right = mid - 1;

    }

  }

  return -1;  // Jika tidak ditemukan

}

void main() {

  // Data acak (misalnya)

  List<int> data = [15, 2, 9, 7, 4, 1, 10];

  // Menyimpan data asli beserta indeksnya

  List<MapEntry<int, int>> indexedData = [];

  for (int i = 0; i < data.length; i++) {

    indexedData.add(MapEntry(data[i], i));  // Menyimpan pasangan (nilai, indeks)

  }

  // Mengurutkan data berdasarkan nilai elemen (bukan indeks asli)

  indexedData.sort((a, b) => a.key.compareTo(b.key));

  // Menentukan target yang ingin dicari

  int target = 9;

  // Melakukan binary search pada data yang sudah diurutkan

  int result = binarySearch(indexedData, target);

  // Output: Menampilkan indeks asli dari target yang dicari

  if (result != -1) {

    print('Indeks asli dari nilai $target adalah $result');

  } else {

    print('Nilai $target tidak ditemukan.');

  }

}

**ANALISA**

 **Data Acak**: Variabel data adalah array yang berisi data yang tidak terurut.

 **Menyimpan Indeks Asli**: List indexedData menyimpan pasangan (nilai, indeks) dari data asli. MapEntry<int, int> digunakan untuk menyimpan pasangan nilai dan indeks.

 **Pengurutan**: Data diurutkan berdasarkan nilai elemen menggunakan indexedData.sort((a, b) => a.key.compareTo(b.key)). key di sini adalah nilai elemen yang akan diurutkan.

 **Binary Search**: Fungsi binarySearch digunakan untuk mencari elemen dalam array yang sudah diurutkan dan mengembalikan indeks asli dari elemen tersebut.

 **Output**: Program akan menampilkan indeks asli dari elemen yang dicari atau memberi tahu bahwa elemen tidak ditemukan.

**HASIL**

**C:\Users\user\Pictures\Screenshots\Screenshot (269).png**

1. **Implementasikan prosedur untuk menyisipkan data sebelum dan sesudah data kunci serta prosedur untuk menghapus data kunci tugas pendahuluan.**

void sisipkanSebelum(List<int> data, int kunci, int dataBaru) {

  int index = data.indexOf(kunci);  // Mencari indeks kunci dalam list

  if (index != -1) {

    data.insert(index, dataBaru);  // Menyisipkan data baru sebelum kunci

    print('Data setelah disisipkan sebelum $kunci: $data');

  } else {

    print('Data kunci $kunci tidak ditemukan.');

  }

}

void sisipkanSesudah(List<int> data, int kunci, int dataBaru) {

  int index = data.indexOf(kunci);  // Mencari indeks kunci dalam list

  if (index != -1) {

    data.insert(index + 1, dataBaru);  // Menyisipkan data baru setelah kunci

    print('Data setelah disisipkan sesudah $kunci: $data');

  } else {

    print('Data kunci $kunci tidak ditemukan.');

  }

}

void hapusDataKunci(List<int> data, int kunci) {

  bool isRemoved = data.remove(kunci);  // Menghapus data kunci pertama yang ditemukan

  if (isRemoved) {

    print('Data setelah $kunci dihapus: $data');

  } else {

    print('Data kunci $kunci tidak ditemukan.');

  }

}

void main() {

  // Data contoh

  List<int> data = [10, 20, 30, 40, 50];

  // Menampilkan data awal

  print('Data awal: $data');

  // Sisipkan data baru sebelum kunci

  sisipkanSebelum(data, 30, 25);

  // Sisipkan data baru sesudah kunci

  sisipkanSesudah(data, 30, 35);

  // Menghapus data kunci

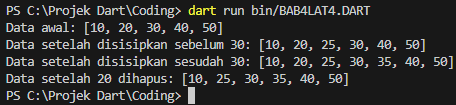
  hapusDataKunci(data, 20);

}

**ANALISA**

1. **Prosedur sisipkanSebelum**:
   * Prosedur ini mencari data yang cocok dengan kunci yang diberikan menggunakan metode indexOf.
   * Jika ditemukan, data baru disisipkan **sebelum** data kunci menggunakan metode insert dengan indeks yang tepat.
2. **Prosedur sisipkanSesudah**:
   * Prosedur ini mencari data kunci, dan jika ditemukan, data baru disisipkan **setelah** data kunci menggunakan metode insert dengan indeks yang tepat (indeks kunci + 1).
3. **Prosedur hapusDataKunci**:
   * Prosedur ini menggunakan metode remove untuk menghapus data yang pertama kali ditemukan yang cocok dengan kunci yang diberikan.

**Output:**

****

**KESIMPULAN**

1. Algoritma pencarian Sequential Search lebih cocok digunakan pada data yang tidak terurut dan berukuran kecil karena metode ini melakukan pencarian secara linear.

2. Algoritma pencarian Binary Search lebih efisien untuk data yang telah terurut karena memiliki kompleksitas waktu O(log n), yang jauh lebih cepat dibandingkan Sequential Search pada data besar.

3. Implementasi algoritma pencarian dalam program sangat bergantung pada struktur data yang digunakan, apakah berbentuk array atau linked list.

4. Dalam konteks pencarian data pegawai, metode Sequential Search lebih fleksibel karena dapat digunakan untuk mencari berdasarkan NIP maupun Nama, sedangkan Binary Search hanya dapat digunakan jika data terurut dan pencarian berdasarkan NIP.

5. Pemilihan metode pencarian harus disesuaikan dengan kebutuhan dan kondisi data agar diperoleh efisiensi terbaik dalam pemrosesan data.

**REFERENSI**

**Chat gpt :** <https://chatgpt.com/>

**LINK Github :**