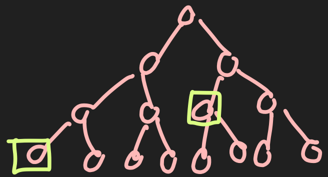
Dynamic Programming

Divide & Conquer + $\boxed{\text{Lookup Table}}$ ส่วนใหญ่เป็น array

Nattee Niparnan

Overview



แก้แล้วไม่ทำซ้ำ (Dynamic Programming)

- The key idea of Divide & Conquer is to break a problem into smaller sub-problems and combine the result of those subproblems
- Some Problem can be divided into subproblems that is $\boxed{\text{overlapping}}$, i.e., same subproblem that happens more than once

  overlapping subproblem

  - If we use general D&C, each copies of the same subproblem will be solved repeatedly, wasting time
  - Dynamic Programming is a technique that use a look up table to store result of each sub-problem and immediately use it if any subproblem is required multiple times

Fibonacci Number

Fibonacci Number

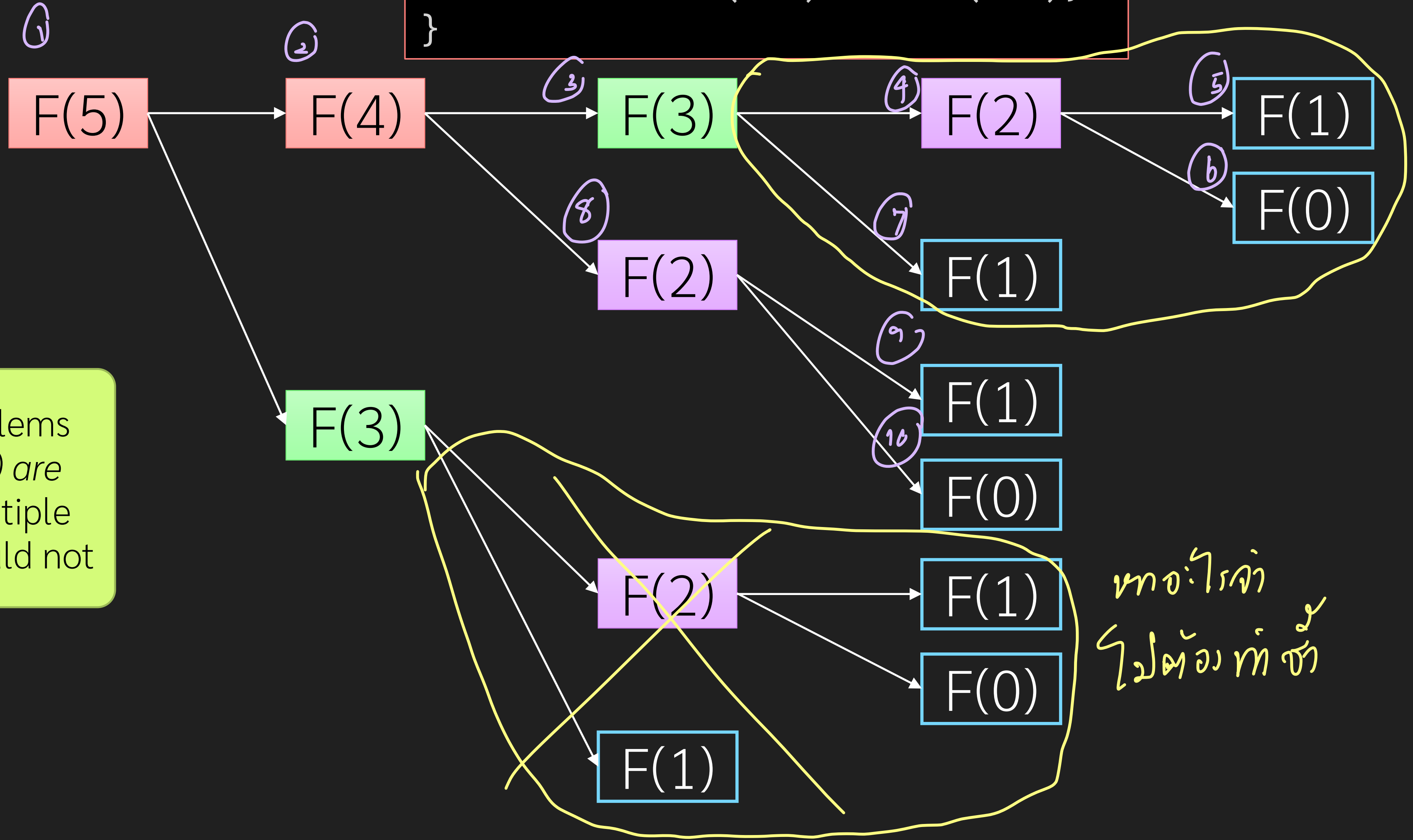
- Problem: compute F(N), the Fibonacci function of N
- Input:
  - An integer N >= 0
- Output:
  - F(N), according to

$$F(N) = \begin{cases} F(N-1) + F(N-2) & ; n > 1 \\ 1 & ; n = 1 \\ 0 & ; n = 0 \end{cases}$$

Can be solve directly using Divide & Conquer

Recursion Tree

```
int fibo(int n) {
  if (n == 1 || n == 0)
    return n;
  if (n >= 2)
    return fibo(n-1) + fibo(n-2);
}
```



Some subproblems *(F(3) and F(2)) are* computed multiple times, they should not

หาอะไรจ้า
ไปต้องทำซ้ำ

Memoization: Simplest form of Dynamic Programming

ท้ !

- Top-Down approach
- Remember what have been done, if the subproblem is needed again, use the remembered result

```
ResultType DC(Problem p) {
  if (p is trivial) {
    solve p directly
    return the result
  } else {


    divide p into p₁,p₂,...,pₙ
    for (i = 1 to n)
      rᵢ = DC(pᵢ)
    combine r₁,r₂,...,rₙ into r

    return r
  }
}
```

```
ResultType DP(Problem p) {
  if (p is trivial) {
    solve p directly
    return the result
  } else {
    if p is solved
      return table.lookup(p);                 use
    divide p into p₁,p₂,...,pₙ
    for (i = 1 to n)
      rᵢ = DP(pᵢ)
    combine r₁,r₂,...,rₙ into r
    table.save(p,r);           save      remember
    return r
  }
}
```

Fibonacci: Top-Down DP

ประกาศเป็น global

- table is an array[1..n] initialized by 0

```
int fibo_memo(int n) {
  if (n == 1 || n == 0)
    return n;

  if (n >= 2) {
    if (table[n] > 0) {
      return table[n];
    }
    int value = fibo_memo(n-1) + fibo_memo(n-2);
    table[n] = value;
    return value;
  }
}
```

ไม่ต้องกลัว table[n] = 0

use

remember

Exercise

- Draw recursion tree when we call $\boxed{\text{fibo\_memo(7)}}$ = 13

```
//table is a global variable
int fibo_memo(int n) {
  if (n == 1 || n == 0)
    return n;

  if (n >= 2) {
    if (table[n] > 0) {
      return table[n];
    }
    int value = fibo_memo(n-1) + fibo_memo(n-2);
    table[n] = value;
    return value;
  }
}
```

→ $\boxed{2^n}$ ถ้าไม่วาดละเอียด = 2n

Top-down จะผูกกับ DC 100%

Bottom-up dynamic programming

- Instead of relying on recursion to discover repetition of subproblems, we analyze the recursion directly and build table constructively from smaller subproblems
  - The initial subproblems are the ones from trivial case of Divide & Conquer recurrent relation

มักจะเร็วกว่า

วิเคราะห์ loop

- Benefit: no-recursion, better runtime performance, (usually) easier to analyze

เราจะเติมตารางตามที่ใช้

- Drawback: sometime, we build unnecessary sub-problem

Fibonacci: Bottom-Up DP

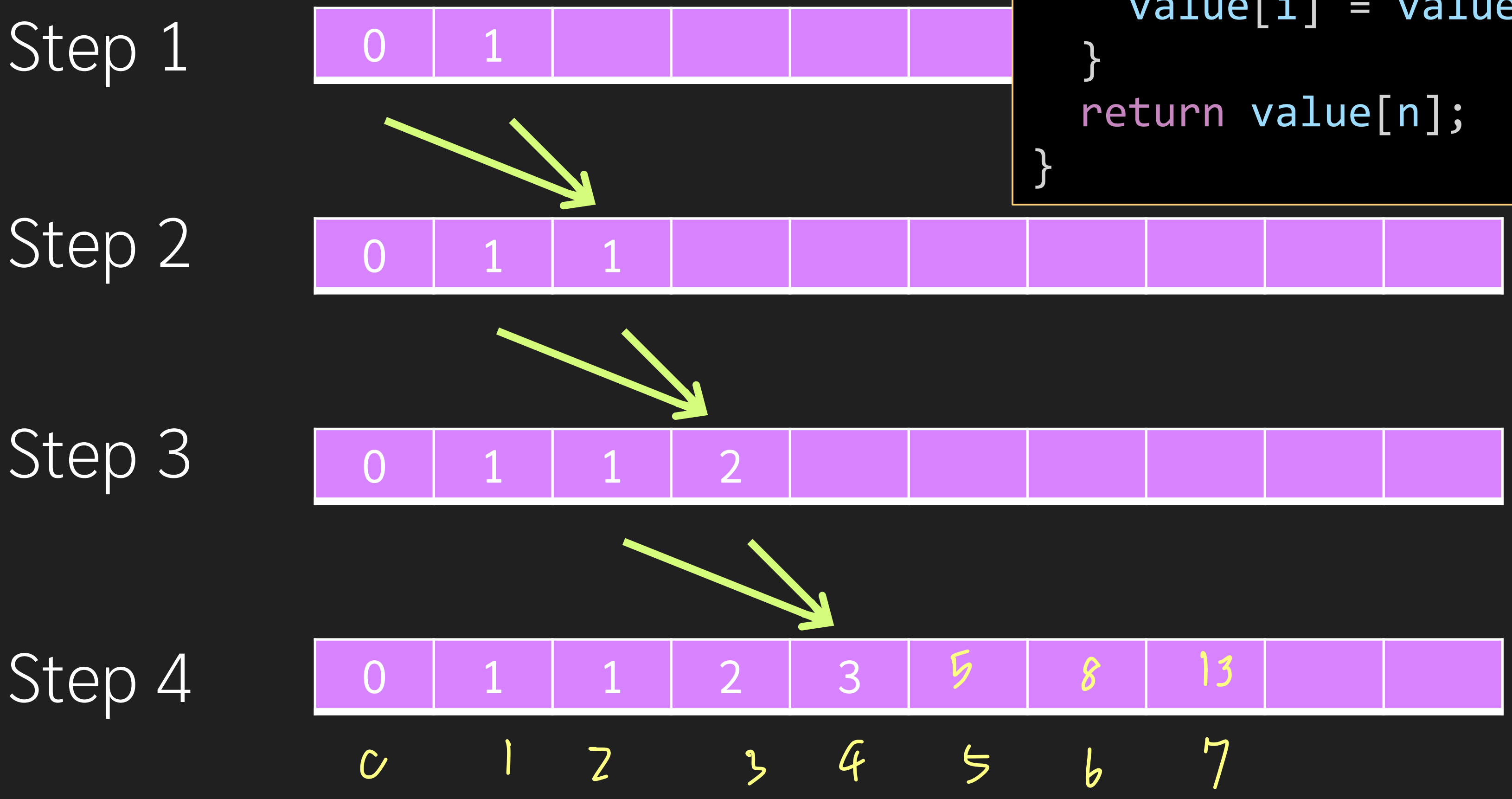
- From the definition of F(N), we know that
  - F(n) needs to know F(n-1) and F(n-2)
  - In other words, if we know F(n-1) and F(n-2), then we can construct F(N)
- Initial Condition:
  - F(0) = 0, F(1) = 1
  - i.e., `table[0] = 0; table[1] = 1;`

    table[2] : [1] + [0]
- From the recurrent
  - `table[3] = table[2] + table[1]`
  - `table[4] = table[3] + table[2]`
  - …

Fibonacci: Bottom Up

```
int fibo_bottom_up(int n) {
  value[0] = 0;
  value[1] = 1;
  for (int i = 2;i <= n;++i) {
    value[i] = value[i-1] + value[i-2];
  }
  return value[n];
}
```

O(n)



สามารถเขียน ทดๆ ฟ้อยได้

Optimized version of Bottom-Up Fibo

- From bottom up approach, we know that we only need two prior Fibonacci numbers (F(n-1) and F(n-2)) to compute the current Fibonacci number (F(n))
    - There is no need to lookup for F(n-3), F(n-4), ... if we know F(n-1), and F(n-2)
    - Hence, no need to use entire table
        - Just remember two previous Fibonacci number

```
def fibo(n)
  if (n == 0 || n == 1)
    return n
  f2 = 0
  f1 = 1
  for i from 2 to n
    #calculate current
    f = f2 + f1
    #prepare f1 and f2 for next round
    f2 = f1
    f1 = f
  end
  return f
end
```

* จำแค่ 2 ตัว (ข้อดี buttom up)

Binomial Coefficient

choose r things from n things

Example 2: Binomial Coefficient

- $C_{n,r}$ =how to choose r things from n things
  - We have a closed form solution
    - $\boxed{C_{n,r} = n!/(r!*(n-r)!)}$
- We also have recurrence relation of $C_{n,r}$

$$\boxed{\begin{aligned} C_{n,r} &= C_{n-1,r} + C_{n-1,r-1} \\ &= 1 \qquad\qquad ; r = 0 \\ &= 1 \qquad\qquad ; r = n \end{aligned}}$$

← หาได้ไหมว่า มีส่วนซ้ำกัน

- What is the subproblem?
- Do we have overlapping subproblem?

- Input:
  - Two integer r and n (0 <= r <= n)
- Output:
  - $C_{n,r}$

Binomial Coefficient

- Each subproblem is represented by 2 numbers, r and n
  - Hence, the table should be 2D

```
int bino_naive(int n,int r) {
  if (r == n) return 1;
  if (r == 0) return 1;

  int result = bino_naive(n-1,r) + bino_naive(n-1,r-1);
  return result;
}
```

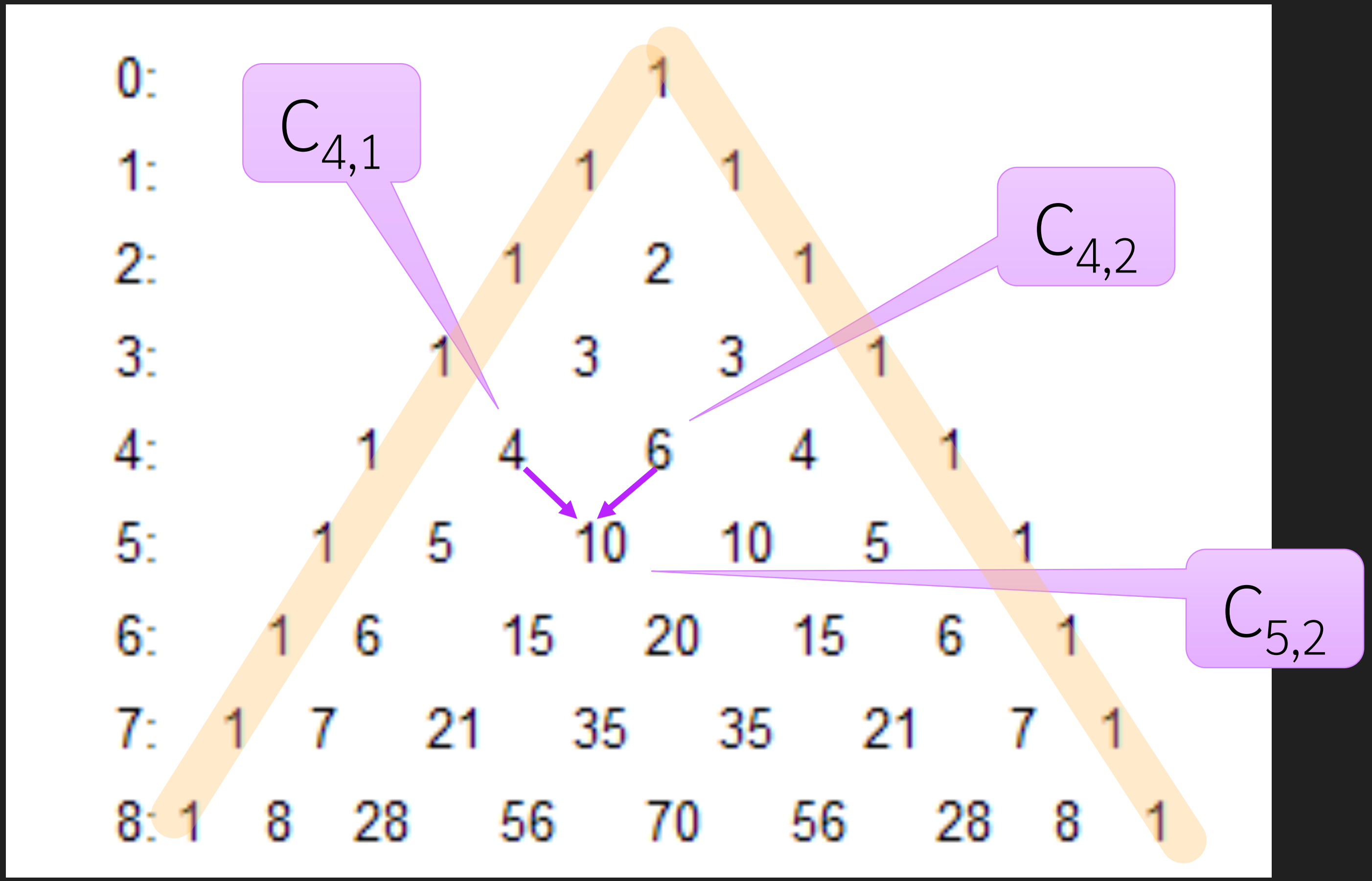
Binomial Coefficient: Top-Down (Memoization)

- table[0..n][0..n] is initialized by -1

```
int bino_memoize(int n,int r) {
  if (r == n) return 1;
  if (r == 0) return 1;

  if (table[n][r] != -1)
    return table[n][r];

  int result = bino_memoize(n-1,r) + bino_memoize(n-1,r-1);
  table[n][r] = result;

  return result;
}
```

Binomial Coefficient: Bottom Up

- Pascal triangle is a by-hand bottom-up DP of Binomial Coeff.



Binomial Coefficient: Bottom Up

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

r : 3

```
int bino_DP(int n,int r) {
  for (int i = 0;i <= n;i++) {
    table[i][0] = 1;
    table[i][i] = 1;
  }
  for (int i = 1;i <= n;i++) {
    for (int j = 1;j < i;j++) {
      table[i][j] = table[i-1][j] + table[i-1][j-1];
    }
  }
  return table[n][r];
}
```

initial condition

min(i,r)

$O(n^2)$

Question

- Is it possible to fill the table in different order?
- Does previous code solve subproblem that we does not need?
  - If yes, how to avoid?

Yep

p&c

$O(n^3) \rightarrow O(n^2)$

$O(n \lg n) \longrightarrow O(n)$

Maximum Subarray Sum

Revisiting

The problem

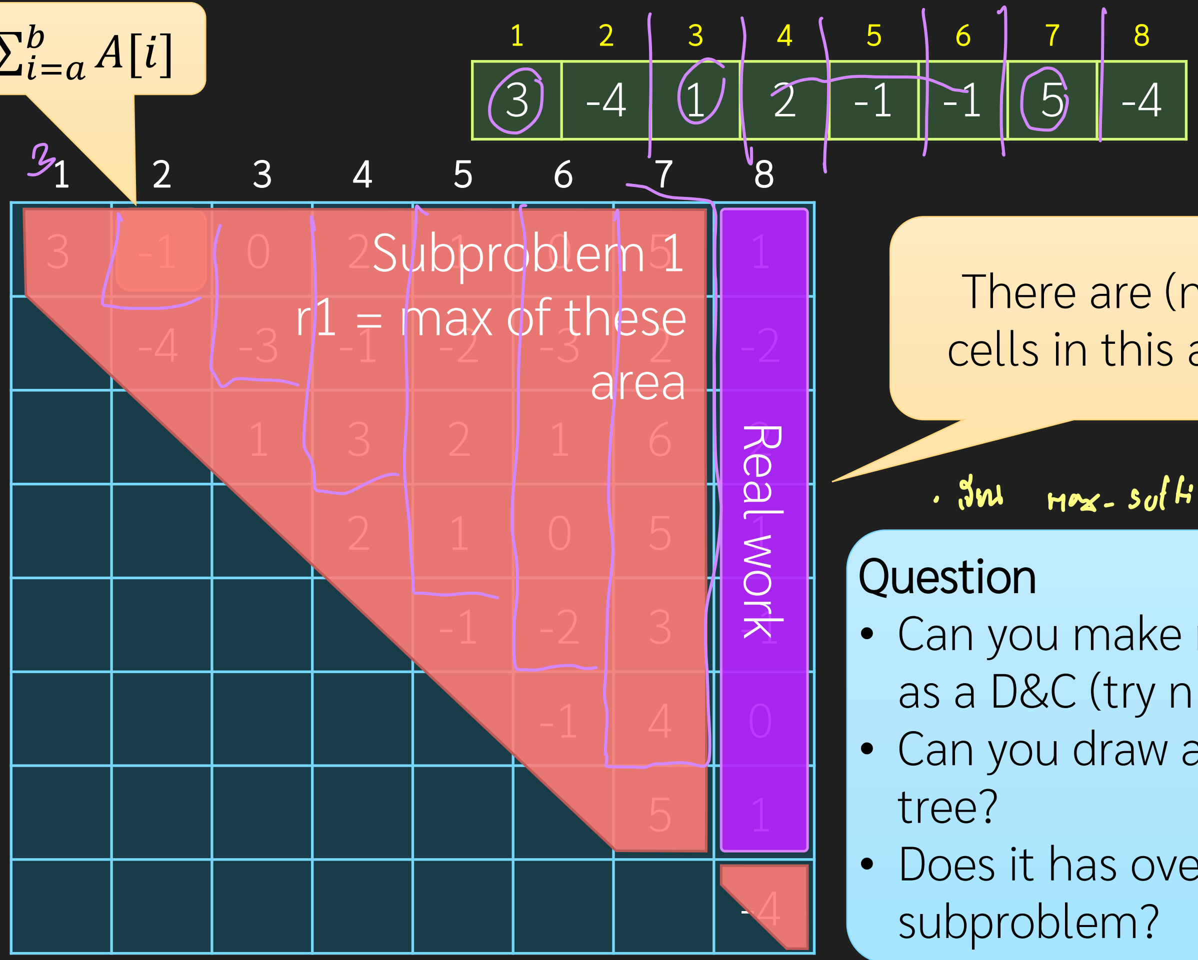
- Given array A[1..n] of numbers, may contain negative number
  - Find a non-empty subarray A[p..q] such that the summation of the values in the subarray is maximum
- Input:
  - A[1..n]
- Output:
  - p and q, where $1 <= p <= q <= n$ and summation of A[p..q] is maximum
- Example:
  - A = [1, 4, 2, 3] output: 1 and 4
  - A = [-2, -1, -3, -5] output: 2 and 2
  - A = [2, 3, -6, 4, -2, 3, -5, -4, 3] output: 4 and 6

$T(n) = T(n-1) + \ldots 1$

D&C by n-1

$B[a][b] = \sum_{i=a}^{b} A[i]$

- Instead of divining n into 2 of n/2 as previously done, we divide by n-1 and 1
  - The real work is solved by another D&C



There are (n-1) cells in this area

$n \rightarrow n-1$

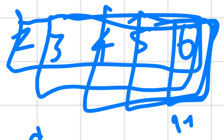
max-suffix D&C ?

Question
- Can you make max_suffix as a D&C (try n -> n-1)
- Can you draw a recursion tree?
- Does it has overlapping subproblem?

```
def mss(A,stop)
  if (stop == 1)
    return A[1]
  r1 = mss(A,stop-1)
  r2 = A[stop]
  r3 = max_suffix(A,stop-1)+A[stop]
  return max(r1,r2,r3)
end
```

```
def max_suffix(A,stop)
  if stop == 1
    return A[1]
  return max(A[stop],
             A[stop]+max_suffix(A,stop-1))
end
```

$$\text{max\_suffix}(A,m) = \max_{1 \le k \le m} \sum_{i=k}^{m} A[i]$$



count : 0

max_sut (A, n, idx, count, max)

count += A[idx]

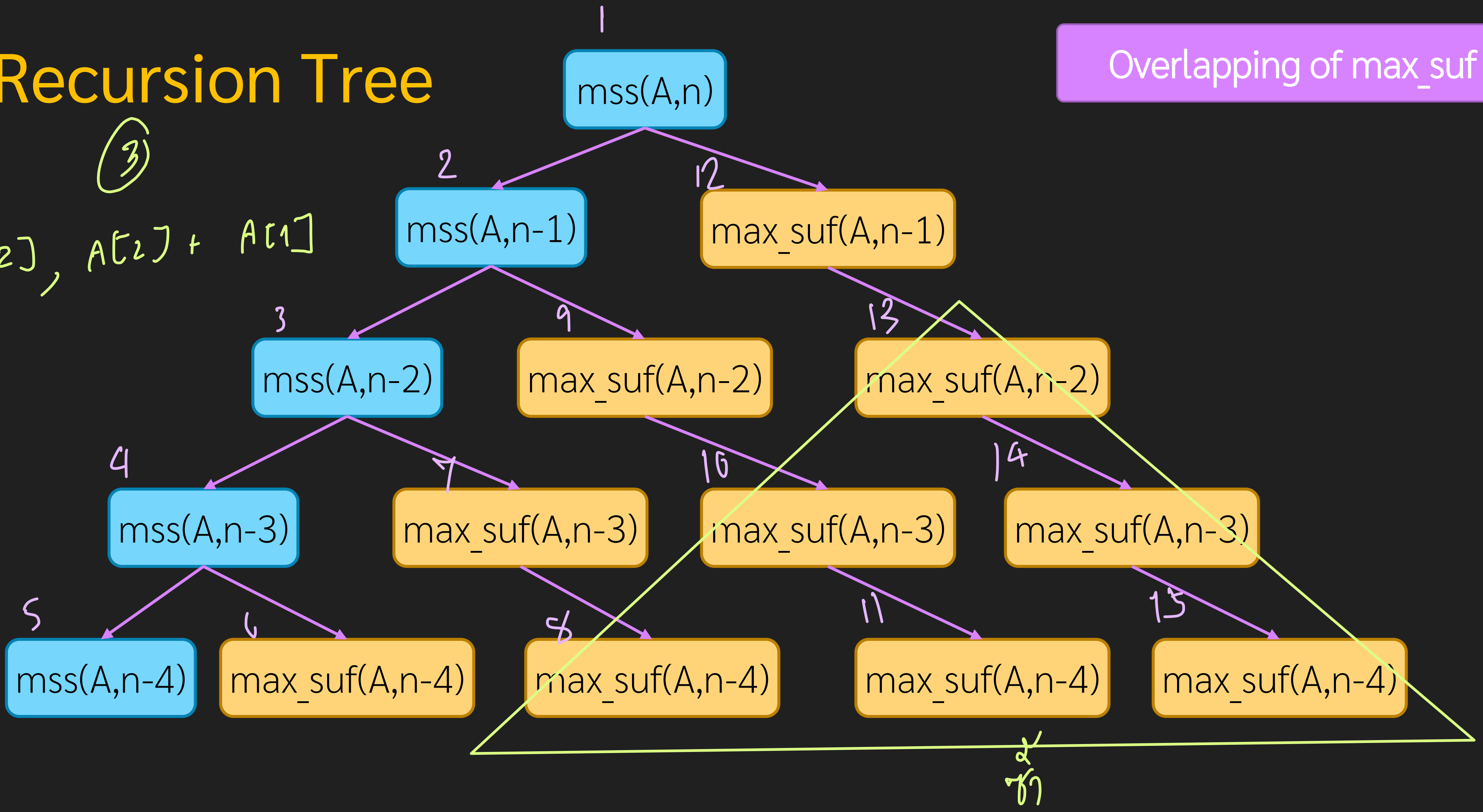
if (idx == n) return A[n]

Recursion Tree

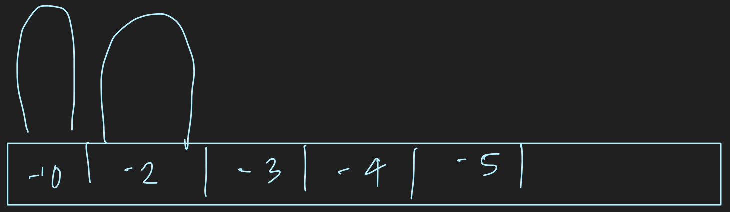
Overlapping of max_suf

3

max(A[2], A[2] + A[1]



MSS with Dynamic Programming




```
def mss(A,stop)
  if (stop == 1)
    return A[1]
  r1 = mss(A,stop-1)
  r2 = A[stop]
  r3 = max_suffix(A,stop-1)+A[stop]
  return max(r1,r2,r3)
end
```

```
def max_suffix(A[1..n],stop,table[1..n],done[1..n])
  if stop == 1
    return A[1]
  if (done[stop])
    return table[stop]

  table[stop] =
    max(A[stop],
        A[stop]+max_suffix(A,stop-1))
  done[stop] = true
  return table[stop]
end
```

- Memoization (top-down) approach
- Since the value of max_suffix can be negative, we need another table to determine whether this subproblem is already solved
  - done[1..n] is initialized as false

i

-10 -2 -3 -4 -5

Bottom-Up approach

- Direct version
  - Build max_sur first
  - Calculate mss from 1 to n
- Optimized version (Kadane's Algorithm)
  - See that we need only one max_suf

เสีย mem สำหรับ array

```
def mss_bottom_up(A[1..n])
  max_suf is array [1..n]
  max_suf[1] = A[1]
  for i from 2 to n
    max_suf[i] = max(max_suf[i-1]+A[i],A[i])

  mss = A[1]
  for i from 2 to n
    mss = max(mss,
              max(A[i],
                  max_suf[i-1]))
  return mss
end
```

Kadane's Algorithm

```
def kadane(A[1..n])
  suf = A[1]
  mss = A[1]
  for i from 2 to n
    suf = max(A[i],suf+A[i])
    mss = max(mss,suf)
  return mss
end
```

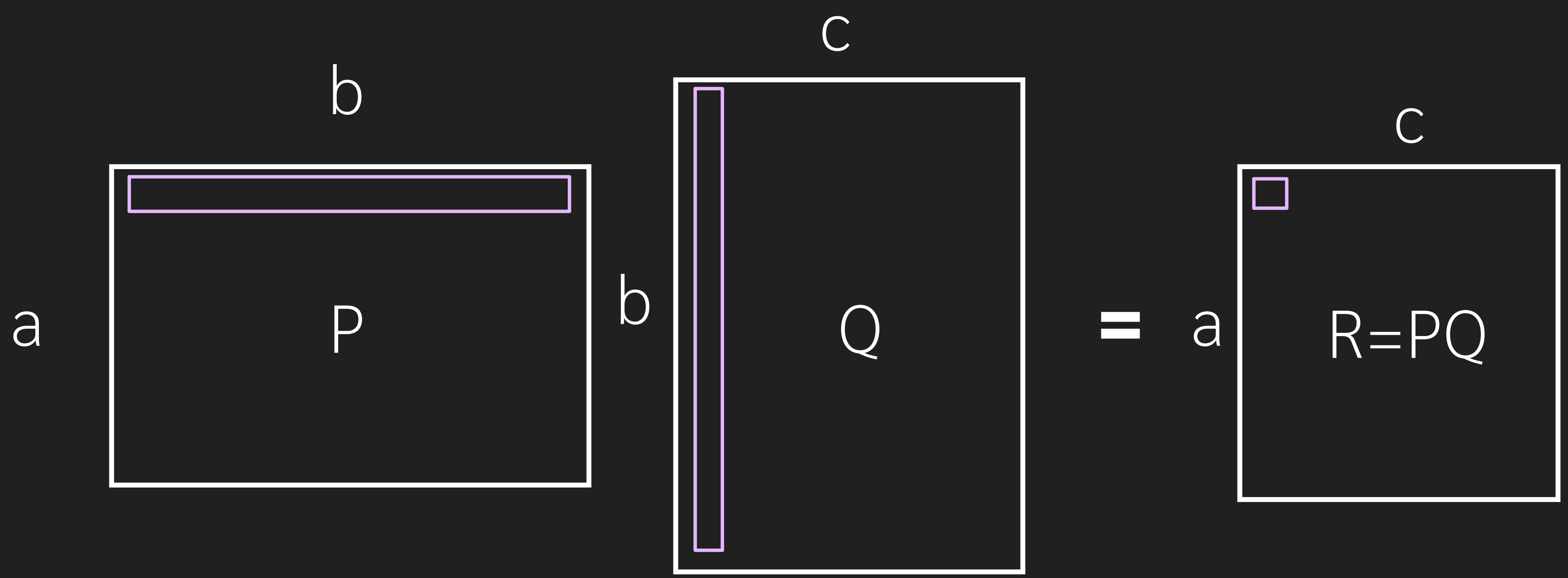
- Calculate both mss and suf on the fly
- Original problem was proposed by Ulf Grenander in 1977
  - Originally 2D problem, convert to 1D to gain insight
- O(n log n) D&C proposed by Michael Shamos
- Joseph Born Kadane heared the problem in a seminar and propose O(n)

MCM

Matrix Chain Multiplication

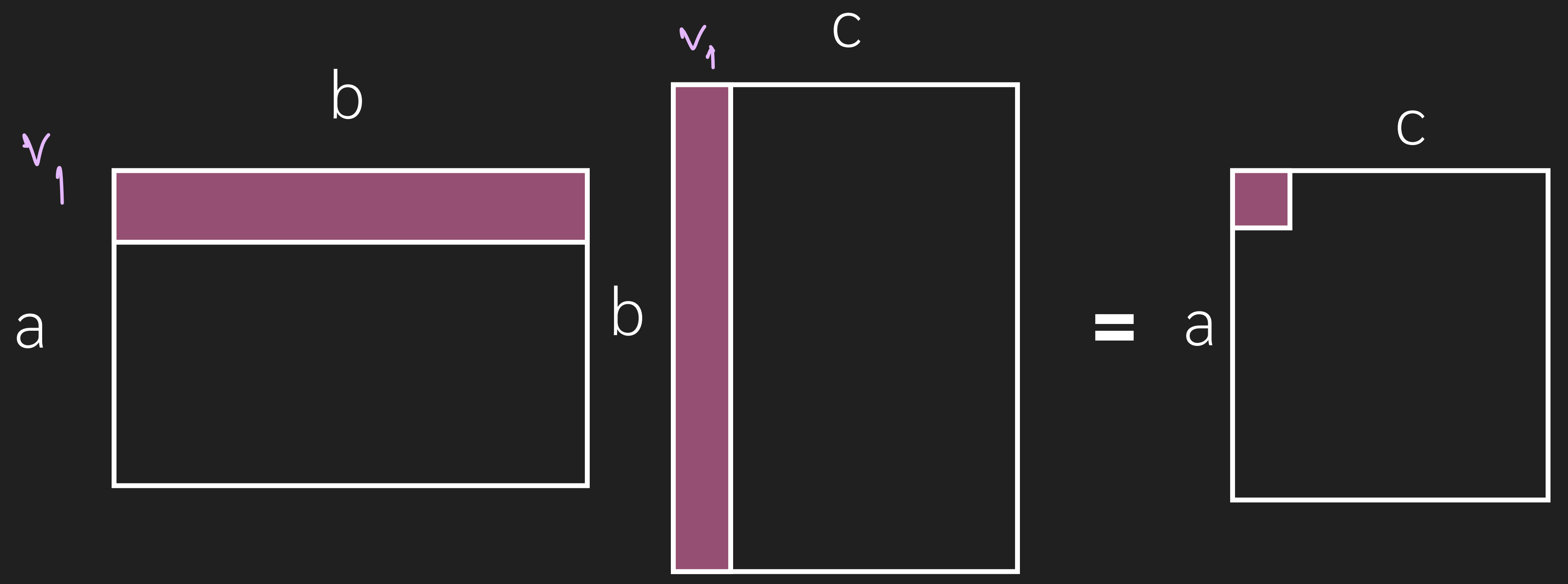
Non-trivial bottom-up

Matrix Multiplication



P = matrix with a rows and b columns
Q = matrix with b rows and c columns

Multiplying the Matrix



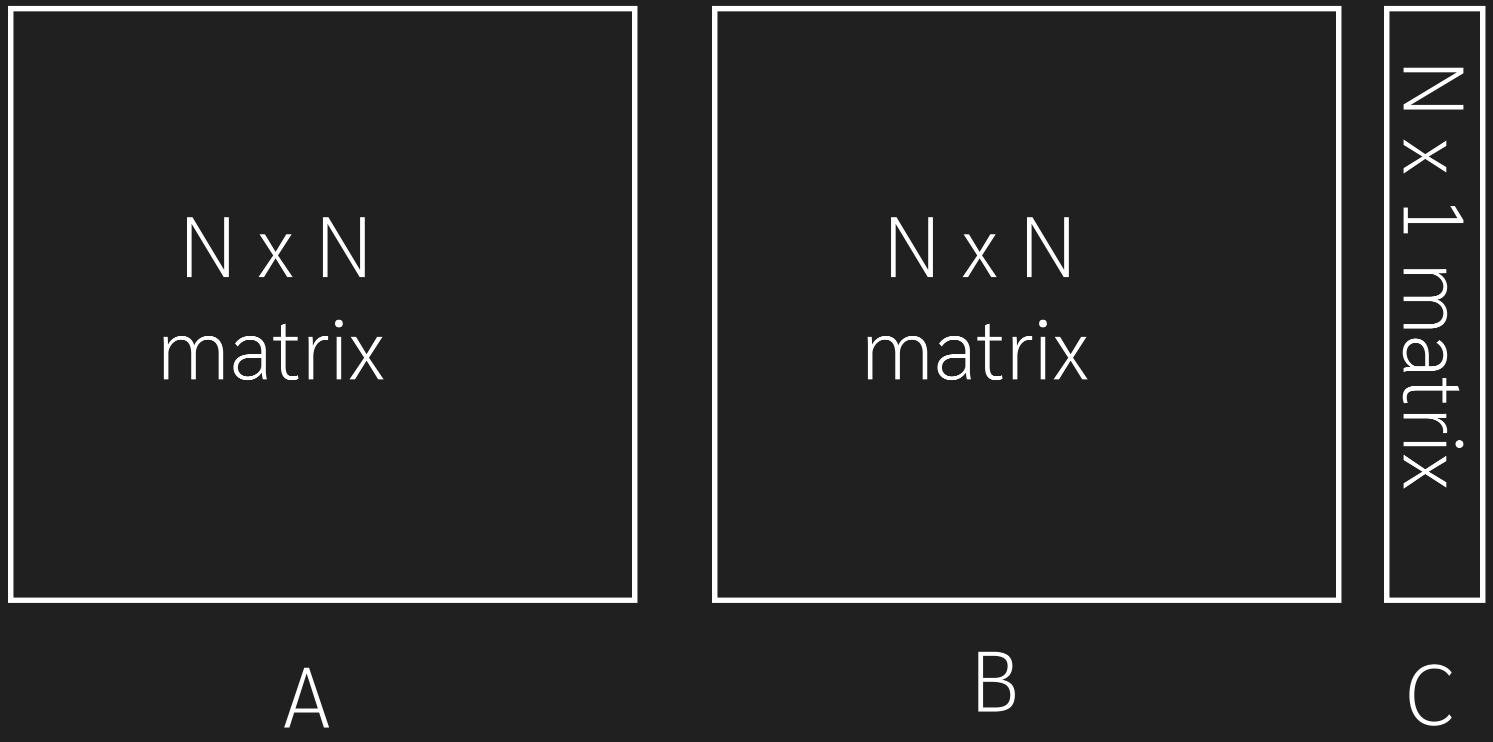
Time used = $\Theta(abc)$

Naïve Method

```
for (i = 1; i <= a;i++) {
  for (j = 1; i <= c;j++) {
      sum = 0;
      for (k = 1;k <= b;k++) {
        sum += P[i][k] * Q[k][j];
      }
      R[i][j] = sum;
  }
}
```

O(abc)

Matrix Chain Multiplication



How to compute ABC ?

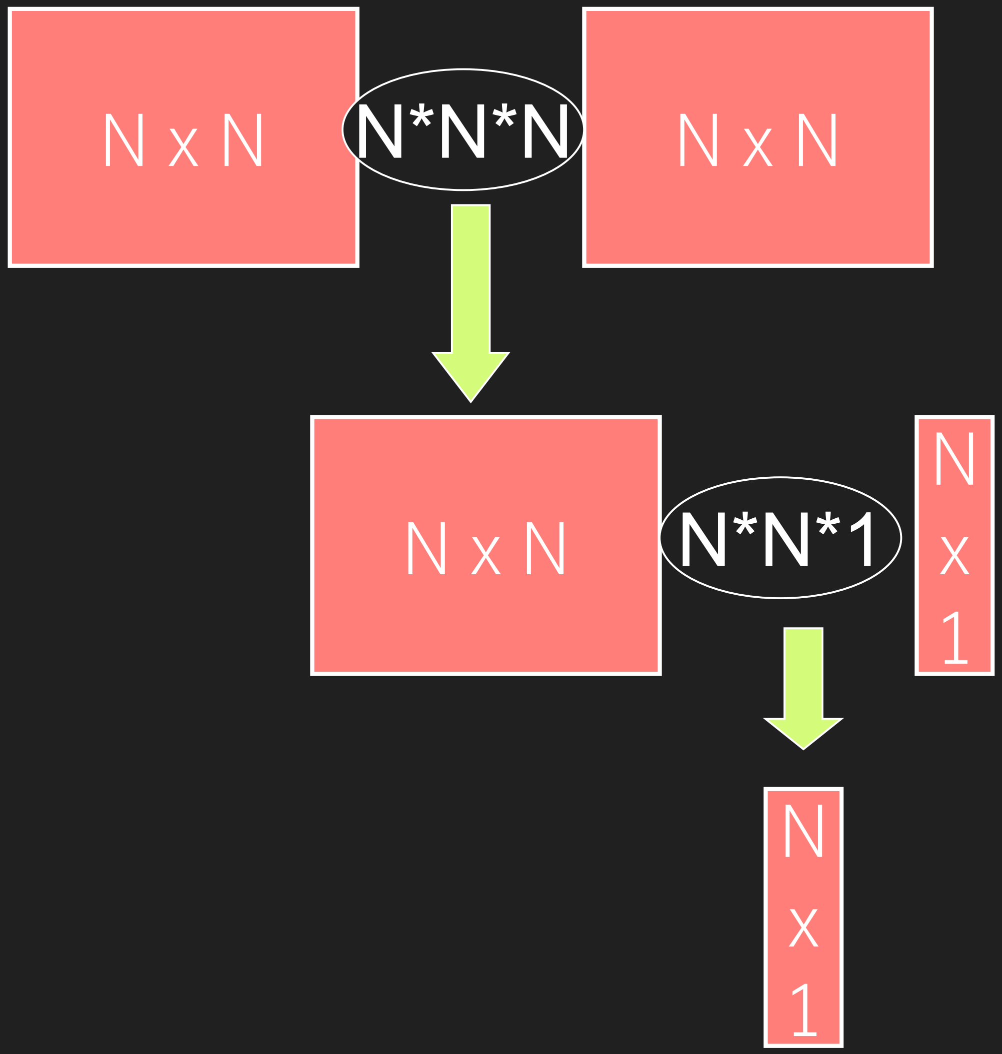
Matrix Multiplication

- ABC = (AB)C = A(BC) ได้ผลลัพธ์เหมือนกัน
- (AB)C differs from A(BC)?
  - Same result, different efficiency
- What is the cost of (AB)C?
- What is the cost of A(BC)?

n n

(AB)C

n n n 1

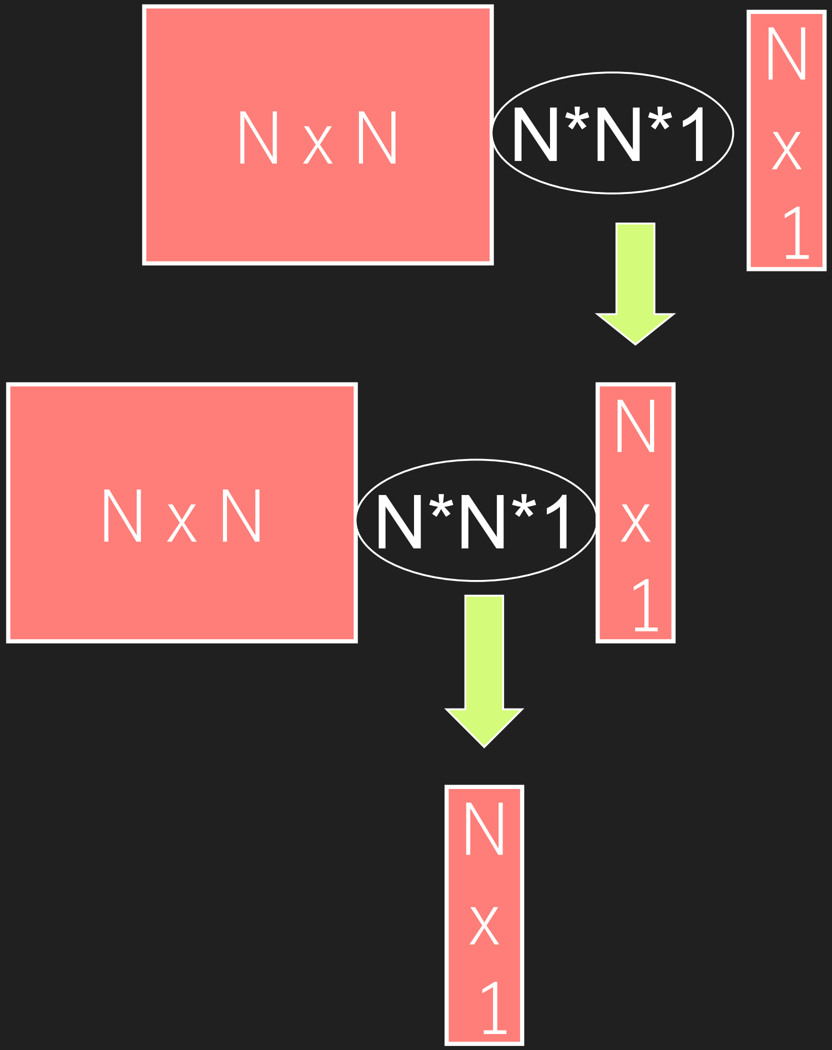


- Total = N^3+N^2

A(BC)

เร็วกว่าเยอะ

- Total = $2N^2$



The Problem

- Input: $B_{1 \ldots n-1}$
  - $a_1, a_2, a_3, \ldots, a_n$
- Output:
  - The order of multiplication
    - How to parenthesize the chain
  - How many multiplication is needed
- Example Instance:
  - Input: 10 10 10 1

Output: $(B_1(B_2B_3))$ 200

These represents the size of the n-1 matrices B_1 .. B_{n-1}

$a_1 \times a_2$	B_1
$a_2 \times a_3$	B_2
$a_3 \times a_4$	B_3
....	
$a_{n-1} \times a_n$	B_{n-1}

10 10 10 1

10 10 (10)(10)(1)

(10)(10)

More Example

INPUT

- $a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$
- $10 \times 5 \times 1 \times 5 \times 10 \times 2$

$B_1 \quad B_2 \quad B_3 \quad B_4 \quad B_5$

Possible Output

$((B_1B_2)(B_3B_4))B_5$

$(B_1B_2)((B_3B_4)B_5)$

$(B_1((B_2B_3)B_4))B_5$

And much more...

Consider the Output

What do

ขั้นสุดท้ายเหมือนกัน

① $(B_1B_2)((B_3B_4)B_5)$

a b c

② $(B_1B_2)(B_3(B_4B_5))$

a b c

have in common?

What do

a

$((B_1B_2)(B_3B_4))B_5$ 1q :c

$(((B_1B_2)B_3)B_4))B_5$

a 1q :c

have in common?

① — $a \cdot b \cdot c$ + แรกไปหาผลคูณของ B_1, B_2

② — + $B_3 .. B_5$

③ $a \cdot q \cdot c$ + แร B_1 B_4

+ B_5 B_5

Solving $B_1\ B_2\ B_3\ B_4 \ldots B_{n-1}$

Min cost of

(1) B_1 Subproblem

(2) Subproblem Subproblem

(3) Subproblem Subproblem

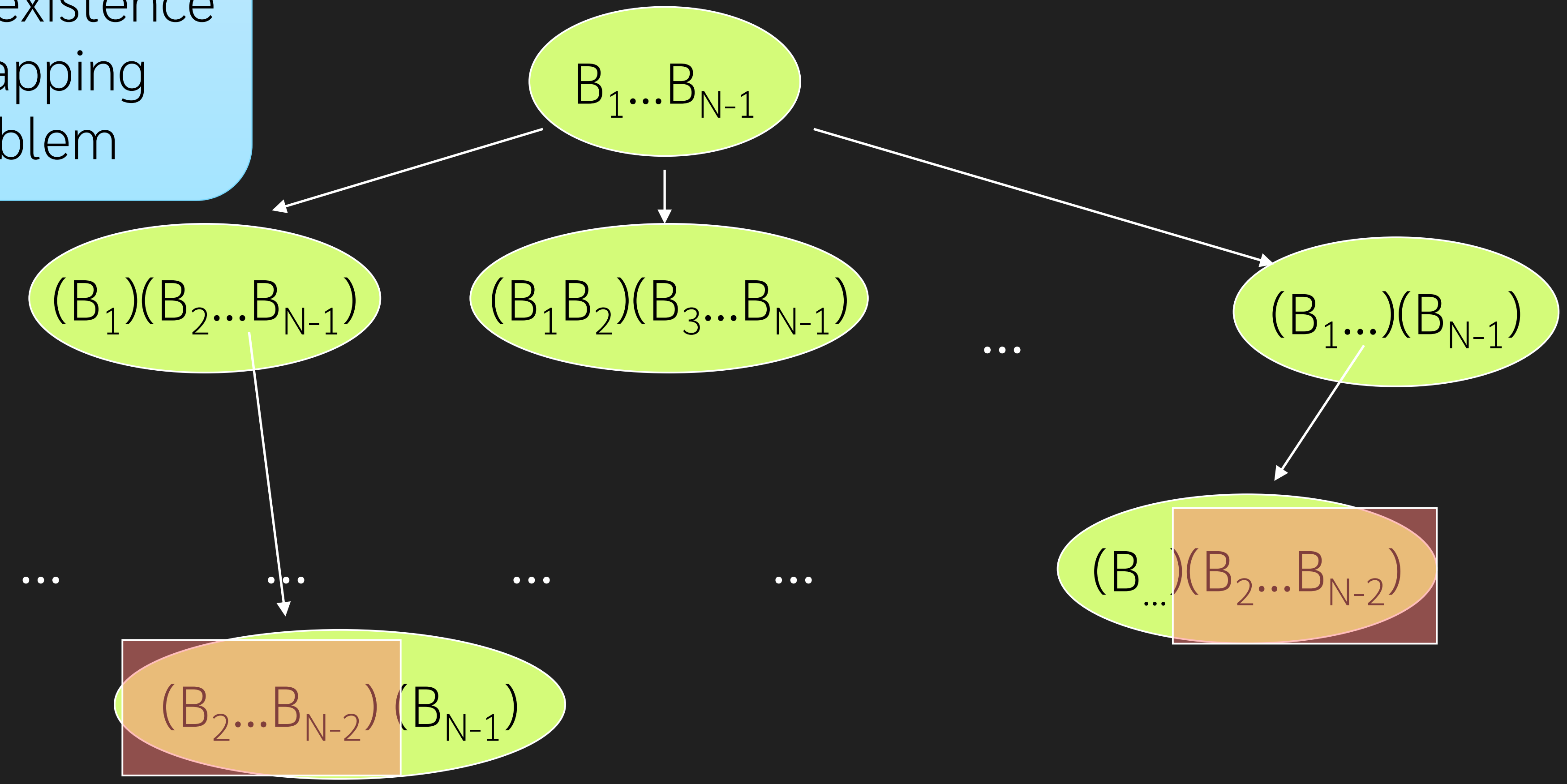
…

(n-2) Subproblem B_{n-1}

- Each options ((1)..(n-2)) has 1 or 2 subproblems
- Sub problem is described by indices of left and right matrix
  - Needs 2 integers to describe a subproblem
- No overlapping subproblem (yet)

Overlapping Subproblem

Have to dig deeper to identify existence of overlapping subproblem



Deriving the Recurrence Relation for D&C

- mcm(l,r) $l \leq r$
  - The least cost to multiply $B_l \dots B_r$

    แรงที่น้อยที่สุดที่ใช้ ไม่ได้สนใจผลลัพธ์การคูณ
- The solution is mcm(1,n-1)

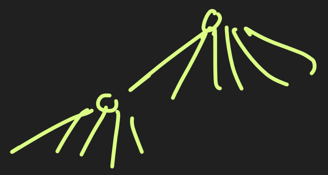
- Initial Case, when (r – l) <= 1 (one or two matrices)
  - mcm(x,x) = 0
  - mcm(x,x+1) = a[x] * a[x+1] * a[x+2]

The Recurrence Relation

- Recursion Case

$$
mcm(l,r) = \text{min of} \begin{cases}
\text{min cost of} & \overbrace{B_l \mid \boxed{mcm(l+1,r)}}^{\text{Subproblems}} & \overbrace{+\ a_l \bullet a_{l+1} \bullet a_{r+1}}^{\text{Final multiplication}} \\
\text{min cost of} & \boxed{mcm(l, l+1)} \mid \boxed{mcm(l+2,r)} & +\ a_l \bullet a_{l+2} \bullet a_{r+1} \\
\text{min cost of} & \boxed{mcm(l, l+2)} \mid \boxed{mcm(l+3,r)} & +\ a_l \bullet a_{l+3} \bullet a_{r+1} \\
\ldots & & \\
\text{min cost of} & \boxed{mcm(l, r-1)} \mid B_r & +\ a_l \bullet\ a_r \bullet a_{r+1}
\end{cases}
$$

Divide & Conquer



```
int mcm(int l,int r) {
  if (l < r) {
    minCost = MAX_INT;
    for (int i = l;i < r;i++) {
      my_cost = mcm(l,i) + mcm(i+1,r) + (a[l] * a[i+1] * a[r+1]);
      minCost = min(my_cost,minCost);
    }
    return minCost;
  } else {
    return 0;
  }
}
```

ลองไปทำ top down ดู

Using bottom-up DP

- Design the table
  - $\boxed{M[i][j]}$ = the best solution (min cost) for multiplying $B_i...B_j$
    - M[i][j] stores mcm(i,j)
  - The solution is at M[1][n-1]
- Trivial Case
  - What is M[x][x] ?
    - No multiplication, M[x][x] = 0
- Simple case
  - What is M[x][x+1] ?
  - $B_x B_{x+1}$
  - Only one solution = $a_x * a_{x+1} * a_{x+2}$

What is M[i,j]?

- General case
  - What is M[x][x+k] ?
  - $B_x B_{x+1} B_{x+2} \ldots B_{x+k}$

$$
\text{min of } \begin{cases}
\overset{1}{M}[\overset{1}{x}][x] + \overset{2}{M}[x+1][\overset{5}{x+k}] & + a_x \bullet a_{x+1} \bullet a_{x+k+1} \\
\overset{1}{M}[x][\overset{2}{x+1}] + \overset{3}{M}[x+2][\overset{5}{x+k}] & + a_x \bullet a_{x+2} \bullet a_{x-k+1} \\
\overset{1}{M}[x][\overset{3}{x+2}] + \overset{4}{M}[x+3][\overset{5}{x+k}] & + a_x \bullet a_{x+3} \bullet a_{x+k+1} \\
\ldots & \\
\overset{1}{M}[x][\overset{4}{x+k-1}] + \overset{5}{M}[x+k][\overset{5}{x+k}] & + a_x \bullet a_{x+k} \bullet a_{x+k+1}
\end{cases}
$$

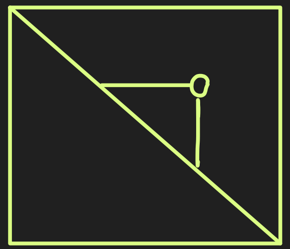
Filling the Table

M[1,1]

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

M[1,6] (our solution)

Filling the Table



Trivial case

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

Filling the Table

Arbitrary case

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

Filling the Table

Arbitrary case

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

Plus $a_1 \bullet a_2 \bullet a_6$

Filling the Table

Arbitrary case

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

Plus $a_1 \bullet a_3 \bullet a_6$

Filling the Table

Arbitrary case

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

Plus $a_1 \cdot a_4 \cdot a_6$

Filling the Table

Arbitrary case

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

Plus $a_1 \bullet a_5 \bullet a_6$

Filling the Table

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

Filling the Table

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

Example

- $a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$
- $10 \times 5 \times 1 \times 5 \times 10 \times 2$

$B_1 \quad B_2 \quad B_3 \quad B_4 \quad B_5$

Example

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 10 \times & 5 \times & 1 \times & 5 \times & 10 \times & 2 \end{array}$$

	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

Example

$a_2 \times a_3 \times a_1$

a_1	a_2	a_3	a_4	a_5	a_6

$\boxed{10 \times 5 \times 1} \times 5 \times 10 \times 2$

	1 a_1a_2	2 a_2a_3	3 a_3a_4	4 a_4a_5	5 a_5a_6
1	0	50			
2		0			
3			0		
4				0	
5					0

Example

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 10 \times & \boxed{5 \times 1 \times 5} & \times 10 \times 2 \end{array}$$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0		
4				0	
5					0

Example

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 10 \times & 5 \times & \boxed{1 \times & 5 \times & 10} & \times 2 \end{array}$$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0	50	
4				0	
5					0

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$10 \times 5 \times 1 \times \boxed{5 \times 10 \times 2}$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0	50	
4				0	100
5					0

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$
$10 \times 5 \times 1 \times 5 \times 10 \times 2$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0	50	
4				0	100
5					0

Example

Option 1 = 0 + 25 + 10 x 5 x 5 = 275

Option 2 = 50 + 0 + 10 x 1 x 5 = 100

m

a_1 a_2 a_3 a_4 a_5 a_6

$\boxed{10}$ x $\boxed{5}$ x 1 x $\boxed{5}$ x 10 x 2

	1	2 (1 2)	3	4	5
1	0	50			
2		0	25		
3			0 (3)(3)	50	
4				0	100
5					0

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$10 \times 5 \times 1 \times 5 \times 10 \times 2$

	1	2	3	4	5
1	0	50	100 (2) mark ไว้		
2		0	25		
3			0	50	
4				0	100
5					0

(2) means that the minimal solution is by dividing at B_2

Example

Option 1 = 0+ 50 + 5x 1 x 10 = 100

Option 2 = 25+ 0 + 5x 5 x 10 = 275

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 10 \times & \boxed{5} \times & \boxed{1} \times & 5 \times & \boxed{10} \times & 2 \end{array}$$

	1	2	3	4	5
1	0	50	100 (2)		
2		0	25		
3			0	50	
4				0	100
5					0

Example

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 10 \times & 5 \times & 1 \times & 5 \times & 10 \times & 2 \end{array}$$

	1	2	3	4	5
1	0	50	100 (2)		
2		0	25	100 (2)	
3			0	50	70 (4)
4				0	100
5					0

Example

Option 1 = 0+ 100 + 10x 5 x 10 = 600

Option 2 = 50+ 50 + 10x 1 x 10 = 200

Option 2 = 100+ 0 + 10x 5 x 10 = 600

a_1 a_2 a_3 a_4 a_5 a_6

$\boxed{10} \times \boxed{5} \times 1 \times 5 \times \boxed{10} \times 2$

	1	2	3	4	5
1	0	50	100 (2)		
2		0	25	100 (2)	
3			0	50	70 (4)
4				0	100
5					0

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$10 \times 5 \times 1 \times 5 \times 10 \times 2$

	1	2	3	4	5
1	0	50	100 (2)	200 (2)	
2		0	25	100 (2)	
3			0	50	70 (4)
4				0	100
5					0

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$10 \times 5 \times 1 \times 5 \times 10 \times 2$

	1	2	3	4	5
1	0	50	100 (2)	200 (2)	
2		0	25	100 (2)	80 (2)
3			0	50	70 (4)
4				0	100
5					0

Example

$(B_1\ B_2)\ |\ (B_3\ B_4\ B_5)$

① ลองเขียน MCM
แบบ bottom-up

② อยากให้เขียนโปรแกรม
บอกวิธีการแบ่ง

$(B_1\ B_2)$ ~

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$10 \times 5 \times 1 \times 5 \times 10 \times 2$

	1	2	3	4	5
1	0	50	100 (2)	200 (2)	140 (2)
2		0	25	100 (2)	80 (2)
3			0	50	70 (4)
4				0	100
5					0

Analysis

- There is $O(n^2)$ cell to be filled
  - Each cell has $O(n)$ options
- This totals to $O(n^3)$

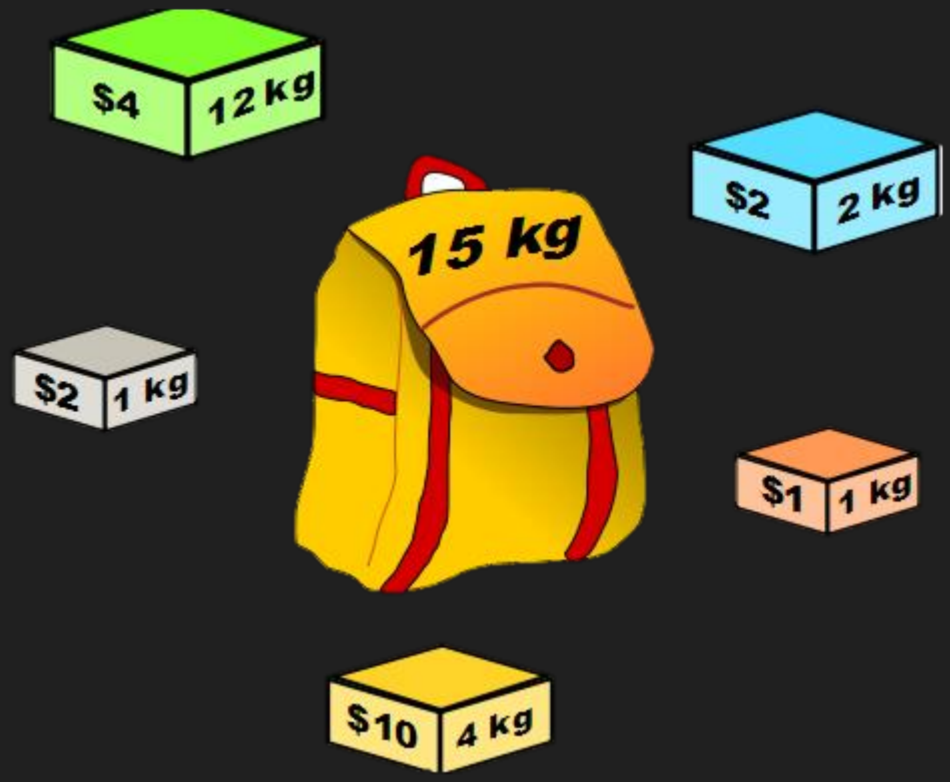
Can you write a code for Bottom-up DP of Matrix Chain Multiplication Problem?

Also can your code build the actual solution (the parenthesis of Bi, not just the minimum cost)

0-1 Knapsack Problem

Knapsack Problem

- Given a sack, able to hold W kg
- Given a list of objects
  - Each has a value and a weight
- Try to pack the object in the sack so that the total value is maximized



from http://en.wikipedia.org/wiki/File:Knapsack.svg

Variation

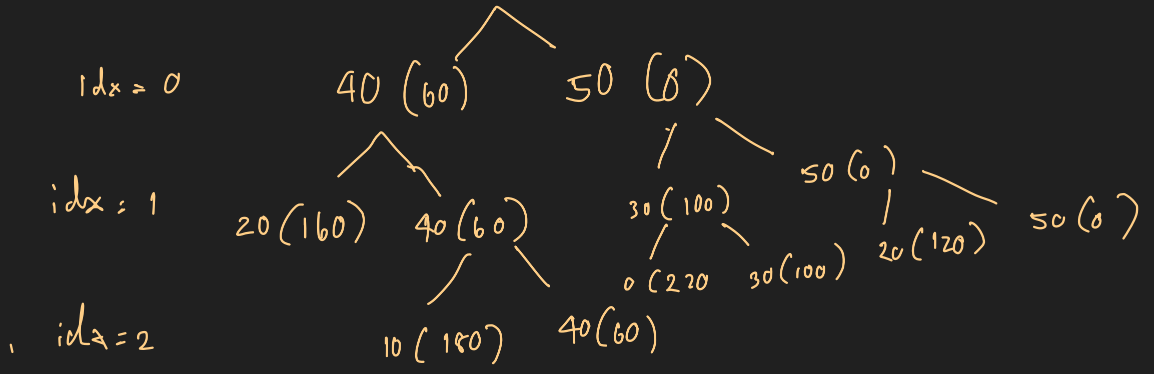
นน/ราคา

- Rational Knapsack เลือกหั่นเป็นชิ้นๆได้ แต่อัตราส่วนเท่าเดิม
  - Object is like a gold bar, we can cut it into pieces, each has the same value/weight ratio
- 0-1 Knapsack
  - Object cannot be broken, we have to choose to take (1) or leave (0) the object
    - W = 50
    - Objects = (60, 10) (100, 20) (120, 30)
    - Best solution = second and third

The Problem

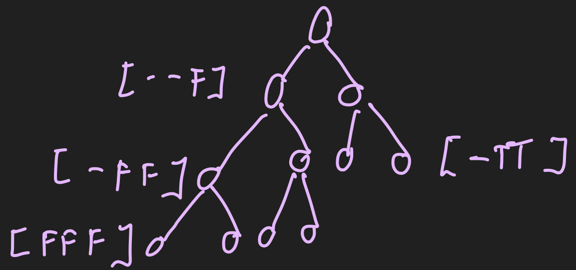
- Input:
  - ความจุ
  - A number W, the capacity of the sack
  - n values of weight and price
    - w_i = weight of the i^{th} items
    - p_i = price of the i^{th} item
- Output:
  - A subset S of {1,2,3,…,n} such that
    - $\sum_{i \in S} p_i$ is maximum
    - $\sum_{i \in S} w_i \leq W$

- Example Instance
  - W = 50
  - Pi = 60, 100, 120   220
  - wi = 10, 20, 30   50 ไม่เกิน
  - Best solution = second and third ✓



Naïve approach

สร้างทุกรูปแบบที่ ปปด




```
def knapsack(W,w[1..n],p[1..n],idx,pick[1..n])
  if (idx == 0)
    sum_price = 0
    sum_weight = 0
    for i from 1 to n
      if pick[i]
        sum_price += p[i]
        sum_weight += w[i]
    if (sum_weight <= W && sum_price > max)
      max = sum_price

  pick[idx] = false
  knapsack(W,w,p,idx-1,pick)
  pick[idx] = true
  knapsack(W,w,p,idx-1,pick)
end
```

↳ true / false

ไป recursive แล้ว

- Try every possible combination of {1,2,3,...n}
- Test whether a combination satisfies the weight constraint
  - If so, remember the best one
  - Start with knapsack(W,w,p,n,[1..n])
  - max is global var
  - $\boxed{\theta(2^n * n)}$

ค่านวณค่า

Another Naïve approach

- Keep track of remaining weight, sum the total price along the way
- What is the benefit of this approach?

knapsack(w, v, p, n, W)

เก็บ น.น. ที่เหลืออยู่

เลือกของชิ้นที่เท่าไหร่

```
def knapsack(W,w[1..n],p[1..n],idx,remain)
  if (idx == 0)
    return 0
  if (remain >= w[idx])
    #r1 is that we don't pick item #idx
    r1 = knapsack(W,w,p,idx-1,remain) ไม่เลือก
    #r2 is that we pick item #idx
    r2 = knapsack(W,w,p,idx-1,remain - w[idx]) + p[idx] เลือก
    return max(r1,r2)
  else
    return knapsack(W,w,p,idx-1,remain) → ไม่มีสิทธิ์เลือก w[idx]
end
```

The Recurrence Relation

- $K(a,b)$ = the best total price when and only item number 1 to number a is considered and the knapsack is of size b

ไม่มีของให้เลือก, ใส่ของเพิ่มไม่ได้แล้ว

- $K(a,b) = 0$ when $a = 0$ or $b = 0$
- $K(a,b) = K(a-1,b)$ when $w_a > b$ น.น. เกิน ก็ทิ้งไปเลย
- $K(a,b) = \max(\, K(a-1,b-w_a) + p_a,$
  $K(a-1,b)\,)$
- The solution is at $K(n,W)$

The Failed Attempt #1

- Let $K(a)$ be the best total value when we consider only item number 1 to number a and the weight limit is W ไม่สนใจ น.น.
    - The answer is at $K(n)$
    - By definition, $K(n)$ and $K(n-1)$ and $K(n-2)$… all consider the $\boxed{\text{same weight limit}}$
- Let's say that the answer contains item number n
    - Also by definition, its means that K(n) = K(n-1) + pn
    - However, K(n-1) will consider the problem thinking that the weight limit is the same (not reduced by weight of item number n)

      เหมือนกัน
    - It is wrong to say that $K(a) = \max(\boxed{K(a-1)} + p_a\,,\ K(a-1))$ เราไม่รู้ว่า น.น.จะเกิน หรือเปล่า
    - It is not possible to have a recurrence relation that does not consider W

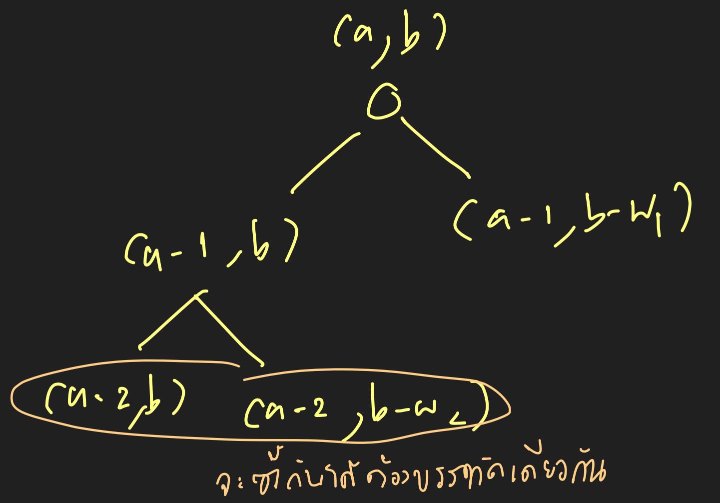
The Failed Attempt #2

สนแค่ น.น.

- Let $K(b)$ be the best total value when the weight limit of the sack is b
  - The answer is at $K(W)$
- If the ith item is in the best solution
  - $K(W) = K(W - w_i) + p_i$
- But, we don't really know that the ith item is in the optimal solution
  - So, we try everything
  - $K(W) = max_{1 \le i \le n}(K(W - w_i) + p_i)$
- Is this our algorithm?
  - Yes, if and only if we allow each item to be selected multiple times (that is not true for this problem)

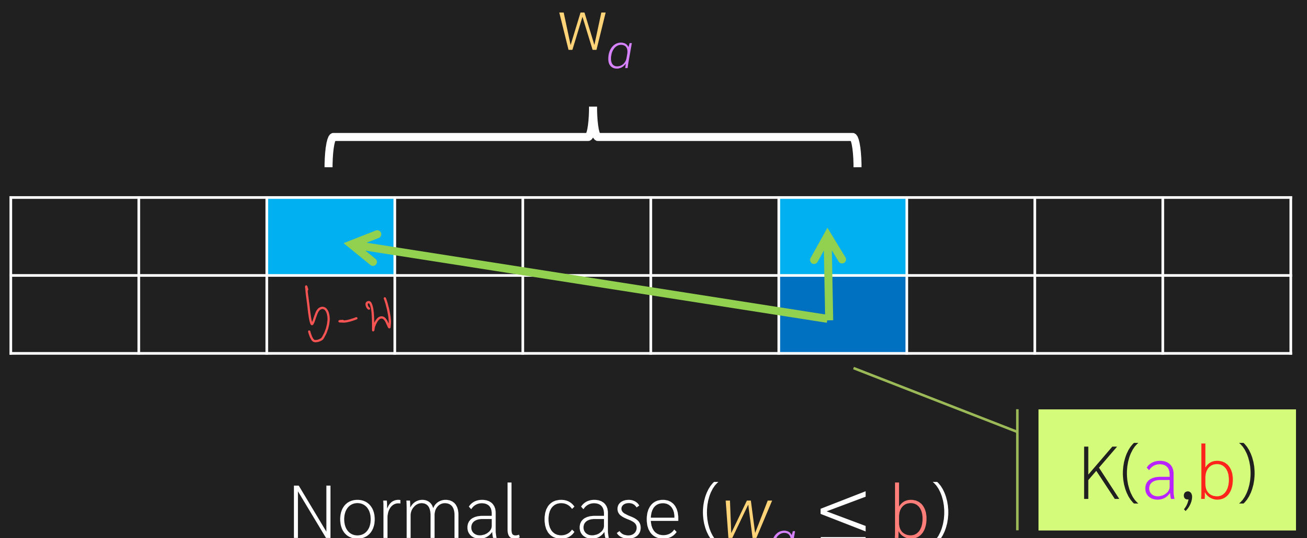
Exercise: Top-Down approach

- Write a top down dynamic programming approach using this recurrence relation
  - $K(a,b) = 0$ when $a = 0$ or $b = 0$
  - $K(a,b) = K(a-1,b)$ when $w_a > b$
  - $K(a,b) = \max(K(a-1,b - w_a) + p_a ,$
    $K(a-1,b))$

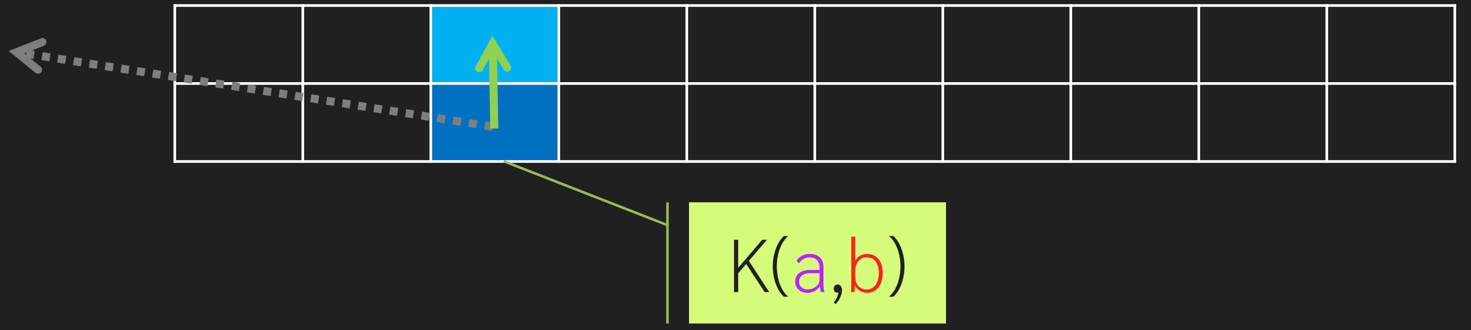


- Which data structure should we use to store result?
  - Should we use 2D array?
  - Should we use associative data structure such as std::map or std::unordered_map?

The Table for Bottom-Up



Normal case ($w_a \le b$)



Too much weight ($w_a > b$)

- Row = item id (a)
- Col = weight (b)

Example

$p = \{4, 2, 2, 1, 10\}$

$w = \{12, 2, 1, 1, 4\}$ $W = 15$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																
2																
3																
4																
5																

Example

$p = \{4, 2, 2, 1, 10\}$

$w = \{12, 2, 1, 1, 4\}$ $W = 15$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0															
2	0															
3	0															
4	0															
5	0															

$K(a,b) = 0$ when $a = 0$ or $b = 0$

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 1 ($p_1=4$ $w_1=12$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0															
2	0															
3	0															
4	0															
5	0															

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 1 ($p_1=4$ $w_1=12$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0				
2	0															
3	0															
4	0															
5	0															

$K(a,b) = K(a-1,b)$ when $w_a > b$

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 1 ($p_1 = 4$ $w_1 = 12$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0															
3	0															
4	0															
5	0															

12 +
ต้องคิดแล้วว่าจะ

$$K(a,b) = \max(\,K(a-1, b - w_a) + p_a\,,\quad K(a-1,b)\,)$$

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 2 ($p_2=2$ $w_2=2$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0														
3	0															
4	0															
5	0															

$K(a,b) = K(a,b-1)$ when $w_b > a$

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 2 ($p_2=2$ $w_2=2$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0														
3	0															
4	0															
5	0															

$K(a,b) = K(a-1,b)$ when $w_a > b$

Example

p = {4, 2, 2, 1, 10}
w ={12, 2, 1, 1, 4}   W = 15

Fill row 2 (p_2=2 w_2=2)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0 +2	0	4	4 +2	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0															
4	0															
5	0															

$K(a,b) = \max(\, K(a-1, b - w_a) + p_a \,,\quad K(a-1,b)\,)$

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 2 ($p_2=2$ $w_2=2$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0															
4	0															
5	0															

$$K(a,b) = \max(\, K(a-1, b - w_a) + p_a \,, \quad K(a-1,b)\,)$$

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 3 ($p_3=2$ $w_3=1$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0															
5	0															

$$K(a,b) = \max(\ K(a-1, b - w_a) + p_a\ ,\quad K(a-1,b)\ )$$

Example

$p = \{4, 2, 2, 1, 10\}$

$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 3 ($p_3 = 2$ $w_3 = 1$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0															
5	0															

ได้ทั้งคู่

+2

Example

$p = \{4, 2, 2, 1, 10\}$

$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 4 ($p_4=1$ $w_4=1$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5	0															

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 4   ($p_4=1$ $w_4=1$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5	0															

เลือกใช้ตัวข้างบน

ทั้ยบน

Example

$p = \{4, 2, 2, 1, 10\}$

$w = \{12, 2, 1, 1, 4\}$ $W = 15$

ค่าเปลี่ยน →
↓
ถูกหยิบไป ?

Fill row 5 ($p_5=10$ $w_5=4$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Fill row 5 ($p_5 = 10$ $w_5 = 4$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Example

$p = \{4, 2, 2, 1, 10\}$
$w = \{12, 2, 1, 1, 4\}$ $W = 15$

Trace the solution backward to get the actual item number
We have item number 5,4,3,2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

เอาของ
ช่องพวกนี้

Bottom-Up Code

recursion

top down = (2^n)

```
set all K[0][*] = 0 and all K[*][0] = 0
for a = 1 to n
  for b = 1 to W
    if (w[a] > b)
      K[a][b] = K[a - 1][b];
    else
      K[a][b] = max( K[a - 1][b - w[a]] + p[a] ,
                     K[a - 1][b] )
return K[n][W];
```

$O(n(W))$

เปรียบเทียบไม่ได้

Can you write a code that generate the list of actual item that we take?

- Does this code generate too much subproblem?
- Does it generates one that we does not need?
- Is it better to use Top-Down approach?
  - Can you show some instance that Top-Down is better than Bottom-up (this code)

สรุปค่าตอบที่เราไปต้อง ใช้หรือเปล่า

เคสไหนที่ Wn จะชนะ 2^n ?

Analysis

- From Bottom-Up, it is clear that this is O(Wn)
- Original generate-all-solution method is $O(2^n)$
- Which one is better
  - In what case that O(Wn) Dynamic Programming will benefit greatly (because there are several overlapping subproblems)

approximation algorithm

ไม่สอบ

Instance

approx → O ดีกว่า 2^n

approxi- strategy

80%

ans

แต่รับประกันว่า ไม่แย่ไปกว่า 80% ของคำตอบที่ดีที่สุด