13 - Pemrograman dinamis (*dynamic programming*)

[KOMS120403]

Desain dan Analisis Algoritma (2022/2023)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 16 (May 2023)

Daftar isi

- Prinsip Pemrograman Dinamis
- Contoh sederhana
- Penerapan lebih lanjut dari pemrograman dinamis
  - Knapsack problem and Memory functions
  - Shortest path problem
  - Traveling salesman problem
  - Binary search tree yang optimal

Figure: Richard Ernest Bellman (1920-1984)

Prinsip pemrograman dinamis (*dynamic programming*) (1)

- Pemrograman tidak mengacu pada pemrograman komputer. Namun dalam hal ini, pemrograman berarti " perencanaan".
- Dinamis digunakan untuk merepresentasikan aspek waktu yang bervariasi dari masalah.

Pemrograman dinamis adalah metode pemecahan masalah dengan menguraikan solusi ke dalam serangkaian *tahapan*, sehingga solusi dari masalah dapat dilihat sebagai **sebuah rangkaian keputusan yang terurut**.

Prinsip pemrograman dinamis (2)

- Biasanya digunakan untuk *optimization problems* (maksimisasi/minimisasi).
- Biasanya, submasalah ini muncul dari **rekurensi** yang menghubungkan solusi masalah yang diberikan dengan solusi dari submasalahnya yang lebih kecil.
- Daripada menyelesaikan submasalah yang tumpang tindih berulang kali, pemrograman dinamis menyarankan untuk menyelesaikan setiap submasalah yang lebih kecil hanya sekali dan mencatat hasilnya dalam tabel yang darinya solusi untuk masalah asli kemudian dapat diperoleh.

Bagian 1. Pemrograman dinamis pada barisan Fibonacci

Bilangan Fibonacci

Ingatlah bahwa deret Fibonacci didefinisikan sebagai berikut.

$$F(n) = \begin{cases} 1, & n = 1 \\ 1, & n = 2 \\ F(n-1) + F(n-2), & n \geq 3 \end{cases}$$

Barisan Fibonacci: $1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Algoritma rekursif:

Algorithm 1 Fibonacci sequence recursively

```
procedure Fib(n)
    if n ≤ 2 then return 1
    end if
    return (Fib(n − 1) + Fib(n − 2))
end procedure
```

Diagram pembentukan barisan Fibonacci

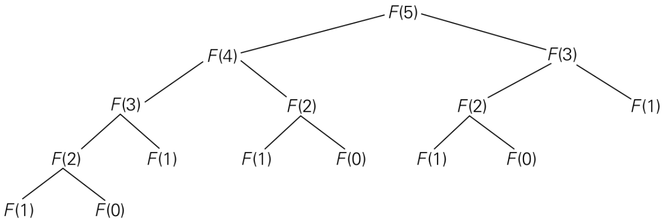


Figure: Pohon panggilan rekursif untuk menghitung angka Fibonacci ke-5 dengan algoritma berbasis definisi.

Tumpang tindih pada konstruksi bilangan Fibonacci

Cara menangani **perhitungan yang tumpang tindih**?

- Buat array 1-dimensi, dan isi dengan $(n+1)$ nilai berurutan dari $F(n)$, mulai dari $F(0)$, dan elemen terakhir adalah $F(n)$.

Algorithm 2 Fibonacci sequence iteratively

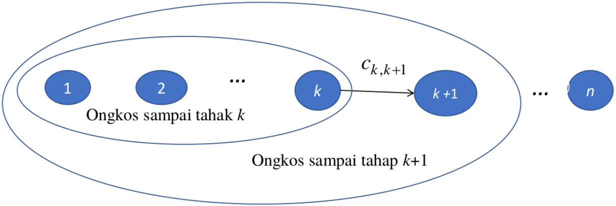
```
procedure FIB2(n)
    F[0] ← 0;  F[1] ← 1
    for i ← 2 to n do
        F(i) ← F(i − 1) + F(i − 2)
    end for
    return F[n]
end procedure
```

Prinsip optimalitas

Suatu masalah dikatakan memenuhi Principle of Optimality **jika solusi optimal untuk setiap instance dari masalah optimisasi terdiri dari solusi optimal untuk sub-instance-nya**; yaitu jika solusi totalnya optimal, maka bagian dari solusi hingga langkah k juga optimal.

- **Implikasi:** jika kita bekerja dari langkah k ke langkah $k+1$, kita dapat menggunakan solusi optimal hingga langkah k, tanpa kembali ke keadaan awal.

Biaya pada langkah $k+1$ = biaya pada langkah k + biaya dari langkah k ke langkah $k+1$



Requirements

Secara umum, **ada beberapa persyaratan untuk menerapkan pemrograman dinamis pada suatu masalah** [*sumber:* Algorithm Design, by Kleinberg and Tardos]:

1. Solusi untuk masalah asli dapat dihitung dari solusi ke submasalah (independen).
2. Ada jumlah submasalah polinomial.
3. Ada pengurutan submasalah sedemikian rupa sehingga solusi submasalah hanya bergantung pada solusi submasalah yang mendahuluinya dalam urutan ini.

Dynamic Programming Steps

1. **Definisikan sub-masalah:** Ini adalah langkah kritis. Biasanya struktur perulangan mengikuti secara alami setelah mendefinisikan submasalah.
2. **Rekurensi:** Tulis solusi untuk (sub-)masalah dalam bentuk solusi untuk submasalah yang lebih kecil, yaitu rekursi. Ini akan memberikan algoritma.
3. **Kebenaran:** Buktikan perulangannya benar, biasanya dengan *induksi*.
4. **Kompleksitas:** Menganalisis kompleksitas waktu, biasanya dengan formula berikut:

$$runtime = \#subproblems \times timeToSolveEachOne$$

Pendekatan

1. Pendekatan **Forward/top-down**

   Perhitungan dilakukan dari tahap $1, 2, \ldots, n-1, n$. Urutan variabel keputusan: $x_1, x_2, \ldots, x_n$.

2. Pendekatan **Backward/bottom-up**

   Perhitungan dilakukan dari tahap $n, n-1, \ldots, 2, 1$. Urutan variabel keputusan: $x_n, x_{n-1}, \ldots, x_1$.

Bagian 2. Contoh sederhana penerapan pemrograman dinamis

1. Masalah barisan koin
2. Masalah penukaran
3. Masalah pengumpulan koin

1. Masalah barisan koin

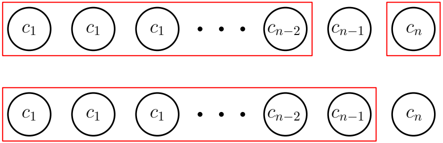
1. Masalah barisan koin (1)

Permasalahan

Diberikan deretan n koin yang nilainya berupa bilangan bulat positif $c_1, c_2, \ldots, c_n$*, belum tentu berbeda. Tujuannya adalah untuk mengambil jumlah uang maksimum dengan batasan bahwa tidak ada dua koin yang berdekatan di baris awal yang dapat diambil.*

Strategi: Misalkan $F(n)$: jumlah maksimum yang dapat diambil dari deretan n koin. Bagaimanakah cara menurunkan rumus rekursif untuk $F(n)$?

- Dua partisi pilihan koin yang diizinkan:
  1. Grup yang menyertakan koin terakhir
  2. Grup yang tidak termasuk koin terakhir



$c_n + F(n-2)$

$F(n-1)$

1. Masalah barisan koin (2)

Fungsi rekursif:

$$\begin{cases} F(n) & = \max\{c_n + F(n-2), F(n-1)\} \text{ for } n > 1 \\ F(0) & = 0,\ F(1) = c_1 \end{cases}$$

Jadi, $F(n)$ dapat dihitung seperti pada deret Fibonacci.

Algorithm 3 Coin row

```
procedure CoinRow(C[1..n])
    F[0] ← 0;  F[1] ← C[1]
    for i ← 2 to n do
        F(i) ← max{C[i] + F(i − 2), F(i − 1)}
    end for
    return F[n]
end procedure
```

1. Masalah barisan koin (3)

$F[0] = 0$, $F[1] = c_1 = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

$F[6] = \max\{2 + 15, 15\} = 17$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	**17**

Figure: Memecahkan masalah baris koin dengan pemrograman dinamis untuk baris koin 5, 1, 2, 10, 6, 2, dengan solusi optimal $\{c_1, c_4, c_6\}$ dan nilai optimal 17.

2. Masalah penukaran

2. Masalah penukaran (1)

Permasalahan

Berikan cek sejumlah n, yang ingikn ditukar dengan koin denominasi $d_1 < d_2 < \cdots < d_m$. Asumsikan bahwa kita memiliki jumlah koin yang tidak terbatas untuk setiap denominasi m $d_1 < d_2 < \cdots < d_m$ di mana $d_1 = 1$. Tentukan jumlah koin minimum yang dibutuhkan.

Strategi: Misalkan $F(n)$: jumlah minimum koin yang nilainya berjumlah n, dan tentukan $F(0) = 0$. Bagaimana cara menurunkan rumus rekursif untuk $F(n)$?

- Jumlah n hanya dapat diperoleh dengan menambahkan satu koin denominasi d_j ke jumlah $n - d_j$ untuk $j = 1, 2, \ldots, m$ sehingga $n \geq d_j$.
- Minimalkan $F(n - d_j) + 1$

Fungsi rekursif:

$$\begin{cases} F(n) & = \min_{j; n \geq d_j} F(n - d_j) + 1 \text{ for } n > 0 \\ F(0) & = 0 \end{cases}$$

2. Masalah penukaran (2)

Algorithm 4 ChangeMaking

procedure MinCoinChange($D[1..m], n$)
  input: positive integer n, and array $D[1..m]$ of increasing positive integers indicating the coin denominations where $D[1] = 1$
  output: the minimum number of coins that add up to n
  $F[0] \leftarrow 0$
  for $i \leftarrow 1$ **to** n **do**
    temp $\leftarrow \infty$; $j \leftarrow 1$
    while $j \leq m$ **and** $i \geq D[j]$ **do**
      temp $\leftarrow \min(F[i - D[j]], \text{temp})$
      $j \leftarrow j + 1$
    end while
    $F[i] \leftarrow \text{temp} + 1$
  end for
  return $F[n]$
end procedure

2. Masalah penukaran (3)

$F[0] = 0$

n	0	1	2	3	4	5	6
F	0						

$F[1] = \min\{F[1-1]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1					

$F[2] = \min\{F[2-1]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2				

$F[3] = \min\{F[3-1], F[3-3]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1			

$F[4] = \min\{F[4-1], F[4-3], F[4-4]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1	1		

$F[5] = \min\{F[5-1], F[5-3], F[5-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	

$F[6] = \min\{F[6-1], F[6-3], F[6-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	**2**

Figure: Penerapan algoritma MinCoinChange untuk nilai $n = 6$ dan satuan koin 1, 3, dan 4.

3. Masalah pengumpulan koin

3. Masalah pengumpulan koin (1)

Permasalahan

Beberapa koin ditempatkan dalam papan berukuran $n \times m$, tidak lebih dari satu koin per sel. Robot, yang terletak di sel kiri atas papan, perlu mengumpulkan koin sebanyak mungkin dan membawanya ke sel kanan bawah.

- *Pada setiap langkah, robot dapat memindahkan satu sel ke kanan atau satu sel ke bawah dari lokasinya saat ini.*
- *Ketika robot mengunjungi sel dengan koin, ia selalu mengambil koin itu.*

Rancang algoritma untuk menemukan jumlah koin maksimum yang dapat dikumpulkan robot dan jalur yang harus diikuti untuk melakukannya.

Strategi: Misalkan $F(i, j)$ menjadi jumlah koin terbesar yang dapat dikumpulkan dan dibawa robot ke sel (i, j) di baris ith dan kolom ke-j di papan.

- (i, j) dapat dijangkau dari sel $(i - 1, j)$ (atas) atau sel $(i, j - 1)$ (kiri).
- Untuk sel di baris pertama (atau kolom pertama), asumsikan bahwa $F(i - 1, j) = 0$ (atau $F(i, j - 1) = 0$).

3. Masalah pengumpulan koin (2)

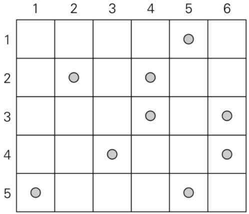
Fungsi rekursif:

$$
\begin{cases}
F(i,j) & = \max\{F(i-1,j), F(i,j-1)\} + c_{ij} \quad \text{for } 1 \le i < n,\ 1 \le j \le m \\
F(0,j) & = 0 \text{ for } 1 \le j \le m \text{ and } F(i,0) \quad \text{for } 1 \le i \le n
\end{cases}
$$

Algorithm 5 Robot coin collection

procedure $\textsc{RobotCoinCollection}(C[1..n, 1..m])$
 $F[1,1] \leftarrow C[1,1]$
 for $j \leftarrow 2$ **to** m **do**
  $F[1,j] \leftarrow F[1,j-1] + C[1,j]$
 end for
 for $i \leftarrow 2$ **to** n **do**
  $F[i,1] \leftarrow F[i-1,1] + C[i,1]$
  for $j \leftarrow 2$ **to** m **do**
   $F[i,j] \leftarrow \max\{F[i-1,j], F[i,j-1]\} + C[i,j]$
  end for
 end for
 return $F[n,m]$
end procedure

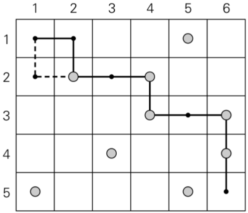
3. Masalah pengumpulan koin (3)



(a)

	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	**5**

(b)



(c)

Figure: (a) Koin untuk dikumpulkan. (b) Hasil algoritma pemrograman dinamis. (c) Dua jalur untuk mengumpulkan 5 koin, jumlah koin maksimum yang mungkin.

Bagian 3. Pemrograman dinamis pada knapsack problem

Knapsack problem (1)

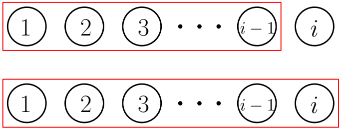
Permasalahan

Diberikan n objek dengan bobot $w_1, \ldots, w_n$ dan nilai $v_1, \ldots, v_n$, serta sebuah ransel berkapasitas W. Temukan subset paling berharga dari barang-barang yang sesuai dengan ransel.

Strategi: Misalkan $F(i,j)$ menjadi nilai solusi optimal untuk instance yang ditentukan oleh i item pertama $1 \leq i \leq n$,

- dengan bobot $w_1, \ldots, w_i$ dan nilai $v_1, \ldots, v_i$
- kapasitas ransel j, dengan $1 \leq j \leq W$.

Kita ingin memutuskan apakah objek i disertakan atau tidak.



$$F(i-1,j)$$

$$v_i + F(i-1, j-w_i)$$

Knapsack problem (2)

$$F(i,j) = \begin{cases} \max\{F(i-1,j), v_i + F(i-1,j-w_i)\} & \text{jika } j - w_i \geq 0 \\ F(i-1,j), & \text{jika } j - w_i < 0 \end{cases}$$

dengan kondisi awal:

$$F(0,j) = 0 \text{ untuk } j \geq 0 \text{ dan } F(i,0) = 0 \text{ untuk } i \geq 0$$

Tujuan: untuk menemukan $F(n, W)$, nilai maksimal subset dari n item yang diberikan yang sesuai dengan ransel kapasitas W; beserta subset yang memenuhi nilai optimal tersebut.

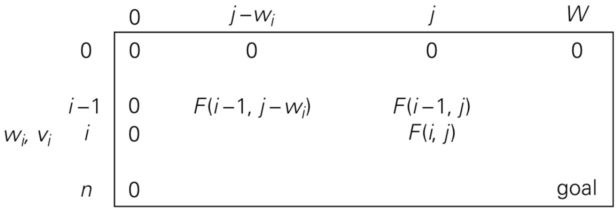


Figure: Tabel untuk memecahkan masalah knapsack dengan pemrograman dinamis.

Knapsack problem (3)

Contoh

item	weight	value
1	2	\$12
2	1	\$10
3	3	\$20
4	2	\$15

capacity $W = 5$

		capacity j					
	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2,\ v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1,\ v_2 = 10$	2	0	10	12	22	22	22
$w_3 = 3,\ v_3 = 20$	3	0	10	12	22	30	32
$w_4 = 2,\ v_4 = 15$	4	0	10	15	25	30	**37**

Figure: Contoh penyelesaian instance masalah knapsack dengan algoritma pemrograman dinamis.

Bagian 4. Fungsi memori (*memory functions*)

(studi kasus pada Knapsack problem)

Prinsip fungsi memori

Pada dasarnya, pemrograman dinamis memecahkan masalah yang memiliki hubungan perulangan.

- Menggunakan perulangan secara langsung dalam algoritma rekursif memiliki kelemahan, yaitu kemungkinan untuk memecahkan sub-masalah umum berkali-kali, menghasilkan kompleksitas yang mencapai eksponensial.
- Teknik pemrograman dinamis memiliki kelemahan bahwa beberapa sub-masalah mungkin tidak perlu dipecahkan.

Tujuan: untuk mendapatkan yang terbaik dari kedua pendekatan, yaitu semua sub-masalah yang diperlukan diselesaikan hanya sekali.

Prinsip fungsi memori

- Teknik menggunakan pendekatan top-down, algoritma rekursif, dengan tabel solusi sub-masalah.
- Sebelum menentukan solusi secara rekursif, algoritma memeriksa apakah sub-masalah sudah diselesaikan dengan memeriksa tabel.
- Jika tabel memiliki nilai yang valid maka algoritma menggunakan nilai tabel yang lain, dilanjutkan dengan solusi rekursif.

Ingat kembali definisi $F(i, j)$, nilai solusi optimal untuk instance yang ditentukan oleh i objek pertama dengan kapasitas knapsack j.

$$F(i,j) = \begin{cases} \max\{F(i-1,j), v_i + F(i-1, j-w_i)\} & \text{jika } j - w_i \geq 0 \\ F(i-1,j), & \text{jika } j - w_i < 0 \end{cases}$$

Algoritma fungsi memori

Algorithm 6 MF Knapsack

```
procedure MFK(i, j)
    input: i ∈ Z≥0 menunjukkan jumlah item pertama yang dipertimbangkan dan j ∈ Z≥0 menunjukkan kapasitas ransel
    output: Nilai subset layak optimal dari item i pertama
    variables: variabel global masukan array bobot Wt[1..n], nilai V[1..n], dan tabel F[0..n, 0..W] yang entri-entrinya diinisialisasi dengan -1 kecuali untuk baris 0 dan kolom 0 diinisialisasi dengan 0.

    if F[i, j] < 0 then
        if j < Wt[i] then
            value ← MFK(i − 1, j)
        else
            value ← max{MFK(i − 1, j), V[i] + MFK(i − 1, j − Wt[i])}
        end if
        F[i, j] ← value
    end if
    return F[i, j]
end procedure
```

Contoh fungsi memori

Mari terapkan prosedur MFK pada contoh sebelumnya:

item	weight	value
1	2	\$12
2	1	\$10
3	3	\$20
4	2	\$15

capacity $W = 5$

		capacity j					
	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2,\ v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1,\ v_2 = 10$	2	0	—	12	22	—	22
$w_3 = 3,\ v_3 = 20$	3	0	—	—	22	—	32
$w_4 = 2,\ v_4 = 15$	4	0	—	—	—	—	**37**

Figure: Contoh penyelesaian instance masalah knapsack dengan algoritma fungsi memori.

Contoh fungsi memori

Implementasi algoritma langkah demi langkah adalah sebagai berikut:

- $N = 4, W = 5, w_1 = 2, w_2 = 1, w_3 = 3, w_4 = 2, v_1 = 12, v_2 = 10, v_3 = 20, v_4 = 15$
- $V[4,5] = \max\{V[3,5], 15 + V[3,3]\}$ (since $v_4 = 15, w_4 = 2, j = 15, w_4 < j \rightarrow$ apply 1st case)
- $V[3,5] = \max\{V[2,5], 20 + V[2,2]\}$
- $V[3,3] = \max\{V[2,3], 20 + V[2,0]\}$
- $V[2,5] = \max\{V[1,5], 10 + V[1,4]\}$
- $V[2,2] = \max\{V[1,2], 10 + V[1,1]\}$
- $V[2,3] = \max\{V[1,3], 10 + V[1,2]\}$
- $V[2,0] = V[1,0] = 0$ (since $w_2 = 1, j = 0, w_2 > j \rightarrow$ apply 2nd case)
- $V[1,5] = \max\{V[0,5], 12 + V[0,3]\} = \max\{0, 12 + 0\} = 12$
- $V[1,4] = \max\{V[0,4], 12 + V[0,2]\} = \max\{0, 12 + 0\} = 12$
- $V[1,2] = \max\{V[0,2], 12 + V[0,0]\} = \max\{0, 12 + 0\} = 12$
- $V[1,1] = V[0,1] = 0$
- $V[1,3] = \max\{V[0,3], 12 + V[0,1]\} = \max\{0, 12 + 0\} = 12$

Selanjutnya, lakukan substitusi mundur:

- $V[2,3] = \max\{V[1,3], 10 + V[1,2]\} = \max\{12, 10 + 12\} = 22$
- $V[2,2] = \max\{V[1,2], 10 + V[1,1]\} = \max\{12, 10 + 0\} = 12$
- $V[2,5] = \max\{V[1,5], 10 + V[1,4]\} = \max\{12, 10 + 12\} = 22$
- $V[3,3] = \max\{V[2,3], 20 + V[2,0]\} = \max\{22, 20 + 0\} = 22$
- $V[3,5] = \max\{V[2,5], 20 + V[2,2]\} = \max\{22, 20 + 12\} = 32$
- $V[4,5] = \max\{V[3,5], 15 + V[3,3]\} = \max\{32, 15 + 22\} = 37$

end of slide...