



# Rekursi Lanjut

Tim Olimpiade Komputer Indonesia

Bagian 1

## Fibonacci



## Soal: Fibonacci

Deskripsi:

- Deret Fibonacci merupakan deret yang mana suatu anggota adalah penjumlahan dari dua anggota sebelumnya, kecuali dua anggota pertama.
- Jika  $f_N$  adalah bilangan Fibonacci ke- $N$ , maka  $f_0 = 0$ ,  $f_1 = 1$ , dan  $f_N = f_{N-1} + f_{N-2}$  untuk  $N > 1$ .
- Beberapa bilangan pertama dari deret Fibonacci adalah 0, 1, 1, 2, 3, 5, 8, 13, 21, ....
- Carilah bilangan Fibonacci ke- $N$ .
- Contoh: Bilangan Fibonacci ke-6 adalah 8. Perhatikan bahwa indeks dimulai dari 0.



## Soal: Fibonacci (lanj.)

Format masukan:

- Sebuah baris berisi sebuah bilangan  $N$ .

Format keluaran:

- Sebuah baris berisi bilangan Fibonacci ke- $N$ .

Batasan:

- $0 \leq N \leq 20$



# Solusi

- Bagaimana cara mendapatkan nilai dari dua bilangan Fibonacci sebelum bilangan Fibonacci ke- $N$ ?
- Apakah kita bisa melakukan rekursi untuk mencari bilangan Fibonacci ke- $(N - 1)$  dan ke- $(N - 2)$ ?



# Penjelasan Solusi Rekursif

## *Base Case*

- Pada batasan soal, nilai  $N$  berkisar antara 0 sampai 20.
- Dari batasan tersebut, kasus terkecil yang sudah pasti diketahui jawabannya adalah  $f_0$  dan  $f_1$ .
- Nilai dari  $f_0 = 0$  dan  $f_1 = 1$ , atau dapat dituliskan  $f_N = N$ , untuk  $0 \leq N \leq 1$ .
- Sehingga,  $N = 0$  dan  $N = 1$  adalah *base case*.



## Penjelasan Solusi Rekursif (lanj.)

### *Recurrence Relation*

- Bagaimana jika  $N > 1$ ?
- Seperti yang sudah didefinisikan,  $f_N = f_{N-1} + f_{N-2}$  untuk  $N > 1$
- Contoh:  $f_5 = f_4 + f_3$ .
- Mencari  $f_4$  dan  $f_3$  sendiri juga memunculkan permasalahan yang lebih kecil, yaitu:
  - $f_4 = f_3 + f_2$
  - $f_3 = f_2 + f_1$
- Hal ini akan terus diulang sampai tercapai *base case*, yaitu  $f_0$  atau  $f_1$ .
- Dengan ini, kita menemukan hubungan rekursif dari  $f_N$ .



## Contoh Solusi: fibonacci\_rekursi.cpp

Perhatikan contoh berikut:

---

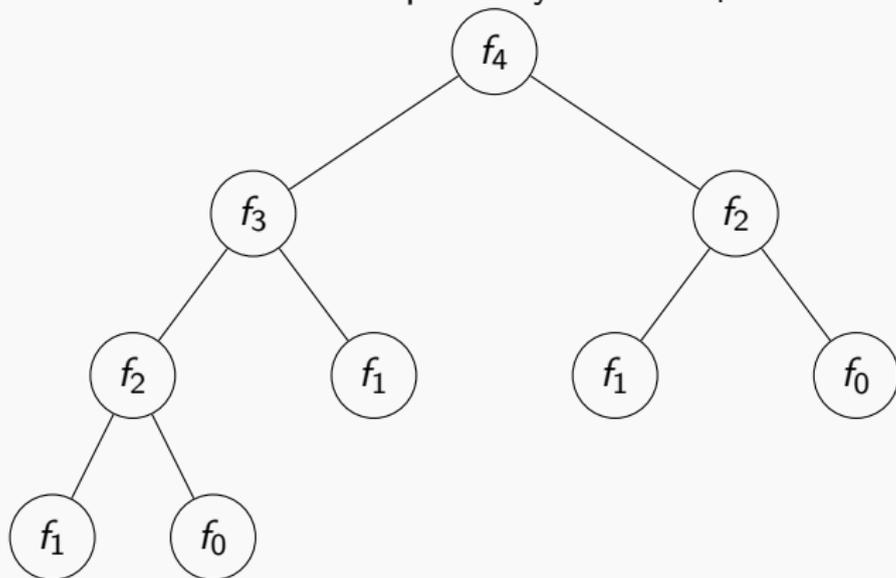
```
int fibonacci(int N) {  
    if (N <= 1) {  
        return N;  
    } else {  
        return fibonacci(N-1) + fibonacci(N-2);  
    }  
}
```

---



## Penjelasan Solusi Rekursif

Alur eksekusi rekursi dapat dimodelkan dengan pohon rekursi. Berikut adalah contoh pohonnya untuk  $f_4$ .



## Penjelasan Solusi Rekursif (lanj.)

Yang terjadi pada program ketika menghitung  $f_4$ :

- Panggil fibonacci(4).
- fibonacci(4) akan memeriksa, apakah  $N = 4$  adalah *base case*.
- Ternyata bukan, karena baru *base case* jika  $N \leq 1$ .
- Dijalankan "return fibonacci(3) + fibonacci(2)".
- Alur rekursi berjalan berurutan, sehingga fibonacci(3) dieksekusi lebih dulu.



## Penjelasan Solusi Rekursif (lanj.)

- fibonacci(3) akan menjalankan "return fibonacci(2) + fibonacci(1)".
- fibonacci(2) akan menjalankan "return fibonacci(1) + fibonacci(0)".
- Ternyata ketika fibonacci(1), 1 termasuk *base case* sehingga fibonacci(1) = 1.
- Ternyata ketika fibonacci(0), 0 termasuk *base case* sehingga fibonacci(0) = 0.
- Kembali ke fibonacci(2), nilai dari fibonacci(2) menjadi  $1 + 0 = 1$ .



## Penjelasan Solusi Rekursif (lanj.)

- Kembali ke fibonacci(3), nilai fibonacci(2) sudah ada nilainya tetapi fibonacci(1) belum sehingga akan dipanggil dan langsung menghasilkan 1 (karena sudah *base case*). Nilai dari fibonacci(3) menjadi  $1 + 1 = 2$ .
- Kembali ke fibonacci(4), nilai fibonacci(3) sudah ada nilainya tetapi fibonacci(2) belum sehingga akan dipanggil dan akan menghasilkan 1 (alur yang terjadi sama seperti sebelumnya). Nilai dari fibonacci(4) menjadi  $2 + 1 = 3$ .

Tantangan: Cobalah membuat pohon rekursi dan alurnya untuk menghitung  $f_5$ !



## Kompleksitas Solusi

- Perhatikan pohon rekursi yang sebelumnya. Berapa kali fungsi akan dipanggil?
- Setiap pemanggilan fungsi akan bercabang 2 dan kedalaman maksimalnya adalah  $N$ .
- Sebagai pendekatan, bisa dikatakan fungsi dipanggil  $2^N$  kali.
- Kompleksitasnya menjadi  $O(2^N)$ .



# Masalah

- Perhatikan kembali pohon rekursi yang sebelumnya. Terlihat  $f_2$  dihitung dua kali.
- Ketika  $f_N$  cukup besar, ada banyak fungsi dengan parameter yang sama namun dihitung berkali-kali. Hal ini berakibat program berjalan lambat.
- Kita dapat mereduksi kompleksitas rekursi Fibonacci menjadi  $O(N)$  dengan teknik yang akan kita pelajari pada pemrograman lanjut.
- Kita juga bisa membuat solusi  $O(N)$  dengan menghitung nilai Fibonacci secara iteratif. Dapatkah Anda membuatnya?



Bagian 2

## Permutasi



## Soal: Permutasi

Deskripsi:

- Pak Dengklek lupa password akun TLX-nya!
- Yang ia ingat hanyalah passwordnya terdiri dari  $N$  angka, dan mengandung masing-masing angka dari 1 sampai  $N$ .
- Misalnya  $N = 3$ , bisa jadi password Pak Dengklek adalah 123, 132, 312, dst
- Bantu Pak Dengklek menuliskan semua kemungkinan passwordnya!



## Soal: Permutasi (lanj.)

Format masukan:

- Sebuah baris berisi sebuah bilangan  $N$ .

Format keluaran:

- Beberapa baris yang merupakan semua kemungkinan password, satu pada setiap barisnya.
- Urutkan keluaran secara leksikografis (seperti pada kamus).

Batasan:

- $1 \leq N \leq 8$



## Soal: Permutasi (lanj.)

Contoh masukan:

3

---

Contoh keluaran:

123

132

213

231

312

321

---



## Solusi

- Sebelum merancang solusi untuk persoalan sebenarnya, mari kita sederhanakan persoalan.
- Misalkan digit-digit boleh berulang, sehingga untuk  $N = 3$ , keluarannya adalah:

---

111  
112  
113  
121  
122  
123  
131  
...  
333

---



## Solusi (lanj.)

- Jika  $N$  selalu 3, terdapat solusi iteratif yang sederhana:

```
for (int i = 1; i <= 3; i++) {  
    for (int j = 1; j <= 3; j++) {  
        for (int k = 1; k <= 3; k++) {  
            printf("%d%d%d\n", i, j, k);  
        }  
    }  
}
```

---



## Solusi (lanj.)

- Namun bagaimana jika  $N = 2$ ? Atau  $N = 4$ ?
- Kedalaman **for loop** tidak bisa diatur untuk memenuhi kebutuhan  $N$  yang beragam!
- Untuk itu, solusi rekursif lebih mudah digunakan untuk persoalan yang disederhanakan ini.



## Ide Rekursif

- Setiap kedalaman **loop** bisa diwujudkan oleh sebuah pemanggilan rekursif.

```
for 1..N
  for 1..N
    for 1..N
      ...
    end
  end
end
end
```

- Dengan menambahkan parameter "kedalaman" pada pemanggilan rekursif, kedalaman dari **loop** dapat diatur.



## Ide Rekursif (lanj.)

---

```
void tulis(int kedalaman) {  
    if (kedalaman >= N) {  
        // Cetak password  
        ...  
    } else {  
        // Masuk ke lapisan lebih dalam  
        for (int i = 1; i <= N; i++) {  
            tulis(kedalaman + 1);  
        }  
    }  
}
```

---



## Ide Rekursif (lanj.)

- Prosedur **tulis** dapat dipanggil dengan perintah:

```
N = 3;  
tulis(0);
```

- Nilai **kedalaman** akan terus bertambah selama kedalaman saat ini belum mencapai  $N$ .
- Hal ini menjadi memicu pemanggilan rekursif lebih dalam.
- Setelah **kedalaman** melebihi  $N$ , artinya tidak perlu lagi menambah "lapisan **loop**", sehingga dicapai *base case* dan dicetak password.



## Ide Rekursif (lanj.)

- Masalah berikutnya adalah bagaimana mencatat password yang sejauh ini telah dibentuk.
- Salah satu solusinya adalah membuat array global yang mencatat digit password dari 1 sampai **kedalaman**.
- Ketika *base case* tercapai, kita bisa mencetak isi array tersebut.
- Kita namakan array tersebut **catat**.



## Ide Rekursif (lanj.)

---

```
void tulis(int kedalaman) {
    if (kedalaman >= N) {
        // Cetak password
        for (int i = 0; i < N; i++) {
            printf("%d", catat[i]); // Cetak
        }
        printf("\n");
    } else {
        // Masuk ke lapisan lebih dalam
        for (int i = 1; i <= N; i++) {
            catat[kedalaman] = i; // Catat di sini
            tulis(kedalaman + 1);
        }
    }
}
```

---



# Solusi untuk Permutasi

- Kita berhasil menyelesaikan masalah yang disederhanakan, saatnya menarik solusi tersebut ke masalah sebenarnya.
- Perbedaan dari masalah yang baru kita selesaikan dengan yang sebenarnya adalah: tidak boleh ada digit yang berulang.
- Sebagai contoh, 122, 212, 311 bukan password yang benar, sementara 123, 213, dan 321 merupakan password yang benar.
- Artinya, jika kita bisa menghindari mencetak password dengan digit berulang, masalah selesai.



# Menghindari Digit Berulang

Solusi yang mungkin:

- Sebelum mencetak, periksa apakah ada digit yang berulang.
- Sebelum melakukan pemanggilan rekursif yang lebih dalam, periksa apakah ada digit berulang yang tercatat.



## Menghindari Digit Berulang (lanj.)

- Solusi pertama kurang cocok digunakan.
- Misalkan untuk  $N = 8$ , dan kedalaman saat ini adalah 2.
- Diketahui bahwa array **catat** sejauh ini berisi  $[1, 1, \dots]$ .
- Tidak ada gunanya untuk meneruskan pemanggilan rekursif lebih dalam, sebab kemungkinan ini sudah pasti tidak dicetak (ada digit '1' berulang).



## Menghindari Digit Berulang (lanj.)

- Menghindari perulangan digit sebelum pemanggilan rekursif lebih efisien untuk digunakan.
- Oleh karena itu kita akan menggunakan cara kedua.
- Hal ini dapat dilakukan dengan menandai digit-digit yang sudah pernah digunakan, dan jangan mencatat digit-digit tersebut.



## Menghindari Digit Berulang (lanj.)

- Kita akan menggunakan array global bertipe **boolean**, yaitu **pernah**.
- Awalnya, seluruh isi array **pernah** adalah **false**.
- **pernah[i]** bernilai **true** jika digit **i** berada di dalam array **catat**.
- Setiap sebelum masuk ke kedalaman rekursif berikutnya, periksa apakah digit yang akan digunakan sudah pernah digunakan.
- Jika belum pernah, baru boleh digunakan.



# Implementasi

---

```
void tulis(int kedalaman) {
    if (kedalaman >= N) {
        // Cetak password
        for (int i = 0; i < N; i++) {
            printf("%d", catat[i]); // Cetak
        }
        printf("\n");
    } else {
        // Masuk ke lapisan lebih dalam
        for (int i = 1; i <= N; i++) {
            if (!pernah[i]) { // i belum pernah?
                pernah[i] = true; // Gunakan
                catat[kedalaman] = i; // Catat di sini
                tulis(kedalaman + 1);
                pernah[i] = false; // Selesai menggunakan
            }
        }
    }
}
```

---



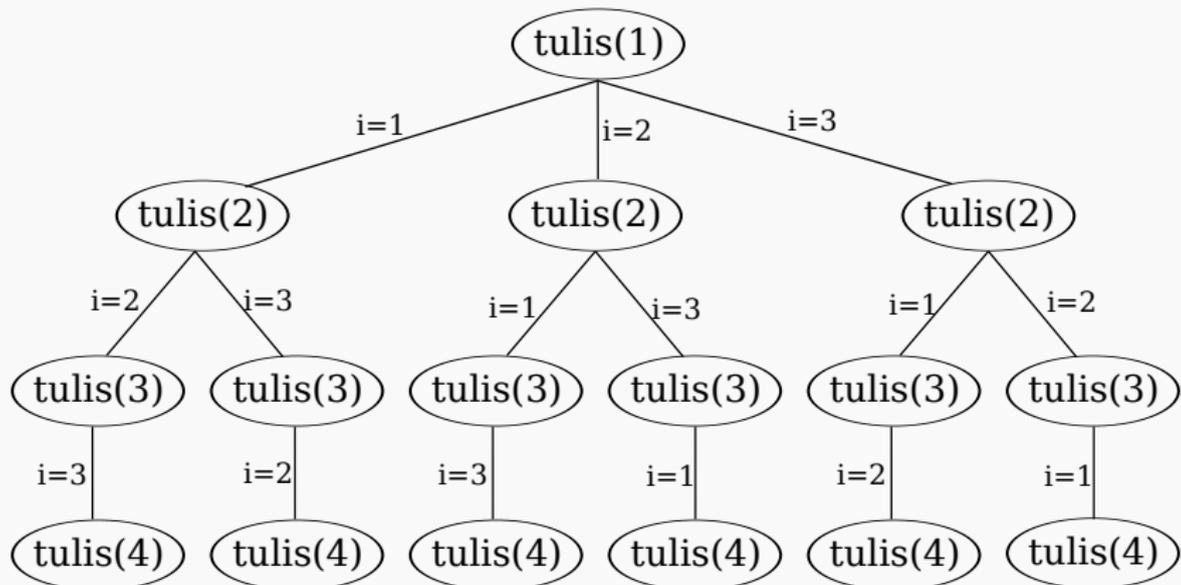
## Menghindari Digit Berulang (lanj.)

- Setelah perintah "tulis(kedalaman + 1)", nilai **pernah[i]** perlu dikembalikan menjadi **false**.
- Sebab setelah keluar dari pemanggilan rekursif tersebut, digit **i** dianggap tidak lagi ada pada **catat**.
- Namun digit **i** bisa saja digunakan untuk beberapa pemanggilan rekursif ke depannya.
- Dengan cara ini, kita memastikan tidak ada digit berulang yang dicetak.



## Kompleksitas

- Jika  $N = 3$ , maka berikut pohon rekursif yang menggambarkan pemilihan  $i$  untuk setiap pemanggilan:



# Kompleksitas

- Dapat diperhatikan bahwa pada kedalaman pertama, terdapat  $N$  cabang rekursif.
- Pada kedalaman kedua, terdapat  $N - 1$  cabang rekursif.
- Seterusnya hingga kedalaman terakhir yang tidak lagi bercabang.
- Kompleksitasnya adalah  $N \times (N - 1) \times (N - 2) \times \dots \times 1$ , atau dengan kata lain  $O(N!)$ .



# Penutup

- Rekursi merupakan topik yang luas untuk dibicarakan.
- Jika Anda belum terbiasa dengan berpikir rekursif, maka latihan yang banyak adalah solusinya.
- Selamat berlatih dengan soal-soal yang ada!

