

ARTIKEL INFORMATIF
JENIS-JENIS ALGORITMA DALAM PEMPROGRAMAN
BESERTA DENGANN CONTOHNYA



SERLIA TURU' ALLO_ D0424312
FAKULTAS TEKNIK SISTEM INFORMASI
UNIVERSITAS SULAWESI BARAT

Dalam era digital saat ini, algoritma memainkan peran penting dalam berbagai aspek kehidupan sehari-hari, mulai dari pencarian informasi di internet hingga dapat menyelesaikan permasalahan dengan mudah. Oleh karena itu, algoritma adalah sekumpulan Langkah-langkah yang terstruktur dan sistematis yang dirancang untuk menyelesaikan suatu masalah untuk mencapai tujuan tertentu. Algoritma biasanya digunakan dalam Bahasa Pemrograman Komputer untuk menjalankan tugas secara efisien untuk memberikan keluaran dan masukan yang diberikan. Algoritma dan pemrograman adalah tempat di mana kreativitas bertemu dengan logika dan imajinasi dipadukan dengan struktur. Memahami jenis-jenis algoritma sangatlah penting, karena setiap jenis algoritma memiliki karakteristik dan aplikasi yang berbeda. Artikel ini akan membahas jenis-jenis algoritma serta contohnya.

Ada beberapa pengertian Algoritma menurut para ahli yaitu:

1. Menurut Donal Knuth dalam buku *“The Art of Computer Programming”* Algoritma adalah langkah-langkah dalam menyelesaikan suatu masalah. Jika seseorang menginginkan algoritma yang baik dalam definisi sederhana dan estetika. Ada kriteria yang harus dimiliki yaitu waktu yang dibutuhkan untuk berjalannya algoritma. Kriteria yang lain adalah adaptasi dari algoritma ke komputer program.
2. Menurut definisi Thomas H. Cormen algoritma sebagai serangkaian instruksi yang dapat dijalankan untuk melakukan komputasi atau pemrosesan. Dalam buku *“Introduction to Algorithms”* yang ditulis oleh Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, dan Clifford Stein, mendefinisikan algoritma sebagai prosedur langkah demi langkah yang memiliki tujuan tertentu.
3. Menurut pandangan Robert E. Tarjan algoritma sebagai prosedur yang terdiri dari Langkah-langkah yang dapat diikuti untuk mencapai tujuan tertentu.

Berikut jenis-jenis Algoritma dalam pemrograman Komputer

Algoritma Recursive

Algoritma Recursive adalah jenis algoritma yang memanggil dirinya sendiri untuk menyelesaikan sub-masalah dari masalah yang kecil hingga masalah yang besar. Menurut Thomas H. Cormen et al. Dalam buku “*Introduction to Algorithms*” Algoritma Recursive adalah cara untuk memecahkan membagi masalah menjadi sub-masalah yang lebih kecil dan serupa. Setiap panggilan Algoritma Recursive menyelesaikan bagian dari masalah hingga mencapai kondisi dasar.

Dalam buku tersebut yang telah di rangkum bersama Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, dan Clifford Stein, mendefinisikan algoritma sebagai prosedur langkah demi langkah yang memiliki tujuan tertentu.

Contohnya Algoritma Recursive:

Faktorial dari bilangan bulat positif n (ditulis $n!$) adalah hasil kali semua bilangan bulat positif dari 1 hingga n . Secara formal:

- $N! = n \times (n-1)! \quad n! = n \times (n-1)! \quad n! = n \times (n-1)! \quad \text{untuk } n > 0$
- $0! = 1$

```
def factorial(n):  
    # Kasus dasar  
    if n == 0:  
        return 1  
    else:  
        # Panggilan rekursif  
        return n * factorial(n - 1)  
  
# Contoh penggunaan  
number = 5  
result = factorial(number)  
print(f"Faktorial dari {number} adalah {result}")
```

Penjelasan Contoh

1. **Definisi Fungsi:** Fungsi `factorial(n)` menerima satu parameter n .
2. **Kasus Dasar:**
 - Jika n adalah 0, fungsi mengembalikan 1. Ini adalah kondisi yang menghentikan rekursi, karena $0! = 1$.
3. **Panggilan Rekursif:**
 - Jika n lebih besar dari 0, fungsi akan mengembalikan hasil kali n dengan hasil dari `factorial(n - 1)`.
 - Ini berarti fungsi memanggil dirinya sendiri dengan nilai n yang lebih kecil hingga mencapai kasus dasar.

4. Contoh Penggunaan:

- Jika kita memanggil `factorial(5)`, berikut adalah urutan panggilan fungsi:
 - $5! = 5 \times 4! = 5 \times 4 \times 3! = 5 \times 4 \times 3 \times 2! = 5 \times 4 \times 3 \times 2 \times 1! = 5 \times 4 \times 3 \times 2 \times 1 \times 0! = 5 \times 4 \times 3 \times 2 \times 1 \times 1 = 120$
 - $4! = 4 \times 3! = 4 \times 3 \times 2! = 4 \times 3 \times 2 \times 1! = 4 \times 3 \times 2 \times 1 \times 0! = 4 \times 3 \times 2 \times 1 \times 1 = 24$
 - $3! = 3 \times 2! = 3 \times 2 \times 1! = 3 \times 2 \times 1 \times 0! = 3 \times 2 \times 1 \times 1 = 6$
 - $2! = 2 \times 1! = 2 \times 1 \times 0! = 2 \times 1 \times 1 = 2$
 - $1! = 1 \times 0! = 1 \times 1 = 1$
 - $0! = 1$ (kasus dasar)
- Jadi, $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Penjelasan Contoh

1. Basis Kasus:

- if $n == 0$ or $n == 1$: Ketika sama n dengan 0 atau 1, fungsi mengembalikan 1.

2. Rekurens:

- return $n * \text{factorial}(n - 1)$: Jika n lebih besar dari 1, fungsi memanggil dirinya sendiri dengan $n - 1$ dan mengalikan hasilnya dengan n .

Proses Rekursi

Jika kita memanggil `factorial(5)`, proses yang terjadi adalah:

- $\text{factorial}(5) \rightarrow 5 * \text{factorial}(4)$
- $\text{factorial}(4) \rightarrow 4 * \text{factorial}(3)$
- $\text{factorial}(3) \rightarrow 3 * \text{factorial}(2)$
- $\text{factorial}(2) \rightarrow 2 * \text{factorial}(1)$
- $\text{factorial}(1) \rightarrow 1$ (basis kasus)

Kemudian, hasilnya akan dihitung kembali dari bawah ke atas:

- $\text{factorial}(2) \rightarrow 2 * 1 = 2$
- $\text{factorial}(3) \rightarrow 3 * 2 = 6$
- $\text{factorial}(4) \rightarrow 4 * 6 = 24$
- $\text{factorial}(5) \rightarrow 5 * 24 = 120$

Hasil

Ketika menjalankan kode, outputnya akan:

Faktorial dari 5 adalah 120

✚ Algoritma sorting

Algoritma sorting adalah metode yang digunakan untuk mengurutkan elemen dalam daftar atau array dalam urutan tertentu, baik secara ascending (menaik) atau descending (menurun). Menurut Robert Sedgewick dalam “Algorithms” Algoritma sorting mengatur elemen dalam urutan tertentu untuk mempermudah pencarian atau pemrosesan lebih lanjut. Contoh yang umum termasuk Bubble Sort, Quick Sort, dan Merge Sort, masing-masing dengan kelebihan dan kekurangan dalam hal kompleksitas waktu dan penggunaan memori.

- **Bubble Sort**

Bubble Sort adalah algoritma paling sederhana mudah dipahami yang bekerja berulang kali melangkahi daftar yang perlu disortir membandingkan elemen yang berdekatan dan menukarnya jika mereka berada pada urutan yang salah.

```
def bubble_sort(arr):
    n = len(arr)
    # Melakukan iterasi untuk setiap elemen dalam array
    for i in range(n):
        # Loop untuk membandingkan elemen
        for j in range(0, n-i-1):
            # Jika elemen saat ini lebih besar dari yang berikutnya, tukar
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Contoh penggunaan
data = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(data)
print("Data terurut:", data)
```

- **Quick Sort**

Quick Sort adalah algoritma pengurutan yang paling cepat dan paling tepat sehingga algoritma Quick Sort paling banyak digunakan dalam praktik karna cara pengerjaannya yang cepat dan efisien.

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[len(arr) // 2] # Memilih pivot
        left = [x for x in arr if x < pivot] # Elemen lebih kecil dari pivot
        middle = [x for x in arr if x == pivot] # Elemen sama dengan pivot
        right = [x for x in arr if x > pivot] # Elemen lebih besar dari pivot
        return quick_sort(left) + middle + quick_sort(right) # Gabungkan hasil

# Contoh penggunaan
data = [10, 7, 8, 9, 1, 5]
sorted_data = quick_sort(data)
print("Data terurut:", sorted_data)
```

- **Merge Sort**

Merge Sort adalah algoritma pengurutan yang menggunakan pendekatan divide and conquer (pembagian dan penaklukan). Algoritma ini membagi daftar yang akan diurutkan menjadi dua sub-daftar, mengurutkan masing-masing sub-daftar secara rekursif, dan kemudian menggabungkan (merge) kedua sub-daftar tersebut untuk menghasilkan daftar yang terurut.

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    # Pembagian
    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid]) # Sub-daftar kiri
    right_half = merge_sort(arr[mid:]) # Sub-daftar kanan

    # Penggabungan
    return merge(left_half, right_half)

def merge(left, right):
    sorted_list = []
    i = j = 0
```

```

# Menggabungkan dua sub-daftar
while i < len(left) and j < len(right):
    if left[i] < right[j]:
        sorted_list.append(left[i])
        i += 1
    else:
        sorted_list.append(right[j])
        j += 1

# Menambahkan sisa elemen dari sub-daftar
sorted_list.extend(left[i:])
sorted_list.extend(right[j:])

return sorted_list

# Contoh penggunaan
data = [38, 27, 43, 3, 9, 82, 10]
sorted_data = merge_sort(data)
print("Data terurut:", sorted_data)

```

Algoritma Searching

Algoritma Searching adalah metode linear dan biner yang digunakan untuk menemukan elemen tertentu dalam Kumpulan data, seperti array dan daftar. Algoritma Searching sangat penting dalam pemrograman komputer karna banyak aplikasi yang bergantung pada kemampuan mencari data yang efisien. Menurut William F. Trench dalam bukunya yang berjudul *"An Introduction to Computer Algorithms,"* Trench mendefinisikan algoritma searching sebagai metode untuk menemukan elemen dalam struktur data. Ia menyatakan bahwa pemilihan metode yang tepat tergantung pada jenis data dan kebutuhan aplikasi.

Contohnya

■ Pencarian Linear

Pencarian linear adalah metode yang paling sederhana, di mana setiap elemen dalam daftar diperiksa satu per satu hingga elemen yang dicari ditemukan.

```

1 def linear_search(arr, target):
2     for index, value in enumerate(arr):
3         if value == target:
4             return index # Mengembalikan indeks elemen yang ditemukan
5     return -1 # Jika elemen tidak ditemukan
6
7 # Contoh penggunaan
8 data = [5, 3, 8, 4, 2]
9 target = 4
10 result = linear_search(data, target)
11 print(f"Elemen {target} ditemukan di indeks: {result}")

```

■ Pencarian Biner

Pencarian biner adalah metode yang lebih efisien tetapi memerlukan daftar yang sudah terurut. Algoritma ini membagi daftar menjadi dua bagian dan memeriksa bagian yang relevan.

```

vscode > python > ...
1 def binary_search(arr, target):
2     left, right = 0, len(arr) - 1
3     while left <= right:
4         mid = (left + right) // 2 # Menentukan titik tengah
5         if arr[mid] == target:
6             return mid # Mengembalikan indeks elemen yang ditemukan
7         elif arr[mid] < target:
8             left = mid + 1 # Fokus pada bagian kanan
9         else:
10            right = mid - 1 # Fokus pada bagian kiri
11    return -1 # Jika elemen tidak ditemukan
12
13 # Contoh penggunaan
14 data = [1, 2, 3, 4, 5, 6, 7, 8, 9] # Data harus terurut
15 target = 6
16 result = binary_search(data, target)
17 print(f"Elemen {target} ditemukan di indeks: {result}")

```

▪ Hasil

```
Elemen 4 ditemukan di indeks: 3
Elemen 6 ditemukan di indeks: 5
```

🌈 Algoritma Backtracking

Algoritma Backtracking merupakan salah satu algoritma yang sangat penting dalam pemrograman karena mampu menyelesaikan permasalahan yang sulit yang efisien atau pencarian Solusi misalnya pada masalah N-Queens. Menurut Cormen, Leiserson, Rivest, dan Stein. Dalam *"Introduction to Algorithms,"* para penulis menyatakan bahwa backtracking dapat digunakan untuk menyelesaikan masalah dengan eksplorasi ruang solusi, dan menghindari langkah yang tidak produktif dengan menggunakan teknik "pruning."

Contohnya Implementasi N-Queens

```
.vscode > python > ...
1 def is_safe(board, row, col):
2     # Cek kolom
3     for i in range(row):
4         if board[i][col] == 1:
5             return False
6     # Cek diagonal kiri atas
7     for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
8         if board[i][j] == 1:
9             return False
10    # Cek diagonal kanan atas
11    for i, j in zip(range(row, -1, -1), range(col, len(board))):
12        if board[i][j] == 1:
13            return False
14    return True
15
16 def solve_n_queens(board, row):
17     if row >= len(board):
18         print_board(board)
19         return
20     for col in range(len(board)):
21         if is_safe(board, row, col):
22             board[row][col] = 1 # Tempatkan ratu
23             solve_n_queens(board, row + 1) # Rekursi untuk baris berikutnya
24             board[row][col] = 0 # Backtrack
25
26 def print_board(board):
27     for row in board:
28         print(" ".join("Q" if x == 1 else "." for x in row))
29     print()
30
31 # Contoh penggunaan
32 N = 4
33 board = [[0 for _ in range(N)] for _ in range(N)]
34 solve_n_queens(board, 0)

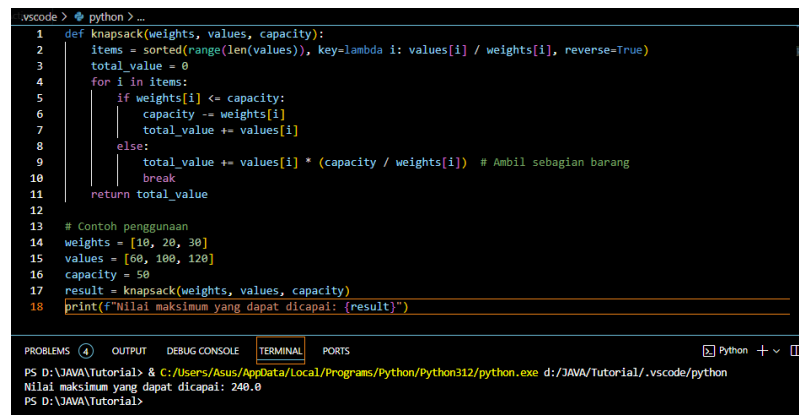
--
.vscode > python > is_safe
16 def solve_n_queens(board, row):
17     if row >= len(board):
18         print_board(board)
19         return
20     for col in range(len(board)):
21         if is_safe(board, row, col):
22             board[row][col] = 1 # Tempatkan ratu
23             solve_n_queens(board, row + 1) # Rekursi untuk baris berikutnya
24             board[row][col] = 0 # Backtrack
25
26 def print_board(board):
27     for row in board:
28         print(" ".join("Q" if x == 1 else "." for x in row))
29     print()
30
31 # Contoh penggunaan
32 N = 4
33 board = [[0 for _ in range(N)] for _ in range(N)]
34 solve_n_queens(board, 0)

--
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\JAVA\Tutorial> & C:/Users/Asus/AppData/Local/Programs/Python/Python312/python.exe d:/JAVA/Tutorial/.vscode/python
- Q - -
- . - Q
Q - . -
- . Q -
- . Q -
Q - . -
- . - Q
- Q - -
PS D:\JAVA\Tutorial>
```

Algoritma Greedy

Algoritma Greedy adalah pendekatan untuk memecahkan masalah optimasi dengan membuat pilihan terbaik (lokal) di setiap langkah, dengan harapan bahwa pilihan tersebut akan mengarah pada solusi optimal secara keseluruhan. Algoritma ini tidak selalu memberikan solusi optimal untuk semua masalah, tetapi sering kali cukup efisien dan efektif untuk masalah tertentu. Menurut V. F. Fomin dan F. A. Fradkin Dalam buku yang berjudul "*Algorithmic Graph Theory*," menjelaskan bahwa algoritma greedy sering digunakan dalam teori graf dan masalah jaringan, di mana pengambilan keputusan lokal dapat menghasilkan solusi yang baik.

Contoh Implementasi



```
.vscode > python > ...
1 def knapsack(weights, values, capacity):
2     items = sorted(range(len(values)), key=lambda i: values[i] / weights[i], reverse=True)
3     total_value = 0
4     for i in items:
5         if weights[i] <= capacity:
6             capacity -= weights[i]
7             total_value += values[i]
8         else:
9             total_value += values[i] * (capacity / weights[i]) # Ambil sebagian barang
10            break
11    return total_value
12
13 # Contoh penggunaan
14 weights = [10, 20, 30]
15 values = [60, 100, 120]
16 capacity = 50
17 result = knapsack(weights, values, capacity)
18 print(f"Nilai maksimum yang dapat dicapai: {result}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\JAVA\tutorial & C:\Users\Asus\AppData\Local\Programs\Python\Python312\python.exe d:/JAVA/tutorial/.vscode/python
Nilai maksimum yang dapat dicapai: 240.0
PS D:\JAVA\tutorial >
```

Algoritma Randomized

Algoritma Randomized adalah algoritma yang kuat dalam pemrograman dan analisis algoritma, memungkinkan penyelesaian masalah yang kompleks dengan cara yang lebih efisien. Pendekatan ini sering kali digunakan untuk meningkatkan efisiensi dalam pemecahan masalah, di mana solusi deterministik mungkin memerlukan waktu yang lebih lama. Dengan menggunakan unsur acak, algoritma randomized dapat menghasilkan solusi yang baik dalam waktu yang lebih singkat.

Menurut Mitzenmacher dan Upfal. Dalam buku "*Probability and Computing*," menggambarkan bagaimana probabilitas dapat digunakan untuk analisis algoritma, termasuk penggunaan teknik acak untuk meningkatkan kinerja dan mengurangi kompleksitas dalam masalah tertentu dan menurut R. Motwani dan P. Raghavan

Dalam buku "*Randomized Algorithms*," mereka membahas berbagai metode dan teknik dalam algoritma randomized, serta memberikan contoh aplikasinya dalam masalah nyata, seperti penjadwalan dan pemrograman graf.

Contoh Implementasi Randomized QuickSort

```
.vscode > python > ...
1 import random
2
3 def randomized_partition(arr, low, high):
4     pivot_index = random.randint(low, high) # Pilih pivot secara acak
5     arr[pivot_index], arr[high] = arr[high], arr[pivot_index] # Tukar pivot dengan elemen terakhir
6     pivot = arr[high]
7     i = low - 1
8     for j in range(low, high):
9         if arr[j] < pivot:
10             i += 1
11             arr[i], arr[j] = arr[j], arr[i] # Tukar elemen
12     arr[i + 1], arr[high] = arr[high], arr[i + 1] # Tempatkan pivot pada posisi yang benar
13     return i + 1
14
15 def randomized_quick_sort(arr, low, high):
16     if low < high:
17         pi = randomized_partition(arr, low, high)
18         randomized_quick_sort(arr, low, pi - 1) # Rekursi untuk bagian kiri
19         randomized_quick_sort(arr, pi + 1, high) # Rekursi untuk bagian kanan
20
21 # Contoh penggunaan
22 data = [10, 7, 8, 9, 1, 5]
23 randomized_quick_sort(data, 0, len(data) - 1)
24 print("Data terurut:", data)
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + v

PS D:\JAVA\Tutorial> & C:/Users/Asus/AppData/Local/Programs/Python/Python312/python.exe d:/JAVA/Tutorial/.vscode/python
Data terurut: [1, 5, 7, 8, 9, 10]
PS D:\JAVA\Tutorial>

Penjelasan Kode

1. **Fungsi `randomized_partition`**
 - Memilih pivot secara acak dari array.
 - Memindahkan pivot ke posisi akhir dan membagi array berdasarkan pivot.
 - Mengembalikan indeks pivot yang baru.
2. **Fungsi `randomized_quick_sort`**
 - Menggunakan rekursi untuk membagi array menjadi dua bagian berdasarkan pivot, dan mengurutkan keduanya secara terpisah.
3. **Contoh Penggunaan:**
 - Array yang tidak terurut diurutkan menggunakan Randomized QuickSort, dan hasilnya dicetak.