


Non-Overlapping Time Intervals

Untuk Kebutuhan Penjadwalan

Disusun Oleh:

Achmad Baihaqie Wibowo 103012400026

Rafi Dzaki Azhari 103012400336

Jean Yudhistira Diva Waluyo 103012400113

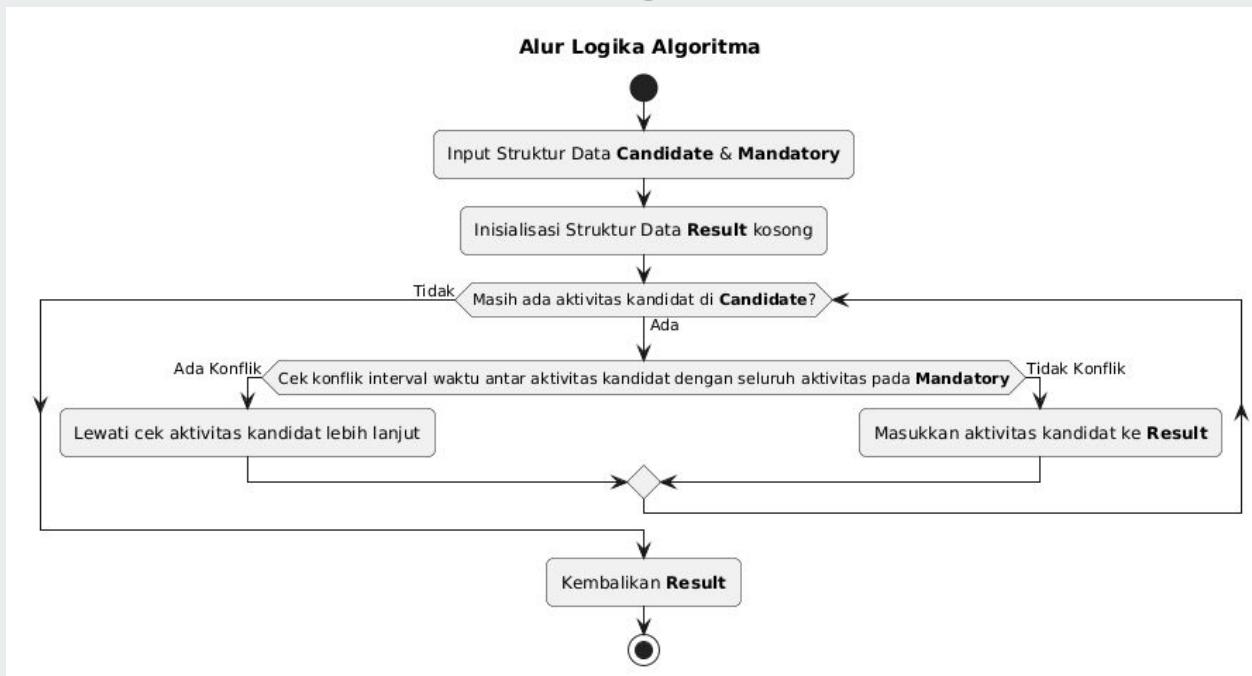
Latar Belakang

- Penjadwalan tentu melibatkan aktivitas-aktivitas berbasis waktu.
- Masalah utama yang muncul adalah aktivitas-aktivitas dalam suatu jadwal berpotensi konflik secara waktu (*time intervals overlapping*).
- Konflik waktu menyebabkan esensi dari jadwal tidak valid secara fungsional-nya.
- Konflik secara waktu = Antar interval waktu dari aktivitas-aktivitas mengalami tumpang tindih (*overlapping*).
- Urgensinya dibutuhkan semacam algoritma untuk mengatasi hal ini *in general case*.

Desain Permasalahan

- Terdapat sebuah jadwal mandatory yang sudah disusun sedemikian sehingga berisi sekumpulan aktivitas tanpa konflik internal secara interval waktu antar aktivitas-nya.
- Terdapat sebuah jadwal kandidat yang berisi sekumpulan aktivitas dan mungkin saja terkandung konflik internal secara interval waktu antar aktivitas-nya.
- Diperlukan sebuah algoritma yang dapat memfilterisasi aktivitas pada jadwal kandidat mana sajakah yang tidak akan menyebabkan tumpang tindih di jadwal mandatory jika suatu aktivitas tersebut ditambahkan ke dalam jadwal mandatory.

Desain Ide Algoritma



Algoritma dengan pendekatan Iteratif

Algorithm 2 Get Conflict Free Activities (Iterative)

```
function GET_CONFLICT_FREE_ACTIVITIES_ITERATIVE
(
    Candidates : array[1..maxC] of Activity,
    Mandatory : array[1..maxM] of Activity,
    nC, nM : integer
) → array of Activity
Kamus
    current : Activity
    startTime, endTime, nR, i : integer
    Result : array[1..maxC] of Activity
Algoritma
    Result ← emptyActivityArray
    nR ← 0
    i ← 1
    while i ≤ nC do
        current ← Candidates[i]
        startTime ← current.start.hour × 60 + current.start.minute
        endTime ← current.end.hour × 60 + current.end.minute
        if IS_TIME_CONFLICT_FREE_ITERATIVE(Mandatory, nM, current, startTime, endTime) then
            nR ← nR + 1
            Result[nR] ← current
        end if
        i ← i + 1
    end while
    return Result
end function
```

Algorithm 1 Check Time Conflict (Iterative)

```
function IS_TIME_CONFLICT_FREE_ITERATIVE
(
    Mandatory : array[1..maxM] of Activity,
    nM : integer,
    current : Activity,
    startTime : integer,
    endTime : integer
) → boolean
Kamus
    i, mandatoryStart, mandatoryEnd : integer
Algoritma
    i ← 1
    while i ≤ nM do
        mandatoryStart ← Mandatory[i].start.hour × 60 + Mandatory[i].start.minute
        mandatoryEnd ← Mandatory[i].end.hour × 60 + Mandatory[i].end.minute
        if current.day = Mandatory[i].day then
            if (startTime < mandatoryEnd and endTime > mandatoryStart) then
                return false
            end if
        end if
        i ← i + 1
    end while
    return true
end function
```

Algoritma dengan Pendekatan Rekursif

Algorithm 4 Get Conflict Free Activities (Recursive)

```
function GET_CONFLICT_FREE_ACTIVITIES_RECURSIVE
(
    Candidates, Result : array[1..maxC] of Activity,
    Mandatory : array[1..maxM] of Activity,
    i, nC, nR, nM : integer
) → array of Activity
Kamus
    current : Activity
    startTime, endTime : integer
Algoritma
    if  $i = 1$  then
        Result ← emptyActivityArray
    end if
    if  $i > nC$  then
        return Result
    end if
    current ← Candidates[i]
    startTime ← current.start.hour × 60 + current.start.minute
    endTime ← current.end.hour × 60 + current.end.minute
    if IS_TIME_CONFLICT_FREE_RECURSIVE(Mandatory, nM, current, startTime, endTime, 1) then
        nR ← nR + 1
        Result[nR] ← current
    end if
    return GET_CONFLICT_FREE_ACTIVITIES_RECURSIVE(
    (
        Candidates,
        Result,
        Mandatory,
        i + 1,
        nC,
        nR,
        nM
    )
)
end function
```

Algorithm 3 Check Time Conflict (Recursive)

```
function IS_TIME_CONFLICT_FREE_RECURSIVE
(
    Mandatory : array[1..maxM] of Activity,
    nM : integer,
    current : Activity,
    startTime : integer,
    endTime : integer,
    i : integer
) → boolean
Kamus
    mandatoryStart, mandatoryEnd : integer
Algoritma
    if  $i > nM$  then
        return true
    end if
    mandatoryStart ← Mandatory[i].start.hour × 60 + Mandatory[i].start.minute
    mandatoryEnd ← Mandatory[i].end.hour × 60 + Mandatory[i].end.minute
    if current.day = Mandatory[i].day then
        if (startTime < mandatoryEnd and endTime > mandatoryStart) then
            return false
        end if
    end if
    return IS_TIME_CONFLICT_FREE_RECURSIVE(Mandatory, nM, current, startTime, endTime, i + 1)
)
end function
```

Parameter Input Size dan Operasi Dasar

Input Size:

- $n_C \rightarrow$ jumlah aktivitas kandidat
- $n_M \rightarrow$ jumlah aktivitas mandatory

Operasi Dasar:

Terjadi di saat melakukan pengecekan konflik antar interval waktu yang dilakukan di dalam fungsi *helper*, yang direpresentasikan oleh operasi perbandingan:

$$(\text{startTime} < \text{mandatoryEnd}) \text{ AND } (\text{endTime} > \text{mandatoryStart})$$

Analisis Kondisi Kasus Terbaik, Terburuk, dan Rata-rata

Best Case

Terjadi ketika tiap aktivitas dari Jadwal Kandidat langsung mengalami konflik pada pemeriksaan pertama di fungsi *helper*. Dengan kata lain, perbandingan (`current.day == Mandatory[i].day`) dan perbandingan (`startTime < mandatoryEnd AND endTime > mandatoryStart`) selalu bernilai *true* pada pemeriksaan pertama di fungsi *helper* untuk tiap iterasi aktivitas kandidatnya.

Worst Case

Terjadi ketika tidak ada konflik waktu untuk seluruh aktivitas kandidat terhadap seluruh aktivitas jadwal wajib yang sudah ada. Dengan kata lain, untuk setiap aktivitas kandidat dilakukan pemeriksaan terhadap seluruh NM aktivitas. Atau mungkin bentrok ditemukan pada elemen terakhir jadwal wajib.

Average Case

Terjadi ketika sebagian aktivitas kandidat mengalami konflik dan sebagian tidak mengalami konflik. Dengan kata lain perbandingan (`startTime < mandatoryEnd AND endTime > mandatoryStart`) tidak selalu bernilai *true* atau *false*.

Perhitungan Kompleksitas Waktu Versi Iteratif

Kasus Terbaik

$$T_{min}(nC, nM) = \sum_{i=1}^{nC} 1$$

$$T_{min}(nC, nM) = (nC - 1 + 1) \times 1$$

$$T_{min}(nC, nM) = nC$$

$$T_{min}(nC, nM) \in \Theta(nC)$$

$\therefore T_{min}(nC, nM)$ berkelas linear

Kasus Terburuk

$$T_{max}(nC, nM) = \sum_{i=1}^{nC} \sum_{j=1}^{nM} 1$$

$$T_{max}(nC, nM) = \sum_{i=1}^{nC} (nM - 1 + 1) \times 1$$

$$T_{max}(nC, nM) = \sum_{i=1}^{nC} nM$$

$$T_{max}(nC, nM) = (nC - 1 + 1) \times nM$$

$$T_{max}(nC, nM) = nC \times nM$$

$$T_{max}(nC, nM) \in \Theta(nC \times nM)$$

$\therefore T_{max}(nC, nM)$ berkelas kuadratik



Perhitungan Kompleksitas Waktu Versi Iteratif

Kasus Rata-rata

$$T_{avg}(nC, nM) = \frac{\sum_{i=1}^{T_{max}(nC, nM)} i}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{1 + 2 + 3 + \dots + T_{max}(nC, nM)}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{\frac{T_{max}(nC, nM) \times [T_{max}(nC, nM) + 1]}{2}}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{T_{max}(nC, nM) + 1}{2}$$

$$T_{avg}(nC, nM) = \frac{nC \times nM + 1}{2} \in \Theta(nC \times nM)$$

$\therefore T_{avg}(nC, nM)$ berkelas kuadratik

Catatan!

Kondisi terburuk dan rata-rata berkelas $\Theta(nC^*nM)$, secara klasifikasi kelas kompleksitas waktu formal dapat digolongkan sebagai kuadratik jika dan hanya jika nC dan nM keduanya tumbuh secara proporsional. Jika nM konstan dengan nC bergerak maka kompleksitas tersebut tumbuh secara linear terhadap pertumbuhan nC dan Jika nC konstan maka kompleksitas tersebut tumbuh secara linear terhadap pertumbuhan nM , fenomena seperti ini dalam dunia matematika khususnya kajian kalkulus multivariat dapat digolongkan fungsi bilinear. Pada kelas algoritma kali ini yang sedang dikaji terlihat bahwa *input size* ada dua, hal ini tetap valid dibuktikan dengan kajian yang sudah umum di ranah analisis algoritma terkait *time complexity* untuk algoritma DFS dan BFS.

Perhitungan Kompleksitas Waktu Versi Rekursif

Kasus Terbaik

Relasi Rekurensi :

$$T_{\min}(nC, nM) = \begin{cases} 1, & nC = 1 \\ T_{\min}(nC - 1, nM) + 1, & nC > 1 \end{cases}$$

Perhitungan (Metode Substitusi):

$$T_{\min}(nC, nM) = T_{\min}(nC - 1, nM) + 1$$

$$T_{\min}(nC, nM) = T_{\min}(nC - 2, nM) + 2$$

$$T_{\min}(nC, nM) = T_{\min}(nC - 3, nM) + 3$$

⋮

$$T_{\min}(nC, nM) = T_{\min}(nC - k, nM) + k$$

Akan berhenti saat $nC - k = 1$

$$\Leftrightarrow k = nC - 1$$

sehingga :

$$T_{\min}(nC, nM) = T_{\min}(1, nM) + nC - 1$$

$$T_{\min}(nC, nM) = 1 + nC - 1$$

$$T_{\min}(nC, nM) = nC \in \Theta(nC)$$

Kasus Terburuk

Relasi Rekurensi :

$$T_{\max}(nC, nM) = \begin{cases} nM, & nC = 1 \\ T_{\max}(nC - 1, nM) + nM, & nC > 1 \end{cases}$$

Perhitungan :

$$T_{\max}(nC, nM) = T_{\max}(nC - 1, nM) + nM$$

$$T_{\max}(nC, nM) = T_{\max}(nC - 2, nM) + 2 \times nM$$

$$T_{\max}(nC, nM) = T_{\max}(nC - 3, nM) + 3 \times nM$$

⋮

$$T_{\max}(nC, nM) = T_{\max}(nC - k, nM) + k \times nM$$

Akan berhenti saat $nC - k = 1$

$$\Leftrightarrow k = nC - 1$$

sehingga :

$$T_{\max}(nC, nM) = T_{\max}(1, nM) + (nC - 1) \times nM$$

$$T_{\max}(nC, nM) = nM + nC \times nM - nM$$

$$T_{\max}(nC, nM) = nC \times nM \in \Theta(nC \times nM)$$



Perhitungan Kompleksitas Waktu Versi Rekursif

Kasus Rata-rata

$$T_{avg}(nC, nM) = \frac{\sum_{i=1}^{T_{max}(nC, nM)} i}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{1 + 2 + 3 + \dots + T_{max}(nC, nM)}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{\frac{T_{max}(nC, nM) \times [T_{max}(nC, nM) + 1]}{2}}{T_{max}(nC, nM)}$$

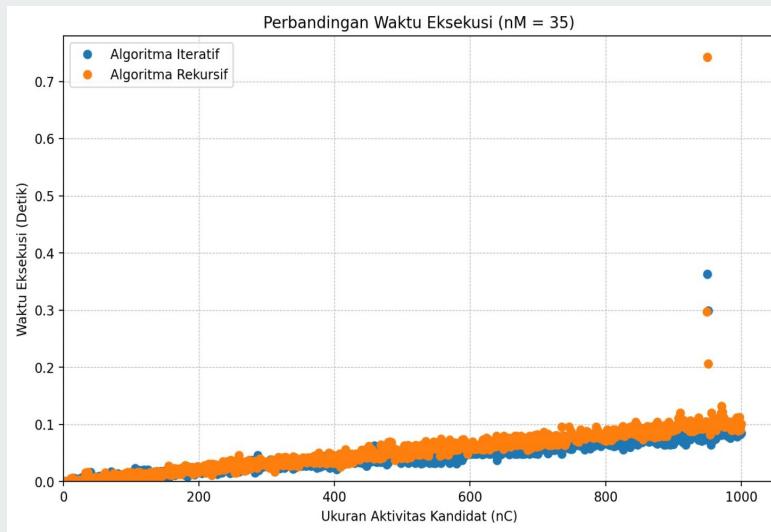
$$T_{avg}(nC, nM) = \frac{T_{max}(nC, nM) + 1}{2}$$

$$T_{avg}(nC, nM) = \frac{nC \times nM + 1}{2} \in \Theta(nC \times nM)$$

$\therefore T_{avg}(nC, nM)$ berkelas kuadratik

Perbandingan Iteratif vs Rekursif

Skenario nM Konstan



Pendekatan Iteratif

0.012482s

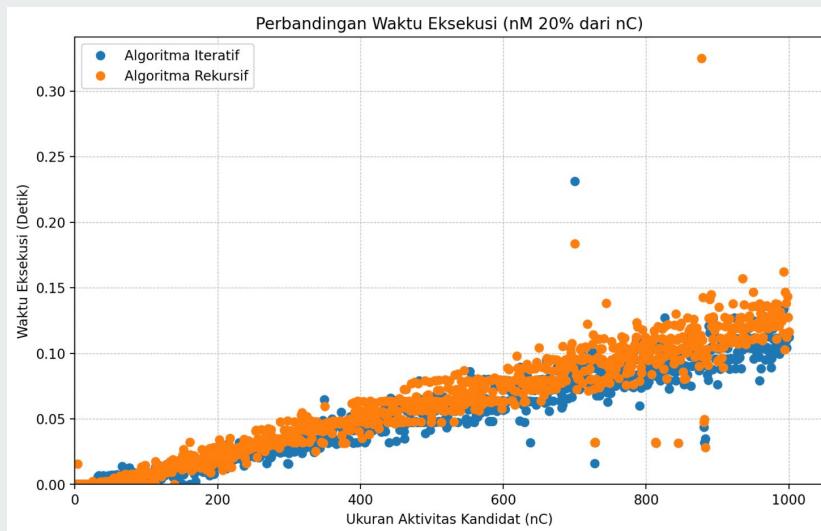
Pendekatan Rekursif

0.014937s

↑ 1.196603x lebih lambat

Perbandingan Iteratif vs Rekursif

Skenario nM dinamis ($nM = 20\% * nC$)



Pendekatan Iteratif

0.015179s

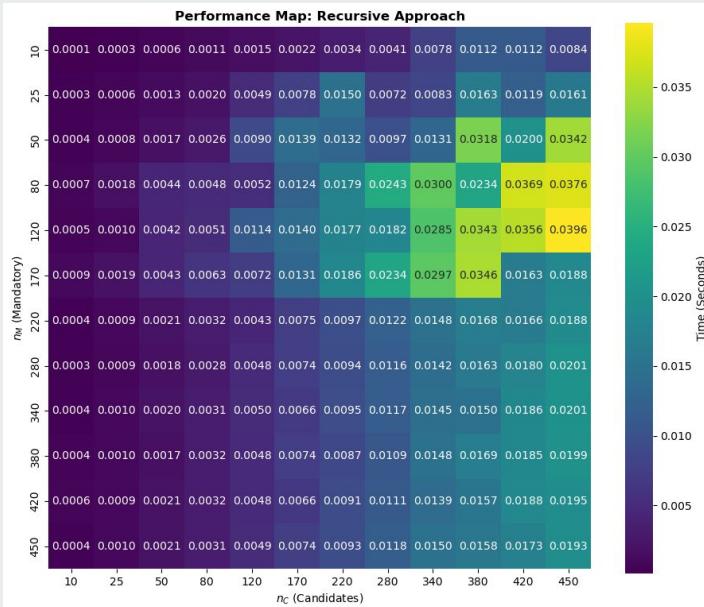
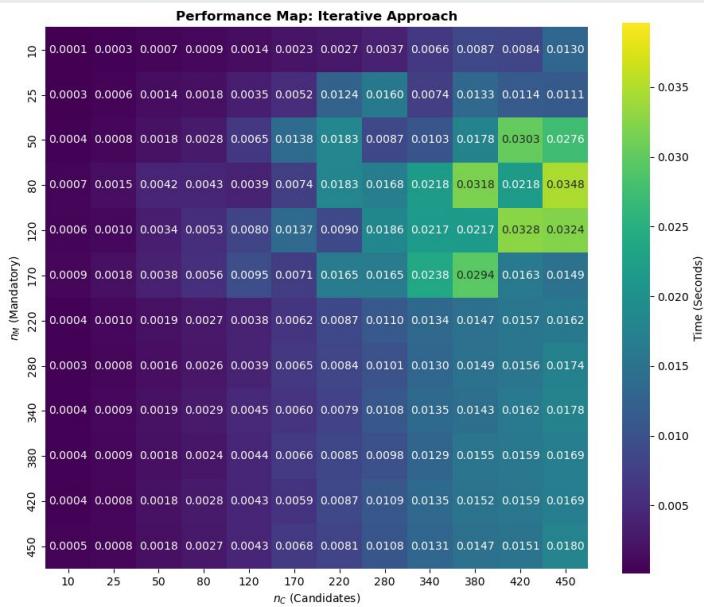
Pendekatan Rekursif

0.017808s

↑ 1.173182x lebih lambat

Perbandingan Iteratif vs Rekursif

Skenario Perspektif 3D dalam Format 2D Visualisasi Heatmap ... (01)



Perbandingan Iteratif vs Rekursif

Skenario Perspektif 3D dalam Format 2D Visualisasi Heatmap ... (02)

```
1 # raw analytic by us :)
2
3 # ratio > 1 == recursive is slower
4 ratio_matrix = results_recur / results_iter
5 diff_matrix = results_recur - results_iter
6
7
8 total_scenarios = results_iter.size
9 recursive_slower_count = np.sum(results_recur > results_iter)
10 avg_ratio = np.mean(ratio_matrix)
11 max_ratio = np.nanmax(ratio_matrix)
12 min_ratio = np.nanmin(ratio_matrix)
13
14 # output insights
15 print("*66")
16 print("HASIL ANALISIS KOMPARATIF: ITERATIF VS REKURSIF (Based on Heatmap)")
17 print("*66")
18
19 print(f"Total Skenario Pengujian: {total_scenarios} Skenario atau Sel")
20 print(f"Jumlah Sel Rekursif > Iteratif : {(recursive_slower_count)} dari {total_scenarios} sel")
21 print(f"Persentase Dominasi Efisiensi : {((recursive_slower_count/total_scenarios)*100:.1f}%)")
22 print("- * 66")
23 print(f"Rata-rata Faktor Overhead Rekursif : {avg_ratio:.2f}x lebih lambat")
24 print(f"Rentang Overhead : {min_ratio:.2f}x s.d. {max_ratio:.2f}x")
25 print("*66")
26
27 if avg_ratio > 1:
28     print("Pendekatan rekursif secara konsisten menunjukkan waktu eksekusi yang lebih tinggi.")
```

```
...
=====
HASIL ANALISIS KOMPARATIF: ITERATIF VS REKURSIF (Based on Heatmap)
=====
Total Skenario Pengujian: 144 Skenario atau Sel
Jumlah Sel Rekursif > Iteratif      : 125 dari 144 sel
Persentase Dominasi Efisiensi       : 86.8%
-----
Rata-rata Faktor Overhead Rekursif : 1.14x lebih lambat
Rentang Overhead                  : 0.45x s.d. 1.97x
=====
Pendekatan rekursif secara konsisten menunjukkan waktu eksekusi yang lebih tinggi.
```

Kesimpulan

- Secara kompleksitas waktu teoritis untuk pendekatan iteratif dan rekursif:
 - *Best Case:* $\Theta(nC)$
 - *Worst Case dan Average Case:* $\Theta(nC^*nM)$
- Pengujian Empiris (*Benchmarking* dan *Visualisasi Perbandingan*):
 - Efisiensi: Pendekatan Iteratif lebih unggul secara konsisten.
 - Stabilitas: Grafik waktu iteratif lebih stabil dan eksekusi lebih cepat.
 - Kelemahan Rekursif: Lebih lambat karena beban tambahan (*overhead*) pada manajemen memori *stack frame*.

Implementasi

Kunjungi tautan repository github di bawah ini:

<https://github.com/wibotron130823/conflict-free-scheduler-app-and-analysis>

Terima Kasih