Algoritma Pencarian

Devit Suwardiyanto, S.Si., M.T.

Algoritma Pencarian Linier (*Linear search/ sequential search*)

- algoritma sederhana untuk mencari nilai target dalam array dan daftar.
- Algoritma memeriksa setiap elemen array hingga menemukan elemen yang cocok dengan nilai target.
- Jika algoritma mencapai akhir array, itu berakhir dengan tidak berhasil.
- Kompleksitas O(n)

Algoritma Pencarian Linier dalam Java

```
public static int search(int[] array, int value) {
    int index = -1;
    for (int i = 0; i < array.length; i++) {
        // periksa jika nilai sama dengan inputan
        if (array[i] == value) {
            index = i;
            break;
        }
    }
    return index;
}
```

```
int[] numbers = {1, 4, 7, 2, 3, 5};

search(numbers, 1); // 0
search(numbers, 4); // 1
search(numbers, 5); // 5
search(numbers, 6); // -1, no value found
```

Algoritma Pencarian Linier dalam Java (sorted arrays)

```
public static int searchInSortedArray(int[] array, int value) {
    int index = -1;
    for (int i = 0; i < array.length; i++) {
        if (array[i] == value) {
            index = i;
            break;
        } else if (array[i] > value) {
           break;
        }
    }
    return index;
}
```

```
searchInSortedArray(new int[] {8, 15, 19, 20, 21}, 10);  // -1, elemen 10 tidak ditemukan


// pencarian hanya sampai elemen 15
```

Algoritma Pencarian Biner (*Binary search*)

- Dimulai dengan membandingkan elemen tengah array dengan nilai target.
- Jika nilai target cocok dengan elemen tengah, posisinya dalam array dikembalikan.
- Jika nilai target kurang dari atau lebih besar dari elemen tengah, pencarian berlanjut di subarray kiri atau kanan, masing-masing, menghilangkan subarray lain dari pertimbangan.
- Itu berulang sampai nilainya ditemukan atau interval pencarian baru kosong.
- Kompleksitas O(log n)

Algoritma Pencarian Biner dalam Java (sorted arrays)

```java
public static int binarySearch(int[] array, int elem, int left, int right) {
    while (left <= right) {
        int mid = left + (right - left) / 2; // indeks tengah
        if (elem == array[mid]) {
            return mid; // elemen ditemukan, kembali ke indeksnya
        } else if (elem < array[mid]) {
            right = mid - 1; // geser ke kiri subarray
        } else {
            left = mid + 1;  // geser ke kanan subarray
        }
    }
    return -1; // elemen tidak ditemukan
}
```

Algoritma Pencarian Biner dalam Java (sorted arrays)
penggunaan method

```
int[] array = { 10, 13, 19, 20, 24, 26, 30, 34, 35 };

int from = 0, to = array.length - 1;

int indexOf10 = binarySearch(array, 10, from, to); // 0
int indexOf19 = binarySearch(array, 19, from, to); // 2
int indexOf26 = binarySearch(array, 26, from, to); // 5
int indexOf34 = binarySearch(array, 34, from, to); // 7
int indexOf35 = binarySearch(array, 35, from, to); // 8

int indexOf5 = binarySearch(array, 5, from, to);   // -1
int indexOf16 = binarySearch(array, 16, from, to); // -1
```

Algoritma Pencarian Biner dalam Java (sorted arrays)
Implementasi Rekursif

```java
public static int binarySearch(int[] array, int elem, int left, int right) {
    if (left > right) {
        return -1; // elemen tidak ditemukan
    }
    int mid = left + (right - left) / 2; // index tengah

    if (elem == array[mid]) {
        return mid; //  elemen ditemukan, kembali ke indeks
    } else if (elem < array[mid]) {
        return binarySearch(array, elem, left, mid - 1); // geser ke kiri
    } else {
        return binarySearch(array, elem, mid + 1, right); // geser ke kanan
    }
}
```

Algoritma Pencarian Lompat (*Jump search/ Block search*)

- Algoritma untuk menemukan posisi elemen dalam array yang diurutkan.
- Berbeda dengan pencarian linier, tidak membandingkan setiap elemen array dengan nilai target.
- Ini membagi array yang diberikan ke dalam urutan blok dan kemudian melompati mereka untuk menemukan blok yang mungkin mengandung elemen target.
- Untuk melakukan itu, algoritma membandingkan batas kanan blok dengan elemen target.
- Kompleksitas O($\sqrt{n}$)

Algoritma Pencarian Lompat dalam Java (1)

```
public static int jumpSearch(int[] array, int target) {
    int currentRight = 0; // batas kanan blok saat ini
    int prevRight = 0; //  batas kanan blok sebelumnya

    /* Jika array kosong, elemen tidak ditemukan */
    if (array.length == 0) {
        return -1;
    }

    /* Periksa elemen pertama */
    if (array[currentRight] == target) {
        return 0;
    }

    /* Menghitung Panjang lompatan dari elemen array*/
    int jumpLength = (int) Math.sqrt(array.length);
```

Algoritma Pencarian Lompat dalam Java (2)

```
    /* mencari blok dimana dimungkinkan terdapat elemen */
    while (currentRight < array.length - 1) {

        /* Menghitung batas kanan blok berikutnya */
        currentRight = Math.min(array.length - 1, currentRight+jumpLength);

        if (array[currentRight] >= target) {
           /* Melakukan pencarian linier dalam blok yang ditemukan */
            return backwardSearch(array, target, prevRight, currentRight);
            break; // Menemukan blok yang mungkin mengandung elemen target
        }
        /* memperbaharui batas blok kanan sebelumnya */
        prevRight = currentRight;
    }

    /* Jika blok terakhir tercapai dan tidak dapat
       berisi nilai target => tidak ditemukan */
    if ((currentRight == array.length - 1) && target > array[currentRight]) {
        return -1;
    }
}
```

Algoritma Pencarian Lompat dalam Java (3)

```
/* method pencarian linier */
public static int backwardSearch
(int[] array, int target, int leftExcl, int rightIncl) {
    for (int i = rightIncl; i > leftExcl; i--) {
        if (array[i] == target) {
            return i;
        }
    }
    return -1;
}
```

Algoritma Pencarian Lompat dalam Java (4), penggunaan

```
int[] array = { 10, 13, 19, 20, 24, 26, 30, 34, 35 };

jumpSearch(array, 10); // 0
jumpSearch(array, 13); // 1
jumpSearch(array, 19); // 2
jumpSearch(array, 20); // 3
jumpSearch(array, 24); // 4
jumpSearch(array, 26); // 5
jumpSearch(array, 30); // 6
jumpSearch(array, 34); // 7
jumpSearch(array, 35); // 8

jumpSearch(array, -10); // -1
jumpSearch(array, 11);  // -1
jumpSearch(array, 27);  // -1
jumpSearch(array, 37);  // -1
```