



Unit 32.

Dynamic Programming

● Learning objectives

- ✓ ນັກຮຽນຈະສາມາດເຂົ້າໃຈການ dynamic programming ແລະ ຄິດໄລ່ລຳດັບ Fibonacci ໂດຍໃຊ້ dynamic programming.
- ✓ ນັກຮຽນຈະສາມາດຄິດໄລ່ຄ່າສຳປະສິດ binomial ໂດຍໃຊ້ສາມແຈຂອງ Pascal.
- ✓ ນັກຮຽນຈະສາມາດແກ້ໄຂບັນຫາການເພີ່ມປະສິດທິພາບໄດ້ໂດຍການໃຊ້ການພົວພັນແບບ recursive ໃນຮູບແບບ bottom-up.

● Learning overview

- ✓ ຮຽນຮູ້ວິທີການກຳນົດລຳດັບ Fibonacci ຊ້ຳຄືນແລະແກ້ໄຂບັນຫາດ້ວຍ dynamic programming.
- ✓ ຮຽນຮູ້ວິທີການຄິດໄລ່ສາມແຈຂອງ Pascal ໂດຍການນຳໃຊ້ຄຸນສົມບັດ recursive ຂອງສຳປະສິດ binomial.
- ✓ ຮຽນຮູ້ວິທີການແກ້ໄຂຄວາມສຳພັນແບບ recursive ແບບ bottom-up ໂດຍການນຳໃຊ້ dynamic programming.

● Concepts you will need to know from previous units

- ✓ ສາມາດກຳນົດບັນຫາບາງສ່ວນແບບ recursively ໂດຍໃຊ້ວິທີການ divide-and-conquer.
- ✓ ສາມາດສ້າງຟັງຊັນ recursive ໂດຍໃຊ້ຄວາມສຳພັນ recursive.
- ✓ ສາມາດເກັບຄ່າໃນຕາຕະລາງໂດຍໃຊ້ lists ແລະ dictionaries.

Keywords

**Dynamic
Programming**

**Recursive
Property**

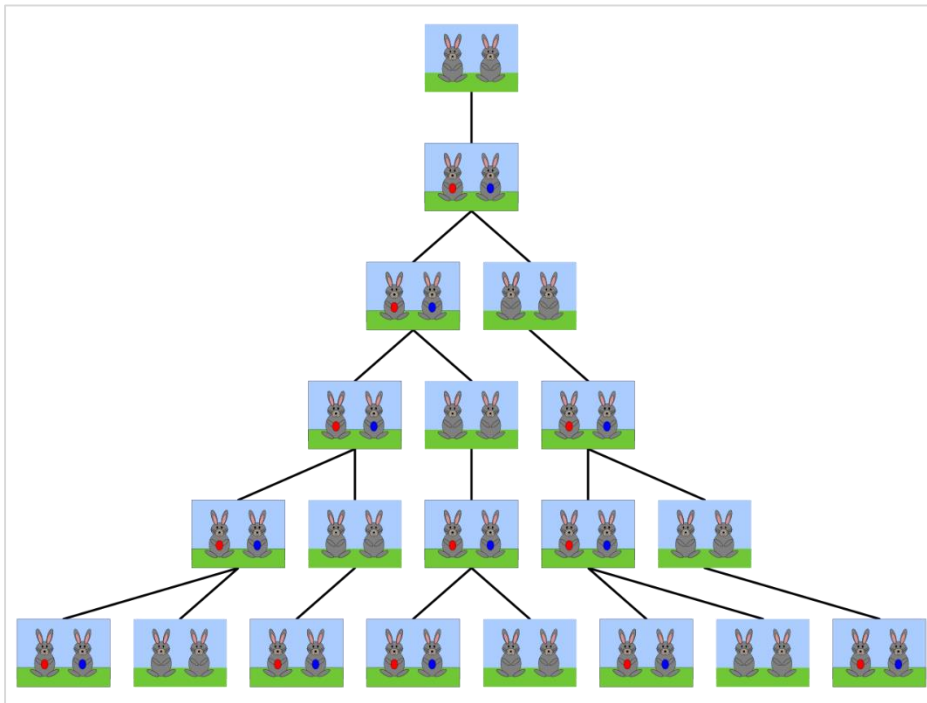
**Bottom-Up
Approach**

Memoization

| Mission

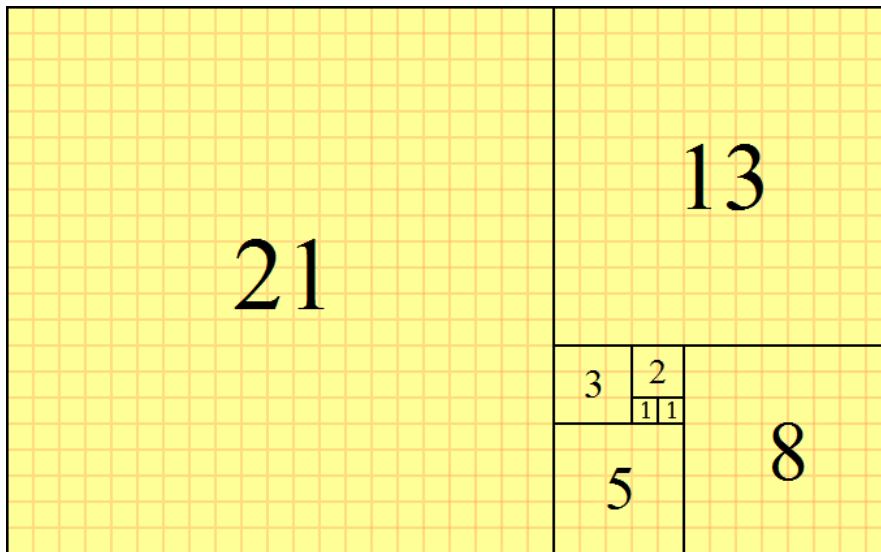
1. Real world problem

1.1. ມີກະຕ່າຍຫຼາຍປານໃດ?



2. Mission

2.1. ບັນຫາລຳດັບຂອງ Fibonacci



https://en.wikipedia.org/wiki/Fibonacci_number#/media/File:34*21-FibonacciBlocks.png

- ▶ Leonardo of Pisa ແມ່ນເປັນທີ່ຮູ້ຈັກໂດຍຊື່ Fibonacci. ບັນຫານີ້ຍົກຂຶ້ນມາໂດຍ Fibonacci ເປັນທີ່ຮູ້ຈັກກັນຢ່າງກວ້າງຂວາງວ່າ ລຳດັບ Fibonacci ຊຶ່ງເປັນບັນຫາທີ່ສຳຄັນຫຼາຍໃນທິດສະດີຄະນິດສາດ.
- ▶ ອັນດັບທີ n ຂອງຕົວເລກ Fibonacci, F_n , ເປັນລຳດັບທີ່ກຳນົດໂດຍຄຳເບື້ອງຕົ້ນຕໍໄປນີ້ ແລະ ການເກີດຂຶ້ນຊ້າຂອງມັນດັ່ງທີ່ສະແດງຢູ່ລຸ່ມນີ້.
 - $F_0 = 0$
 - $F_1 = 1$
 - $F_n = F_{(n-1)} + F_{(n-2)}$
- ▶ ລຳດັບ Fibonacci ເປັນດັ່ງຕໍ່ໄປນີ້.
 - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, ...
- ▶ ໃຫ້ຂຽນຂັ້ນຕອນວິທີເພື່ອຊອກຫາພຶດທີ່ n ຂອງອັນດັບ Fibonacci.

3. ການແກ້ໄຂບັນຫາ

3.1. ຄຳນວນຄ່າ Fibonacci n ໂດຍໃຊ້ recursive

- ບັນຫາຂອງການຄຳນວນຄ່າ Fibonacci ຈຳນວນ n ພົດ ສາມາດແກ້ໄຂໄດ້ໂດຍການນຳໃຊ້ສົມຜົນ recursive ດັ່ງຕໍ່ໄປນີ້.
- ແນວໃດກໍຕາມ, ວິທີການແກ້ໄຂບັນຫາແບບ recursive ເຫຼົ່ານີ້ບໍ່ມີປະສິດທິພາບເທົ່າທີ່ຄວນ.

```
1 def fib1(n):
2     if n == 0 or n == 1:
3         return n
4     else:
5         return fib1(n - 1) + fib1(n - 2)
```

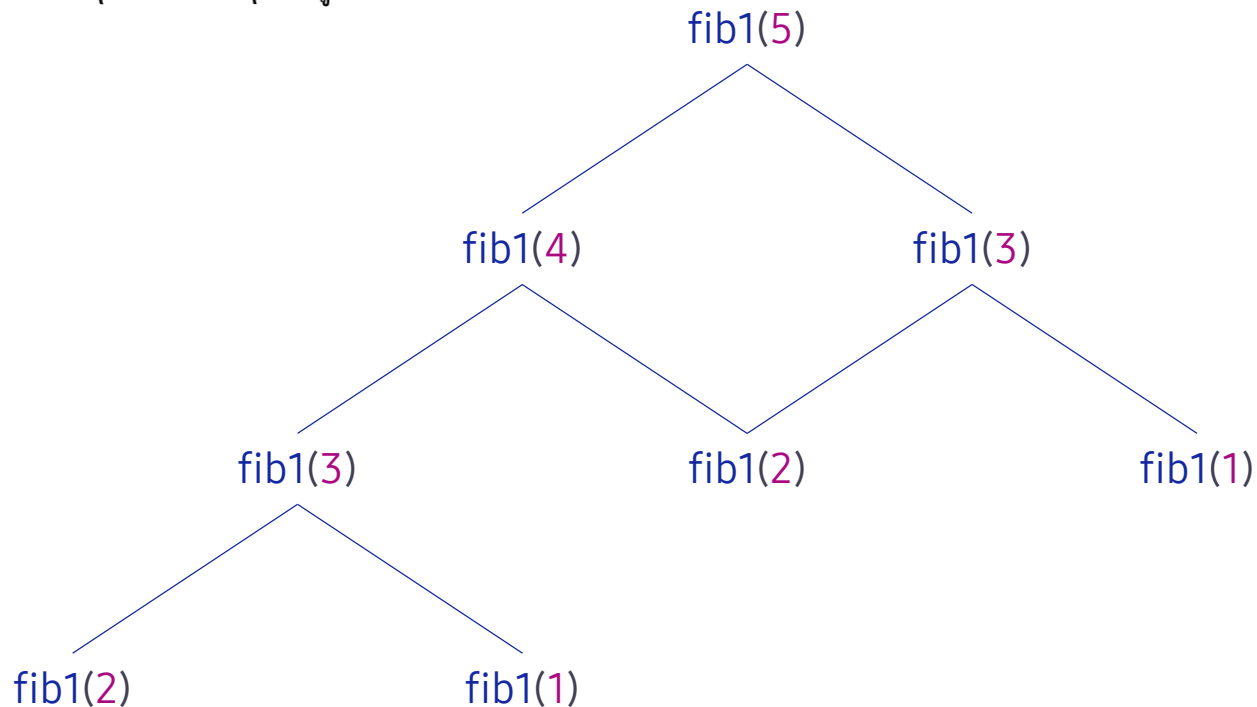
```
1 N = int(input("Input a number: "))
2 print(fib1(N))
```

Input a number: 10
55

3. ການແກ້ໄຂບັນຫາ

3.1. ຄຳນວນຄ່າ Fibonacci n ໂດຍໃຊ້ recursive

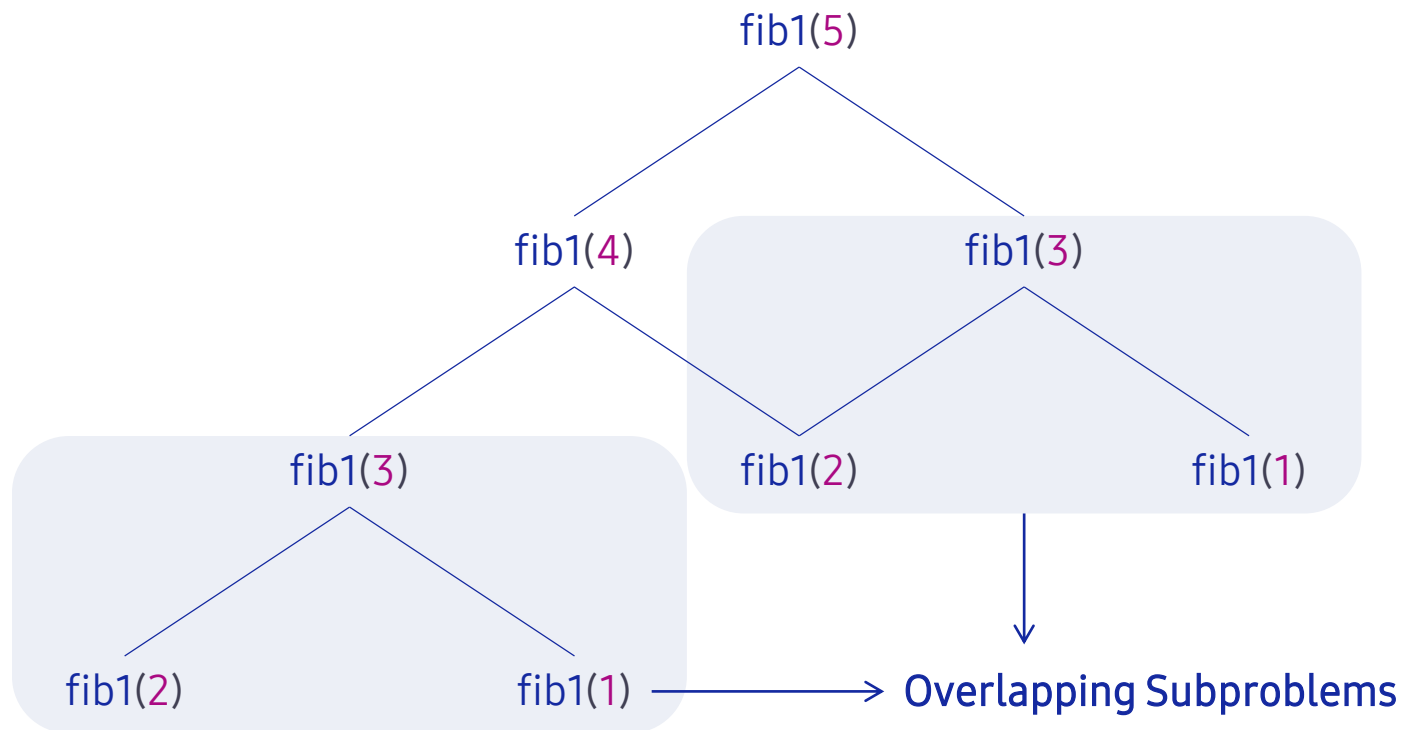
- ຕົວຢ່າງ, ພິຈາລະນາ ການເອິ້ນໃຊ້ $\text{fib1}(5)$.
- ຟັງຊັນ recursive ຖືກເອິ້ນໃຊ້ໃນຮູບແບບຕໍ່ໄປນີ້.



3. ການແກ້ໄຂບັນຫາ

3.1. ຄຳນວນຄ່າ Fibonacci n ໂດຍໃຊ້ recursive

- | ໃນນີ້, ໃຫ້ຂໍສັງເກດວ່າ $\text{fib1}(3)$, $\text{fib1}(2)$ ແລະ $\text{fib1}(1)$ ຖືກເອີ້ນສອງຄັ້ງ, ຕາມລຳດັບ.
- | ບັນຫາຂອງການເອີ້ນໃຊ້ແບບ recursive ຊຳກັ້ນດັ່ງກ່າວເອີ້ນວ່າບັນຫາຍ່ອຍທີ່ທັບຊ້ອນກັນ.



3. ການແກ້ໄຂບັນຫາ

3.2. ຄຳນວນຫາຄ່າ Fibonacci ຈຳນວນ n ພົດ ໂດຍໃຊ້ Dynamic Programming

| ເພື່ອແກ້ໄຂບັນຫາຍ່ອຍທີ່ທັບຊ້ອນກັນ, ສາມາດແກ້ໄຂບັນຫາແບບ bottom-up ໂດຍການເອີ້ນໃຊ້ຕົວເອງຄືນ ຕາມຮູບຂ້າງລຸ່ມນີ້ ແລະ ແກ້ໄຂດ້ວຍ loops.

```
1 F = {0: 0, 1: 1}
2
3 def fib2(n):
4     if n in F:
5         return F[n]
6     else:
7         F[n] = fib2(n - 1) + fib2(n - 2)
8         return F[n]
```

```
1 N = int(input("Input a number: "))
2 print(fib2(N))
```

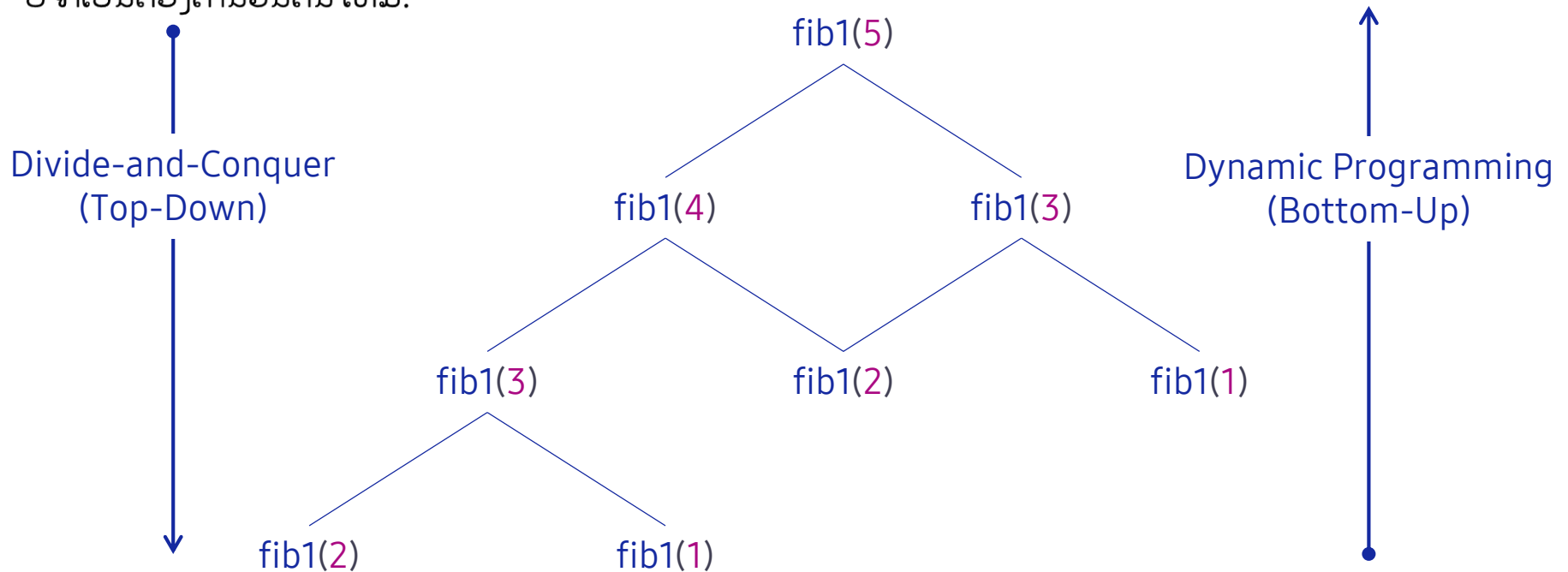
Input a number: 10
55

| Key concept

1. Dynamic Programming

1.1. Dynamic Programming .vs. Divide-and-Conquer


- Dynamic programming ແມ່ນວິທີການທີ່ຄືກັນກັບວິທີການ divide-and-conquer, ທີ່ໄດ້ກ່າວເຖິງໃນຫົວຂໍ້ທີ 31, ໃນນັ້ນທັງສອງວິທີແມ່ນການແກ້ໄຂບັນຫາແບບ recursive.
- ເນື່ອງຈາກວິທີການ divide-and-conquer ເປັນການຄຳນວນແບບຊ້ຳຄືນທີ່ບໍ່ມີປະສິດທິພາບເມື່ອບັນຫາຍ່ອຍທັບຊ້ອນກັນເກີດຂຶ້ນ, ວິທີການການ dynamic programming ເກັບ ບັນຫາຍ່ອຍໄວ້ໃນຕົວປ່ຽນ ຫຼື ຕາຕະລາງ ເພື່ອໃຫ້ບັນຫາຍ່ອຍທີ່ຄຳນວນແລ້ວບໍ່ຈຳເປັນຕ້ອງຄຳນວນຄືນໃໝ່.




1. Dynamic Programming

1.2. ການອອກແບບຂັ້ນຕອນວິທີຂອງ Dynamic Programming

- | Dynamic programming ແມ່ນເທັກນິກການອອກແບບຂັ້ນຕອນວິທີທີ່ວິເຄາະຄຸນລັກສະນະການເຮັດຊຳຄືນຂອງບັນຫາທີ່ໃຫ້ມາ ແລະ ຫຼັງຈາກນັ້ນໄດ້ຮັບຄຳຕອບໃນຮູບແບບ bottom-up.
- | ເຕັກນິກການເກັບຄຳຕອບຂອງບັນຫາຍ່ອຍທີ່ທັບຊ້ອນກັນຢູ່ໃນຕາຕະລາງເພື່ອແກ້ໄຂບັນຫາຍ່ອຍທີ່ຊ້ອນກັນເອີ້ນວ່າ memorization.

 Focus ຂັ້ນຕອນການພັດທະນາຂອງຂັ້ນຕອນວິທີແບບ dynamic programming ດັ່ງຕໍ່ໄປນີ້.

- ▶ ສ້າງສົມຜົນ recursive ເພື່ອຄຳນວນຫາຄຳຕອບຂອງກໍລະນີຂໍ້ມູນປ້ອນເຂົ້າຂອງບັນຫາ
- ▶ ຄຳຕອບຂອງກໍລະນີ ຂໍ້ມູນປ້ອນເຂົ້າທັງໝົດແມ່ນໄດ້ຮັບໂດຍວິທີ bottom-up ທີ່ແກ້ໄຂກໍລະນີຂໍ້ມູນປ້ອນເຂົ້າຂະໜາດນ້ອຍກ່ອນ.

 Focus Dynamic programming ໃຊ້ memoization ສຳລັບການຄິດໄລ່ແບບ bottom-up.

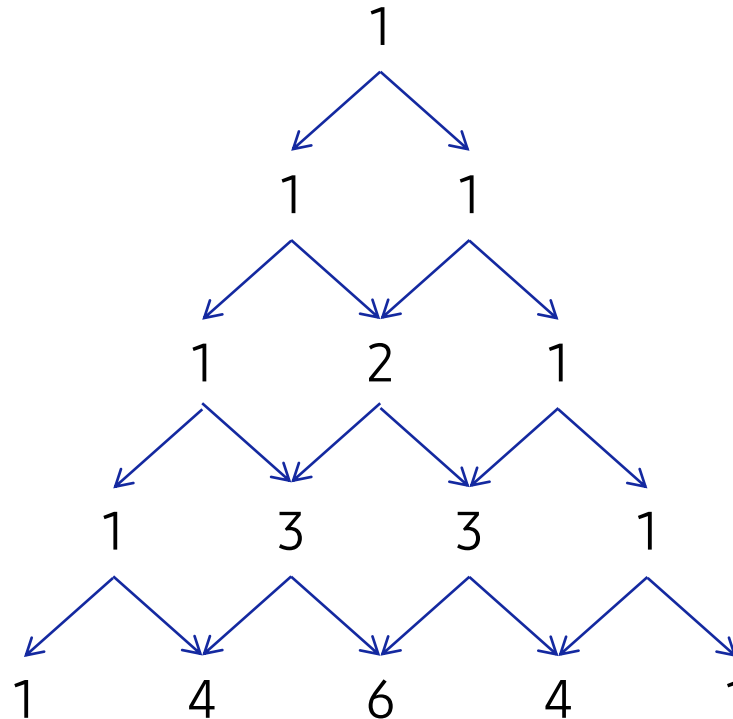
- ▶ ຄຳຕອບສຳລັບບັນຫາຍ່ອຍຖືກເກັບໄວ້ໃນຕາຕະລາງ (ຫຼືຫນ່ວຍຄວາມຈຳ).
- ▶ ເມື່ອພົບບັນຫາຍ່ອຍດຽວກັນ, ມັນຈະເອີ້ນເອົາຄຳຕອບຈາກຕາຕະລາງ (ຫຼືຫນ່ວຍຄວາມຈຳ) ໂດຍບໍ່ມີການຄິດໄລ່ຄືນໃໝ່.

1. Dynamic Programming

1.3. ຮູບສາມແຈ Pascal

- ສາມແຈຂອງ Pascal ແມ່ນຂັ້ນຕອນວິທີທີ່ມີຊື່ສຽງທີ່ນຳໃຊ້ໃນ dynamic programming.
- ສາມແຈຂອງ Pascal ແມ່ນ array ຂອງຄ່າສຳປະສິດຖານສອງໃນຄະນິດສາດໃນຮູບສາມແຈ.

Pascal's Triangle

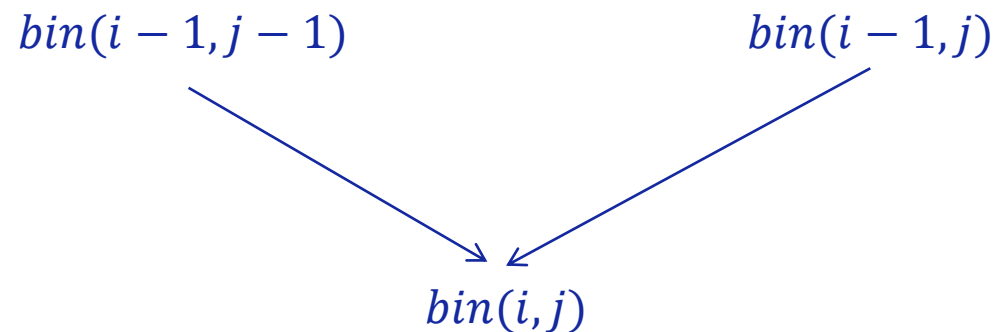


1. Dynamic Programming

1.3. ຮູບສາມແຈ Pascal

ໃນສາມແຈຂອງ Pascal, ຄ່າສໍາປະສິດ binomial ມີຄຸນສົມບັດ recursive ດັ່ງຕໍ່ໄປນີ້ ຂຶ້ນກັບ n ແລະ k .

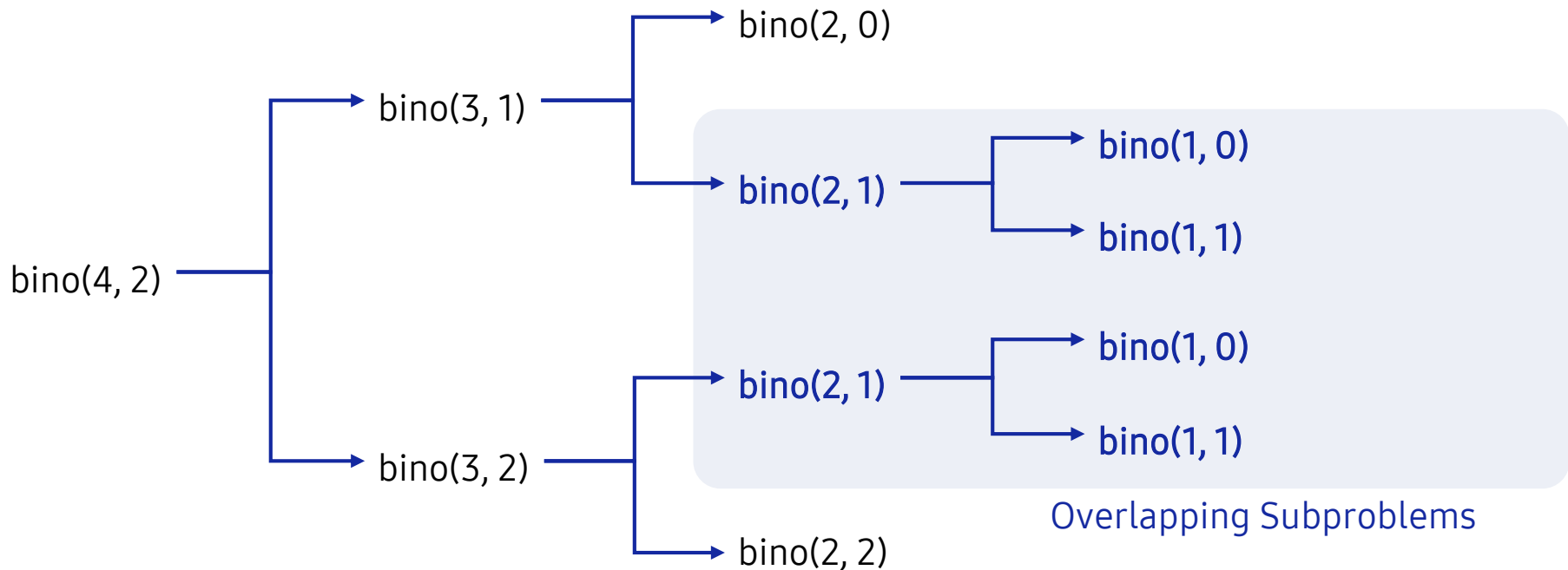
$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \text{ or } k = n \end{cases}$$



1. Dynamic Programming

1.4. ສາມແຈ Pascal ດ້ວຍເທັກນິກ Divide-and-Conquer

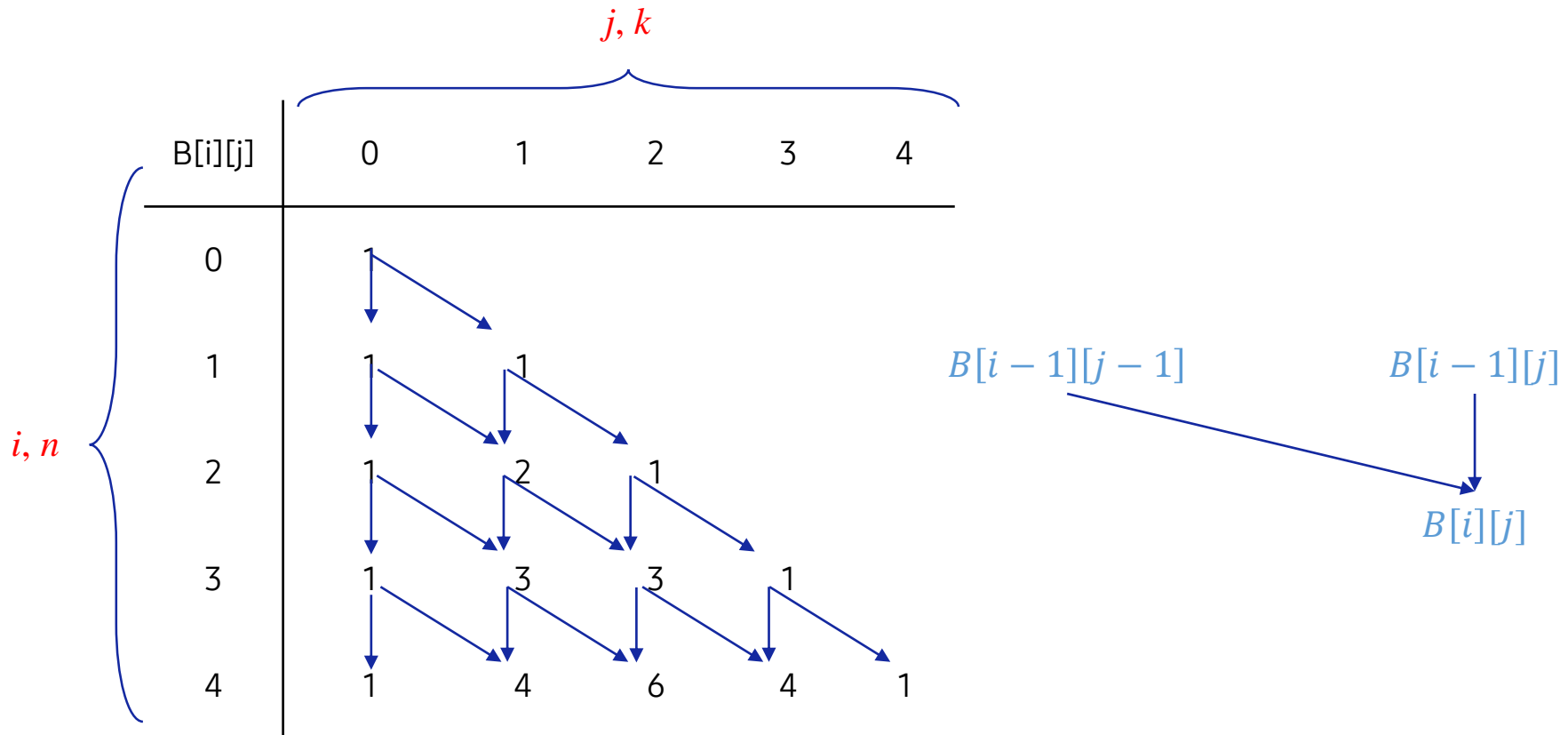
ແນວໃດກໍ່ຕາມ, ການແກ້ໄຂແບບ recursive ເຊັ່ນ: ລຳດັບ Fibonacci, ເຮັດໃຫ້ເກີດບັນຫາຍ່ອຍທັບຊ້ອນກັນ ດັ່ງຕໍ່ໄປນີ້.



1. Dynamic Programming

1.4. ສາມແຈ Pascal ດ້ວຍເທັກນິກ Divide-and-Conquer

ເພື່ອນຳໃຊ້ dynamic programming ກັບສາມແຈຂອງ Pascal, ສາມາດນຳໃຊ້ຕາຕະລາງ ດັ່ງທີ່ສະແດງຂ້າງລຸ່ມນີ້.



| Let's code

1. Dynamic Programming

1.1. ຄຳນວນຄ່າ Fibonacci ຈຳນວນ n ພົດດ້ວຍ Recursion

ຟັງຊັນ `fib1()` ແມ່ນສ້າງຕາມຫຼັກການ recursive ຂອງລຳດັບ Fibonacci ໂດຍຜ່ານຟັງຊັນ recursive.

```
1 def fib1(n):  
2     if n == 0 or n == 1:  
3         return n  
4     else:  
5         return fib1(n - 1) + fib1(n - 2)
```



Line 2-5

- Line 2-3: ເງື່ອນໄຂການສິ້ນສຸດການເຮັດວຽກຂອງຟັງຊັນ recursive ຈະສິ້ນສຸດຖ້າ n ເທົ່າ 0 ຫຼື 1.
- Line 4-5: ຟັງຊັນ recursive ສົ່ງຄືນຜົນບວກຂອງພົດທີ $n-1$ ແລະ $n-2$ ໂດຍກຳນົດແບບ recursive ຂອງລຳດັບ Fibonacci.

1. Dynamic Programming

1.2. ຄຳນວນຄ່າ Fibonacci ຈຳນວນ n ພົດດ້ວຍ Memoization

ຟັງຊັນ fib2() ຖືກສ້າງໂດຍຜ່ານຟັງຊັນ recursive ເປັນການແກ້ໄຂແບບ bottom-up ໂດຍການນຳໃຊ້ວິທີການ dynamic programming.

```
1 F = {0: 0, 1: 1}
2
3 def fib2(n):
4     if n in F:
5         return F[n]
6     else:
7         F[n] = fib2(n - 1) + fib2(n - 2)
8         return F[n]
```



Line 1-5

- F ຖືກສ້າງຂຶ້ນເປັນ dictionary ແລະ ຄ່າ F[0] ແລະ F[1] ແມ່ນເລີ່ມຕົ້ນເປັນ 0 ແລະ 1 ຕາມລຳດັບ.
- Line 4-5: ຖ້າ n ມີຢູ່ໃນ dictionary F, ມັນເປັນບັນຫາຍ່ອຍທີ່ໄດ້ຄິດໄລ່ແລ້ວ ແລະສາມາດສົ່ງຄືນ F[n] ໄດ້ທັນທີ.

1. Dynamic Programming

1.2. ຄຳນວນຄ່າ Fibonacci ຈຳນວນ n ພຶດດ້ວຍ Memoization

```
1 F = {0: 0, 1: 1}
2
3 def fib2(n):
4     if n in F:
5         return F[n]
6     else:
7         F[n] = fib2(n - 1) + fib2(n - 2)
8         return F[n]
```



Line 6-8

- ຖ້າຄ່າຂອງ n ບໍ່ມີຢູ່ໃນ Dictionary F, ມັນແມ່ນບັນຫາບາງສ່ວນທີ່ຍັງບໍ່ທັນໄດ້ຄິດໄລ່, ດັ່ງນັ້ນຈຶ່ງໄດ້ໃຊ້ການເອີ້ນໃຊ້ຄືນໃໝ່.
- Line 7-8: ຄ່າທີ່ຄິດໄລ່ຜ່ານການເອີ້ນໃຊ້ recursive ຈະຖືກເກັບໄວ້ໃນ F ເພື່ອບໍ່ໃຫ້ຖືກຄິດໄລ່ຄືນໃໝ່ ແລະ ຫຼັງຈາກນັ້ນສົ່ງຄືນຄ່າ F[n].

1. Dynamic Programming

1.2. ຄຳນວນຄ່າ Fibonacci ຈຳນວນ n ພົດດ້ວຍ Memoization

- ຟັງຊັນ fib3() ປະຕິບັດ memoization ໂດຍໃຊ້ list ແທນທີ່ຈະເປັນ dictionary.
- ໃນກໍລະນີນີ້, ຄ່າຂອງອົງປະກອບຂອງພົດ Fibonacci, ທີ່ບໍ່ທັນໄດ້ຄິດໄລ່, ອາດຈະຖືກກຳນົດຄ່າເປັນ -1 ເພື່ອກຳນົດວ່າການຄິດໄລ່ທີ່ຊ້າກັນ.

```
1 def fib3(F, n):
2     if n <= 1:
3         return F[n]
4     else:
5         if F[n] < 0:
6             F[n] = fib3(F, n - 1) + fib3(F, n - 2)
7         return F[n]
```

Line 1-7

- Line 2-3: ຖ້າ n ມີຄ່າຕໍ່າກວ່າ 1, ໃຫ້ສົ່ງຄືນຄ່າ F[n].
- Line 5-7: ເມື່ອຄ່າຂອງ F[n] ຕໍ່າກວ່າ 0, ຈະມີການເອິ້ນໃຊ້ຄືນ, ຫຼື F[n] ຖືກສົ່ງຄືນທັນທີ.

1. Dynamic Programming

1.2. ຄຳນວນຄ່າ Fibonacci ຈຳນວນ n ພຶດດ້ວຍ Memoization

ເມື່ອເອີ້ນໃຊ້ຟັງຊັນ fib3(), list F, ເຊິ່ງເກັບຄ່າຂອງພຶດທີ່ບໍ່ໄດ້ຄຳນວນເປັນ -1 ຊຶ່ງຈະຕ້ອງຖືກເກັບໄວ້ເປັນຂໍ້ມູນປ້ອນເຂົ້າ.

```

1 def fib3(F, n):
2     if n <= 1:
3         return F[n]
4     else:
5         if F[n] < 0:
6             F[n] = fib3(F, n - 1) + fib3(F, n - 2)
7         return F[n]
```

```

1 N = int(input("Input a number: "))
2 F = [0, 1] + [-1] * (N - 1)
3 print(fib3(F, N))
```

Input a number: 10
55

Line 2-3

- ອົງປະກອບທີ 0 ແລະ ທີ 1 ຂອງລາຍການ F ແມ່ນເລີ່ມຕົ້ນເປັນ 0 ແລະ 1 ຕາມລຳດັບ ແລະອົງປະກອບ N - 1 ທີ່ຍັງເຫຼືອແມ່ນເລີ່ມຕົ້ນເປັນ -1.
- ສົ່ງ F ແລະ N ໄປຫາຄ່າພາລາມິຕິຂອງຟັງຊັນ fib3().

1. Dynamic Programming

1.3. ແກ້ໄຂສາມແຈ Pascal ດ້ວຍ Recursion

ການນຳໃຊ້ຄຸນສົມບັດ recursive ຂອງສຳປະສິດ binomial, $\text{bin}(n, k)$ ສາມາດຄິດໄລ່ໄດ້ດັ່ງຕໍ່ໄປນີ້.

```
1 def bin1(n, k):
2     if k == 0 or n == k:
3         return 1
4     else:
5         return bin1(n - 1, k - 1) + bin1(n - 1, k)
```

```
1 n = int(input("Input the value of n: "))
2 k = int(input("Input the value of k: "))
3 print(f"binomial({n}, {k}) is {bin1(n, k)}")
```

Input the value of n: 5

Input the value of k: 3

binomial(5, 3) is 10



Line 1-5

- Line 2-3: ຖ້າ k ເປັນ 0 ຫຼື n , ໃຫ້ສົ່ງຄືນ 1.
- Line 4-5: ຖ້າ k ບໍ່ແມ່ນ 0 ຫຼື 1 ການເອິ້ນໃຊ້ແບບ recursive ແມ່ນເຮັດໂດຍໃຊ້ລັກສະນະ recursive ຂອງຄ່າສຳປະສິດ binomial.

1. Dynamic Programming

1.4. ແກ້ໄຂສາມແຈ Pascal ດ້ວຍ Dynamic Programming

ໂດຍການນຳໃຊ້ dynamic programming, ສຳນວນ recursive ສາມາດແກ້ໄຂໄດ້ໃນລັກສະນະ bottom-up ໂດຍໃຊ້ loops ດັ່ງທີ່ສະແດງຂ້າງລຸ່ມນີ້.

```
1 def bin2(n, k):
2     B = [[0] * (i + 1) for i in range(n + 1)]
3     for i in range(n + 1):
4         for j in range(i + 1):
5             if j == 0 or j == i:
6                 B[i][j] = 1
7             else:
8                 B[i][j] = B[i - 1][j - 1] + B[i - 1][j]
9     return B[n][k]
```



Line 1-2

- ສຳລັບພາຣາມິເຕີຂໍ້ມູນປ້ອນເຂົ້າ n ແລະ k , ສາມແຈຂອງ Pascal ຖືກກຳນົດຄ່າໃນຮູບແບບຂອງ ລາຍການ.
- ລາຍການ B ມີ $n + 1$ ແຖວ, ແລະ ແຖວ i ແມ່ນລາຍການທີ່ມີຄວາມຍາວຂອງ $i + 1$, ເຊິ່ງຄ່າທັງໝົດແມ່ນເລີ່ມຕົ້ນເປັນ 0.

1. Dynamic Programming

1.4. ແກ້ໄຂສາມແຈ Pascal ດ້ວຍ Dynamic Programming

```
1 def bin2(n, k):
2     B = [[0] * (i + 1) for i in range(n + 1)]
3     for i in range(n + 1):
4         for j in range(i + 1):
5             if j == 0 or j == i:
6                 B[i][j] = 1
7             else:
8                 B[i][j] = B[i - 1][j - 1] + B[i - 1][j]
9     return B[n][k]
```

Line 3-9

- ຈາກແຖວທີ n ຂອງສາມແຈຂອງ Pascal, ພວກເຮົາຄິດໄລ່ແຕ່ລະຕົວຄູນ binomial ຈາກ bottom-up.
- ແຕ່ລະຕົວຄູນ binomial $B[i][j]$ ແມ່ນ $B[i-1][j-1]$ ແລະ $B[i-1][j]$ ໄດ້ຖືກຄິດໄລ່ແລ້ວ, ດັ່ງນັ້ນພວກມັນສາມາດຄິດໄລ່ໄດ້ທັນທີໂດຍຜ່ານການບວກ.

1. Dynamic Programming

1.4. ແກ້ໄຂສາມແຈ Pascal ດ້ວຍ Dynamic Programming

ໂດຍການໄດ້ຮັບຄ່າຂອງ n ແລະ k ຈາກຜູ້ໃຊ້, ຄ່າສໍາປະສິດ binomial $\text{bin}(n, k)$ ອາດຈະໄດ້ຮັບຜົນດັ່ງທີ່ສະແດງຂ້າງລຸ່ມນີ້.

```
1 def bin2(n, k):
2     B = [[0] * (i + 1) for i in range(n + 1)]
3     for i in range(n + 1):
4         for j in range(i + 1):
5             if j == 0 or j == i:
6                 B[i][j] = 1
7             else:
8                 B[i][j] = B[i - 1][j - 1] + B[i - 1][j]
9     return B[n][k]
```

```
1 n = int(input("Input the value of n: "))
2 k = int(input("Input the value of k: "))
3 print(f"binomial({n}, {k}) is {bin2(n, k)}")
```

Input the value of n: 5
Input the value of k: 3
binomial(5, 3) is 10

| Pop quiz

Q1. ວິເຄາະຜົນການປະຕິບັດຂອງ code ຕໍ່ໄປນີ້ ແລະ ປຽບທຽບກັບການປະຕິບັດໜ້າທີ່ຂອງ fib1(), fib2(), ແລະ fib3().

```
1 def fib4(n):
2     if n <= 1:
3         return n
4     else:
5         a, b = 0, 1
6         for i in range(2, n + 1):
7             a, b = b, a + b
8         return b
```

```
1 N = int(input("Input a number: "))
2 print(fib4(N))
```

Input a number:

Q2. ວິເຄາະຜົນການປະຕິບັດຂອງ code ຕໍ່ໄປນີ້ ແລະປຽບທຽບກັບການປະຕິບັດໜ້າທີ່ຂອງ bin1() ແລະ bin2().

```
1 def bin3(n, k):
2     B = [0] * (n + 1)
3     for i in range(n + 1):
4         for j in range(n, -1, -1):
5             if j == 0 or j == i:
6                 B[j] = 1
7             else:
8                 B[j] = B[j] + B[j - 1]
9     return B[k]
```

```
1 for i in range(10):
2     for j in range(i + 1):
3         print(bin3(i, j), end=" ")
4     print()
```

| Pair programming



Pair Programming Practice

| ແນວທາງ, ກົນໄກ ແລະ ແຜນສຸກເສີນ

ການກະກຽມການສ້າງໂປຣແກຣມຮ່ວມກັນເປັນຄູ່ກ່ຽວຂ້ອງກັບການໃຫ້ຄໍາແນະນໍາແລະກົນໄກເພື່ອຊ່ວຍໃຫ້ນັກຮຽນຈັບຄູ່ຢ່າງຖືກຕ້ອງແລະ ໃຫ້ເຂົາເຈົ້າເຮັດວຽກເປັນຄູ່. ຕົວຢ່າງ, ນັກຮຽນຄວນປຸງ "ເຮັດ." ການກະກຽມທີ່ມີປະສິດຕິຜົນຕ້ອງໃຫ້ມີແຜນການສຸກເສີນໃນກໍລະນີທີ່ຄູ່ຮ່ວມງານຫນຶ່ງບໍ່ຢູ່ຫຼືຕັດສິນໃຈທີ່ຈະບໍ່ເຂົ້າຮ່ວມດ້ວຍເຫດຜົນໃດຫນຶ່ງ ຫຼືດ້ວຍເຫດຜົນອື່ນ. ໃນກໍລະນີເຫຼົ່ານີ້, ມັນເປັນສິ່ງສໍາຄັນທີ່ຈະເຮັດໃຫ້ມັນຊັດເຈນວ່ານັກຮຽນທີ່ມີປະຕິບັດໜ້າທີ່ຢ່າງຫ້າວຫັນຈະບໍ່ຖືກລົງໂທດຍ້ອນວ່າການຈັບຄູ່ບໍ່ໄດ້ຜິດ.

| ການຈັບຄູ່ທີ່ຄ້າຍຄືກັນ, ບໍ່ຈໍາເປັນຕ້ອງເທົ່າທຽມກັນ, ຄວາມສາມາດເປັນຄູ່ຮ່ວມງານ

ການຂຽນໂປຣແກຣມຄູ່ຈະມີປະສິດທິພາບເມື່ອນັກຮຽນຕັ້ງໃຈຮ່ວມກັນເຮັດວຽກ, ຊຶ່ງວ່າບໍ່ຈໍາເປັນຕ້ອງມີຄວາມຮູ້ເທົ່າທຽມກັນ, ແຕ່ຕ້ອງມີຄວາມສາມາດເຮັດວຽກເປັນຄູ່ຮ່ວມງານ. ການຈັບຄູ່ນັກຮຽນທີ່ບໍ່ສາມາດເຂົ້າກັນໄດ້ມັກຈະເຮັດໃຫ້ການມີສ່ວນຮ່ວມທີ່ບໍ່ສົມດຸນກັນ. ຄຸສອນຕ້ອງເນັ້ນຫນັກວ່າການຂຽນໂປຣແກຣມຄູ່ບໍ່ແມ່ນຍຸດທະສາດ “divide-and-conquer”, ແຕ່ຈະເປັນຄວາມພະຍາຍາມຮ່ວມມືເຮັດວຽກທີ່ແທ້ຈິງໃນທຸກໆດ້ານສໍາລັບໂຄງການທັງຫມົດ. ຄູຄວນຫຼີກເວັ້ນການຈັບຄູ່ນັກຮຽນທີ່ອ່ອນຫຼາຍກັບນັກຮຽນທີ່ເກັ່ງຫຼາຍ.

| ກະຕຸ້ນນັກຮຽນໂດຍການໃຫ້ສິ່ງຈູງໃຈພິເສດ

ການສະເໜີແຮງຈູງໃຈພິເສດສາມາດຊ່ວຍກະຕຸ້ນນັກຮຽນໃຫ້ຈັບຄູ່, ໂດຍສະເພາະກັບນັກຮຽນທີ່ມີຄວາມສາມາດຫຼາຍ. ຈະເຫັນວ່າມັນເປັນປະໂຫຍດທີ່ຈະໃຫ້ນັກຮຽນຈັບຄູ່ເຮັດວຽກຮ່ວມກັນພຽງແຕ່ຫນຶ່ງຫຼືສອງວຽກເທົ່ານັ້ນ.



Pair Programming Practice

| ປ້ອງກັນການໂກງໃນການເຮັດວຽກຮ່ວມກັນ

ສິ່ງທ້າທາຍສໍາລັບຄູແມ່ນເພື່ອຊອກຫາວິທີທີ່ຈະປະເມີນຜົນໄດ້ຮັບຂອງບຸກຄົນ, ໃນຂະນະທີ່ນໍາໃຊ້ຜົນປະໂຫຍດຂອງການຮ່ວມມື. ຈະຮູ້ໄດ້ແນວໃດວ່ານັກຮຽນຕັ້ງໃຈເຮັດວຽກ ຫຼື ກົງແຮງງານຜູ້ຮ່ວມງານ? ຜູ້ຊ່ຽວຊານແນະນໍາໃຫ້ທົບທວນຄືນການອອກແບບບັກສູດ ແລະ ການປະເມີນ ພ້ອມທັງປຶກສາຫາລືຢ່າງຈະແຈ້ງ ແລະ ຊັດເຈນກ່ຽວກັບພຶດຕິກຳຂອງນັກຮຽນທີ່ຈະຖືກຕິດຄວາມວ່າຂີ້ຕົວະ. ຜູ້ຊ່ຽວຊານເນັ້ນໜັກໃຫ້ຄູເຮັດການມອບໝາຍໃຫ້ມີຄວາມໝາຍຕໍ່ນັກຮຽນ ແລະ ອະທິບາຍຄຸນຄ່າຂອງສິ່ງທີ່ນັກຮຽນຈະຮຽນຮູ້ໂດຍການເຮັດສໍາເລັດ.

| ສະພາບແວດລ້ອມການຮຽນຮູ້ໃນການຮ່ວມມື

ສະພາບແວດລ້ອມການຮຽນຮູ້ໃນຮ່ວມກັນເກີດຂຶ້ນໄດ້ທຸກເວລາທີ່ຜູ້ສອນຮຽນຮູ້ອົງໃຫ້ນັກຮຽນເຮັດວຽກຮ່ວມກັນໃນກິດຈະກຳການຮຽນຮູ້. ສະພາບແວດລ້ອມການຮຽນຮູ້ຮ່ວມກັນສາມາດມີສ່ວນຮ່ວມທັງກິດຈະກຳທີ່ເປັນທາງການ ແລະ ບໍ່ເປັນທາງການ ແລະ ອາດຈະບໍ່ລວມເຖິງການປະເມີນໂດຍກົງ. ຕົວຢ່າງ, ນັກສຶກສາຄູ່ເຮັດວຽກມອບໝາຍຮ່ວມກັນໃນການຂຽນໂປຣແກຣມ; ນັກສຶກສາກຸ່ມນ້ອຍໆສິນທະນາຄໍາຕອບທີ່ເປັນໄປໄດ້ຕໍ່ກັບຄໍາຖາມຂອງອາຈານໃນລະຫວ່າງການບັນຍາຍ; ແລະ ນັກຮຽນເຮັດວຽກຮ່ວມກັນນອກຫ້ອງຮຽນເພື່ອຮຽນຮູ້ແນວຄວາມຄິດໃໝ່. ການຮຽນຮູ້ການຮ່ວມມືແມ່ນແຕກຕ່າງຈາກໂຄງການທີ່ນັກຮຽນ “divide and conquer.” ເມື່ອນັກຮຽນແບ່ງວຽກກັນ, ແຕ່ລະຄົນຮັບຜິດຊອບພຽງແຕ່ສ່ວນໜຶ່ງຂອງການແກ້ໄຂບັນຫາ ແລະ ຈະບໍ່ຄ່ອຍມີບັນຫາຫຍັງໃນການເຮັດວຽກຮ່ວມກັບຄົນອື່ນໃນທີມ. ໃນສະພາບແວດລ້ອມການເຮັດວຽກຮ່ວມກັນ, ນັກຮຽນມີສ່ວນຮ່ວມໃນການສິນທະນາປຶກສາຫາລືເຊິ່ງກັນແລະກັນ.

Q1. ດ້ວຍຈຳນວນຖ້ວນ n ໃນອາເຣໜຶ່ງມິຕິ, ໃຫ້ສ້າງຂັ້ນຕອນວິທີເພື່ອຊອກຫາວ່າເມື່ອໃດຜົນບວກຂອງຄ່າຕໍ່ເນື່ອງໃນອາເຣມີຄ່າສູງທີ່ສຸດ.

```
1 S = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
2 M = maximum_subarray(S)
3 print(M)
```

6

nums



maximum = 6

