Greedy

Tim Olimpiade Komputer Indonesia

Pendahuluan

Melalui dokumen ini, kalian akan:

- Memahami konsep *Greedy*.
- Menyelesaikan beberapa contoh persoalan *Greedy* sederhana.

Greedy

Greedy merupakan sebuah teknik dalam strategi penyelesaian masalah, bukan suatu algoritma khusus.

Konsep Greedy

Suatu persoalan dapat diselesaikan dengan teknik *Greedy* jika persoalan tersebut memiliki memiliki properti berikut:

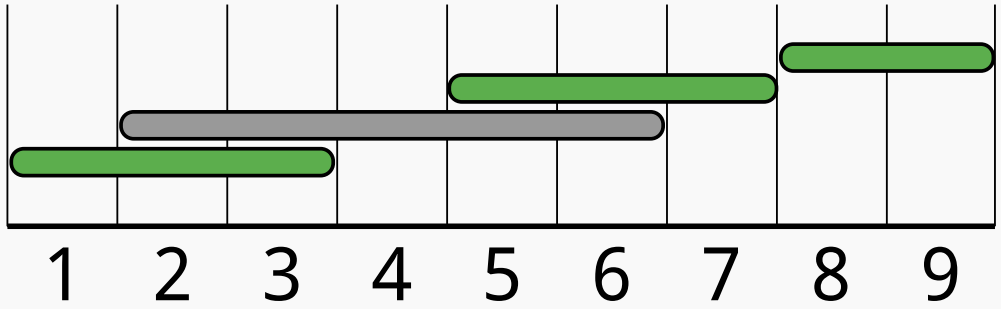
- Solusi optimal dari persoalan dapat ditentukan dari solusi optimal sub-persoalan tersebut.
- Pada setiap sub-persoalan, ada suatu langkah yang bisa dilakukan yang mana langkah tersebut menghasilkan solusi optimal pada sub-persoalan tersebut. Langkah ini disebut juga *Greedy Choice*.

Contoh Soal: Activity Selection

- Diberikan N buah aktivitas.
- Aktivitas ke-i dinyatakan dalam $< a_i.start, a_i.end >$.
- Artinya, aktivitas ini dimulai pada waktu $a_i.start$ dan berakhir pada waktu $a_i.end$.
- Pada setiap satuan waktu, Anda dapat mengikuti paling banyak satu aktivitas.
- Anda ingin mengatur jadwal sedemikian sehingga Anda bisa ikut aktivitas sebanyak mungkin.

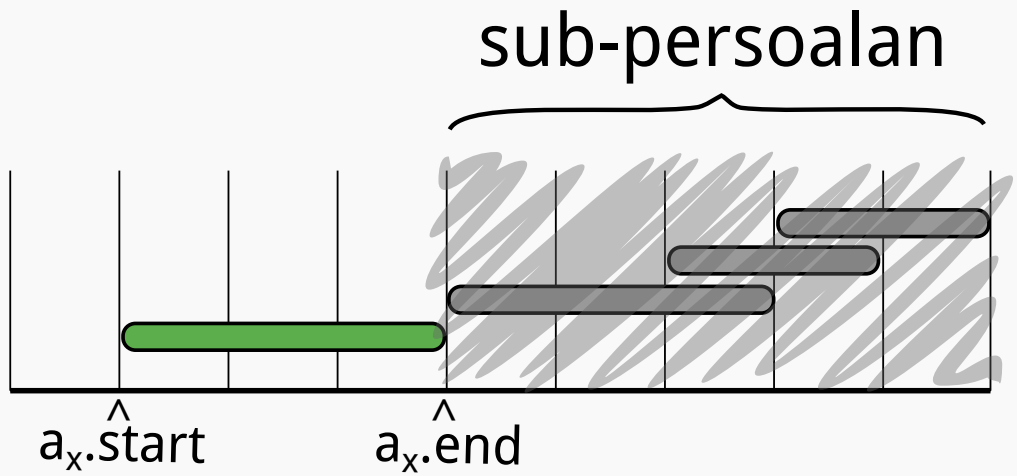
Contoh Activity Selection

- Sebagai contoh, diberikan 4 buah aktivitas: $[< 1, 3 >, < 2, 6 >, < 5, 7 >, < 8, 9 >]$.
- Anda dapat hadir di 3 aktivitas berbeda yang tidak saling tindih, yaitu $< 1, 3 >, < 5, 7 >, dan < 8, 9 >$.



Solusi Activity Selection

- Misalkan kegiatan pertama yang kita ikuti adalah kegiatan ke-x.
- Kegiatan selanjutnya yang diikuti haruslah memiliki waktu awal $\geq a_x.end$.
- Lebih jauh lagi, ternyata kita mendapat persoalan yang serupa, hanya saja ukurannya lebih kecil.
- Dengan kata lain, kita memperoleh sub-persoalan.



Solusi Activity Selection (lanj.)

Pertanyaan: aktivitas mana yg akan pertama kali dipilih?

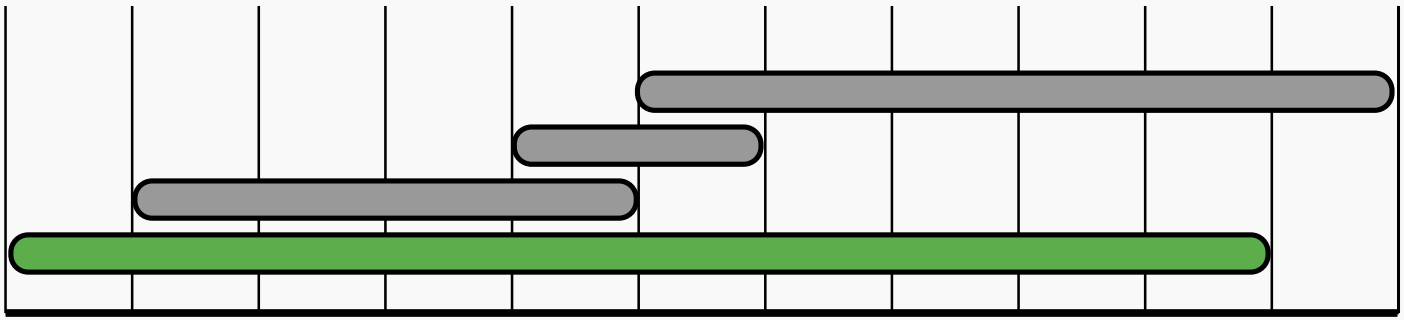
Perhatikan pilihan berikut:

- Memilih aktivitas dengan waktu mulai paling awal.
- Memilih aktivitas dengan durasi paling singkat.
- Memilih aktivitas dengan waktu akhir paling awal.

Memilih Aktivitas Pertama

Memilih aktivitas dengan waktu mulai paling awal:

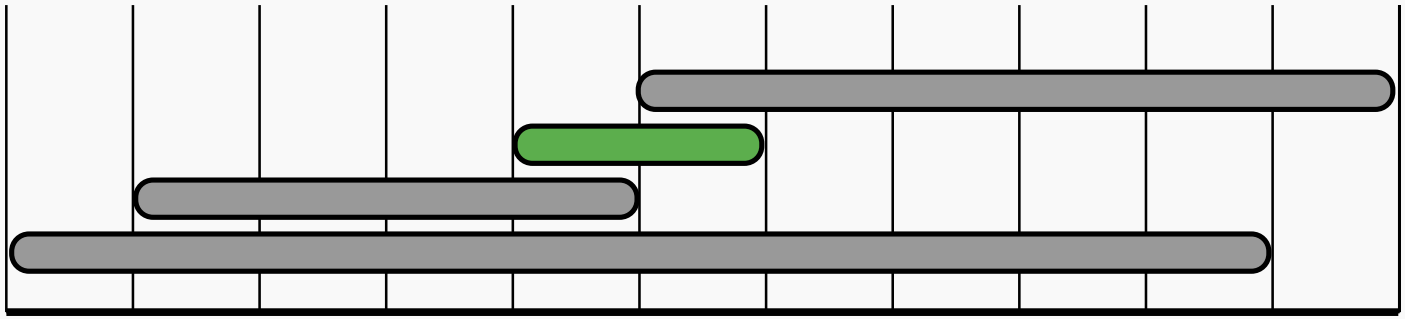
- Bisa jadi ada aktivitas yang mulai lebih awal, tetapi memiliki durasi yang sangat panjang sehingga menyita waktu.
- Memilih aktivitas yang mulai paling awal belum pasti optimal.



Memilih Aktivitas Pertama (lanj.)

Memilih aktivitas dengan durasi paling singkat:

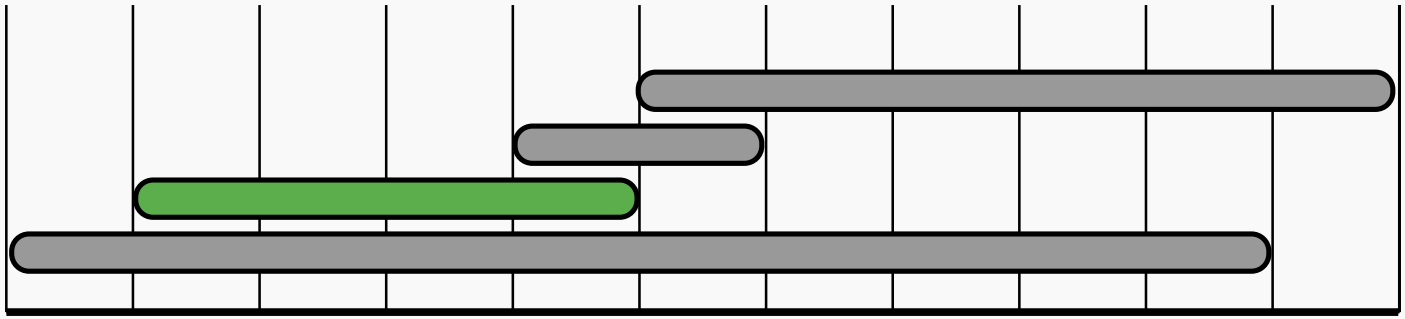
- Bisa jadi aktivitas dengan durasi paling singkat ini memotong dua aktivitas lain yang sebenarnya dapat kita ikuti.
- Pilihan ini juga belum pasti menghasilkan solusi optimal.



Memilih Aktivitas Pertama (lanj.)

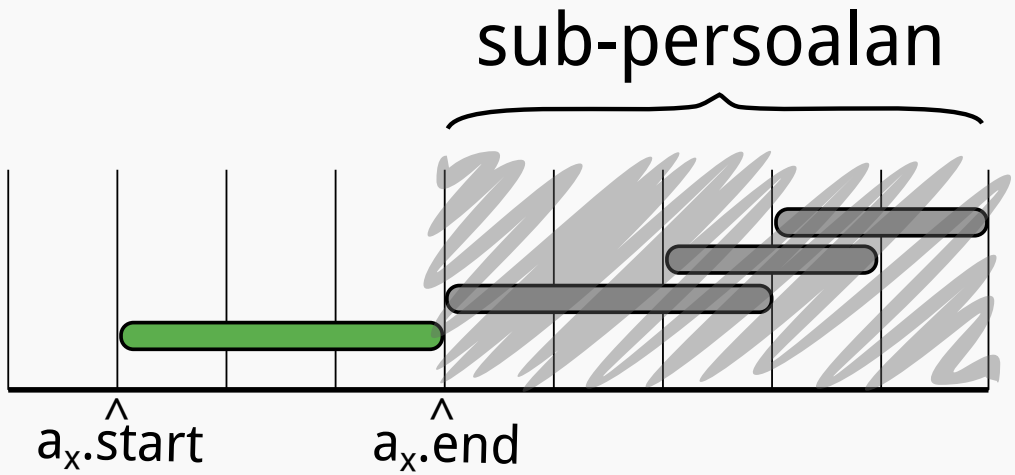
Memilih aktivitas dengan waktu akhir paling awal:

- Dengan memilih aktivitas yang selesai lebih awal, kita mempunyai sisa waktu lebih banyak untuk aktivitas lainnya.
- Tanpa peduli kapan aktivitas ini mulai atau berapa durasinya, memilih yang selesai lebih awal pasti menguntungkan.
- Pilihan ini adalah merupakan *Greedy Choice*, yang selalu menghasilkan solusi optimal.



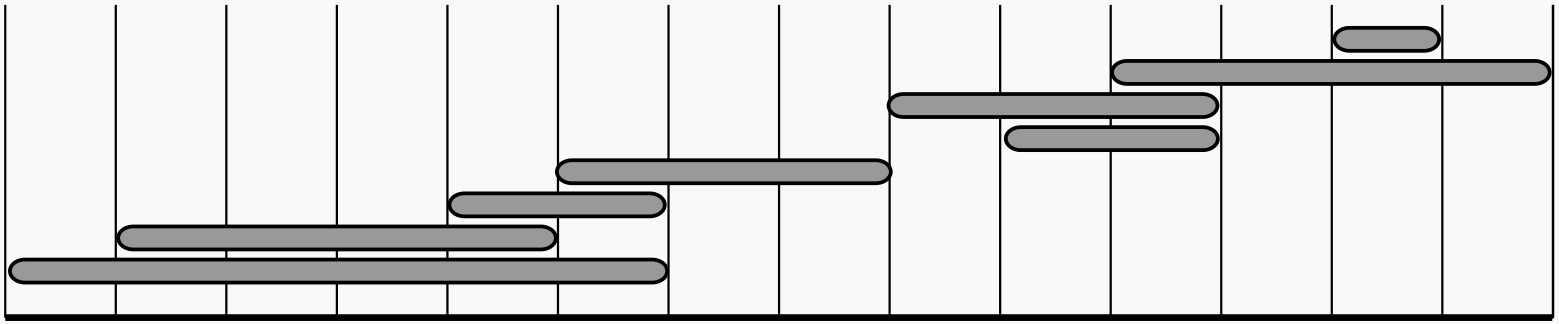
Penyelesaian Activity Selection

- Kini kita dapat menentukan aktivitas yang akan diikuti pertama kali.
- Selanjutnya kita mendapatkan sub-persoalan, yang ternyata dapat diselesaikan dengan cara serupa!



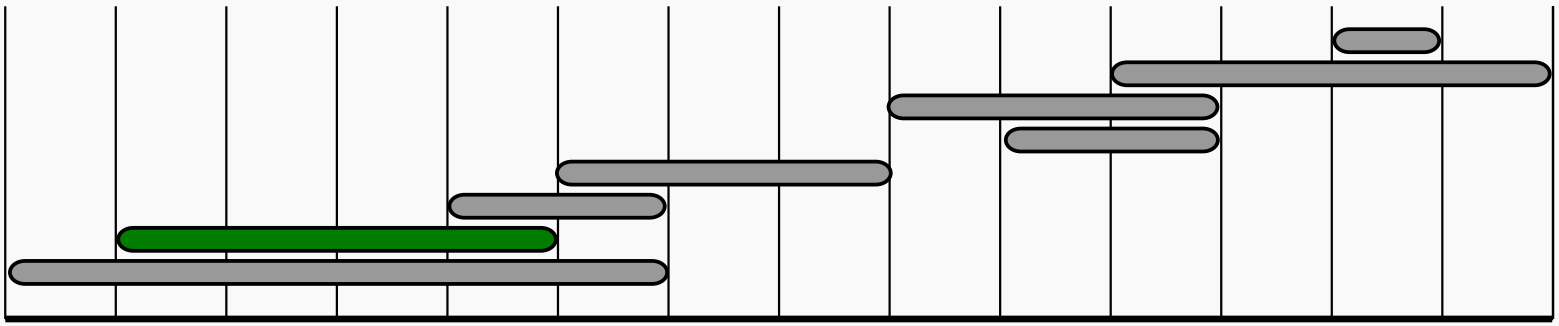
Contoh Eksekusi Activity Selection

Berikut contoh cara pemilihan aktivitas yang optimal.



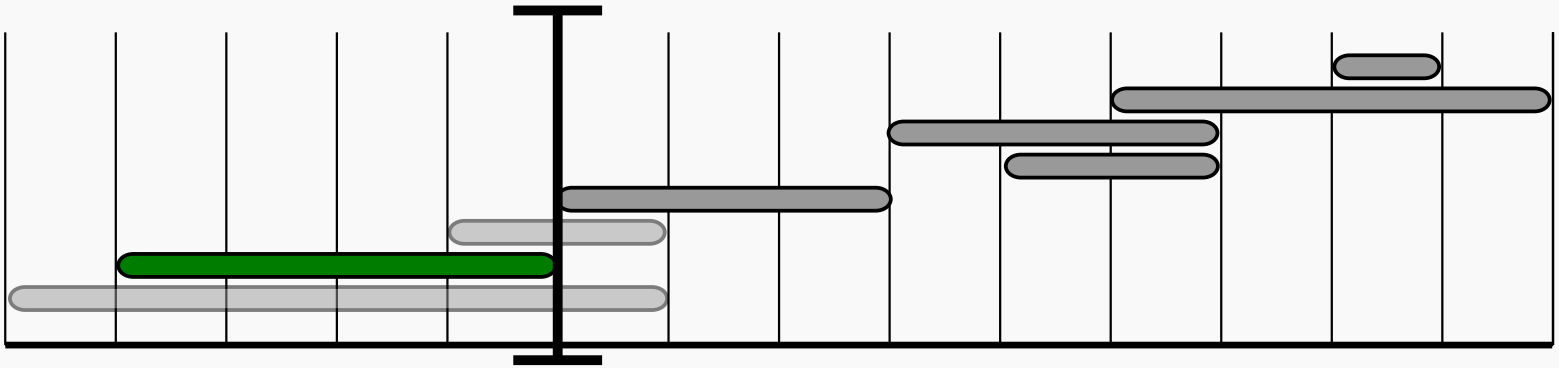
Contoh Eksekusi Activity Selection (lanj.)

Dimulai dari memilih aktivitas pertama.



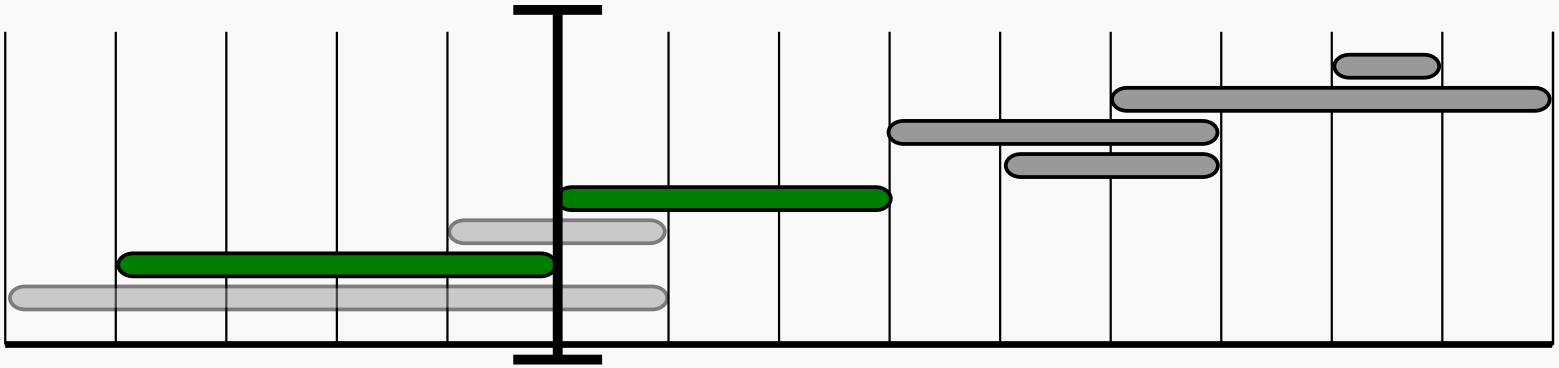
Contoh Eksekusi Activity Selection (lanj.)

Selanjutnya kita mendapatkan sub-persoalan.
Beberapa aktivitas kini tidak dapat dipilih lagi.



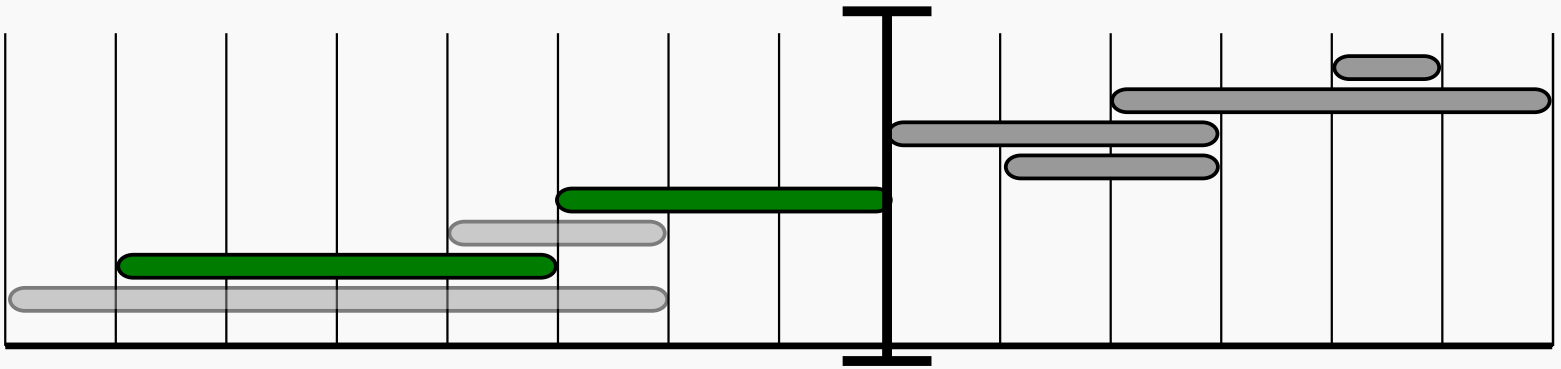
Contoh Eksekusi Activity Selection (lanj.)

Masalah yang kita hadapi serupa dengan masalah sebelumnya.
Kita tinggal memilih aktivitas yang berakhir paling awal.



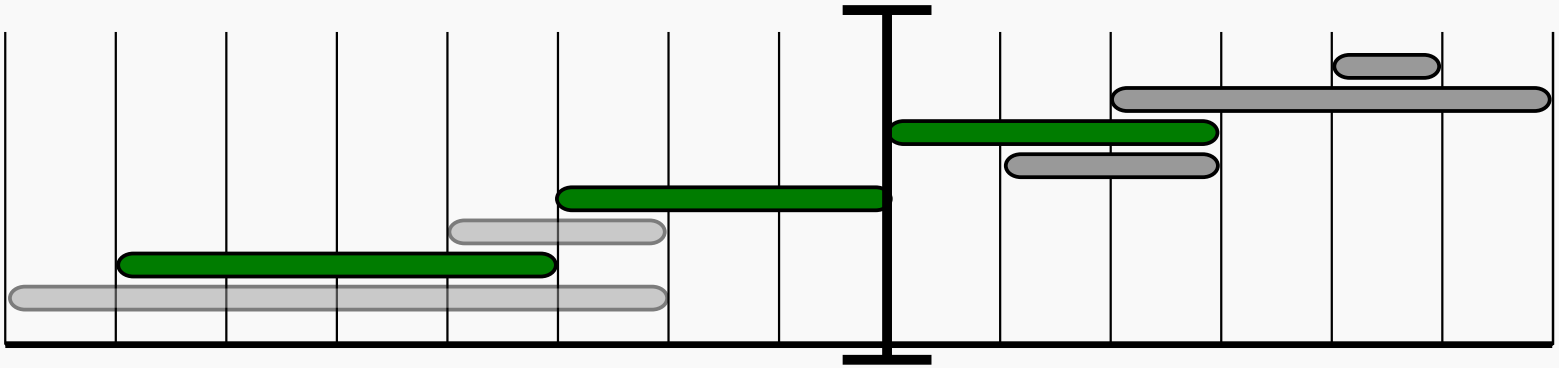
Contoh Eksekusi Activity Selection (lanj.)

Kembali kita mendapatkan sub-persoalan....



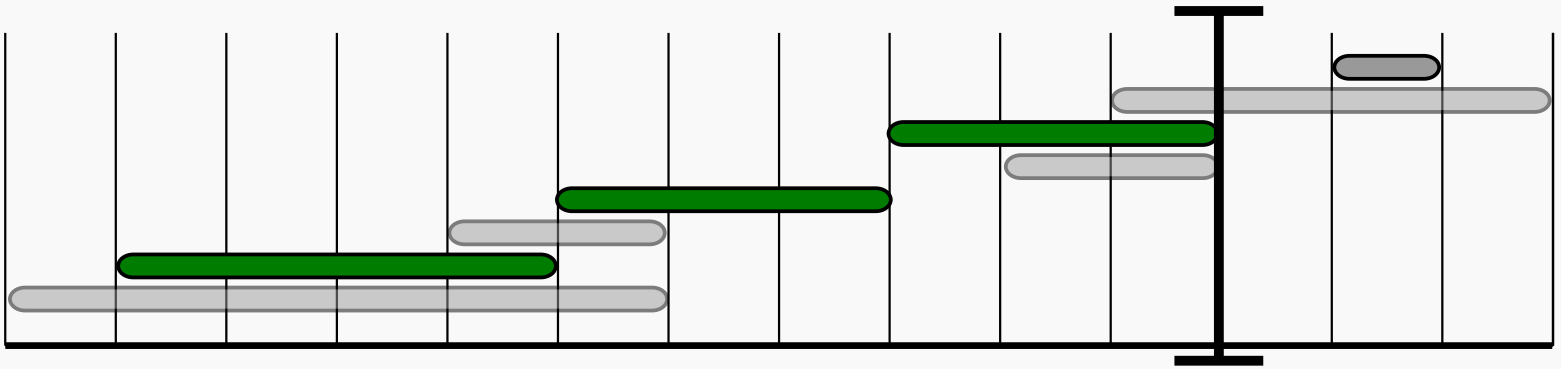
Contoh Eksekusi Activity Selection (lanj.)

Pilih lagi aktivitas yang berakhir paling awal.



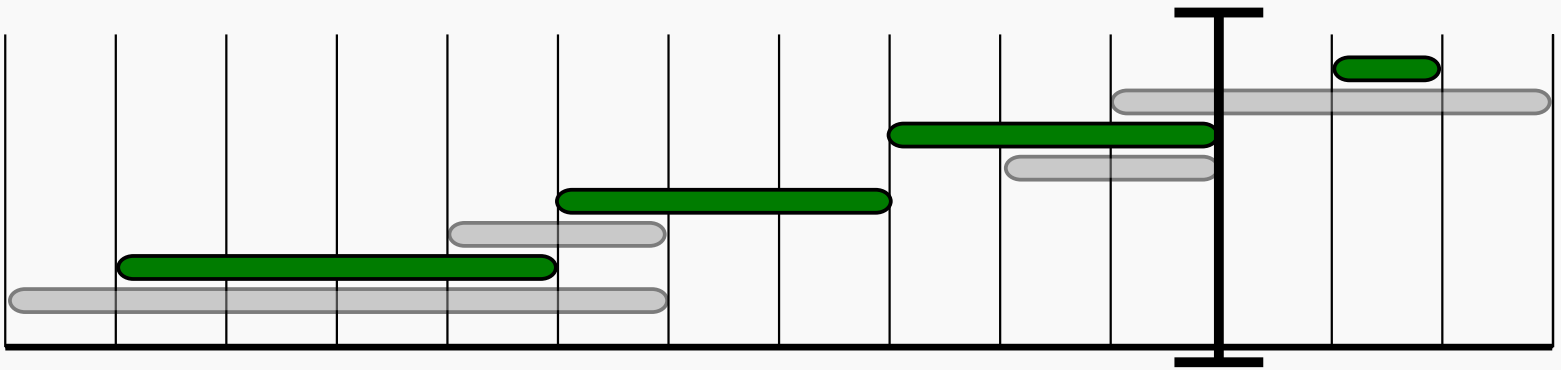
Contoh Eksekusi Activity Selection (lanj.)

Didapatkan lagi sub-persoalan....



Contoh Eksekusi Activity Selection (lanj.)

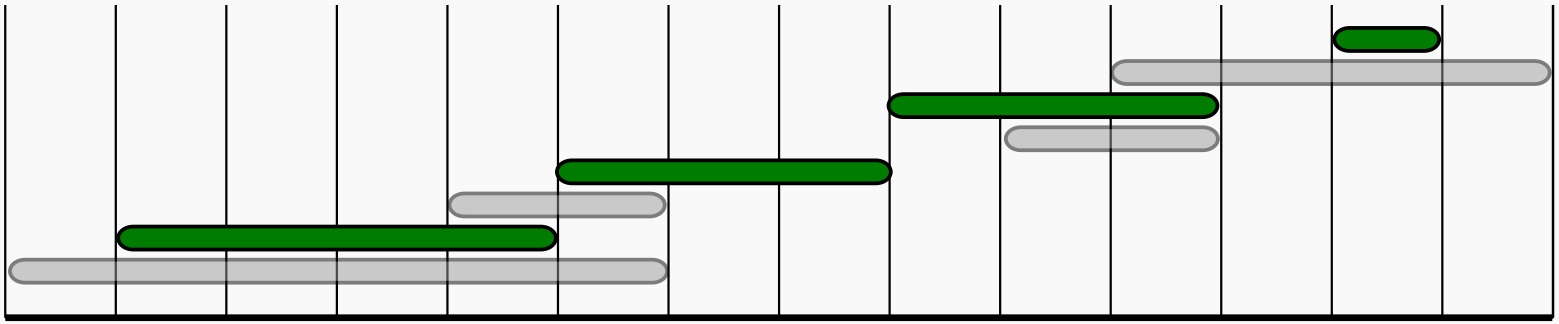
Pilih lagi aktivitas yang berakhir paling awal.



Contoh Eksekusi Activity Selection (lanj.)

Selesai!

Tidak ada cara lain yang memberikan hasil lebih optimal.



Implementasi Solusi Activity Selection

```
SOLVEACTIVITYSELECTION(a[], N)
1   // Urutkan a secara menaik berdasarkan a[i].end
2   SORTBYENDINGTIME(a, N)

3   selectedCount = 0
4   startTime = 1
5   for i = 1 to N
6       if (a[i].start >= startTime)
7           selectedCount = selectedCount + 1
8           startTime = a[i].end + 1
9   return selectedCount
```

Analisis Kompleksitas

- Mengurutkan aktivitas berdasarkan waktu berakhirnya dapat dilakukan dalam $O(N \log N)$, jika *Quicksort* digunakan.
- Setelah diurutkan, pemilihan aktivitas dapat dilakukan dalam $O(N)$.
- Kompleksitas akhirnya $O(N \log N)$.
- Cepat dan efisien!

Selingan

- *Greedy Choice* memungkinkan kita untuk memilih suatu keputusan yang dijamin akan menghasilkan solusi optimal, tanpa peduli ke depannya seperti apa.
- Hal ini memberi kesan "rakus", yaitu hanya mementingkan masalah yang sedang dihadapi dan selalu mengambil keputusan terbaik saat ini.
- Inilah sebabnya teknik ini dinamakan *Greedy*.

Permasalahan pada Algoritma Greedy

Perhatikan contoh soal berikut:

- Anda ingin menukar uang \$12 dengan pecahan koin \$5, \$2, dan \$1.
- Anda ingin menukar dengan jumlah koin sekecil mungkin.

Permasalahan pada Algoritma Greedy

- *Greedy Choice* yang terpikirkan adalah dengan memilih pecahan koin dengan nominal terbesar yang mungkin untuk tiap sub-persoalan.
- Pertama kita pilih pecahan $5, sehingga tersisa $7 lagi yang harus dipecah.
- Selanjutnya kita pilih $5 lagi dan menyisakan $2 untuk dipecah.
- Akhirnya, kita pilih $2 sebagai pecahan terakhir.
- Solusi dari kasus ini adalah dengan menggunakan 3 keping koin.

Permasalahan pada Algoritma Greedy (lanj.)

Dengan soal yang sama, bagaimana jika pecahan koin yang tersedia bernilai \$5, \$4, dan \$1?

Permasalahan pada Algoritma Greedy (lanj.)

- Dengan algoritma *Greedy*, kita akan menukar \$12 dengan pecahan \$5, \$5, \$1, dan \$1.
- Padahal ada solusi yang lebih baik, yaitu menggunakan 3 keping koin pecahan \$4.
- Pada kasus tersebut, *Greedy Choice* yang tidak selalu dapat menghasilkan solusi optimal.
- Permasalahan ini tidak dapat diselesaikan oleh algoritma *Greedy*.

Permasalahan pada Algoritma Greedy (lanj.)

- Pembuktian kebenaran algoritma *Greedy* tidaklah mudah.
- Biasanya akan ada beberapa pilihan *Greedy Choice* yang ada, yang mana tidak semuanya bisa menghasilkan solusi optimal.
- Ketika menemukan suatu *Greedy Choice*, sangat dianjurkan untuk menguji kebenaran dari pilihan tersebut sebelum diimplementasikan.

Permasalahan pada Algoritma Greedy (lanj.)

- Pengujian yang dapat dilakukan adalah dengan mencoba membuat contoh kasus yang dapat menggagalkan *Greedy Choice* tersebut.
- Teknik ini biasa disebut *proof by counter-example*.
- Jika ditemukan satu saja contoh kasus yang mana *Greedy Choice* yang diajukan tidak menghasilkan solusi optimal, maka *Greedy Choice* tersebut dinyatakan salah.

Saran

- Algoritma *Greedy* terkadang mudah untuk dipikirkan dan mudah untuk diimplementasikan, namun sulit untuk dibuktikan kebenarannya.
- Pembuktian kebenaran algoritma *Greedy* bisa jadi membutuhkan pembuktian matematis yang kompleks dan memakan waktu.
- Pada suasana kompetisi, intuisi dan pengalaman sangat membantu untuk menyelesaikan soal bertipe *Greedy*
- Berhati-hati dan telili saat mengerjakan soal bertipe *Greedy*. Perhatikan setiap detil yang ada, karena bisa berakibat fatal.

Penutup

- Untuk dapat menguasai *Greedy*, Anda perlu banyak berlatih dan berpikir secara cerdik.
- Selamat berlatih untuk mengasah "kerakusan" Anda :)