



Laporan Praktikum Algoritma dan Pemrograman

Semester Genap 2023/2024

NIM	71230971
Nama Lengkap	James Marvin Santoso
Minggu ke / Materi	13 / Fungsi Rekursif

SAYA MENYATAKAN BAHWA LAPORAN PRAKTIKUM INI SAYA BUAT DENGAN USAHA SENDIRI TANPA MENGGUNAKAN BANTUAN ORANG LAIN. SEMUA MATERI YANG SAYA AMBIL DARI SUMBER LAIN SUDAH SAYA CANTUMKAN SUMBERNYA DAN TELAH SAYA TULIS ULANG DENGAN BAHASA SAYA SENDIRI.

SAYA SANGGUP MENERIMA SANKSI JIKA MELAKUKAN KEGIATAN PLAGIASI, TERMASUK SANKSI TIDAK LULUS MATA KULIAH INI.

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN DUTA WACANA
YOGYAKARTA
2024

BAGIAN 1: MATERI MINGGU INI

PENGERTIAN REKURSIF

Rekursi adalah teknik pemrograman di mana sebuah fungsi memanggil dirinya sendiri untuk menyelesaikan masalah. Teknik ini sering digunakan untuk memecahkan masalah yang dapat dipecah menjadi sub-masalah yang lebih kecil dan memiliki struktur yang mirip dengan masalah asli. Rekursi memungkinkan pemrogram menyelesaikan masalah kompleks dengan pendekatan yang lebih sederhana dan elegan dibandingkan dengan metode iteratif tradisional.

Meskipun rekursi dapat memberikan solusi yang elegan dan singkat untuk beberapa masalah, penggunaannya harus dilakukan dengan hati-hati. Rekursi dapat menyebabkan overhead memori dan waktu yang besar jika tidak digunakan dengan benar, terutama dalam kasus di mana kedalaman rekursi bisa menjadi sangat besar. Sebagai panduan umum, jika masalah dapat diselesaikan dengan cara iteratif yang sederhana, lebih baik memilih pendekatan iteratif daripada rekursif.

Dasar-Dasar Rekursi

- Fungsi Rekursif: Fungsi yang memanggil dirinya sendiri. Fungsi rekursif harus memiliki dua elemen utama:
 - a) Basis Kasus (Base Case): Kondisi yang menghentikan rekursi untuk mencegah loop tak terbatas dan stack overflow. Basis kasus adalah kondisi paling sederhana yang dapat diselesaikan secara langsung.
 - b) Kasus Rekursif (Recursive Case): Bagian di mana fungsi memanggil dirinya sendiri dengan argumen yang dimodifikasi sehingga mendekati basis kasus. Kasus rekursif menguraikan masalah besar menjadi sub-masalah yang lebih kecil.

Contoh umum dari penggunaan rekursi adalah fungsi faktorial.

Faktorial dari n (ditulis sebagai $n!$) adalah hasil dari perkalian semua bilangan bulat positif dari 1 hingga n . Fungsi faktorial rekursif dapat didefinisikan sebagai berikut:

- Basis Kasus: Faktorial dari 0 adalah 1 ($0! = 1$).
- Kasus Rekursif: Faktorial dari n adalah n dikali faktorial dari $n-1$ ($n! = n * (n-1)!$).

```
def faktorial(n):  
    if n == 0:  
        return 1 # Basis Kasus  
    else:  
        return n * faktorial(n - 1) # Kasus Rekursif
```

Kode tersebut mendefinisikan fungsi rekursif `faktorial` yang menghitung faktorial dari bilangan `n`, di mana jika `n` adalah 0 mengembalikan 1 (basis kasus), dan jika bukan,

mengembalikan hasil perkalian `n` dengan pemanggilan fungsi `faktorial` dengan argumen `n-1` (kasus rekursif).

Rekursi dan Loop

Rekursi dapat digunakan sebagai alternatif dari loop seperti for atau while. Misalnya, mencetak elemen dari sebuah list dapat dilakukan baik dengan loop maupun dengan rekursi. Namun, tidak semua masalah cocok diselesaikan dengan rekursi. Penggunaan rekursi yang tidak bijaksana dapat menyebabkan stack overflow jika basis kasus tidak ditentukan dengan benar atau jika kedalaman rekursi terlalu besar.

Rekursi dapat digunakan sebagai alternatif dari loop seperti for atau while. Misalnya, mencetak elemen dari sebuah list dapat dilakukan baik dengan loop maupun dengan rekursi. Namun, tidak semua masalah cocok diselesaikan dengan rekursi. Penggunaan rekursi yang tidak bijaksana dapat menyebabkan stack overflow jika basis kasus tidak ditentukan dengan benar atau jika kedalaman rekursi terlalu besar.

- Tipe Rekursi
 - a) Rekursi Langsung: Fungsi memanggil dirinya sendiri secara langsung.
 - b) Rekursi Tidak Langsung: Fungsi A memanggil fungsi B yang pada akhirnya memanggil kembali fungsi A.
 - c) Rekursi Ekor (Tail Recursion): Panggilan rekursif berada di akhir fungsi. Ini dapat dioptimalkan oleh beberapa kompiler menjadi loop yang lebih efisien.

Contoh dalam Python:

```
def faktorial_tail(n, hasil=1):  
    if n == 0:  
        return hasil  
    else:  
        return faktorial_tail(n - 1, n * hasil)
```

Fungsi faktorial_tail menghitung faktorial dari bilangan n dengan menggunakan rekursi tail, di mana jika n adalah 0 mengembalikan hasil (basis kasus), dan jika bukan, memanggil dirinya sendiri dengan argumen n-1 dan n * hasil (kasus rekursif) untuk mengakumulasi hasil faktorial.

Contoh Rekursi Lanjutan :

- 1) Fibonacci: Deret Fibonacci adalah deret angka di mana setiap angka adalah jumlah dari dua angka sebelumnya. Fungsi rekursif untuk menghitung bilangan Fibonacci ke-n:

```
def fibonacci(n):  
    if n <= 1:  
        return n  
    else:  
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Fungsi `fibonacci` dalam Python menghitung bilangan Fibonacci ke-n secara rekursif dengan memeriksa apakah n kurang dari atau sama dengan 1 untuk mengembalikan nilai n sebagai kasus dasar, dan jika tidak, fungsi tersebut memanggil dirinya sendiri dengan argumen n-1 dan n-2, kemudian menjumlahkan hasil dari kedua pemanggilan tersebut untuk mendapatkan bilangan Fibonacci ke-n.

- 2) Menara Hanoi: Permainan teka-teki yang melibatkan pemindahan tumpukan cakram dari satu tiang ke tiang lain dengan aturan tertentu. Fungsi rekursif untuk memecahkan Menara Hanoi:

```
def hanoi(n, asal, tujuan, bantu):  
    if n > 0:  
        hanoi(n - 1, asal, bantu, tujuan)  
        print(f"Pindahkan cakram dari {asal} ke {tujuan}")  
        hanoi(n - 1, bantu, tujuan, asal)
```

Fungsi `hanoi` adalah implementasi rekursif untuk menyelesaikan masalah menara Hanoi dengan cara memindahkan cakram dari tiang asal ke tujuan, dengan bantuan tiang bantu, sesuai aturan bahwa cakram yang lebih besar tidak boleh diletakkan di atas cakram yang lebih kecil.

Rekursi dalam Pemrograman Fungsional

Dalam pemrograman fungsional, rekursi adalah konsep dasar yang menggantikan penggunaan loop iteratif. Bahasa pemrograman seperti Haskell dan Lisp menggunakan rekursi untuk mengelola iterasi dan pengulangan. Rekursi memungkinkan ekspresi yang lebih alami dan elegan dari masalah yang memerlukan iterasi kompleks.

- **Manfaat:**

- a) Menyederhanakan kode untuk masalah kompleks.
- b) Lebih mudah dipahami dan dipelihara dalam banyak kasus.
- c) Membuat solusi lebih elegan dan intuitif.

- **Kekurangan:**

- a) Dapat menyebabkan tumpukan panggilan yang dalam dan penggunaan memori yang besar.
- b) Rentan terhadap stack overflow jika basis kasus tidak ditentukan dengan benar atau jika masalah terlalu besar untuk diselesaikan secara rekursif.
- c) Rekursi dapat kurang efisien dibandingkan dengan loop iteratif dalam beberapa kasus karena overhead pemanggilan fungsi berulang.

Studi Kasus dan Implementasi

- Studi kasus dari algoritma rekursif, seperti:
 - a) Pencarian Biner: Algoritma pencarian efisien yang bekerja pada list yang telah diurutkan dengan membagi ruang pencarian menjadi dua setiap kali.
 - b) Merge Sort: Algoritma pengurutan berbasis rekursi yang membagi list menjadi dua, mengurutkan kedua bagian, dan kemudian menggabungkannya kembali.
 - c) Quick Sort: Algoritma pengurutan cepat yang menggunakan teknik pembagian dan penaklukan (divide and conquer) dengan memilih elemen pivot dan mengurutkan elemen di sekitar pivot.

Optimasi Rekursi

Untuk mengatasi beberapa kekurangan rekursi, teknik optimasi seperti memoization dan dynamic programming digunakan. Memoization menyimpan hasil dari sub-masalah yang telah dihitung sebelumnya sehingga tidak perlu dihitung ulang. Dynamic programming memperluas ide ini dengan menggunakan tabel untuk menyimpan hasil dari sub-masalah, memungkinkan algoritma rekursif yang lebih efisien.

Rekursi dalam Struktur Data

Rekursi sering digunakan dalam operasi pada struktur data seperti pohon dan graf. Misalnya, traversal pohon biner (pre-order, in-order, dan post-order) biasanya diimplementasikan menggunakan rekursi. Berikut adalah contoh traversal in-order pada pohon biner:

```
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

def inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.data, end=' ')
        inorder_traversal(root.right)
```

Kode tersebut mengimplementasikan struktur data pohon biner dengan kelas `Node`, di mana setiap node memiliki nilai dan dua referensi ke node anak kiri dan kanan, serta fungsi `inorder_traversal` untuk melakukan penelusuran in-order pada pohon biner tersebut.

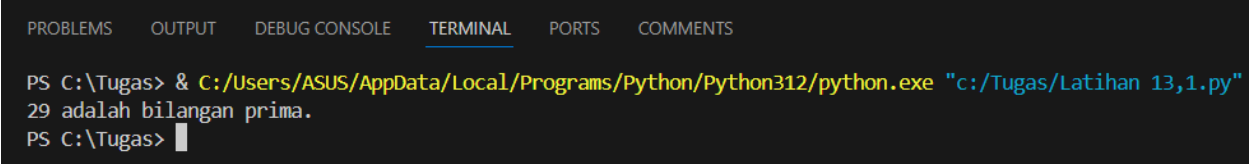
BAGIAN 2: LATIHAN MANDIRI

SOAL 13.1

- Source Code :

```
def is_prime(n, i=2):  
    if n < 2:  
        return False  
    if i * i > n:  
        return True  
    if n % i == 0:  
        return False  
    return is_prime(n, i + 1)  
  
number = 29  
if is_prime(number):  
    print(f"{number} adalah bilangan prima.")  
else:  
    print(f"{number} bukan bilangan prima.")
```

- Output :



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS  
  
PS C:\Tugas> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe "c:/Tugas/Latihan 13,1.py"  
29 adalah bilangan prima.  
PS C:\Tugas> █
```

- Penjelasan :

Fungsi `is_prime` digunakan untuk memeriksa apakah suatu bilangan `n` adalah bilangan prima dengan melakukan iterasi dari 2 hingga akar dari `n`, dan mengembalikan `True` jika `n` adalah bilangan prima, `False` jika tidak.

SOAL 13.2

- Source Code :

```
def is_palindrome(s):  
    s = ''.join(char.lower() for char in s if char.isalnum())  
  
    def check_palindrome(s, left, right):  
        if left >= right:  
            return True  
        if s[left] != s[right]:  
            return False  
        return check_palindrome(s, left + 1, right - 1)  
  
    return check_palindrome(s, 0, len(s) - 1)  
  
kalimat = "Nama saya andi saya umur 18"  
if is_palindrome(kalimat):  
    print(f"{kalimat}" adalah palindrom.)  
else:  
    print(f"{kalimat}" bukan palindrom.)
```

- Output :

```
PS C:\Tugas> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe "c:/Tugas/Latihan 13,2.py"  
"Nama saya andi saya umur 18" bukan palindrom.  
PS C:\Tugas> █
```

- Penjelasan :

Fungsi `is_palindrome` digunakan untuk memeriksa apakah sebuah string `s` adalah palindrom, dengan membersihkan karakter non-alfanumerik dan mengonversi huruf-hurufnya menjadi huruf kecil, lalu menggunakan rekursi untuk membandingkan karakter dari kedua ujung string hingga mencapai titik tengahnya.

SOAL 13.3

- Source Code :

```
def sum_odd_series(n):  
    if n < 1:  
        return 0  
    if n == 1:  
        return 1  
    if n % 2 == 0:  
        n -= 1  
    return n + sum_odd_series(n - 2)  
  
n = 9  
result = sum_odd_series(n)  
print(f"Jumlah deret bilangan ganjil hingga {n} adalah {result}.")
```

- Output :

```
PS C:\Tugas> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe "c:/Tugas/Latihan 13,3.py"  
Jumlah deret bilangan ganjil hingga 9 adalah 25.  
PS C:\Tugas>
```

- Penjelasan :

Fungsi `sum_odd_series` digunakan untuk menghitung jumlah dari deret bilangan ganjil hingga suatu bilangan bulat positif `n`. Pertama, fungsi melakukan pengecekan untuk kasus dasar, yaitu ketika `n` kurang dari 1, di mana hasilnya adalah 0, dan ketika `n` sama dengan 1, di mana hasilnya adalah 1. Selanjutnya, jika `n` adalah bilangan genap, fungsi mengurangi `n` menjadi bilangan ganjil sebelum melanjutkan dengan rekursi. Setiap rekursi mengurangi `n` sebesar 2 dan menambahkannya ke hasil akhir. Dalam contoh ini, jika `n` adalah 9, hasilnya adalah jumlah dari $1 + 3 + 5 + 7 + 9 = 25$.

SOAL 13.4

- Source Code :

```
def sum_of_digits(n):  
    if len(n) == 1:  
        return int(n)  
    return int(n[0]) + sum_of_digits(n[1:])  
  
number = "234"  
result = sum_of_digits(number)  
print(f"Jumlah digit dari {number} adalah {result}.")
```

- Output :

```
PS C:\Tugas> & C:/users/ASUS/Appdata/Local/Programs/Python/Python312/python.exe -c:/Tugas/Latinah_13,4.py  
Jumlah digit dari 234 adalah 9.  
PS C:\Tugas> █
```

- Penjelasan :

Fungsi `sum_of_digits(n)` pada kode tersebut menghitung jumlah digit dalam string angka `n`. Jika panjang string `n` adalah 1, fungsi mengembalikan nilai integer karakter tunggal tersebut; jika tidak, fungsi mengembalikan jumlah nilai integer dari karakter pertama string `n` dan hasil rekursif dari `sum_of_digits(n[1:])`, yaitu string `n` tanpa karakter pertama. Contoh penggunaan: inisialisasi `number = "234"`, lalu `result = sum_of_digits(number)`, hasilnya adalah 9 (2 + 3 + 4), yang kemudian dicetak menggunakan `print(f"Jumlah digit dari {number} adalah {result}.")`.

SOAL 13.5

- Source Code :

```
def combination(n, k):  
    if k == 0 or k == n:  
        return 1  
    return combination(n - 1, k - 1) + combination(n - 1, k)  
  
n = 5  
k = 2  
result = combination(n, k)  
print(f"C({n}, {k}) = {result}")
```

- Output :

```
PS C:\Tugas> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe "c:/Tugas/Latihan 13,5.py"  
C(5, 2) = 10  
PS C:\Tugas> █
```

- Penjelasan :

Kode ini adalah implementasi dalam Python untuk menghitung kombinasi $C(n, k)$, di mana $C(n, k)$ merupakan jumlah cara untuk memilih k item dari total n item tanpa memperhatikan urutan. Fungsi `combination(n, k)` menggunakan pendekatan rekursif dengan kondisi dasar jika k sama dengan 0 atau sama dengan n , di mana jumlah kombinasi adalah 1. Jika tidak, fungsi ini mengembalikan jumlah kombinasi dari dua kasus: ketika item ke- n dipilih, sisa item yang dipilih adalah $C(n-1, k-1)$, dan ketika item ke- n tidak dipilih, sisa item yang dipilih adalah $C(n-1, k)$. Contoh penggunaan adalah dengan menginisialisasi $n = 5$ dan $k = 2$, kemudian mencetak hasilnya.

- Link GitHub :

<https://github.com/Jamesmarvins/Tugas-Alpro-13.git>