



MODUL PERKULIAHAN

TFC251 – Praktikum Struktur Data

Searching Dalam Python

Penyusun Modul	:	Suamanda Ika Novichasari, M.Kom
Minggu/Pertemuan	:	5
Sub-CPMK/Tujuan Pembelajaran	:	1. Mahasiswa mampu menerapkan algoritma Searching pada bahasa pemrograman python
Pokok Bahasan	:	1. Studi Kasus Pencarian dalam array

**Program Studi Teknologi Informasi (S1)
Fakultas Teknik
Universitas Tidar
Tahun 2024**



Materi 5

SEARCHING DALAM PYTHON

Petunjuk Praktikum :

- Cobalah semua contoh penyelesaian kasus dengan kode program yang terdapat pada semua sub bab dalam modul ini menggunakan google colab.
- Dokumentasikan kegiatan dalam bentuk laporan studi kasus sesuai template laporan praktikum yang telah ditentukan.

Metode sistematis dalam Python untuk menemukan elemen tertentu dalam kumpulan data, seperti array, daftar, atau kamus, disebut algoritma pencarian. Python menawarkan berbagai metode pencarian kepada para pemrogram untuk mengambil informasi dengan efisien dari berbagai jenis struktur data.

Ada dua jenis Algoritma Pencarian yang umum digunakan:

- Pencarian Sekuensial: Di sini, kita secara berurutan menelusuri semua elemen dan memeriksa masing-masing elemen.
- Pencarian Interval: Pada jenis pencarian ini, array dibagi menjadi 2 atau 3 interval pada setiap langkah. Setelah pembagian array menjadi interval, kita menentukan interval di mana kemungkinan elemen yang dicari berada, kemudian kita bagi interval tersebut menjadi 2 atau 3 sub-interval dan terus mengulangi proses ini sampai panjang sub-interval menjadi 0.

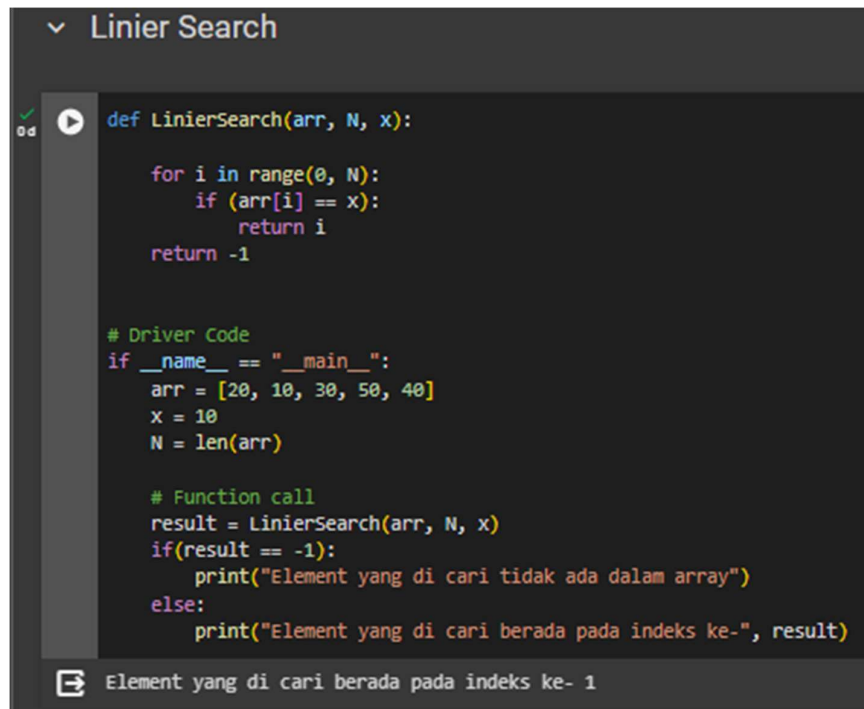
Contoh dari Pencarian Sekuensial adalah Pencarian Linier, sementara contoh dari Pencarian Interval termasuk Pencarian Biner dan Pencarian Ternary.

5.1 Linear Search

Linear Search adalah metode pencarian berurutan yang dimulai dari satu titik dan melewati setiap elemen dari daftar sampai elemen yang dicari ditemukan. Jika elemen yang dicari tidak ditemukan, pencarian akan terus dilanjutkan hingga akhir himpunan data.

Pada Linear Search, setiap elemen dianggap sebagai kemungkinan pencocokan dengan kunci pencarian, dan diperiksa untuk kesesuaiannya. Jika ada elemen yang cocok dengan kunci pencarian, pencarian dianggap berhasil dan indeks elemen tersebut akan dikembalikan. Jika tidak ada elemen yang cocok dengan kunci pencarian, pencarian akan menghasilkan pesan "Tidak ada hasil yang cocok".

Contoh: Misalnya, kita memiliki array `arr[] = { 20, 10, 30, 50, 40 }` dan kunci = 10.



```
def LinierSearch(arr, N, x):  
    for i in range(0, N):  
        if (arr[i] == x):  
            return i  
    return -1  
  
# Driver Code  
if __name__ == "__main__":  
    arr = [20, 10, 30, 50, 40]  
    x = 10  
    N = len(arr)  
  
    # Function call  
    result = LinierSearch(arr, N, x)  
    if(result == -1):  
        print("Element yang di cari tidak ada dalam array")  
    else:  
        print("Element yang di cari berada pada indeks ke-", result)
```

Element yang di cari berada pada indeks ke- 1

Langkah 1: Dimulai dari elemen pertama (indeks 0) dan membandingkan kunci dengan setiap elemen (`arr[i]`).

- Membandingkan kunci dengan elemen pertama `arr[0]`. Karena tidak cocok, iterator bergerak ke elemen berikutnya sebagai kemungkinan pencocokan.

Langkah 2: Ketika membandingkan `arr[1]` dengan kunci, nilai cocok.

Jadi, Linear Search akan memberikan pesan berhasil dan mengembalikan indeks elemen saat kunci ditemukan (di sini adalah 1).

Pada kasus terbaik, kunci mungkin berada pada indeks pertama. Jadi kompleksitas pada kasus terbaik adalah $O(1)$. Pada kasus terburuk, kunci mungkin berada pada indeks terakhir, yaitu berlawanan dengan ujung dari mana pencarian dimulai dalam daftar. Jadi kompleksitas pada kasus terburuk adalah $O(N)$, di mana N

adalah ukuran dari daftar. Ruang Tambahan: $O(1)$ karena kecuali untuk variabel yang digunakan untuk mengiterasi melalui daftar, tidak ada variabel lain yang digunakan.

Keuntungan Pencarian Linear:

- Pencarian linear dapat digunakan tanpa memperhatikan apakah array tersebut telah diurutkan atau tidak. Ini dapat digunakan pada array dari jenis data apa pun.
- Tidak memerlukan memori tambahan.
- Ini adalah algoritma yang cocok untuk dataset kecil.

Kekurangan Pencarian Linear:

- Pencarian linear memiliki kompleksitas waktu $O(N)$, yang membuatnya lambat untuk dataset besar.
- Tidak cocok untuk array besar.
- Kapan Menggunakan Pencarian Linear?
- Ketika kita berurusan dengan dataset kecil.
- Ketika Anda mencari dataset yang disimpan dalam memori berurutan.

5.2 Binary Search

Pencarian Biner adalah algoritma pencarian yang digunakan dalam array yang sudah diurutkan dengan cara membagi interval pencarian menjadi dua bagian secara berulang. Konsep dari pencarian biner adalah menggunakan informasi bahwa array tersebut sudah diurutkan dan mengurangi kompleksitas waktu menjadi $O(\log N)$.

Dalam algoritma ini, ruang pencarian dipisahkan menjadi dua bagian dengan menemukan indeks tengah "mid". Kemudian membandingkan elemen tengah dari ruang pencarian dengan kunci. Jika kunci ditemukan pada elemen tengah, proses dihentikan. Jika kunci tidak ditemukan pada elemen tengah, pilih mana dari dua bagian yang akan digunakan sebagai ruang pencarian selanjutnya. Jika kunci lebih kecil dari elemen tengah, maka sisi kiri digunakan untuk pencarian berikutnya. Jika kunci lebih besar dari elemen tengah, maka sisi kanan digunakan untuk pencarian berikutnya.

Proses ini berlanjut sampai kunci ditemukan atau seluruh ruang pencarian habis.

Misalkan kita memiliki array `arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}`, dan target = 72. Langkah Pertama: Hitung nilai tengah dan bandingkan elemen tengah dengan kunci.

- Jika kunci lebih kecil dari elemen tengah, pindah ke kiri dan jika lebih besar dari tengah, pindahkan ruang pencarian ke kanan.
- Kunci (yaitu, 72) lebih besar dari elemen tengah saat ini (yaitu, 16). Ruang pencarian bergerak ke kanan.

Langkah Kedua: Jika kunci cocok dengan nilai elemen tengah, elemen tersebut ditemukan dan pencarian dihentikan.

Algoritma Pencarian Biner dapat diimplementasikan dengan dua cara berikut:

- **Algoritma Pencarian Biner Iteratif**

```
Binary Search

# It returns location of x in given array arr
def binarySearch(arr, l, r, x):

    while l <= r:

        mid = l + (r - l) // 2

        # Check if x is present at mid
        if arr[mid] == x:
            return mid

        # If x is greater, ignore left half
        elif arr[mid] < x:
            l = mid + 1

        # If x is smaller, ignore right half
        else:
            r = mid - 1

    # If we reach here, then the element
    # was not present
    return -1

# Driver Code
if __name__ == '__main__':
    arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
    x = 72

    # Function call
    result = binarySearch(arr, 0, len(arr)-1, x)
    if result != -1:
        print("Element is present at index", result)
    else:
        print("Element is not present in array")

Element is present at index 8
```

- Algoritma Pencarian Biner Rekursif

```

Binary Search dengan rekursif

# Returns index of x in arr if present, else -1
def binarySearch(arr, l, r, x):

    # Check base case
    if r >= l:

        mid = l + (r - l) // 2

        # If element is present at the middle itself
        if arr[mid] == x:
            return mid

        # If element is smaller than mid, then it
        # can only be present in left subarray
        elif arr[mid] > x:
            return binarySearch(arr, l, mid-1, x)

        # Else the element can only be present
        # in right subarray
        else:
            return binarySearch(arr, mid + 1, r, x)

    # Element is not present in the array
    else:
        return -1

# Driver Code
if __name__ == '__main__':
    arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
    x = 72

    # Function call
    result = binarySearch(arr, 0, len(arr)-1, x)

    if result != -1:
        print("Element terdapat pada indeks ke-", result)
    else:
        print("Element tidak ada dalam array")

Element terdapat pada indeks ke- 8

```

5.3 Jump Search

Seperti Pencarian Biner, Jump Search adalah algoritma pencarian untuk array yang terurut. Ide dasarnya adalah untuk memeriksa lebih sedikit elemen (dibandingkan dengan pencarian linier) dengan melompati beberapa langkah tetap atau melewati beberapa elemen daripada mencari semua elemen.

Sebagai contoh, misalkan kita memiliki array `arr[]` dengan ukuran `n` dan sebuah blok (yang akan dilewati) dengan ukuran `m`. Kemudian kita mencari di indeks-indeks `arr[0]`,

$arr[m], arr[2m], \dots, arr[km]$, dan seterusnya. Setelah kita menemukan interval ($arr[km] < x < arr[(k+1)m]$), kita melakukan operasi pencarian linier dari indeks km untuk menemukan elemen x .

Mari kita pertimbangkan array berikut: (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610). Panjang array ini adalah 16. Pencarian Jump akan menemukan nilai 55 dengan langkah-langkah berikut dengan asumsi bahwa ukuran blok yang akan dilewati adalah 4.

LANGKAH 1: Loncat dari indeks 0 ke indeks 4;

LANGKAH 2: Loncat dari indeks 4 ke indeks 8;

LANGKAH 3: Loncat dari indeks 8 ke indeks 12;

LANGKAH 4: Karena elemen di indeks 12 lebih besar dari 55, kita akan mundur satu langkah untuk kembali ke indeks 8.

LANGKAH 5: Lakukan pencarian linier dari indeks 8 untuk mendapatkan elemen 55.

Berdasarkan kinerjanya, jika kita membandingkannya dengan pencarian linier dan biner, maka pencarian ini lebih baik daripada pencarian linier tetapi tidak lebih baik dari pencarian biner. Urutan peningkatan kinerja adalah sebagai berikut:

pencarian linier < pencarian jump < pencarian biner

Dalam kasus terburuk, kita harus melakukan n/m lompatan, dan jika nilai terakhir yang diperiksa lebih besar dari elemen yang dicari, kita melakukan $m-1$ perbandingan tambahan untuk pencarian linier. Oleh karena itu, jumlah total perbandingan dalam kasus terburuk akan menjadi $((n/m) + m-1)$. Nilai dari fungsi $((n/m) + m-1)$ akan minimum saat $m = \sqrt{n}$. Oleh karena itu, ukuran langkah terbaik adalah $m = \sqrt{n}$.

Pencarian Jump adalah algoritma untuk menemukan nilai tertentu dalam sebuah array yang terurut dengan melompati langkah-langkah tertentu dalam array tersebut.

Langkah-langkahnya ditentukan oleh akar kuadrat dari panjang array. Berikut adalah algoritma langkah demi langkah untuk pencarian jump:

- Tentukan ukuran langkah m dengan mengambil akar kuadrat dari panjang array n .
- Mulai dari elemen pertama array dan lompat m langkah sampai nilai pada posisi tersebut lebih besar dari nilai target.

- Begitu nilai yang lebih besar dari target ditemukan, lakukan pencarian linier dimulai dari langkah sebelumnya sampai target ditemukan atau jelas bahwa target tidak ada dalam array.
- Jika target ditemukan, kembalikan indeksinya. Jika tidak, kembalikan -1 untuk menunjukkan bahwa target tidak ditemukan dalam array.

▼ Jump Search

```

import math

def jumpSearch( arr , x , n ):

    # Finding block size to be jumped
    step = math.sqrt(n)

    # Finding the block where element is
    # present (if it is present)
    prev = 0
    while arr[int(min(step, n)-1)] < x:
        prev = step
        step += math.sqrt(n)
        if prev >= n:
            return -1

    # Doing a linear search for x in
    # block beginning with prev.
    while arr[int(prev)] < x:
        prev += 1

    # If we reached next block or end
    # of array, element is not present.
    if prev == min(step, n):
        return -1

    # If element is found
    if arr[int(prev)] == x:
        return prev

    return -1

# Driver code to test function
arr = [ 0, 1, 1, 2, 3, 5, 8, 13, 21,
        34, 55, 89, 144, 233, 377, 610 ]
x = 55
n = len(arr)

# Find the index of 'x' using Jump Search
index = jumpSearch(arr, x, n)

# Print the index where 'x' is located
print("Number", x, "is at index", "%.0f"%index)

```

Number 55 is at index 10

Mencari data dalam berbagai struktur data adalah bagian penting dari hampir semua aplikasi. Terdapat banyak algoritma yang tersedia untuk digunakan saat melakukan pencarian, dan setiap algoritma memiliki implementasi yang berbeda serta bergantung pada struktur data yang berbeda pula untuk menyelesaikan tugasnya.

Kemampuan untuk memilih algoritma yang tepat untuk setiap tugas adalah keterampilan penting bagi para pengembang, dan dapat membuat perbedaan antara aplikasi yang cepat, handal, dan stabil dengan aplikasi yang gagal karena permintaan yang sederhana.

Studi Kasus 1

Seorang dosen ingin mengetahui nilai terendah dan tertinggi dari 40 mahasiswa. Bantulah dosen tersebut untuk mencari nilai terendah dan tertinggi dengan menggunakan bahasa pemrograman python.

Penyelesaian:

Permasalahan pencarian nilai maksimum dan minimum dari sebuah array dapat menggunakan Algoritma Pencarian Linear:

Dimulai dengan menginisialisasi nilai min dan max sebagai nilai minimum dan maksimum dari dua elemen pertama. Mulai dari elemen ke-3, bandingkan setiap elemen dengan max dan min, dan ubah max dan min sesuai (yaitu, jika elemennya lebih kecil dari min, maka ubah min, jika elemennya lebih besar dari max, maka ubah max, jika tidak, abaikan elemen tersebut).

▼ Linier Search

```
# Python program of above implementation
# structure is used to return two values from minMax()

class pair:
    def __init__(self):
        self.min = 0
        self.max = 0

def getMinMax(arr: list, n: int) -> pair:
    minmax = pair()

    # If there is only one element then return it as min and max both
    if n == 1:
        minmax.max = arr[0]
        minmax.min = arr[0]
        return minmax

    # If there are more than one elements, then initialize min
    # and max
    if arr[0] > arr[1]:
        minmax.max = arr[0]
        minmax.min = arr[1]
    else:
        minmax.max = arr[1]
        minmax.min = arr[0]

    for i in range(2, n):
        if arr[i] > minmax.max:
            minmax.max = arr[i]
        elif arr[i] < minmax.min:
            minmax.min = arr[i]

    return minmax
```

```
# Driver Code
if __name__ == "__main__":
    arr = [1000, 11, 445, 1, 330, 3000]
    arr_size = 6
    minmax = getMinMax(arr, arr_size)
    print("Minimum element is", minmax.min)
    print("Maximum element is", minmax.max)

# This code is contributed by
# sanjeev2552
```

```
➞ Minimum element is 1
Maximum element is 3000
```

Studi Kasus 2

Diberikan sebuah array yang tidak terurut dengan ukuran n . Elemen-elemen array berada dalam rentang 1 hingga n . Satu angka dari himpunan $\{1, 2, \dots, n\}$ hilang dan satu angka muncul dua kali dalam array. Temukan kedua angka ini.

Penyelesaian:

Input: $\text{arr}[] = \{3, 1, 3\}$

Output: Hilang = 2, Terulang = 3

Penjelasan: Dalam array ini, 2 yang hilang dan 3 muncul dua kali.

Input: $\text{arr}[] = \{4, 3, 6, 2, 1, 1\}$

Output: Hilang = 5, Terulang = 1

Untuk dapat menyelesaikan persoalan tersebut dapat menggunakan operasi *count()*.

Pendekatan:

Buat array sementara $\text{temp}[]$ dengan ukuran n dan semua nilai awalnya diatur sebagai 0.

Traverse array input $\text{arr}[]$, dan lakukan langkah berikut untuk setiap $\text{arr}[i]$:

Jika $\text{temp}[\text{arr}[i]-1] == 0$, atur $\text{temp}[\text{arr}[i]-1] = 1$;

Jika $\text{temp}[\text{arr}[i]-1] == 1$, outputkan " $\text{arr}[i]$ " // angka yang terulang

Traverse $\text{temp}[]$ dan outputkan ' $i+1$ ' yang sesuai dengan elemen array $\text{temp}[]$ yang memiliki nilai 0. (Ini adalah angka yang hilang).

```
04 ▶ def printTwoElements(arr):
    n = len(arr)
    temp = [0] * n # Creating temp array of size n with initial values as 0.
    repeatingNumber = -1
    missingNumber = -1

    for i in range(n):
        temp[arr[i] - 1] += 1
        if temp[arr[i] - 1] > 1:
            repeatingNumber = arr[i]
    for i in range(n):
        if temp[i] == 0:
            missingNumber = i + 1
            break

    print("The repeating number is", repeatingNumber, ".")
    print("The missing number is", missingNumber, ".")

arr = [7, 3, 4, 5, 5, 6, 2]
printTwoElements(arr)

The repeating number is 5 .
The missing number is 1 .
```

Studi Kasus 3

Misalkan Anda memiliki sebuah array `arr[]` yang berisi elemen-elemen unik dengan ukuran `N`. Array ini awalnya terurut tetapi kemudian diputar di sekitar titik yang tidak diketahui. Tugas Anda adalah menentukan apakah terdapat sepasang elemen dalam array tersebut yang jumlahnya sama dengan nilai yang ditentukan `X`. Implementasikan dalam bahasa pemrograman python !

Input: `arr[] = {11, 15, 6, 8, 9, 10}`, `X = 16`

Output: true

Explanation: Terdapat Pasangan (6, 10) dengan jumlah 16

Input: `arr[] = {11, 15, 26, 38, 9, 10}`, `X = 35`

Output: true

Explanation: Terdapat Pasangan (26, 9) dengan jumlah 35

Input: `arr[] = {11, 15, 26, 38, 9, 10}`, `X = 45`

Output: false

Explanation: Tidak ada pasangan dengan jumlah 45.