



# Dynamic Programming: Studi Kasus

Tim Olimpiade Komputer Indonesia

# Pendahuluan

Melalui dokumen ini, kalian akan:

- Menyelesaikan beberapa contoh persoalan DP sederhana.
- Membiasakan diri untuk "berpikir secara DP".



# Contoh Soal 1: Knapsack

- Diberikan  $N$  buah barang, dinomori dari 1 sampai  $N$ .
- Barang ke- $i$  memiliki harga  $v_i$  rupiah dan berat  $w_i$  gram.
- Kita memiliki tas yang berkapasitas  $G$  gram.
- Kita ingin memasukkan beberapa barang ke dalam tas, sedemikian sehingga jumlah berat dari barang-barang yang kita masukan tidak lebih dari kapasitas tas dan jumlah harganya sebanyak mungkin.



# Observasi

- Untuk setiap barang, kita harus memutuskan apakah barang ini diambil atau tidak.
- Jika diambil, kapasitas tas kita berkurang, dan harga barang yang kita dapatkan bertambah.
- Jika tidak diambil, tidak terjadi perubahan.



# Formulasi

- Definisikan sebuah fungsi  $dp(i, c)$  sebagai jumlah harga maksimum yang mungkin diperoleh, jika kita hanya mempunyai barang ke-1 sampai ke- $i$  dan sisa kapasitas tas kita adalah  $c$  gram.
- Untuk menghitung fungsi  $dp(i, c)$  kita bisa mencoba-coba apakah kita akan memasukkan barang ke- $i$  ke tas atau tidak.



## Formulasi Rekurens

- Jika kita memasukkan barang ke- $i$  ke tas, maka kita akan menyisakan barang ke-1 sampai ke- $(i - 1)$  dan sisa kapasitas tas menjadi  $c - w_i$ .
- Harga barang yang didapatkan pada kasus ini adalah  $dp(i - 1, c - w_i)$  ditambah dengan harga yang kita peroleh pada barang ke- $i$ .
- Dapat dituliskan  $dp(i, c) = dp(i - 1, c - w_i) + v_i$ .
- Kasus ini hanya boleh dipertimbangkan jika  $c \geq w_i$ .



## Formulasi Rekurens (lanj.)

- Jika kita tidak memasukkan barang ke- $i$  ke tas, maka kita akan menyisakan barang ke-1 sampai ke- $(i - 1)$  dan sisa kapasitas tas masih tetap  $c$ .
- Harga barang didapatkan pada kasus ini adalah  $dp(i - 1, c)$ , tanpa tambahan apapun (kita tidak mengambil barang ke- $i$ ).
- Dapat dituliskan  $dp(i, c) = dp(i - 1, c)$ .



## Formulasi Rekurens (lanj.)

- Dari kedua pilihan keputusan tersebut, kita tertarik dengan yang menghasilkan nilai terbesar.
- Cukup bandingkan mana yang lebih besar, antara:
  - $dp(i - 1, c - w_i) + v_i$ , atau
  - $dp(i - 1, c)$
- Dapat dituliskan:
$$dp(i, c) = \max(dp(i - 1, c - w_i) + v_i, dp(i - 1, c)).$$
- Kembali ditekankan bahwa pilihan memasukkan barang ke- $i$  hanya boleh dipertimbangkan jika  $c \geq w_i$ .





# Formulasi Base Case

- Jika  $i = 0$ , maka berarti tidak ada lagi barang yang tersedia.
- Ini berarti  $dp(i, c) = 0$ .
- Kasus ini menjadi *base case*.



## Formulasi Akhir

$dp(i, c)$  dapat dirumuskan sebagai berikut:

$$dp(i, c) = \begin{cases} 0, & i = 0 \\ dp(i - 1, c), & i > 0 \wedge c < w_i \\ \max(dp(i - 1, c - w_i) + v_i, dp(i - 1, c)), & i > 0 \wedge c \geq w_i \end{cases}$$



# Analisis Kompleksitas

- Terdapat  $O(N)$  nilai berbeda untuk nilai  $i$  dan  $O(G)$  nilai berbeda untuk nilai  $c$  pada  $dp(i, c)$ .
- Dibutuhkan  $O(1)$  untuk menghitung  $dp(i, c)$ .
- Sehingga untuk menghitung seluruh nilai  $dp(i, c)$  untuk seluruh  $i$  dan  $c$  dibutuhkan waktu  $O(NG)$ .



## Solusi Top-Down

Kita implementasikan  $dp(i, c)$  sebagai fungsi  $SOLVE(i, c)$ :

$SOLVE(i, c)$

```
1  if ( $i == 0$ )  
2      return 0  
3  if  $computed[i][c]$   
4      return  $memo[i][c]$   
5   $best = SOLVE(i - 1, c)$   
6  if ( $c \geq w[i]$ )  
7       $best = \max(best, SOLVE(i - 1, c - w[i]) + v[i])$   
8   $computed[i][c] = true$   
9   $memo[i][c] = best$   
10 return  $best$ 
```

Jawaban akhirnya ada pada  $SOLVE(N, G)$ .



## Solusi Bottom-Up

SOLVE()

```
1  // Base case
2  for  $c = 0$  to  $G$ 
3       $dp[0][c] = 0$ 

4  // Isi "tabel" dari kasus yang kecil ke besar
5  for  $i = 1$  to  $N$ 
6      for  $c = 0$  to  $G$ 
7           $best = dp[i - 1][c]$ 
8          if ( $c \geq w[i]$ )
9               $best = \max(best, dp[i - 1][c - w[i]] + v[i])$ 
10          $dp[i][c] = best$ 

11 return  $dp[N][G]$ 
```



## Contoh Soal 2: Memotong Kayu

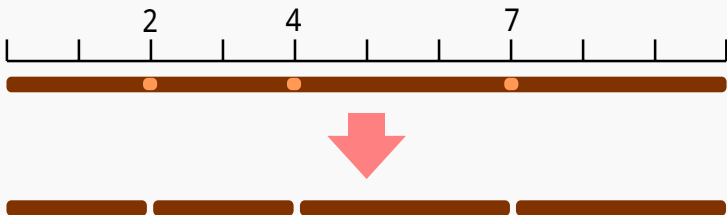
Diadopsi dari UVa 10003 - Cutting Sticks

- Kita akan memotong sebuah batang kayu dengan panjang  $M$  meter pada  $N$  titik menjadi  $N + 1$  bagian.
- Titik ke- $i$  berada di  $L_i$  meter dari ujung kiri, dengan  $1 \leq i \leq N$ .
- Untuk memotong sebatang kayu menjadi dua, kita memerlukan usaha sebesar panjang kayu yang sedang kita potong.
- Cari urutan pemotongan sedemikian sehingga total usaha yang dibutuhkan minimum!



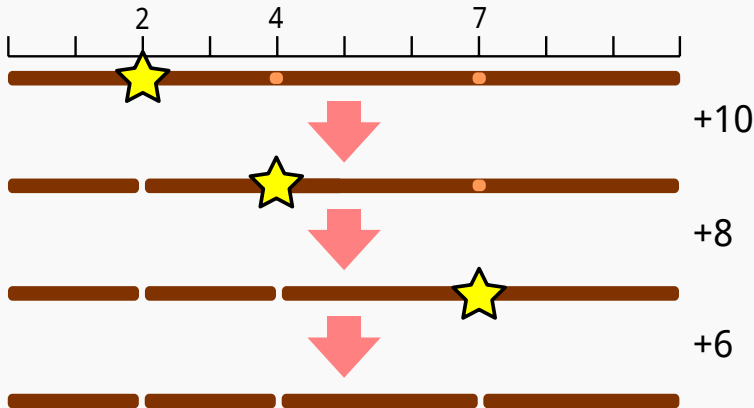
## Contoh Soal 2: Memotong Kayu (lanj.)

Sebagai contoh, terdapat sebuah kayu dengan panjang 10 meter dan terdapat 3 titik potongan pada 2 meter, 4 meter, dan 7 meter dari ujung kiri.



## Contoh Soal 2: Memotong Kayu (lanj.)

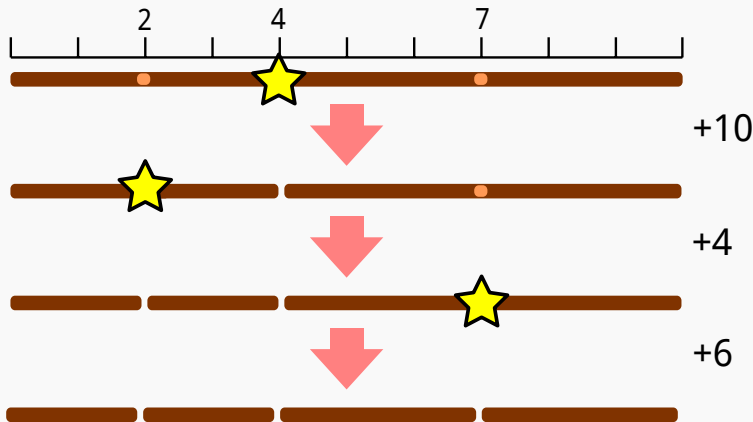
Kita bisa memotong pada titik 2, titik 4, lalu titik 7 dan memerlukan usaha  $10 + 8 + 6 = 24$ .





## Contoh Soal 2: Memotong Kayu (lanj.)

Cara lain adalah memotong pada titik 4, titik 2, lalu titik 7 dan memerlukan usaha  $10 + 4 + 6 = 20$ .



## Solusi Greedy?

- Apakah strategi *greedy* dengan memotong "setengah-tengahnya" selalu menghasilkan solusi optimal?
- Bagaimana dengan kasus jika  $M = 2000$  dan  $L = [1, 2, 3, 4, 5, 1000]$ ?
- Kita akan coba menggunakan DP untuk persoalan ini.



# Observasi

- Untuk pemotongan pertama, terdapat  $N$  pilihan lokasi pemotongan.
- Jika kita memotong di posisi  $L_m$ , maka didapatkan dua batang.
- Batang pertama perlu dipotong di titik  $L_1, L_2, \dots, L_{m-1}$  dan batang kedua di  $L_{m+1}, L_{m+2}, \dots, L_N$ .
- Ternyata kita mendapatkan sub-persoalan yang serupa.
- Pemotongan bisa dilanjutkan secara rekursif, dan kita pilih posisi pemotongan yang ke depannya membutuhkan usaha terkecil.

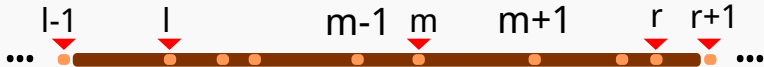


## Formulasi Rekurens

- Definisikan sebuah fungsi  $dp(l, r)$  sebagai jumlah usaha minimum yang mungkin diperoleh, jika kita hanya perlu memotong di  $L_l, L_{l+1}, \dots, L_r$ .
- Untuk menghitung  $dp(l, r)$  kita dapat mencoba titik mana yang kita potong pertama kali.
- Jika kita memotong di  $L_m$  ( $l \leq m \leq r$ ), maka kita akan mendapatkan dua potongan.



## Formulasi Rekurens (lanj.)



- Total usaha yang dibutuhkan jika kita melakukan pemotongan di  $L_m$  adalah jumlah dari:
  - Total usaha minimum dari potongan pertama, yaitu  $dp(l, m - 1)$ .
  - Total usaha minimum dari potongan kedua, yaitu  $dp(m + 1, r)$ .
  - Usaha untuk pemotongan ini, yaitu  $L_{r+1} - L_{l-1}$ .
- Untuk mempermudah, asumsikan  $L_0 = 0$  dan  $L_{N+1} = M$ .



# Formulasi Base Case

- Ketika  $l > r$ , artinya sudah tidak ada pemotongan yang perlu dilakukan.
- Dengan demikian, usaha yang dibutuhkan adalah 0, atau  $dp(l, r) = 0$ .



## Formulasi Akhir

Dapat dirumuskan:

$$dp(l, r) = \begin{cases} 0, & l > r \\ \min_{l \leq m \leq r} dp(l, m-1) + dp(m+1, r) + (L_{r+1} - L_{l-1}), & l \leq r \end{cases}$$



# Analisis Kompleksitas

- Terdapat  $O(N)$  nilai berbeda untuk nilai  $l$  dan  $O(N)$  nilai berbeda untuk nilai  $r$  pada  $dp(l, r)$ .
- Dibutuhkan iterasi sebanyak  $O(N)$  untuk menghitung  $dp(l, r)$ .
- Sehingga untuk menghitung seluruh nilai  $dp(l, r)$  untuk seluruh  $l$  dan  $r$  dibutuhkan waktu  $O(N^3)$ .





## Solusi Top-Down

Kita implementasikan  $dp(l, r)$  sebagai fungsi  $SOLVE(l, r)$ :

$SOLVE(l, r)$

```
1  if ( $l > r$ )
2      return 0
3  if  $computed[l][r]$ 
4      return  $memo[l][r]$ 
5
6   $best = \infty$ 
7   $cost = L[r + 1] - L[l - 1]$ 
8  for  $m = l$  to  $r$ 
9       $best = \min(best, SOLVE(l, m - 1) + SOLVE(m + 1, r) + cost)$ 
10  $computed[l][r] = true$ 
11  $memo[l][r] = best$ 
12 return  $best$ 
```

Jawaban akhirnya ada pada  $SOLVE(1, N)$ .



## Solusi Bottom Up

SOLVE()

```
1 // Base case
2 for  $l = 0$  to  $N + 1$ 
3     for  $r = 0$  to  $l - 1$ 
4          $dp[l][r] = 0$ 

5 // Isi "tabel" mulai dari kasus yang kecil
6 for  $gap = 0$  to  $N$ 
7     for  $l = 1$  to  $N - gap$ 
8          $r = l + gap$ 
9          $best = \infty$ 
10         $cost = L[r + 1] - L[l - 1]$ 
11        for  $m = l$  to  $r$ 
12             $best = \min(best, dp[l][m - 1] + dp[m + 1][r] + cost)$ 
13         $dp[l][r] = best$ 

14 return  $dp[1][N]$ 
```



## Pengisian "Tabel" DP

- Perhatikan bahwa pada metode *bottom-up*, pengisian "tabel" dilakukan secara "tidak biasa".
- Kita perlu mengisi mulai dari:
  - $dp[1][1], dp[2][2], \dots, dp[N][N]$ ,
  - lalu  $dp[1][2], dp[2][3], \dots, dp[N-1][N]$ ,
  - lalu  $dp[1][3], dp[2][4], \dots, dp[N-2][N]$ ,
  - lalu  $dp[1][4], dp[2][5], \dots, dp[N-3][N]$ ,
  - dan seterusnya sampai  $dp[1][N]$ .
- Ingat bahwa pengisian "tabel" harus dilakukan dari kasus yang kecil ke besar.
- Definisi "kasus kecil" pada masalah ini adalah kayu dengan titik-titik pemotongan yang lebih sedikit.



## Pengisian "Tabel" DP (lanj.)

- Dari contoh ini kita mempelajari bahwa urutan pengisian "tabel" pada DP *bottom-up* tidak selalu biasa.
- Jika urutan pengisiannya salah, maka hasil akhir yang didapatkan juga bisa jadi salah.
- Hal ini terjadi ketika kita hendak menyelesaikan kasus yang besar, tetapi hasil untuk kasus-kasus yang lebih kecil belum tersedia.
- Untuk mengetahui urutan pengisian "tabel", Anda perlu mengamati apa definisi "kasus kecil" pada masalah yang dihadapi.



# Penutup

- DP merupakan topik yang cukup luas untuk dibicarakan.
- Banyak berlatih mengerjakan soal DP dapat melatih kita untuk mendapatkan rumus DP yang sesuai dengan masalah yang dihadapi.
- Topik optimisasi lainnya pada DP akan dibahas pada kesempatan yang lain.

