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

Formulasi Rekurens

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

Formulasi Rekurens (lanj.)

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

Formulasi Rekurens (lanj.)

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

Formulasi Base Case

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

Formulasi Akhir

$g(x, y)$ dapat dirumuskan sebagai berikut:

$$g(x,y) = \begin{cases} 0, & x = 0 \\ g(x-1, y), & x > 0 \wedge y < w_x \\ \max(g(x-1, y-w_x) + v_x, g(x-1, y)), & x > 0 \wedge y \geq w_x \end{cases}$$

Analisis Kompleksitas

- Terdapat $O(N)$ nilai berbeda untuk nilai x dan $O(G)$ nilai berbeda untuk nilai y pada $g(x, y)$.
- Dibutuhkan $O(1)$ untuk menghitung $g(x, y)$.
- Sehingga untuk menghitung seluruh nilai $g(x, y)$ untuk seluruh x dan y dibutuhkan waktu $O(NG)$.

Solusi Top Down

Kita implementasikan $g(x, y)$ sebagai fungsi $\textsc{solve}(x, y)$:

$\textsc{solve}(x, y)$

if $(x == 0)$
  return 0
elseif $\mathit{hasComputed}[x][y]$
  return $\mathit{memo}[x][y]$
else
  $\mathit{best} = \textsc{solve}(x - 1, y)$
  if $(y \geq w[x])$
    $\mathit{best} = \max(\mathit{best}, \textsc{solve}(x - 1, y - w[x]) + v[x])$
  $\mathit{hasComputed}[x][y] = \mathit{true}$
  $\mathit{memo}[x][y] = \mathit{best}$
  return best

Jawaban ada pada $\textsc{solve}(N, G)$

Solusi Bottom Up

SOLVE()

for $y = 0$ **to** G
    $g[0][y] = 0$ // Base case...

// Isi "tabel" dari kasus yang kecil ke besar
for $x = 1$ **to** N
    for $y = 0$ **to** G
        $best = g[x-1][y]$
        if $(y \geq w[x])$
            $best = \max(best, g[x-1][y-w[x]] + v[x])$
        $g[x][y] = best$
return $g[N][G]$

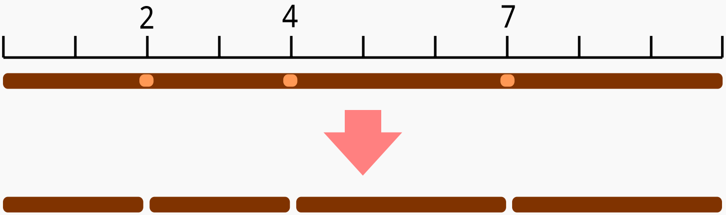
Contoh Soal 2: Memotong Kayu

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!

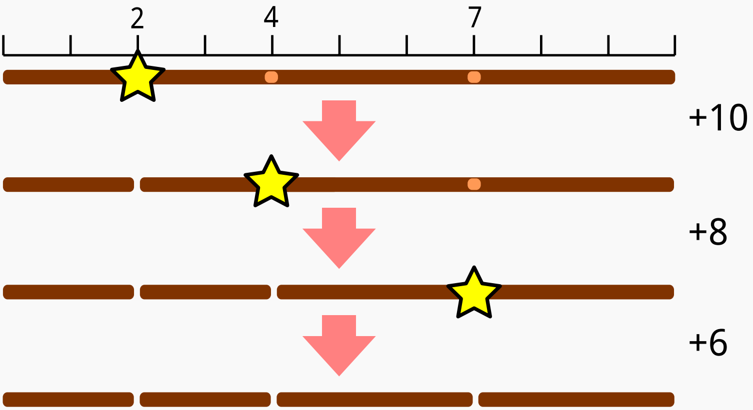
Contoh Soal 2: Memotong Kayu (lanj.)

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.



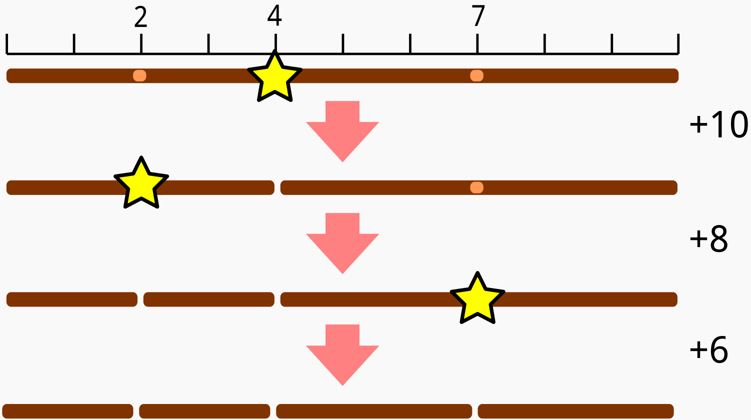
Contoh Soal 2: Memotong Kayu (lanj.)

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?

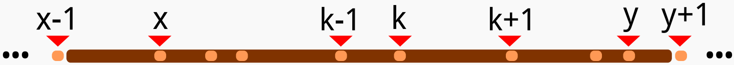
- Apakah strategi *Greedy* dengan memotong "setengah-tengahnya" selalu menghasilkan solusi optimal?
- Bagaimana dengan kasus jika $M = 2000$ dan $L = [1, 2, 3, 4, 5, 1000]$?
- Kita akan coba menggunakan DP untuk persoalan ini.

Observasi

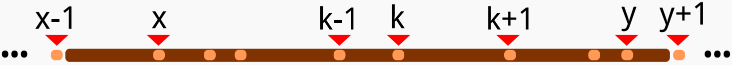
- Untuk pemotongan pertama, terdapat N pilihan lokasi pemotongan.
- Jika kita memotong di posisi L_x, maka didapatkan dua batang.
- Batang pertama perlu dipotong di titik $L_1, L_2, ..., L_{x-1}$ dan batang kedua di $L_{x+1}, L_{x+2}, ..., L_N$.
- Ternyata kita mendapatkan sub-persoalan yang serupa.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

Formulasi Rekurens

- Definisikan sebuah fungsi $g(x, y)$ sebagai jumlah usaha minimum yang mungkin diperoleh, jika kita hanya perlu memotong di $L_x, L_{x+1}, ..., L_y$.
- Untuk menghitung $g(x, y)$ kita dapat mencoba titik mana yang kita potong pertama kali.
- Jika kita memotong di L_k ($x \leq k \leq y$), maka kita akan mendapatkan dua potongan.



Formulasi Rekurens (lanj.)



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

Formulasi Base Case

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

Formulasi Akhir

Dapat dirumuskan:

$$g(x, y) = \begin{cases} 0, & x > y \\ \min_{x \leq k \leq y} g(x, k-1) + g(k+1, y) + (L_{y+1} - L_{x-1}), & x \leq y \end{cases}$$

Analisis Kompleksitas

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

Solusi Top Down

Kita implementasikan $g(x, y)$ sebagai fungsi $\textsc{solve}(x, y)$:

```
SOLVE(x, y)
if (x > y)
    return 0
elseif hasComputed[x][y]
    return memo[x][y]
else
    best = ∞
    cost = L[y + 1] − L[x − 1]
    for k = x to y
        best = min(best, SOLVE(x, k − 1) + SOLVE(k + 1, y) + cost)
    hasComputed[x][y] = true
    memo[x][y] = best
    return best
```

Jawaban ada pada $\textsc{solve}(1, N)$

Solusi Bottom Up

SOLVE()

```
// Inisialisasi base case
for x = 0 to N + 1
    for y = 0 to x − 1
        g[x][y] = 0

// Isi "tabel" mulai dari kasus yang kecil
for gap = 0 to N
    for x = 1 to N − gap
        y = x + gap
        best = ∞
        cost = L[y + 1] − L[x − 1]
        for k = x to y
            best = min(best, g[x][k − 1] + g[k + 1][y] + cost)
        g[x][y] = best

return g[1][N]
```

Pengisian "Tabel" DP

- Perhatikan bahwa pada metode *bottom up*, pengisian "tabel" dilakukan secara "tidak biasa".
- Kita perlu mengisi mulai dari:
  - $g[1][1], g[2][2], ..., g[N][N]$,
  - lalu $g[1][2], g[2][3], ..., g[N-1][N]$,
  - lalu $g[1][3], g[2][4], ..., g[N-2][N]$,
  - lalu $g[1][4], g[2][5], ..., g[N-3][N]$,
  - dan seterusnya sampai $g[1][N]$.
- 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 in 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.