Dynamic Programming

ค่ายคอมพิวเตอร์โอลิมปิก สอวน. ค่าย 2 /2566
ศูนย์โรงเรียนสามเสนวิทยาลัย - มหาวิทยาลัยธรรมศาสตร์
ระหว่างวันที่ 18 มีนาคม - 3 เมษายน 2567

Pisit Makpaisit

COMPUTER SCIENCE

Dynamic Programming

- เทคนิคการแก้ปัญหาด้วยการแบ่งปัญหาออกเป็นปัญหาย่อย (Subproblem)
- เก็บคำตอบที่ดีที่สุดของปัญหาย่อย เพื่อนำกลับมาใช้อีกครั้ง
- ลดเวลาจากการทำงานที่ซ้ำซ้อนลง

Fibonacci Sequence

0 1 1 2 3 5 8 13 21 34 55 89 ...

Recurrence Relation:

$$F_n = F_{n-1} + F_{n-2}$$

$$F_0 = 0, F_1 = 1$$

Recursive Fibonacci Sequence

Algorithm 31 recursiveFibonacci

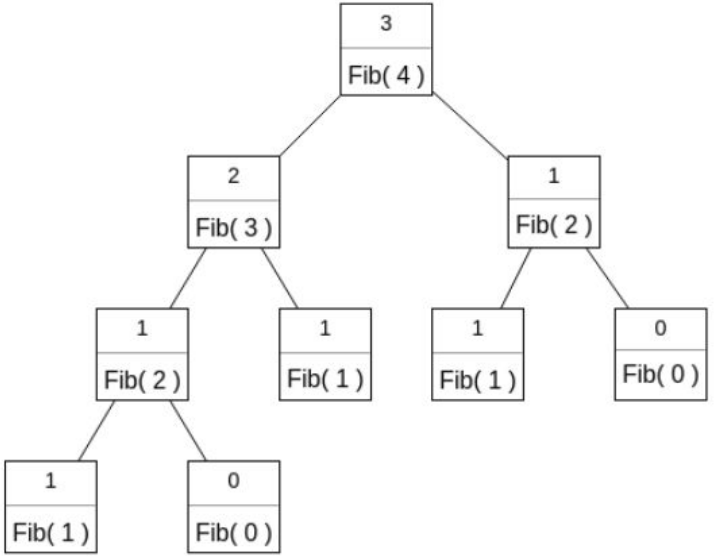
```
1: function Fib(n)
2:     if n ≤ 1 then
3:         return n
4:     else
5:         return Fib(n − 1) + Fib(n − 2)
```

เขียนตาม recurrence relation ได้โดยตรง ด้วย recursive function

Recursive Fibonacci Sequence

เวลาการทำงานเป็น Exponential

(แต่ละฟังก์ชันเรียกไปอีกเฉลี่ยประมาณ 2 ครั้ง)



Memoization Fibonacci Sequence

Algorithm 32 memoizationFibonacci

```
1: Initial array F with −1
2: function Fib(n)
3:     if n ≤ 1 then
4:         return n
5:     if F[n] = −1 then
6:         F[n] ← Fib(n − 1) + Fib(n − 2)
7:     return F[n]
```

เก็บค่าที่คำนวนไว้แล้วใน memory (อาร์เรย์) จะได้ไม่ต้องคำนวณใหม่ (เวลา Θ(n))

เรียกวิธีนี้ว่า Memoization หรือ Dynamic Programming แบบ Top down

Dynamic Programming Fibonacci

Algorithm 33 bottomUpDpFibonacci

```
1: function Fib(n)
2:     Initial array F of size n + 1
3:     F[0] = 0, F[1] = 1
4:     for i = 2 to n + 1 do
5:         F[i] ← F[i − 1] + F[i − 2]
6:     return F[n]
```

Bottom up dynamic programming ใช้การเติมตารางเพื่อหาคำตอบ (เวลา Θ(n))

n :	0	1	2	3	4	5	6	7	8	9	10	...
F :	0	1	1	2	3	5	8	13	21	34	55	...

Assignment 0

เขียนโปรแกรมหาค่า Fibonacci ลำดับที่ n ด้วยเทคนิค Memoization และ Dynamic programming แบบ Bottom up

Binomial Coefficient

- จำนวนวิธีในการเลือกของ k ชิ้นจากของ n ชิ้น
- ค่าสัมประสิทธิ์ของแต่ละพจน์ในการกระจาย $(x + y)^n$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Recurrence Relation:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } n, k > 0 \\ 1 & \text{if } k = 0 \text{ or } n = k \end{cases}$$

Binomial Coefficient

Algorithm 34 recursiveBinomialCoefficient

```
1: function C(n, k)
2:     if k = 0 or k = n then
3:         return 1
4:     if k < 0 or k > n then
5:         return 0
6:     return C(n − 1, k − 1) + C(n − 1, k)
```

คำตอบย่อยที่เป็นไปได้มีค่าได้ตั้งแต่ 0 - n และ 0 - k ซึ่งเป็นข้อมูลใน 2 มิติ ดังนั้นอาร์เรย์ที่ใช้สำหรับเก็บคำตอบย่อยจะต้องมีขนาด (n + 1) x (k + 1)

Assignment 1

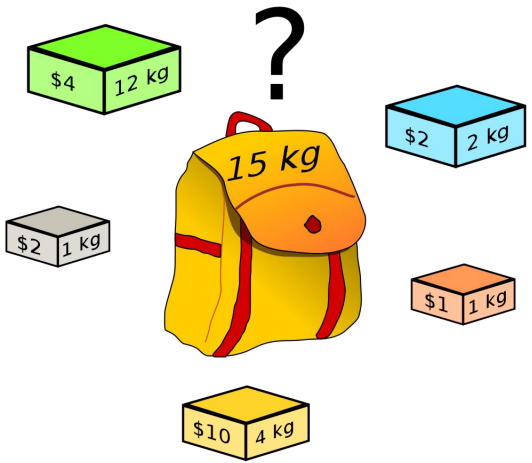
เขียนโปรแกรมสำหรับหาค่า Binomial Coefficient (รับค่า n, k เข้ามาในโปรแกรม และแสดงค่า C(n,k) ออกมาทางหน้าจอ) โดยใช้การเขียนแบบ Top down dynamic programming

Assignment 2

เขียนโปรแกรมสำหรับหาค่า Binomial Coefficient (รับค่า n, k เข้ามาในโปรแกรม และแสดงค่า C(n,k) ออกมาทางหน้าจอ) โดยใช้การเขียนแบบ Bottom up dynamic programming

0/1 Knapsack Problem

มีของ n ชิ้น ชิ้นที่ i มีน้ำหนัก w_i และมีมูลค่า v_i มีเป้อยู่หนึ่งใบที่สามารถรับน้ำหนักได้ W ให้หามูลค่ารวมของสิ่งของสูงสุดที่สามารถใส่ลงในเป้ได้ โดยที่นำหนักไม่เกินน้ำหนักที่เป้จะรับได้



https://th.wikipedia.org/wiki/ปัญหาถุงกระสอบ

0/1 Knapsack Problem

สมมติให้ **OPT(i,j)** แทน มูลค่ารวมของสิ่งของทั้งหมดในเป้ เมื่อพิจารณาสิ่งของตั้งแต่ชิ้นที่ 1 ถึงชิ้นที่ i และน้ำหนักที่รับได้ของเป้เป็น j

- OPT(4, 30) = มูลค่ารวมสูงสุด ถ้าคิดแค่ของชิ้นที่ 1-4 และกระเป๋ารับน้ำหนักได้ 30
- OPT(1, 22) = มูลค่ารวมสูงสุด ถ้าคิดแค่ของชิ้นที่ 1 และกระเป๋ารับน้ำหนักได้ 22
- OPT(8, 0) = มูลค่ารวมสูงสุด ถ้าคิดแค่ของชิ้นที่ 1-8 และกระเป๋ารับน้ำหนักได้ 0

คำตอบของปัญหาคือการหา OPT(n, W)
(ปัญหานี้มีของทั้งหมด n ชิ้น และเป้รับน้ำหนักได้ไม่เกิน W)

0/1 Knapsack Problem

Base case

กรณีที่ W = 0 หรือเป้รับน้ำหนักไม่ได้เลย มูลค่ารวมจะเป็น 0

$$OPT(i,0) = 0$$

ของทั้งหมดมี 0 ชิ้นมูลค่ารวมก็จะต้องได้ 0 เช่นกัน เพราะไม่มีสิ่งของให้ใส่

$$OPT(0,j) = 0$$

0/1 Knapsack Problem

General Case

สมมติว่ามีของทั้งสิ้น i ชิ้น และเราสนใจเพียงแค่ว่าจะเอาชิ้นที่ i นี้ใส่ลงในเป้หรือไม่ เราไม่สามารถรู้คำตอบได้ทันทีว่าควรที่จะนำใส่เข้าไปในเป้หรือไม่ แต่เราสามารถดูจากคำตอบของปัญหาย่อยได้ว่าใส่หรือไม่ใส่จะได้มูลค่ามากกว่ากัน

เอาชิ้นที่ i ใส่เป้ จะได้มูลค่าที่ดีที่สุดคือ $v_i + OPT(i-1,j-w_i)$

มูลค่าที่ดีที่สุด ได้จาก มูลค่าของของชิ้นที่ i บวกด้วยมูลค่ารวมของสิ่งของทั้งหมดในเป้ เมื่อพิจารณาสิ่งของตั้งแต่ชิ้นที่ 1 ถึงชิ้นที่ i-1 และน้ำหนักที่รับได้ของเป้เป็น $j-w_i$ ที่ต้องเป็น i-1 เพราะเราเลือกของชิ้นที่ i มาแล้วทำให้ไม่ต้องสนใจอีก ส่วนน้ำหนักของเป้ที่รับได้ก็จะต้องลดลงไปเท่ากับน้ำของของชิ้นที่ i (w_i)

0/1 Knapsack Problem

ไม่เอาของชิ้นที่ i ใส่เป้ มูลค่าที่ดีที่สุดก็ให้พิจารณาจากมูลค่าที่ดีที่สุดเมื่อพิจารณาของตั้งแต่ชิ้นที่ 1 จนถึงชิ้นที่ i-1 หรือก็คือ

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

ดังนั้นรูปทั่วไปก็คือการหาว่ามูลค่าระหว่างการเลือกหรือไม่เลือกของชิ้นที่ i แบบไหนจะดีกว่า จึงได้ว่า

$$OPT(i,j) = \max(OPT(i-1,j), v_i+OPT(i-1,j-w_i)) \quad \text{if } i > 0 \text{ and } j \geq w_i$$

0/1 Knapsack Problem

Recurrence relation

$$OPT(i,j) = \begin{cases} max(OPT(i-1,j), v_i + OPT(i-1, j-w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ OPT(i-1,j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

0/1 Knapsack Problem

Algorithm 35 discreteKnapsack

```
1:  function Knapsack(v, w, W)
2:      for i = 0 to n + 1 do T[i][0] ← 0
3:      for j = 0 to W + 1 do T[0][j] ← 0
4:      for i = 1 to n + 1 do
5:          for j = 1 to W + 1 do
6:              if j < w_i then
7:                  T[i][j] ← T[i − 1][j]
8:              else
9:                  T[i][j] ← Max(T[i − 1][j], v_i + T[i − 1][j − w_i])
10:     return T[n][W]
```

0/1 Knapsack Problem

- โปรแกรมจะทำการไล่เติมคำตอบในแถวที่ 0 และหลักที่ 0 ก่อน
- จากนั้นก็เติมคำตอบในแถวที่ 1, 2, 3, ..., n
- คำตอบจะอยู่ช่อง T[n][W] ซึ่งแทนค่าของ OPT(n, W)
- เวลาการทำงานของโปรแกรมคือการเติมตารางขนาด (n + 1) x (W + 1) นั่นคือใช้เวลาเท่ากับ Θ(nW)

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0								
1								
2								
3								
4(n)								

for $i = 0$ to $n + 1$ do $T[i][0] \leftarrow 0$

for $j = 0$ to $W + 1$ do $T[0][j] \leftarrow 0$

for $i = 1$ to $n + 1$ do

    for $j = 1$ to $W + 1$ do

        if $j < w_i$ then

            $T[i][j] \leftarrow T[i-1][j]$

        else

            $T[i][j] \leftarrow \text{Max}(T[i-1][j], v_i + T[i-1][j-w_i])$

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0	0							
1	0							
2	0							
3	0							
4(n)	0							

for $i = 0$ to $n + 1$ do $T[i][0] \leftarrow 0$

for $j = 0$ to $W + 1$ do $T[0][j] \leftarrow 0$

for $i = 1$ to $n + 1$ do

    for $j = 1$ to $W + 1$ do

        if $j < w_i$ then

            $T[i][j] \leftarrow T[i-1][j]$

        else

            $T[i][j] \leftarrow \text{Max}(T[i-1][j], v_i + T[i-1][j-w_i])$

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4(n)	0							

for $i = 0$ to $n + 1$ do $T[i][0] \leftarrow 0$

for $j = 0$ to $W + 1$ do $T[0][j] \leftarrow 0$

for $i = 1$ to $n + 1$ do

    for $j = 1$ to $W + 1$ do

        if $j < w_i$ then

            $T[i][j] \leftarrow T[i-1][j]$

        else

            $T[i][j] \leftarrow \text{Max}(T[i-1][j], v_i + T[i-1][j-w_i])$

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0	0	0	0	0	0	0	0	0
1	0	1						
2	0							
3	0							
4(n)	0							

for $i = 0$ to $n + 1$ do $T[i][0] \leftarrow 0$

for $j = 0$ to $W + 1$ do $T[0][j] \leftarrow 0$

for $i = 1$ to $n + 1$ do

  for $j = 1$ to $W + 1$ do

    if $j < w_i$ then

      $T[i][j] \leftarrow T[i-1][j]$

    else

      $T[i][j] \leftarrow \text{Max}(T[i-1][j], v_i + T[i-1][j-w_i])$

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0	0	0	0	0	0	0	0	0
1	0	1	1					
2	0							
3	0							
4(n)	0							

for $i = 0$ to $n + 1$ do $T[i][0] \leftarrow 0$

for $j = 0$ to $W + 1$ do $T[0][j] \leftarrow 0$

for $i = 1$ to $n + 1$ do

  for $j = 1$ to $W + 1$ do

    if $j < w_i$ then

      $T[i][j] \leftarrow T[i-1][j]$

    else

      $T[i][j] \leftarrow \text{Max}(T[i-1][j], v_i + T[i-1][j-w_i])$

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1
2	0							
3	0							
4(n)	0							

for $i = 0$ to $n + 1$ do $T[i][0] \leftarrow 0$

for $j = 0$ to $W + 1$ do $T[0][j] \leftarrow 0$

for $i = 1$ to $n + 1$ do

    for $j = 1$ to $W + 1$ do

        if $j < w_i$ then

            $T[i][j] \leftarrow T[i-1][j]$

        else

            $T[i][j] \leftarrow \text{Max}(T[i-1][j], v_i + T[i-1][j-w_i])$

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1
2	0	1						
3	0							
4(n)	0							

for $i = 0$ to $n + 1$ do $T[i][0] \leftarrow 0$

for $j = 0$ to $W + 1$ do $T[0][j] \leftarrow 0$

for $i = 1$ to $n + 1$ do

    for $j = 1$ to $W + 1$ do

        if $j < w_i$ then

            $T[i][j] \leftarrow T[i-1][j]$

        else

            $T[i][j] \leftarrow \text{Max}(T[i-1][j], v_i + T[i-1][j-w_i])$

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1
2	0	1	1	4				
3	0							
4(n)	0							

for $i = 0$ to $n + 1$ do $T[i][0] \leftarrow 0$

for $j = 0$ to $W + 1$ do $T[0][j] \leftarrow 0$

for $i = 1$ to $n + 1$ do

  for $j = 1$ to $W + 1$ do

    if $j < w_i$ then

      $T[i][j] \leftarrow T[i-1][j]$

    else

      $T[i][j] \leftarrow \text{Max}(T[i-1][j], v_i + T[i-1][j-w_i])$

0/1 Knapsack Problem Example

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

	0	1	2	3	4	5	6	7(W)
0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1
2	0	1	1	4	5	5	5	5
3	0	1	1	4	5	6	6	9
4(n)	0	1	1	4	5	7	8	9

คำตอบ

0/1 Knapsack Problem Example

เขียน Pseudo-code ของ DP Knapsack problem ถ้าเขียนแบบ Memoization

W = 7, v = [1, 4, 5, 7], w = [1, 3, 4, 5]

จำลองการทำงานและเติมคำตอบลงในตารางของ Pseudo-code นี้

Dynamic Programming

- DP แบบ bottom-up จะต้องระวังเรื่องการเติมตารางให้ดี จะต้องตรวจสอบการขึ้นต่อกันของข้อมูลให้ดีก่อนว่าจะต้องเติมช่องไหนก่อน เพราะหากเติมผิดลำดับคำตอบที่ได้ก็จะไม่ถูกต้อง
- ตรงข้ามกับการเขียนแบบ top-down ที่เราไม่ต้องกังวลเรื่องลำดับ

Dynamic Programming

ขั้นตอนในการแก้ปัญหาด้วย Dynamic programming

1	ทำความเข้าใจปัญหา จุดประสงค์ของปัญหา และแทนด้วยสัญลักษณ์	ยากในขั้นตอนการของแปลงปัญหาเป็นสัญลักษณ์
2	เขียน recurrence relation	base case ที่จะหยุดการคำนวณต่อและส่วนของ general case ที่อธิบายความสัมพันธ์ระหว่างปัญหาและปัญหาย่อย ปัญหาย่อยแต่ละปัญหาสามารถรวมคำตอบกันได้อย่างไร
3	เขียนโปรแกรมตาม recurrence relation	Top down หรือ Bottom up

Assignment 3

เขียนโปรแกรมแก้ปัญหา Knapsack problem แบบ top down และ bottom up โปรแกรมสามารถกรอกข้อมูลของน้ำหนักสูงสุดที่รับได้ W รวมทั้งน้ำหนักและมูลค่าของของแต่ละชิ้น

(ส่งเป็น 2 โปรแกรม)

Coin Changing Problem (Minimum Number of Coins)

ปัญหาการทอนเหรียญ

มีเหรียญทั้งหมด n ชนิด เหรียญชนิดที่ i มีมูลค่า v_i ต้องการทอนเหรียญเป็นจำนวนเงิน C โดยใช้เหรียญให้น้อยที่สุด จะต้องใช้กี่เหรียญในการทอน

https://moneyhub.in.th/article/%E0%B9%80%E0%B8%AB%E0%B8%A3%E0%B8%B5%E0%B8%A2%E0%B8%8D%E0%B8%9A%E0%B8%B2%E0%B8%97-%E0%B8%81%E0%B9%87%E0%B9%80%E0%B8%87%E0%B8%B4%E0%B8%99-%E0%B8%97%E0%B8%B3%E0%B9%84%E0%B8%A1%E0%B8%9A%E0%B8%B2%E0%B8%87/

Coin Changing Problem

M(i, j) แทนจำนวนเหรียญที่น้อยที่สุดในการทอนเงิน j โดยใช้เหรียญแบบที่ 1 - i ดังนั้นคำตอบของปัญหานี้คือการหาค่าของM(n, C)

Coin Changing Problem

Base Case

เนื่องจากเราทราบว่าในการทอนเงิน 0 บาทจะใช้ 0 เหรียญจะได้ว่า

$$M(i,0) = 0$$

ถ้าไม่มีเหรียญเลยก็จะทอนไม่ได้ (ให้เป็นค่า infinity)

$$M(0,j) = \infty$$

Coin Changing Problem

General Case

ในรูปทั่วไปเราจะต้องคำนึงถึงความสัมพันธ์ของปัญหาและปัญหาย่อย จำนวนเหรียญที่น้อยที่สุดในการทอนเมื่อพิจารณา i เหรียญมีความสัมพันธ์อย่างไรกับ i-1 เหรียญ ซึ่งปัญหานี้จะใกล้เคียงกับปัญหา Knapsack คือให้เราพิจารณาว่าจะใช้เหรียญที่ i ในการทอนหรือไม่

ถ้าใช้แสดงว่าจำนวนเหรียญที่น้อยที่สุดในการทอนจะเป็น

$$M(i,j) = 1 + M(i,j-v_i)$$

ถ้าไม่ใช้

$$M(i,j) = M(i-1,j)$$

Coin Changing Problem

ซึ่งเราจะต้องหาค่าที่น้อยกว่าระหว่างสองค่านี้ (เพราะเราต้องการค่าน้อยสุดที่เป็นไปได้ของจำนวนเหรียญที่จะทอน) และเราจะต้องคิดถึงกรณีที่เหรียญที่ i มีมูลค่าสูงกว่าเงินที่ต้องการทอน j ซึ่งในกรณีนี้ค่าที่ได้จะมาจาก M(i-1,j) เท่านั้น

Recurrence relation ที่ได้จึงเป็น

$$M(i,j) = \begin{cases} min(M(i-1,j), 1 + M(i, j-v_i)) & \text{if } i > 0 \text{ and } j \geq v_i \\ M(i-1,j) & \text{if } j < v_i \\ 0 & \text{if } j = 0 \\ \infty & \text{if } i = 0 \end{cases}$$

Assignment 4

เขียนโปรแกรมแก้ปัญหาการทอนเหรียญด้วยวิธี Dynamic programming โดยรับจำนวนเงินที่ต้องทอน และมูลค่าของเหรียญต่างๆ ที่มี

Dynamic Programming

Dynamic programming ลดเวลาการทำงานด้วยการเก็บคำตอบที่จะต้องคำนวณซ้ำเอาไว้ แต่ไม่ใช่ทุกปัญหาที่สามารถใช้เทคนิคนี้เพื่อเพิ่มความเร็วได้

ปัญหาที่จะใช้ DP ได้ จะต้องมีคุณสมบัติ 2 อย่าง

1. คำตอบที่ดีที่สุดของปัญหาจะต้องสามารถหาได้จากคำตอบที่ดีที่สุดของปัญหาย่อย (***Optimal substructure***)
    a. เช่น ในตัวอย่างของ Fibonacci และ Binomial coefficient ปัญหาขนาด n และ n, k สามารถหาคำตอบได้จากปัญหาขนาด n-1 และ n - 1, k - 1
2. ปัญหาย่อยที่เกิดขึ้นจะต้องมีการคำนวณซ้ำ (***Overlapping subproblems***)

Longest Common Subsequence

subsequence ส่วนหนึ่งของสตริงที่เรียงลำดับกันแต่ไม่ต้องต่อเนื่องกัน เช่น PSEUDO มี subsequence หลายคำ เช่น PSUO, SUD, PED, UO, E, SEDO เป็นต้น

common subsequence คือ subsequence ของ 2 สตริง ที่เหมือนกัน เช่น กำหนดให้มีคำว่า ANALYSIS และ ALGORITHMS จะพบว่า subsequence ของทั้งสองคำที่เหมือนกัน เช่น AL, ALI, ALS, IS, ALIS เป็นต้น

longest common subsequence หมายถึง common subsequence ที่ยาวที่สุด ของทั้ง 2 สตริง เช่นจากตัวอย่างที่แล้ว LCS คือ ALIS

Longest Common Subsequence

Longest common subsequence

กำหนดให้สตริง $X = (x_1, x_2, x_3, \ldots, x_n)$ และ $Y = (y_1, y_2, y_3, \ldots, y_m)$ จงหาความยาวของ common subsequence ของ X และ Y ที่ยาวที่สุด

Longest Common Subsequence

- วิธี brute force คือการหา subsequence ทั้งหมดของ X และ Y และนำมาเปรียบเทียบกันว่ามีคู่ใดที่เหมือนกันและยาวที่สุด
- เปรียบเทียบได้กับการหาสับเซต ซึ่งมีความเป็นได้ทั้งหมด 2^n และ 2^m สำหรับ X และ Y ซึ่งจะใช้ไม่ได้ในทางปฏิบัติเมื่อสตริงทั้งสองมีความยาวมาก

Longest Common Subsequence

ให้ LCS(i,j) เป็น longest common subsequence ของสตริง X ความยาว i และสตริง Y ความยาว j

คำตอบที่ต้องการคือการหา LCS(n,m)

Longest Common Subsequence

Base Case

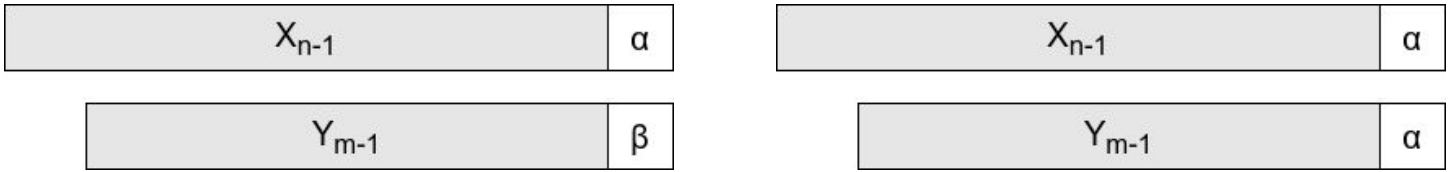
กรณีที่ X หรือ Y ตัวใดตัวหนึ่งมีความยาว 0 จะทำให้ LCS ของทั้งคู่จะกลายเป็น 0 นั่นคือ

$$LCS(i,j) = 0 \qquad \text{if } i = 0 \text{ or } j = 0$$

Longest Common Subsequence

General Case

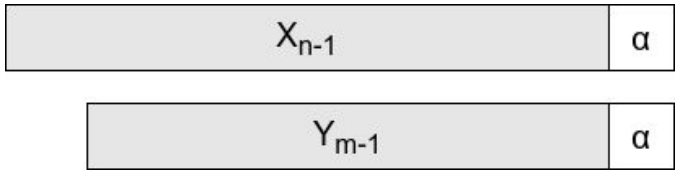
กรณีทั่วไปของความสัมพันธ์จะแบ่งออกเป็น 2 กรณี คือ กรณีที่ตัวสุดท้ายของทั้งคู่เท่ากัน และกรณีที่ตัวสุดท้ายของทั้งคู่ไม่เท่ากัน



Longest Common Subsequence

ถ้าหากตัวสุดท้ายเท่ากัน longest common subsequence ก็จะมีค่าอย่างน้อย 1 แน่นอน จากตัวอักษรที่เหมือนกัน ส่วนที่เหลือก็ให้ LCS ไปหาต่อในขนาดของสตริงที่เล็กลง นั่นคือ

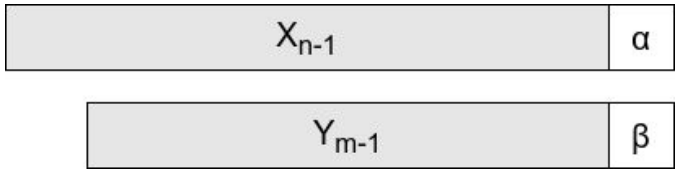
$$LCS(i,j) = 1 + LCS(i-1,j-1) \quad \text{if } x_i = y_j$$



Longest Common Subsequence

ส่วนกรณีที่ตัวสุดท้ายไม่เท่ากันนั้นจะต้องนำ 2 แบบมาเทียบกันว่าแบบไหนให้ค่าดีกว่ากัน ระหว่างการตัดตัวสุดท้ายของ X ออกหรือการตัดตัวสุดท้ายของ Y ออก แล้วนำมาเปรียบเทียบกัน นั่นคือ

$$LCS(i,j) = \max(LCS(i-1,j),LCS(i,j-1)) \qquad \text{if } x_i \neq y_j$$



Longest Common Subsequence

Algorithm 36 longestCommonSubsequence

```
1:  function LCS(X[1..n], Y[1..m])
2:      for i = 0 to n + 1 do T[i][0] ← 0
3:      for j = 0 to m + 1 do T[0][j] ← 0
4:      for i = 1 to n + 1 do
5:          for j = 1 to m + 1 do
6:              if X_i = Y_j then
7:                  T[i][j] ← 1 + T[i − 1][j − 1]
8:              else
9:                  T[i][j] ← Max(T[i − 1][j], T[i][j − 1])
10:     return T[n][m]
```

สตริง X และ Y จะเริ่มตัวแรกที่ตำแหน่ง 1 ซึ่งต่างจากสตริงในภาษา C/C++ ดังนั้นเมื่อนำไปเขียนโปรแกรมจริง จะต้องระวังเรื่องตำแหน่งของตัวอักษรให้ถูกต้อง เวลาการทำงานจะเป็น Θ(nm) หรือเท่ากับการเติมตาราง T ทั้งหมดนั่นเอง

Assignment 5

เขียนโปรแกรมที่รับสตริงทั้งหมด 2 สตริง และแสดงค่าของ Longest Common Subsequence ของทั้งสองตัว

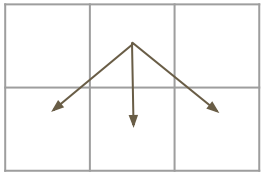
Assignment 6

เขียน recurrence relation ของปัญหาต่อไปนี้ พร้อมทั้งวิเคราะห์ความซับซ้อนของเวลาในการทำงาน

1. Friends Pairing - มีกลุ่มเพื่อน n คน สามารถแบ่งเป็นกลุ่มละ 1 หรือ 2 คนได้ทั้งหมดกี่แบบ เช่น n = 3 สามารถแบ่งเป็น A/B/C, AB/C, AC/B, A/BC ทั้งหมด 4 แบบ
2. Subset Sum - ตอบแค่ True/False ว่ามี subset ที่มีผลรวมเท่ากับ k หรือไม่

Assignment 7

เกม Puzzle หนึ่งต้องเริ่มเดินจากแถวบนสุด (ช่องใดก็ได้) และลงมายังแถวล่างสุด (ช่องได้ก็ได้) โดยที่ผลรวมของค่าแต่ละช่องที่เดิมผ่านมากที่สุด เขียนโปรแกรมสำหรับหาผลรวมที่มากที่สุด เงื่อนไขการเดินคือสามารถเดินลงไปยังช่องด้านล่างตามรูปเท่านั้น



34	21	22	34
45	70	43	65
25	62	15	26
15	19	32	24
30	60	50	80

Matrix Chain Multiplication

กำหนดให้เมทริกซ์ A ขนาด n x m และเมทริกซ์ B ขนาด m x p

ให้ C เป็นผลคูณของเมทริกซ์ A และ B (C = A x B)

การคูณเมทริกซ์ขนาด n x m และ m x p จะเกิดการคูณสเกลาร์ทั้งหมด n x m x p และได้ผลลัพท์เป็นเมทริกซ์ขนาด n x p

$$\overset{4\times 2 \text{ matrix}}{\begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix}} \overset{2\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix}} = \overset{4\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & c_{12} & c_{13} \\ \cdot & \cdot & \cdot \\ \cdot & c_{32} & c_{33} \\ \cdot & \cdot & \cdot \end{bmatrix}}$$

https://en.wikipedia.org/wiki/Matrix_multiplication

Matrix Chain Multiplication

หากต้องการคูณเมทริกซ์ $A_1A_2A_3...A_n$ ที่มีขนาดเท่ากับ $p_0 \times p_1$, $p_1 \times p_2$, $p_2 \times p_3$, ..., $p_{n-1} \times p_n$ จะเกิดการคูณสเกลาร์กี่ครั้ง ?

เนื่องจากการคูณเมทริกซ์มีคุณสมบัติการเปลี่ยนหมู่ (Associative Property) ซึ่งหมายถึงว่าไม่ว่าจะคูณคู่ใดก่อนก็จะให้คำตอบที่เท่ากัน ดังนั้นเราจะได้ว่า

$$(A(BC)) = ((AB)C)$$

จำนวนการคูณสเกลาร์ที่เกิดขึ้นของทั้งสองแบบเท่ากันหรือไม่ ?

Matrix Chain Multiplication

สมมติเมทริกซ์

A ขนาด 5 x 100

B ขนาด 100 x 10

C ขนาด 10 x 50

(A(BC))

B x C 100 x 10 x 50 = 50,000 => 100 x 50

A x (BC) 5 x 100 x 50 = 25,000

50,000 + 25,000 = 75,000

((AB)C)

A x B 5 x 100 x 10 = 5,000 => 5 x 10

(AB) x C 5 x 10 x 50 = 2,500

5,000 + 2,500 = 7,500

Matrix Chain Multiplication

ปัญหา Matrix Chain Multiplication

หากต้องการคูณเมทริกซ์ $A_1A_2A_3...A_n$ ที่มีขนาดเท่ากับ $p_0 \times p_1$, $p_1 \times p_2$, $p_2 \times p_3$, ..., $p_{n-1} \times p_n$ จะเกิดการคูณสเกลาร์น้อยที่สุดกี่ครั้ง ถ้าทำการจัดกลุ่มการคูณอย่างถูกต้อง

Matrix Chain Multiplication

ให้ M(i,j) เป็นจำนวนการคูณสเกลาร์ที่น้อยที่สุดของการคูณเมทริกซ์ A_i จนถึง A_j

คำตอบที่ต้องการคือ M(1,n)

Matrix Chain Multiplication

Base case

มีเมทริกซ์เพียงตัวเดียว จะไม่เกิดการคูณใดๆ

$$M(i,j) = 0 \qquad \text{if } i = j$$

Matrix Chain Multiplication

General case

หาจำนวนการคูณที่น้อยที่สุดของการคูณเมทริกซ์ตัวที่ i ถึง j ด้วยการลองทดสอบแบ่งครึ่งการคูณทุกแบบและดูว่าวิธีการแบ่งแบบไหนที่ทำให้เกิดการคูณน้อยที่สุด

$$M(i,j) = \min_{i \le k < j} M(i,k) + M(k+1,j) + (p_{i-1} \times p_k \times pj), \qquad \text{if } i < j$$

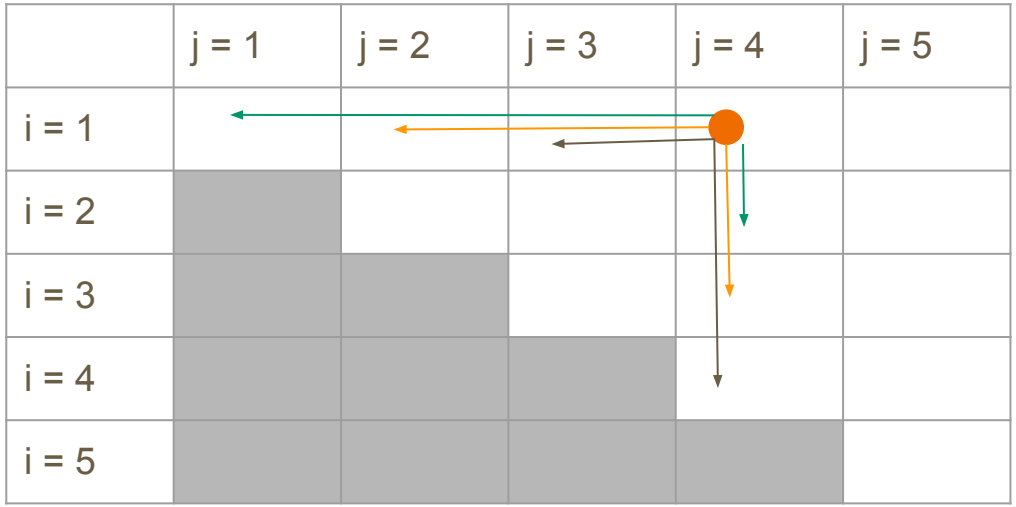
Matrix Chain Multiplication

Algorithm 38 MatrixChainMultiplication

```
1:  function MCM(p[0..n])
2:      for i = 1 to n do M[i][i] ← 0
3:      for L = 2 to n do                                   ▷ L แทนความยาวของ chain
4:          for i = 1 to n − L + 1 do
5:              j ← i + L − 1
6:              M[i][j] ← ∞
7:              for k ← i to j - 1 do
8:                  q ← M[i][k] + M[k + 1][j] + (p[i − 1] × p[k] × p[j])
9:                  if q < M[i][j] then
10:                     M[i][j] ← q
11:     return M[1][n]
```

Matrix Chain Multiplication

A_1 ขนาด 2 x 4
A_2 ขนาด 4 x 2
A_3 ขนาด 2 x 1
A_4 ขนาด 1 x 5
A_5 ขนาด 5 x 2

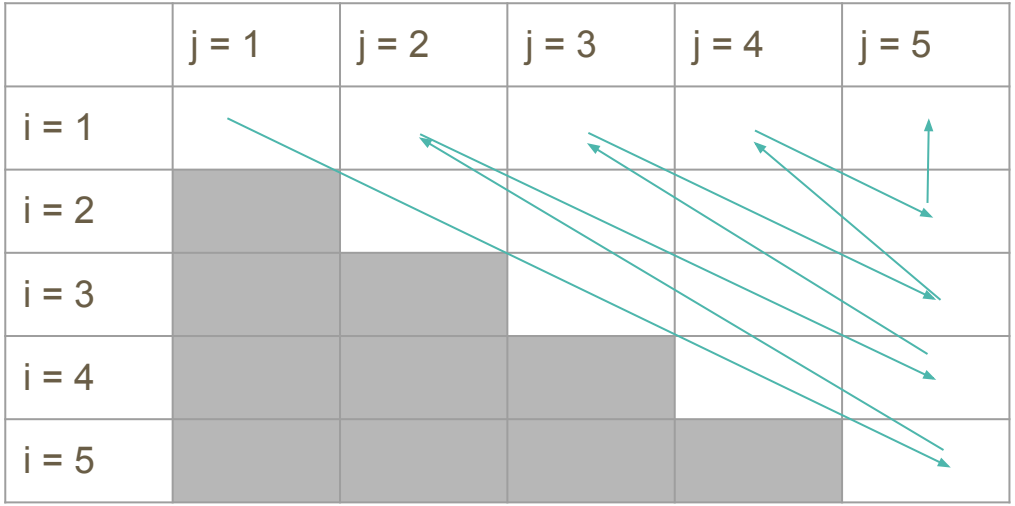


M(1,4) จะต้องใช้ข้อมูลจาก M(1,1), M(2,4), M(1,2), M(3,4), M(1,3), M(4,4)

$$M(i,j) = \min_{i \le k \le j} M(i,k) + M(k+1,j) + (p_{i-1} \times p_k \times pj), \qquad \text{if } i < j$$

Matrix Chain Multiplication

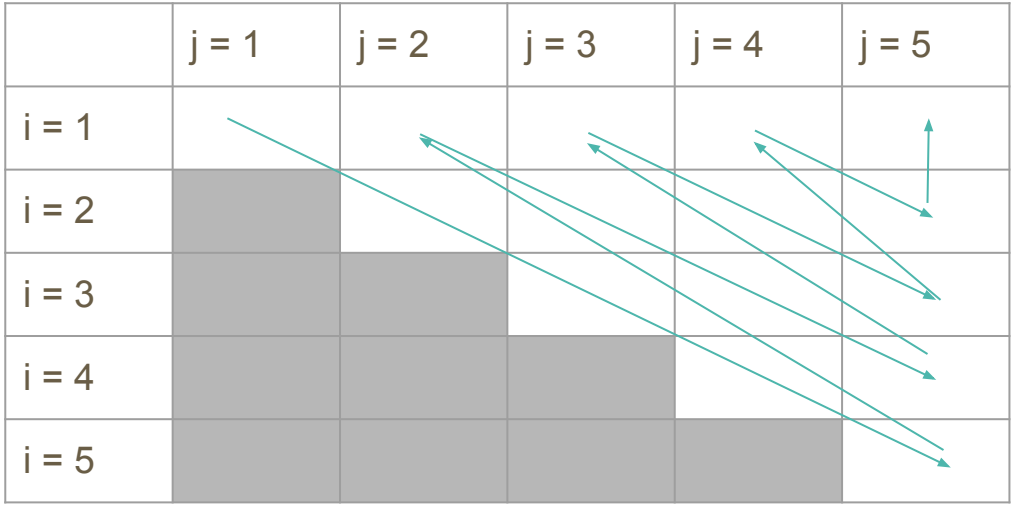
A_1 ขนาด 2 x 4
A_2 ขนาด 4 x 2
A_3 ขนาด 2 x 1
A_4 ขนาด 1 x 5
A_5 ขนาด 5 x 2



หาคำตอบเป็นแนวทแยงจากด้านล่างก่อน เพื่อให้สามารถหาคำตอบในชั้นบนถัดไปได้

Matrix Chain Multiplication

A_1 ขนาด 2 x 4
A_2 ขนาด 4 x 2
A_3 ขนาด 2 x 1
A_4 ขนาด 1 x 5
A_5 ขนาด 5 x 2



มีลำดับแบบอื่นในการคำนวณเพื่อให้ได้คำตอบหรือไม่?

Longest Increasing Subsequence

increasing subsequence คือ subsequence ที่ยาวที่สุดที่มีค่ามากขึ้นเรื่อยๆ สำหรับอาร์เรย์ A ความยาว n

2	12	8	0	23	11	52	31	51	59	80	22

Longest Increasing Subsequence

ให้ L(i) เป็นความยาวของ LIS ที่มี i เป็นตัวสุดท้ายใน subsequence

LIS ของอาร์เรย์ความยาว n หาได้จาก

$$\max_{1 \leq i \leq n} L(i)$$

Longest Increasing Subsequence

Base case

กรณีที่มีตัวเดียว (ตัวแรกเริ่มนับที่ 1)

$L(1) = 1$

General case

ไปดูว่า L(k) เมื่อ k < i มีตัวไหนที่มีค่ามากที่สุด ภายใต้เงื่อนไขว่า A[k] < A[i]

$L(i) = 1 + \max(L(k))$ where $1 \leq k < i$ and $A[k] < A[i]$;

$L(i) = 1$, if no such k exists.

Longest Increasing Subsequence

เติมตาราง n ช่อง

แต่ละช่องใช้เวลา ~n

Time complexity : $\Theta(n^2)$

Space complexity : $\Theta(n)$

ทำให้เร็วกว่านี้ได้หรือไม่?

Observations

กำหนดอาร์เรย์ [2, 5, 4, 8, 1, 9, 3]

(1) [2] - Candidate list
[2]

(2) [2, 5] - Candidate list
[2]
[5]
[2, 5]

(3) [2, 5, 4] - Candidate list
[2]
[4]
[5]
[2, 4]
[2, 5]

(4) [2, 5, 4, 8] - Candidate list
[2]
[4]
[5]
[8]
[2, 4]
[2, 5]
[2, 8]
[2, 4, 8]
[2, 5, 8]

...

* Candidate list แสดง Increasing Subsequence ทั้งหมด เมื่อพิจารณาอาร์เรย์ (ย่อย) ที่กำหนดให้

Observations

ในชุดของ Candidate list มีบางอาร์เรย์ที่ dominate ตัวอื่น

(1) [2] - Candidate list
[2]

(2) [2, 5] - Candidate list
[2]
[5]
[2, 5]

(3) [2, 5, 4] - Candidate list
[2]
[4]
[5]
[2, 4]
[2, 5]

(4) [2, 5, 4, 8] - Candidate list
[2]
[4]
[5]
[8]
[2, 4]
[2, 5]
[2, 8]
[2, 4, 8]
[2, 5, 8]

- ใน candidate list ตัวที่มีความยาวเท่ากันจะมีตัวที่ดีที่สุดอยู่
- ในความยาวเท่ากันเก็บแค่ตัวเดียวที่ตัวสุดท้ายน้อยที่สุด

Observations

กำหนดอาร์เรย์ [2, 5, 4, 8, 1, 9, 3]

(1) [2] - Candidate list
[2]

(2) [2, 5] - Candidate list
[2]
[2, 5]

(3) [2, 5, 4] - Candidate list
[2]
[2, 4]

(4) [2, 5, 4, 8] - Candidate list
[2]
[2, 4]
[2, 4, 8]

(5) [2, 5, 4, 8, 1] - Candidate list
[1]
[2, 4]
[2, 4, 8]

(6) [2, 5, 4, 8, 1, 9] - Candidate list
[1]
[2, 4]
[2, 4, 8]
[2, 4, 8, 9]

Observations

กำหนดอาร์เรย์ [2, 5, 4, 8, 1, 9, 3]

(1) [2] - Candidate list
[2]

(2) [2, 5] - Candidate list
[2]
[2, 5]

(3) [2, 5, 4] - Candidate list
[2]
[2, 4]

(4) [2, 5, 4, 8] - Candidate list
[2]
[2, 4]
[2, 4, 8]

(5) [2, 5, 4, 8, 1] - Candidate list
[1]
[2, 4]
[2, 4, 8]

(6) [2, 5, 4, 8, 1, 9] - Candidate list
[1]
[2, 4]
[2, 4, 8]
[2, 4, 8, 9]

Case 1: A[i] น้อยกว่าสมาชิกตัวที่มีความยาว 1
ให้สร้าง list ใหม่แทน list ความยาว 1 เดิม

Case 2: A[i] มากกว่าตัวสุดท้ายของ list ที่ยาวที่สุด
สร้าง list ใหม่ที่ copy จาก list ที่ยาวที่สุด และนำ A[i] เข้าไปต่อ

Case 3: กรณีอื่นๆ
หาว่า A[i] น้อยกว่าตัวสุดท้ายของ list ใด ให้ copy list ที่สั้นกว่าอยู่ 1 และนำ A[i] เข้าไปต่อ

Observations

ไม่จำเป็นที่จะต้องเก็บ list ทั้งหมด สามารถเก็บแค่ความยาว และค่าของตัวสุดท้าย

(1) [2] - Candidate list

[2]	(1, 2)

(2) [2, 5] - Candidate list

[2]	(1, 2)
[2, 5]	(2, 5)

(3) [2, 5, 4] - Candidate list

[2]	(1, 2)
[2, 4]	(2, 4)

(4) [2, 5, 4, 8] - Candidate list

[2]	(1, 2)
[2, 4]	(2, 4)
[2, 4, 8]	(3, 8)

(5) [2, 5, 4, 8, 1] - Candidate list

[1]	(1, 1)
[2, 4]	(2, 4)
[2, 4, 8]	(3, 8)

(6) [2, 5, 4, 8, 1, 9] - Candidate list

[1]	(1, 1)
[2, 4]	(2, 4)
[2, 4, 8]	(3, 8)
[2, 4, 8, 9]	(4, 9)

ค่าของตัวสุดท้ายจะเรียงกันเสมอ (ใช้ binary search ในการหาได้)

Longest Increasing Subsequence

Algorithm 39 logLinearTimeLongestIncreasingSubsequence

```
1: function LIS(A[1..n])
2:     L[1] ← A[1]
3:     LISlen ← 0
4:     for i = 2 to n + 1 do
5:         k ← BinarySearch(L, 1, i − 1, A[i])      ▷ ค้นหา A[i] ใน L ที่ตำแหน่ง 1 ถึง i − 1 (inclusion)
6:         L[k] ← A[i]
7:         if k > LISlen then
8:             LISlen ← k
9:     return LISlen
```

Longest Increasing Subsequence

- สร้างอาร์เรย์ L ความยาว n สำหรับเก็บตัวสุดท้ายของ candidate list
- วนลูป n รอบ แต่ละรอบค้นหาค่าใน L ด้วย binary search (O(logn))

Time complexity : O(nlogn)

Space complexity : Θ(n)

References

1. Ahmed Shamsul Arefin.Art of Programming Contest. 2006.
2. Anany Levitin.Introduction to The Design & Analysis of Algorithms. Pearson Education, 2nd edition, 2007.
3. Steven S. Skiena.The Algorithm Design Manual. Springer, 2nd edition, 2008.
4. Jon Kleinberg & Éva Tardos.Algorithm Design. Pearson Education, 2006.
5. สมชาย ประสิทธิ์จูตระกูล.การออกแบบและวิเคราะหอัลกอริทึม. ภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย, พิมพ์ครั้งที่ 4, 2553