Unit 23.

Queue

Learning objectives

- ✓ ເຂົ້າໃຈແນວຄວາມຄິດຂອງ queue ແລະ ສາມາດກຳນົດ queue ເປັນປະເພດຂໍ້ມູນນາມມະທຳ.
- ✓ ສາມາດສ້າງໂຄງສ້າງຂໍ້ມູນ queue ເປັນ class ໂດຍໃຊ້ພາສາ python.
- ✓ ສາມາດນຳໃຊ້ໂຄງສ້າງຂໍ້ມູນ queue ເພື່ອແກ້ໄຂບັນຫາ Josephus.

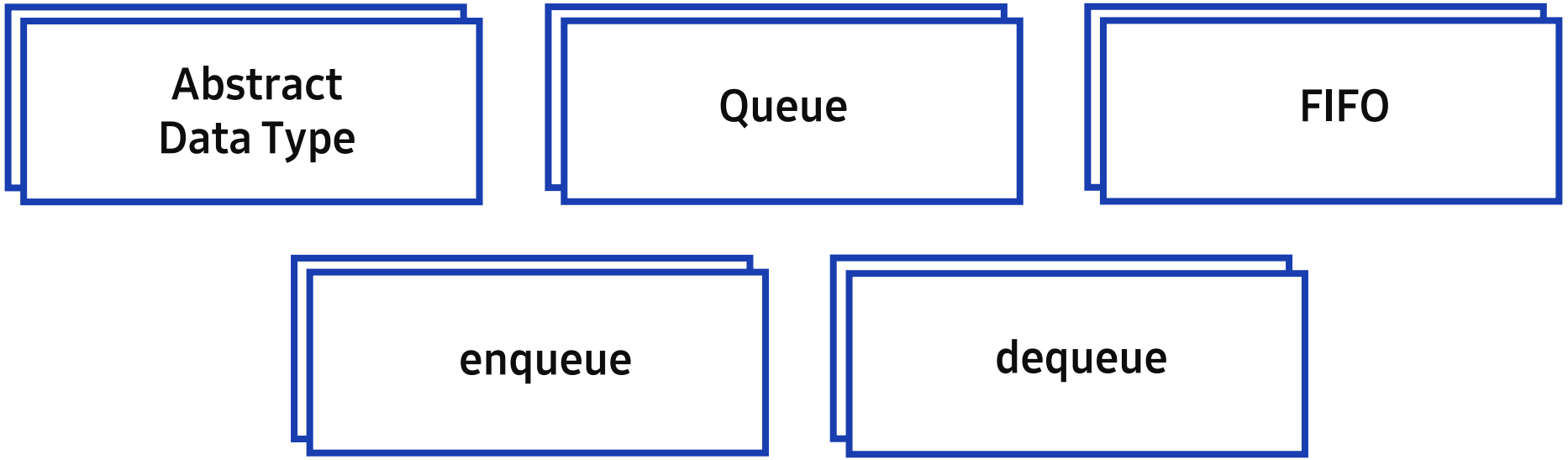
Learning overview

- ✓ ຮຽນຮູ້ວິທີການກຳນົດໂຄງສ້າງຂໍ້ມູນ queue ເປັນ class
- ✓ ຮຽນຮູ້ວິທີການນຳໃຊ້ queue class ກັບ methods ຂອງມັນ
- ✓ ຮຽນຮູ້ວິທີການແກ້ໄຂບັນຫາ Josephus ໂດຍໃຊ້ໂຄງສ້າງຂໍ້ມູນ queue

Concepts you will need to know from previous units

- ✓ ວິທີການປະກາດ list, ການເພີ່ມ ແລະ ການລຶບຂໍ້ມູນ
- ✓ ວິທີການກຳນົດ class constructor ແລະ ກຳນົດຕົວປ່ຽນ ແລະ methods
- ✓ ວິທີການສ້າງ class instance ເພື່ອເຂົ້າຫາຕົວປ່ຽນ ແລະ ເອີ້ນໃຊ້ methods

Keywords



Mission

1. Real world problem

1.1. ເລື່ອງ Josephus ໃນປະຫວັດສາດ

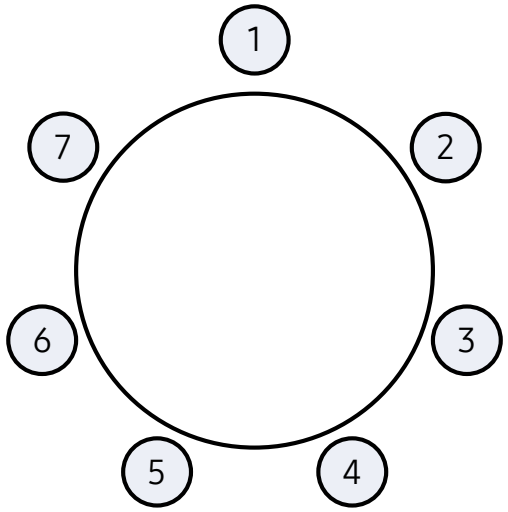
https://en.wikipedia.org/wiki/Josephus#/media/File:Josephus.jpg

- ເລື່ອງ Josephus ໃນປະຫວັດສາດໄດ້ກາຍເປັນບັນຫາການຂຽນ code ທີ່ຫນ້າສົນໃຈ.
- Josephus ເປັນນັກປະຫວັດສາດຊາວຢິວໃນສະຕະວັດທີ 1. ຫຼັງຈາກທີ່ໄດ້ຮັບການພ່າຍແພ້ຈາກສົງຄາມກັບ Rome, ທັງຫມົດ 41 ຄົນ ລວມທັງລາວເອງໄດ້ຫນີເຂົ້າໄປໃນຖ້ຳ.
- ຢູ່ໃນຖ້ຳ, ພວກເຂົາຢືນຢູ່ໃນວົງມົນ ແລະ ໄດ້ເຮັດຂໍ້ຕົກລົງວ່າທຸກໆຄົນທີ່ 3 ຈະຂ້າຕົວຕາຍຈົນກ່ວາບໍ່ມີໃຜລອດຊີວິດ.
- Josephus ຄິດວ່າການຂ້າຕົວເອງແມ່ນບໍ່ມີຄວາມຫມາຍ, ດັ່ງນັ້ນລາວໄດ້ຄິດໄລ່ແລະຢຶດເອົາຕຳແຫນ່ງທີ່ຈະເຮັດໃຫ້ລາວຢູ່ລອດຈົນເຖິງຄົນສຸດທ້າຍ. ຈາກນັ້ນ, ນິທານເລົ່າວ່າໃນທີ່ສຸດລາວໄດ້ລອດຊີວິດໂດຍການຊັກຊວນຄົນອື່ນທີ່ຍັງເຫຼືອ.

1. Real world problem

1.1. ເລື່ອງ Josephus ໃນປະຫວັດສາດ

- ສົມມຸດວ່າມີ 7 ຄົນນັ່ງອ້ອມໂຕະ.
- ແຕ່ລະຄົນແມ່ນໄດ້ກຳນົດໝາຍໃຫ້ ດ້ວຍຕົວເລກຈາກ 1 ຫາ 7 ຕາມເຂັມໂມງ. ວິທີການຕໍ່ໄປນີ້ຈະຖືກໃຊ້ເພື່ອເລືອກຕົວແທນ.



- ຍົກເວັ້ນທຸກໆຄົນທີ 3 ອອກຈາກຕາຕະລາງໂດຍເລີ່ມຕົ້ນດ້ວຍເລກທີ 1 ຕາມເຂັມໂມງ.
- ເຮັດຊ້ຳວິທີດຽວກັນຈາກຜູ້ທີ່ນັ່ງຢູ່ຂ້າງຫນຶ່ງທີ່ໄດ້ຖືກຍົກເວັ້ນກ່ອນຫນ້ານີ້.

- ຖ້າວິທີການຂ້າງເທິງນີ້ຖືກນຳໃຊ້ເພື່ອເລືອກເອົາຜູ້ຕາງຫນ້າຜູ້ຫນຶ່ງທີ່ຍັງຄົງຢູ່ສຸດທ້າຍ, ແລ້ວໃຜຈະເປັນຜູ້ຕໍ່ໄປ?

1. Real world problem

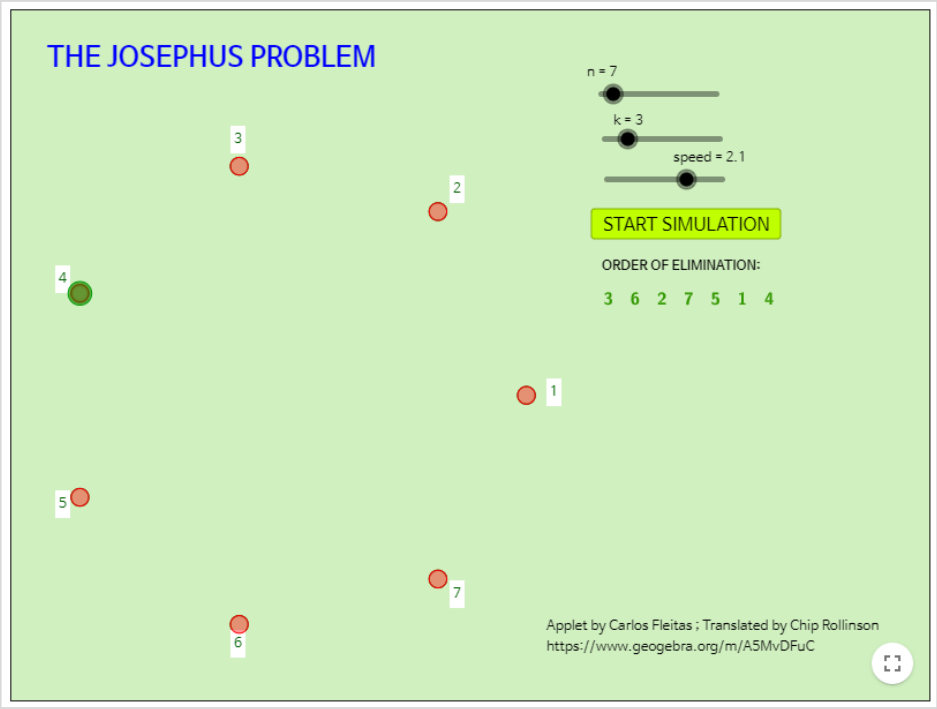
1.1. ເລື່ອງ Josephus ໃນປະຫວັດສາດ

ปัญหา Josephus สามารถกำหนดดังต่อไปนี้.

- N ຄົນມາເຕົ້າໂຮມກັນຮອບໂຕະ ແລະແຕ່ລະຄົນໄດ້ຮັບຕົວເລກຈາກ 1 ຫາ N ຕາມເຂັມໂມງ.
- ເລືອກ K ທີ່ນ້ອຍກວ່າ ຫຼື ເທົ່າກັນກັບ N, ແລະຍົກເວັ້ນຄົນທີ K ຈາກ ໝາຍເລກ 1.
- ຍົກເວັ້ນຄົນທີ່ K ຕາມເຂັມໂມງຈົນກ່ວາ $(N-1)$ ຄົນຖືກລົບລ້າງ.
- ຄິດໄລ່ຈຳນວນຜູ້ທີ່ຈະຢູ່ລອດຈົນກ່ວາສຸດທ້າຍເມື່ອສອງຈຳນວນບວກ N ແລະ $K(\leq N)$ ທີ່ໃຫ້ມາ.

1. Real world problem

1.2. ເຄື່ອງຈັກກຳນົດລຳດັບ Josephus



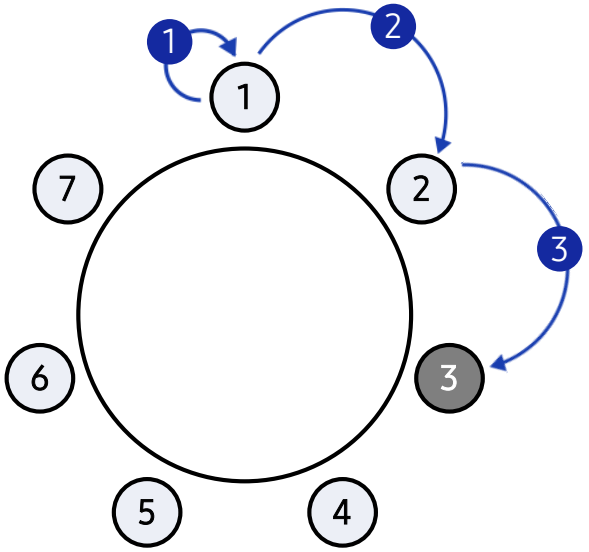
https://www.geogebra.org/m/ExvvrBbR

- GeoGebra ໄດ້ສ້າງ applet ສຳລັບການຈຳລອງບັນຫາ Josephus.
- ໃນປະຫວັດສາດ, ໂຈເຊັບໄດ້ເຂົ້າຮ່ວມໃນເກມທີ່ທຸກໆ ຄົນທີ 3 ຂ້າຕົວເອງໃນຈຳນວນ 41 ຄົນທີ່ນັ່ງຢູ່ໃນ ວົງມົນ.
- ດັ່ງນັ້ນ, ໃນບັນຫາ Josephus, N = 41 ແລະ K = 3. ແລ້ວໂຈເຊບໄດ້ເລກໃດ?
- ໂຈເຊັບໄດ້ລອດຊີວິດກັບຄົນອື່ນທີ່ຍັງຢູ່ລອດຈົນກ່ວາ ຄົນສຸດທ້າຍ. ແລ້ວ, ຕົວເລກຂອງລາວແມ່ນຫຍັງ?
- ໃຊ້ applet ເພື່ອກວດສອບເບິ່ງຄົນສອງຄົນທີ່ລອດ ຊີວິດຈາກເລື່ອງໂຈເຊັບ.

2. Mission

2.1. ບັນຫາ Josephus

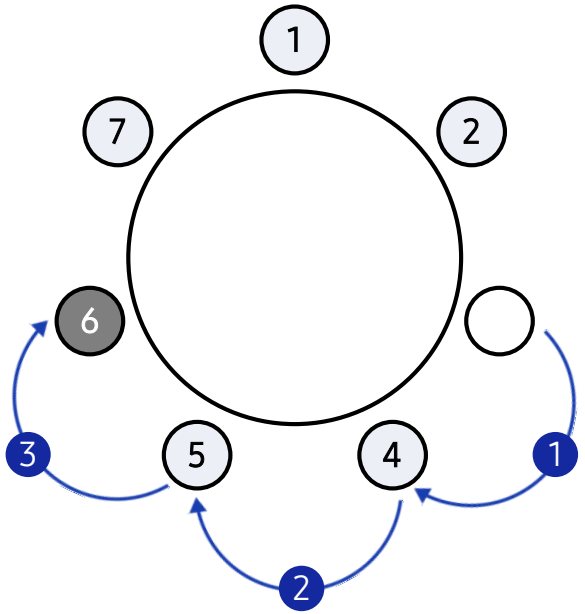
- ບັນຫາທຳອິດທີ່ພວກເຮົາມີແມ່ນບັນຫາ Josephus ໃນຕຳແໜ່ງທີ່ N = 7, K = 3.
- ເພື່ອແກ້ໄຂບັນຫານີ້, ເລີ່ມຕົ້ນດ້ວຍໝາຍເລກທີ່ 1 ທັງ 7 ຄົນແມ່ນນັ່ງອ້ອມຂ້າງໂຕະແລະຍົກເວັ້ນທຸກຄົນທີ 3.



2. Mission

2.1. ບັນຫາ Josephus

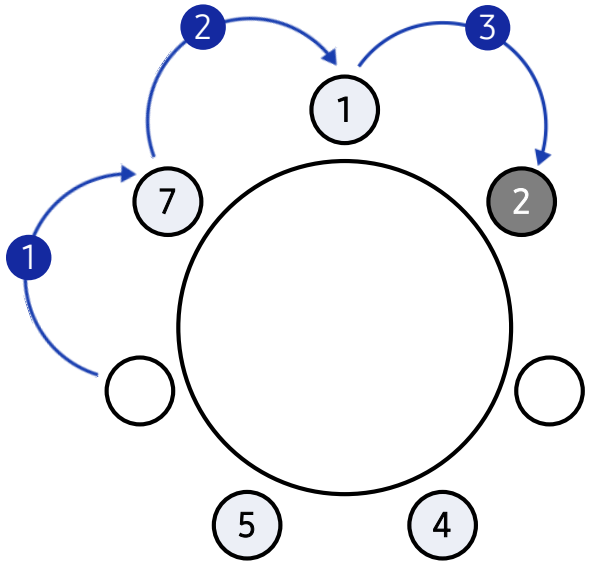
ຫຼັງຈາກເລກ 3 ຖືກລົບລ້າງ, ໝາຍເລກ 6 ຜູ້ທີ່ເປັນບຸກຄົນທີ 3 ຈາກໝາຍເລກ 4 ຄວນຖືກລົບລ້າງ.



2. Mission

2.1. ບັນຫາ Josephus

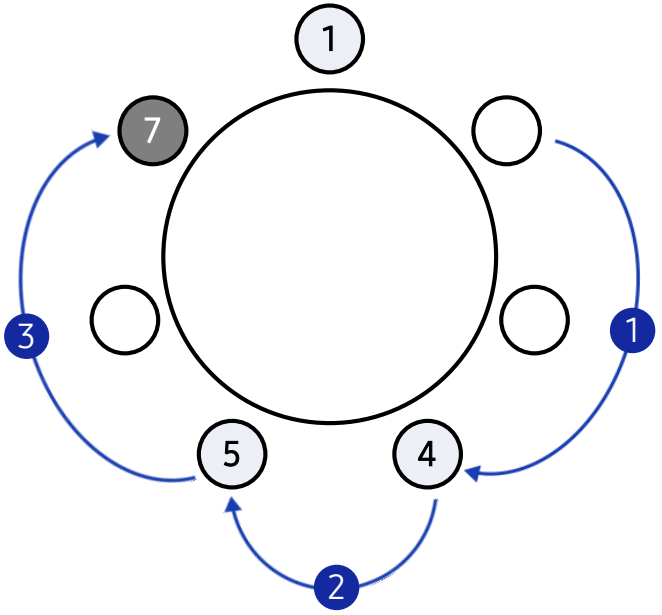
- ເມື່ອໝາຍເລກ 6 ຖືກຕັດອອກແລ້ວ, ອັນດັບທີ 3 ຕໍ່ໄປແມ່ນເບີ 2.
- ລົບລ້າງໝາຍ 2 ຈາກຕາຕະລາງ.



2. Mission

2.1. ບັນຫາ Josephus

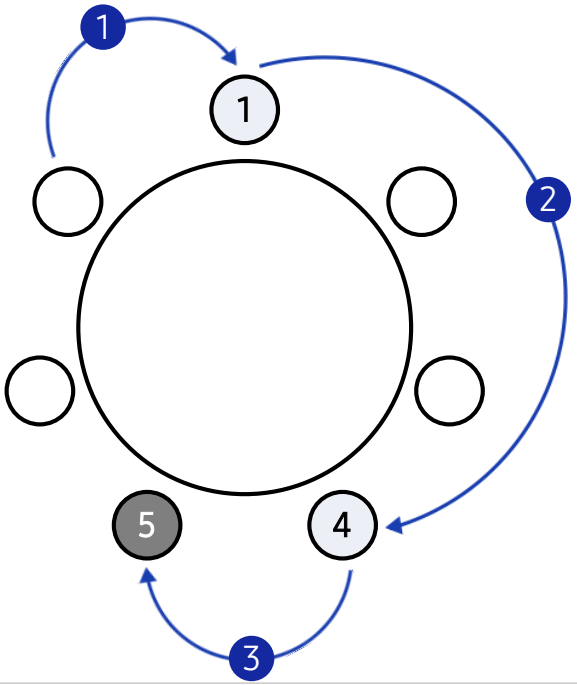
- ຂ້າມຄົນທີ່ຖືກຍົກລົບລ້າງແລ້ວ.
- ຫຼັງຈາກໝາຍເລກ 2 ຖືກຕັດອອກ, ບຸກຄົນທີ 3 ຖັດໄປແມ່ນໝາຍເລກ 7, ລົບເລກ 7.



2. Mission

2.1. ບັນຫາ Josephus

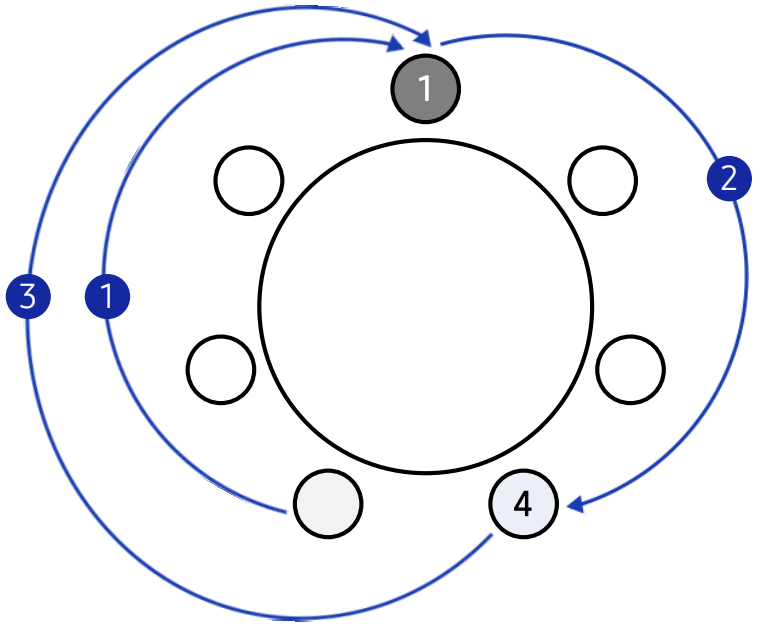
- ຫຼັກການເຊັ່ນດຽວກັນຖືກໃຊ້ ແລະ ຄົນຕໍ່ໄປທີ່ຈະຖືກລົບລ້າງຕໍ່ຈາກໝາຍເລກ 7 ແມ່ນໝາຍເລກ 5.
- ລົບລ້າງໝາຍເລກ 5.



2. Mission

2.1. ບັນຫາ Josephus

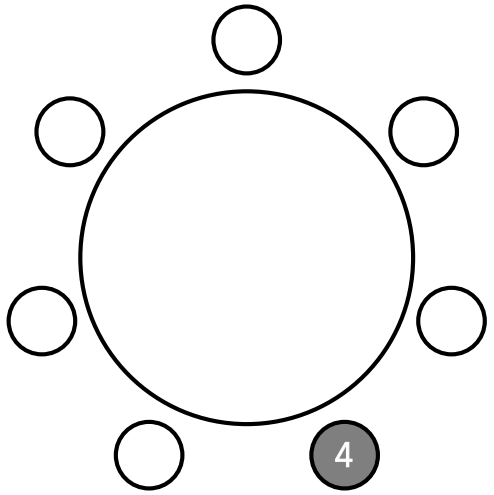
- ມີພຽງແຕ່ສອງຄົນເທົ່ານັ້ນທີ່ຍັງເຫຼືອສຸດທ້າຍ, ດັ່ງນັ້ນເມື່ອເລີ່ມຕົ້ນຈາກໝາຍເລກ 1, ບຸກຄົນທີ 3 ກາຍເປັນ ໝາຍເລກ 1.
- ດັ່ງນັ້ນ, ລົບລ້າງໝາຍເລກ 1.



2. Mission

2.1. ບັນຫາ Josephus

ຫຼັງຈາກລົບລ້າງໝາຍເລກ 1, ຍັງເຫຼືອພຽງແຕ່ຄົນສຸດທ້າຍ. ຕົວແທນສຸດທ້າຍແມ່ນໝາຍເລກ 4.



3. ການແກ້ໄຂບັນຫາ

3.1. ວິທີແກ້ໄຂບັນ Josephus

ໄດ້ຮັບ N ແລະ K ຂໍ້ມູນຂອງຜູ້ໃຊ້ທີ່ປ້ອນເຂົ້າເພື່ອພິມລໍາດັບຂອງ Josephus.

```
1 N = int(input("Input the number of people(N): "))
2 K = int(input("Input the number to be skipped(K): "))
3 print(josephus_sequence(N, K))
```

```
Input the number of people(N): 7
Input the number to be skipped(K): 3
[3, 6, 2, 7, 5, 1, 4]
```

3. ການແກ້ໄຂບັນຫາ

3.2. code ສຸດທ້າຍຂອງບັນຫາ Josephus

```python
def josephus_sequence(n, k):
    sequence = []
    queue = []
    for i in range(1, n + 1):
        queue.append(i)
    j = 1
    while len(queue) > 1:
        item = queue.pop(0)
        if j % k == 0:
            sequence.append(item)
        else:
            queue.append(item)
        j += 1
    sequence.append(queue.pop(0))
    return sequence
```

Key concept

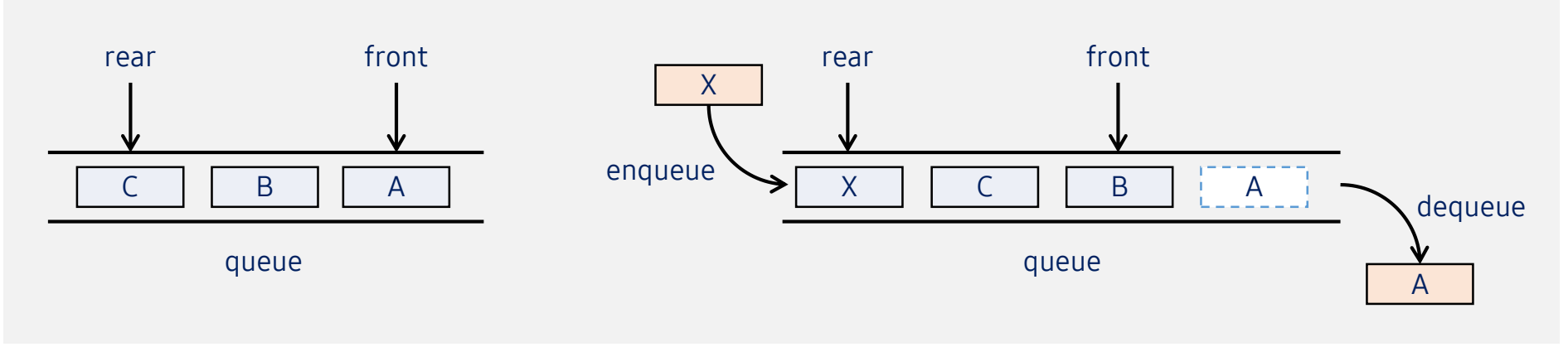
1. Queue ແມ່ນຫຍັງ?

1.1. ຄຳນິຍາມຂອງ queue

- ຄືອແມ່ນປະເພດຂໍ້ມູນນາມມະທຳທີ່ເກັບຂໍ້ມູນລຽງກັນຢ່າງມີລຳດັບ ມີທາງເຂົ້າ ແລະ ທາງອອກແຍກຈາກກັນ ການເອົາຂໍ້ມູນອອກ ແມ່ນເອົາອອກຈາກທາງຫນ້າ ແລະ ເອົາຂໍ້ມູນເຂົ້າຢູ່ທາງຫລັງ.
- ຂໍ້ມູນທີ່ເອົາເຂົ້າກ່ອນຈະຖືກເອົາອອກກ່ອນ ແລະ ຂໍ້ມູນທີ່ເອົາເຂົ້ານຳຫຼັງແມ່ນເອົາອອກນຳຫຼັງ, ຄືອແມ່ນໂຄງສ້າງຂໍ້ມູນທີ່ເຮັດວຽກ ດ້ວຍວິທີການ FIFO (First-In-First-Out).

Focus ຄືອມີສອງຕືອດຳເນີນການຫຼັກ.

- enqueue: ເຮັດໜ້າທີ່ເພີ່ມຂໍ້ມູນໃໝ່ເຂົ້າດ້ານຫຼັງຂອງຄືอ
- dequeue: ເອົາຂໍ້ມູນອອກຈາກທາງໜ້າຂອງຄືอ



1. Queue ແມ່ນຫຍັງ?

1.2. ຕົວຢ່າງ ການໃຊ້ queue ໃນຊີວິດປະຈຳວັນ

- ຄິວແມ່ນພົບເຫັນເປັນປະຈຳໃນຊີວິດຂອງພວກເຮົາ.
- ຕົວຢ່າງ, ສຳລັບການຈອງຄິວຊື້ປີ້ຍົນຢູ່ສະໜາມບິນ, ປະຊາຊົນຈຳເປັນຕ້ອງລໍຖ້າຢູ່ໃນຄິວ ແລະ ຜູ້ທຳອິດທີ່ເຂົ້າມາໃນຄິວແມ່ນໄດ້ຮັບການບໍລິການກ່ອນ.
- ນອກເໜືອຈາກນັ້ນ, ຄິວແມ່ນພົບເຫັນທົ່ວໄປໃນຮ້ານອາຫານ ແລະ ທະນາຄານ.

2. ການສ້າງ Queue

2.1. ການສ້າງ Queue ໃນຮູບແບບ class

ໃນພາສາ Python, ປະເພດ list ສາມາດຖືກນຳໃຊ້ເພື່ອກຳນົດໂຄງສ້າງຂໍ້ມູນຄືອເປັນ class.

```
1  class Queue:
2
3      def __init__(self):
4          self.queue = []
5
6      def is_empty(self):
7          return True if len(self.queue) == 0 else False
8
9      def enqueue(self, item):
10         self.queue.append(item)
11
12     def dequeue(self):
13         return None if self.is_empty() else self.queue.pop(0)
```

2. ການສ້າງ Queue

2.1. ການສ້າງ Queue ໃນຮູບແບບ class

ຄ້າຍກັບ class ຂອງ stack, class ຂອງຄິວກຳນົດຄິວເປັນປະເພດຂໍ້ມູນ list ຜ່ານ constructor method.

```
1  class Queue:
2
3      def __init__(self):
4          self.queue = []
5
6      def is_empty(self):
7          return True if len(self.queue) == 0 else False
8
9      def enqueue(self, item):
10         self.queue.append(item)
11
12     def dequeue(self):
13         return None if self.is_empty() else self.queue.pop(0)
```

Line 1-4

- ກຳນົດໂຄງສ້າງຂໍ້ມູນຄິວເປັນ queue class.
- Constructor __init__ (self) ເລີ່ມຕົ້ນ ເອົາ field ຂໍ້ມູນສະມາຊິກຂອງຄິວເຂົ້າໄປໃນ list ທີ່ຫວ່າງເປົ່າ.

2. ການສ້າງ Queue

2.1. ການສ້າງ Queue ໃນຮູບແບບ class

'enqueue' ຈະຖືກສ້າງເປັນ method ຂອງ queue class ດັ່ງສະແດງຂ້າງລຸ່ມນີ້.

```
1  class Queue:
2
3      def __init__(self):
4          self.queue = []
5
6      def is_empty(self):
7          return True if len(self.queue) == 0 else False
8
9      def enqueue(self, item):
10         self.queue.append(item)
11
12     def dequeue(self):
13         return None if self.is_empty() else self.queue.pop(0)
```

Line 9-10

- Method enqueue() ໄດ້ຮັບ 'self' ແລະ ຂໍ້ມູນທີ່ຈະເພີ່ມໃສ່ຄືอเป็นพารามิเตอร์.
- 'enqueue' ເພີ່ມຂໍ້ມູນໃສ່ໃນຕອນທ້າຍຂອງ list self.queue ເຊິ່ງເປັນຊ່ອງສະມາຊິກຂອງ queue class.

2. ການສ້າງ Queue

2.1. ການສ້າງ Queue ໃນຮູບແບບ class

'dequeue' ຈະຖືກສ້າງເປັນ method ຂອງຄົວ class ດັ່ງທີ່ສະແດງຂ້າງລຸ່ມນີ້.

```
1  class Queue:
2
3      def __init__(self):
4          self.queue = []
5
6      def is_empty(self):
7          return True if len(self.queue) == 0 else False
8
9      def enqueue(self, item):
10         self.queue.append(item)
11
12     def dequeue(self):
13         return None if self.is_empty() else self.queue.pop(0)
```

Line 12-13

- Method dequeue() ໄດ້ຮັບ 'self' ເປັນພາຣາມິເຕີ.
- 'dequeue' ຈະສົ່ງຄືນຄ່າ None ຖ້າ self.is_empty() ມີຄ່າເປັນຈິງ(True) ແລະ ຖ້າບໍ່ແມ່ນຄວາມຈິງ, ມັນຈະລຶບອົງປະກອບສຸດທ້າອົດຂອງ self.queue.

2. ການສ້າງ Queue

2.1. ການສ້າງ Queue ໃນຮູບແບບ class

Method is_empty() ທີ່ໃຊ້ເພື່ອກວດສອບຄືວ່າງເປົ່າຫຼືບໍ່ ມັນຈະຖືກກຳນົດເປັນ method ຂອງ queue class.

```
1  class Queue:
2
3      def __init__(self):
4          self.queue = []
5
6      def is_empty(self):
7          return True if len(self.queue) == 0 else False
8
9      def enqueue(self, item):
10         self.queue.append(item)
11
12     def dequeue(self):
13         return None if self.is_empty() else self.queue.pop(0)
```

Line 6-7

- Method is_empty() ຈະສົ່ງຄ່າກັບຄືນເປັນ True ຖ້າຄວາມຍາວຂອງ self.queue ມີຄ່າເປັນ 0 ແລະ ຖ້າບໍ່ເປັນດັ່ງນັ້ນ, ມັນຈະສົ່ງຄືນຄ່າ False.

2. ການສ້າງ Queue

2.1. ການສ້າງ Queue ໃນຮູບແບບ class

Code ຕໍ່ໄປນີ້ທົດສອບໂຄງສ້າງຂໍ້ມູນຄິວທີ່ຖືກກຳນົດເປັນ class.

```
1 queue = Queue()
2 queue.enqueue("A")
3 queue.enqueue("B")
4 print(queue.dequeue())
5 queue.enqueue("C")
6 print(queue.dequeue())
7 print(queue.dequeue())
8 print(queue.dequeue())
```

```
A
B
C
None
```

Line 1-8

- ຕົວປ່ຽນຄິວຈະບັນທຶກ ຕົວຢ່າງຂອງ object queue class ທີ່ສ້າງຂຶ້ນໂດຍ constructor Queue().
- Methods enqueue() ແລະ dequeue() ຂອງ object ຄິວ ສາມາດຖືກເອີ້ນໃຊ້ໂດຍໃຊ້ຕົວປະຕິບັດການອ້າງອີງ '.'
- ພິມຄ່າທີ່ສົ່ງຄືນຂອງຟັງຊັນ queue.dequeue() ເປັນ print() ເພື່ອກວດສອບລຳດັບການເອົາຂໍ້ມູນອອກຈາກຄິວ.

One More Step

- จะเธัดแมอใดถ้าต้องการรู้จำนวนอิงปะกอบใน Queue?
- Method size() ที่ใช้จำนวนอิงปะกอบในถือปะจุบันสามาดถึกสร้างโดยการสิ่งถืมถอาม ยาว list ถือของ queue class ปัดจุบัน.

```
1  class Queue:
2
3      # ......
4
5      def size(self):
6          return len(self.queue)
```

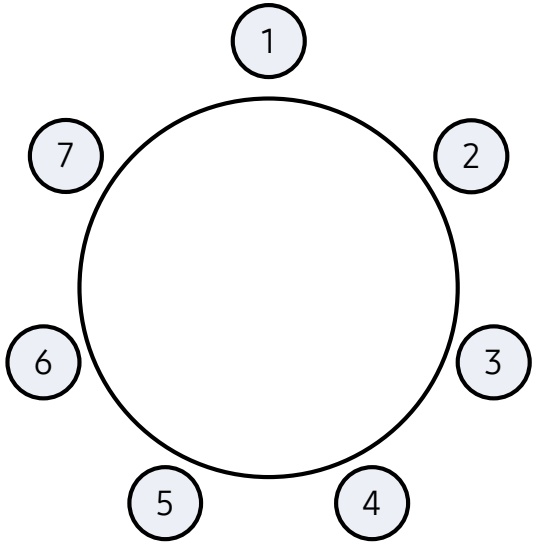
Let’s code

1. ບັນຫາ Josephus

1.1. ນິຍາມບັນຫາ

ຄິດໄລ່ຈຳນວນຄົນທີ່ຖືກຂ້າຕ໋ອຕາຍຈາກກຸ່ມເມື່ອນຳໃຊ້ກົດລະບຽບຕໍ່ໄປນີ້ກັບສອງຕ໋ອເລກທຳມະຊາດ N ແລະ K.

- *N* ຄົນມານັ່ງອ້ອມໂຕະມົນ ແລະ ແຕ່ລະຄົນຖືກກຳນົດດ້ວຍຕ໋ອເລກຈາກ 1 ຫາ N ຕາມທິດຂອງເຂັມໂມງ.
- ສຳລັບ K ທີ່ນ້ອຍກວ່າ ຫຼື ເທົ່າກັບ N, ເອົາຄົນທີ K ອອກໄປ ໂດຍເລີ່ມຕົ້ນນັບຈາກຄົນທີ 1.
- ເອົາຄົນທີ K ອອກໄປ ຕາມທິດຂອງເຂັມໂມງ ຈົນກວ່າທຸກຄົນຈະຖືກເອົາອອກໃຫ້ໝົດ.

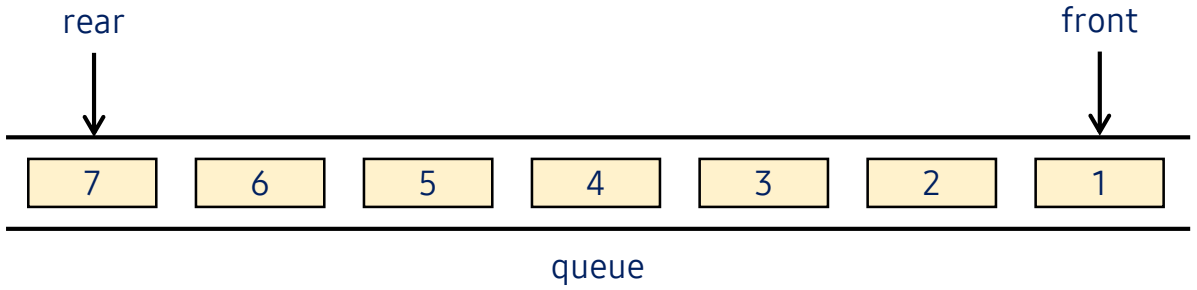


ຖ້າ N=7, K=3, ລຳດັບຂອງການເອົາຄົນ ອອກຈາກໂຕະແມ່ນ [3, 6, 2, 7, 5, 1, 4]

1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

- ໃຊ້ໂຄງສ້າງຂໍ້ມູນ Queue ເພື່ອແກ້ໄຂບັນຫາ Josephus ໄດ້ຢ່າງງ່າຍດາຍ.
- ທຳອິດ, ເອົາຄົນທີ 1 ຫາ N ເຂົ້າໄປໃນຄິວຕາມລຳດັບ.
- ຖ້າ N=7, ມັນຈະມີລັກສະນະດັ່ງຕໍ່ໄປນີ້.
- ເນື່ອງຈາກວ່າຍັງບໍ່ທັນໄດ້ເອົາຄົນອອກຈາກໂຕະເທື່ອ, ອັນດັບດັ່ງກ່າວແມ່ນຖືກກຳນົດເປັນ list ເປົ່າ.

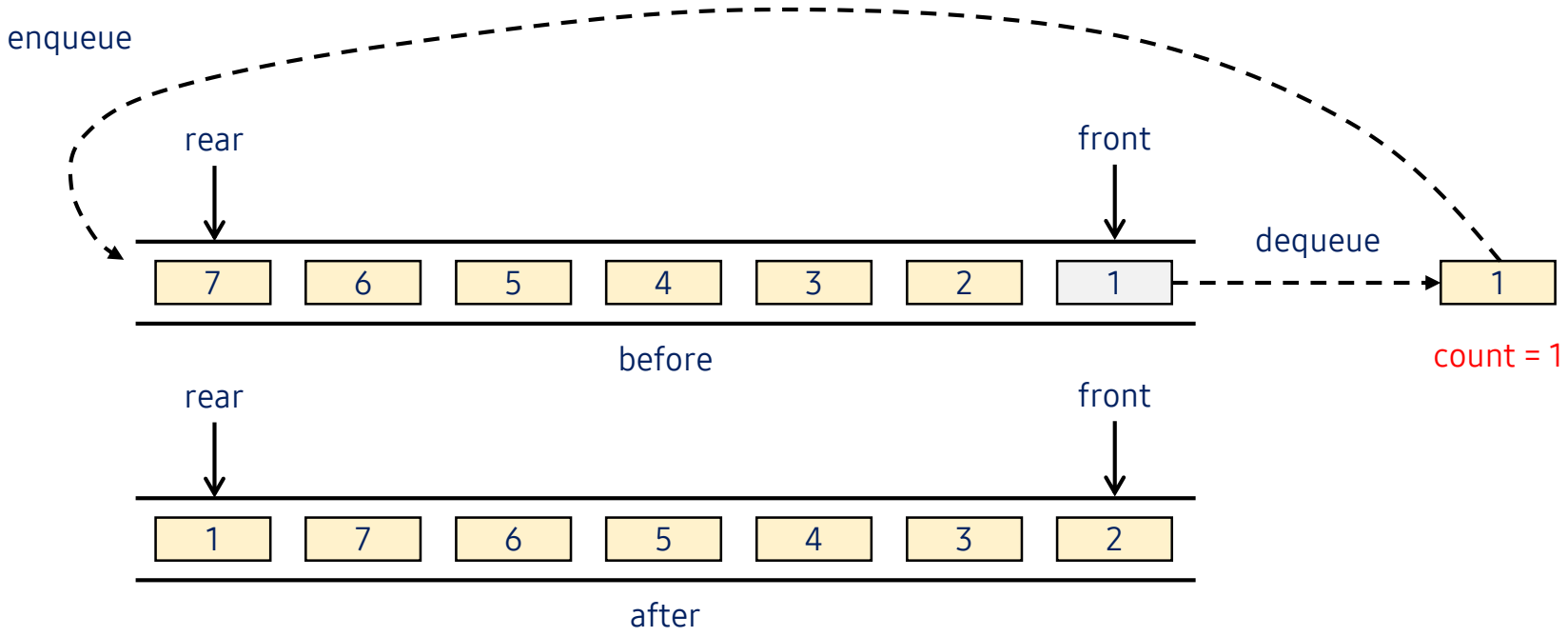


sequence = []

1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

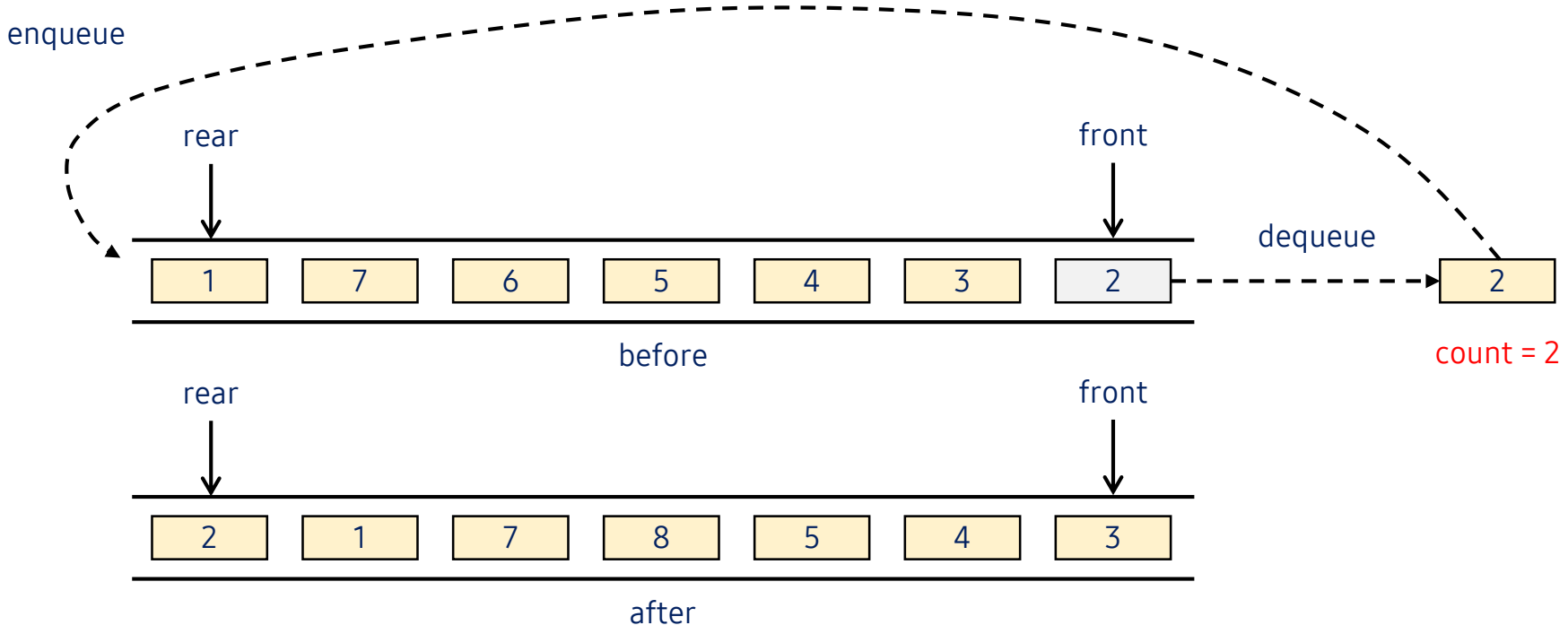
- ເອົາອົງປະກອບໜຶ່ງອອກຈາກຄິວ ແລະ ເພີ່ມມັນກັບຄືນເຂົ້າໄປໃນ Queue. ຢ່າງໃດກໍ່ຕາມ, ຢ່າເພີ່ມອົງປະກອບຕົວທີ K ກັບຄືນເຂົ້າໄປໃນຄິວ.
- ເລີ່ມຕົ້ນໂດຍການເອົາອົງປະກອບທຳອິດອອກ ແລະ ເພີ່ມມັນກັບຄືນເຂົ້າໄປໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດ້ວຍ່າວໂດຍໃຊ້ queue

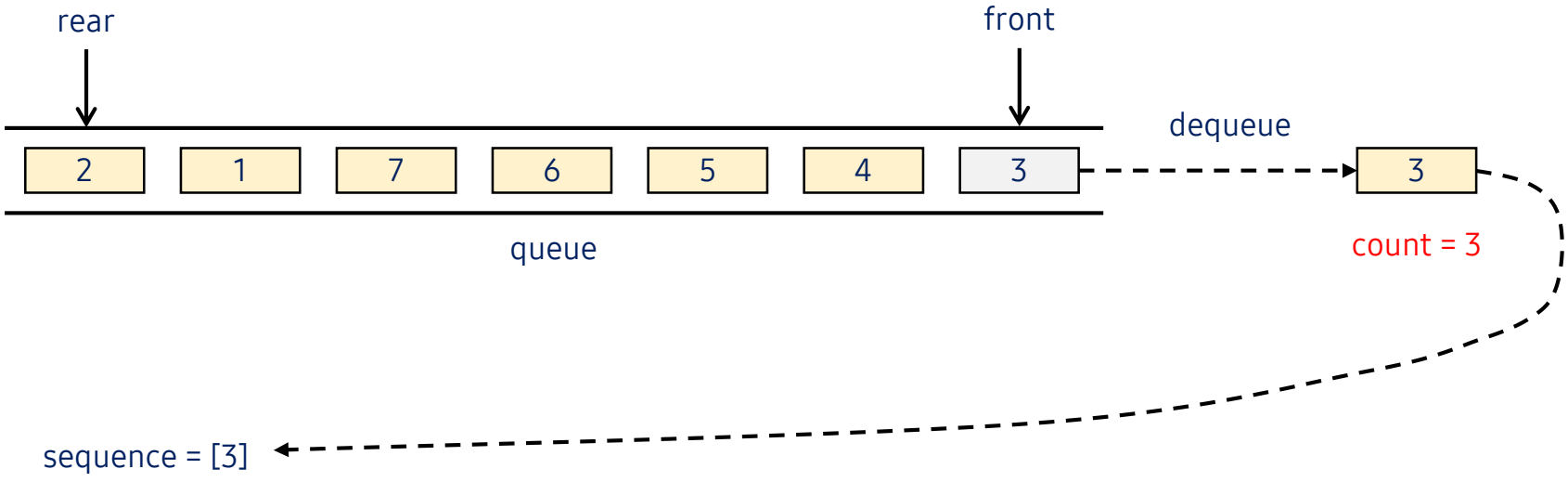
- ເມື່ອເອົາອົງປະກອບທີ 2 ອອກມາ, ແລ້ວເພີ່ມອົງປະກອບທີ 2 ກັບຄືນເຂົ້າໄປໃນຄິວ.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

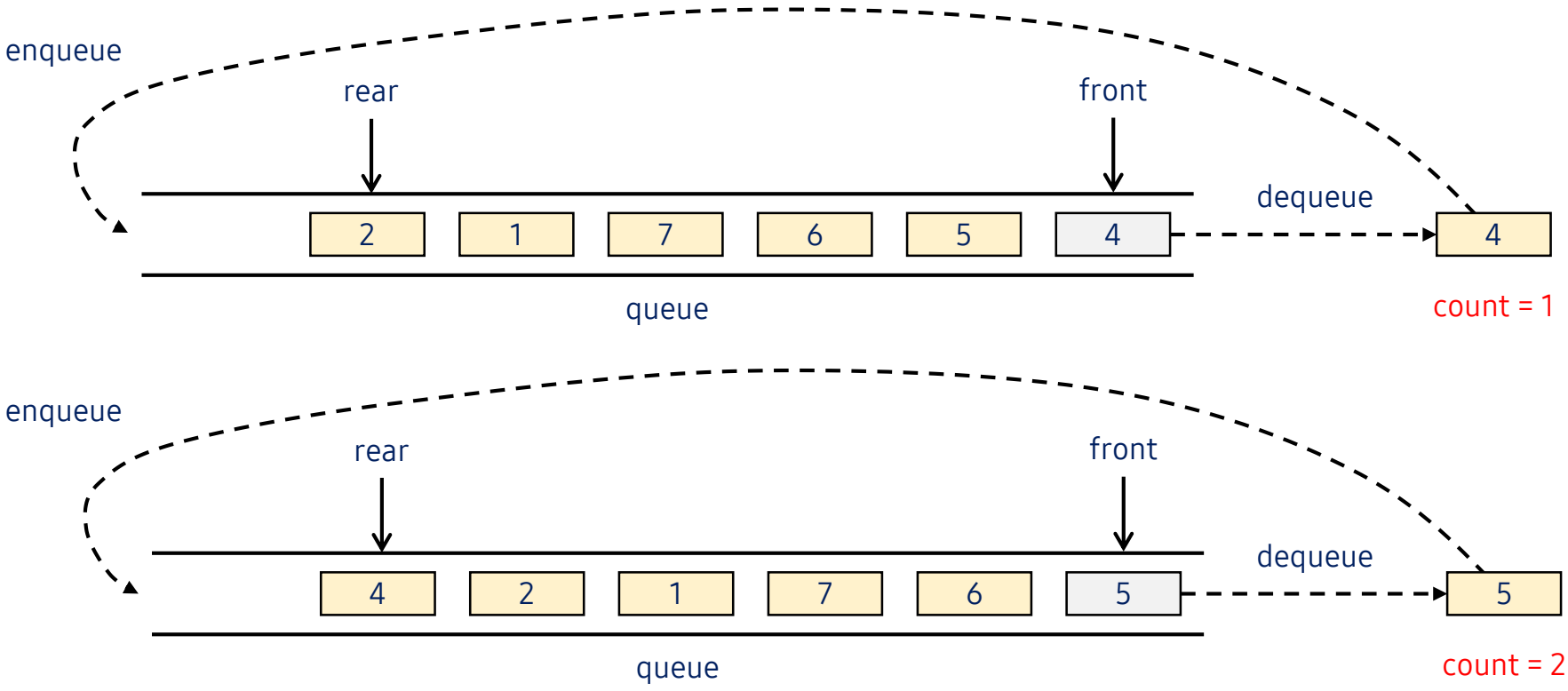
- ເມື່ອເອົາອົງປະກອບຕົວທີ 3 ອອກມາ, ແລ້ວລົບລ້າງມັນໂດຍບໍ່ຕ້ອງເພີ່ມກັບຄືນເຂົ້າໄປໃນຄິວ.(K=3).
- ໝາຍເລກ 3 ທີ່ຖືກລົບລ້າງອອກໄປຈະຖືກເພີ່ມເຂົ້າໄປໃສ່ໃນ list ຕາມອັນດັບ.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

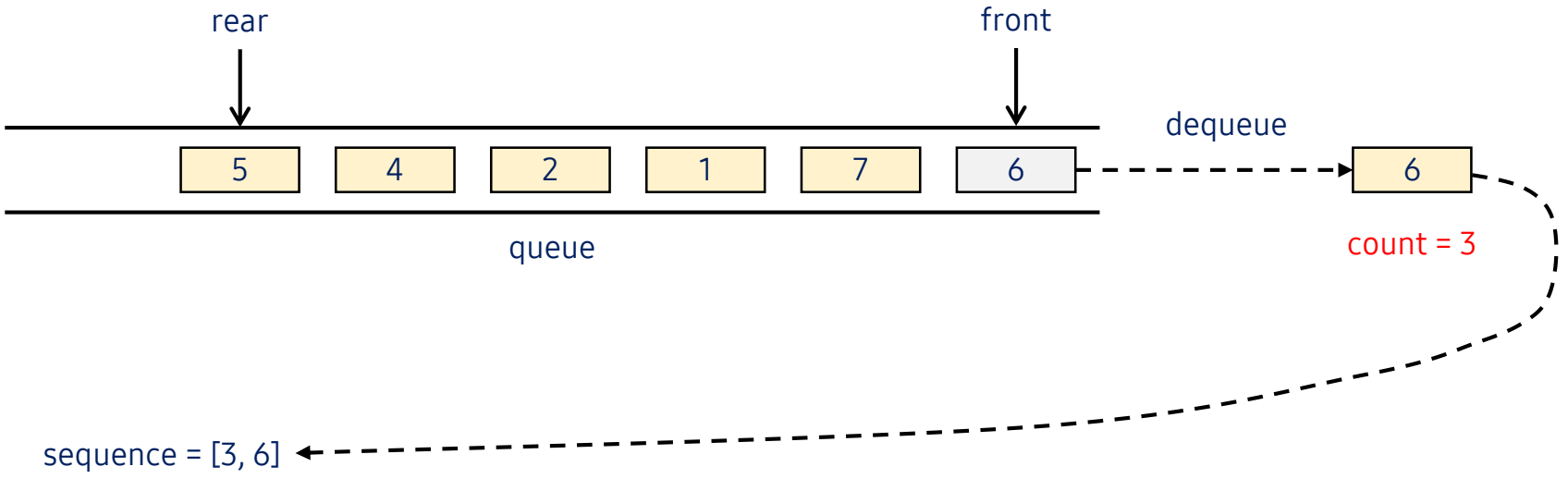
ຕໍ່ໄປ, ເອົາອົງປະກອບທີ 4 ແລະ 5 ອອກຈາກຄິວ, ຫຼັງຈາກນັ້ນເພີ່ມພວກມັນກັບຄືນເຂົ້າໄປໃນ Queue ອີກ.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

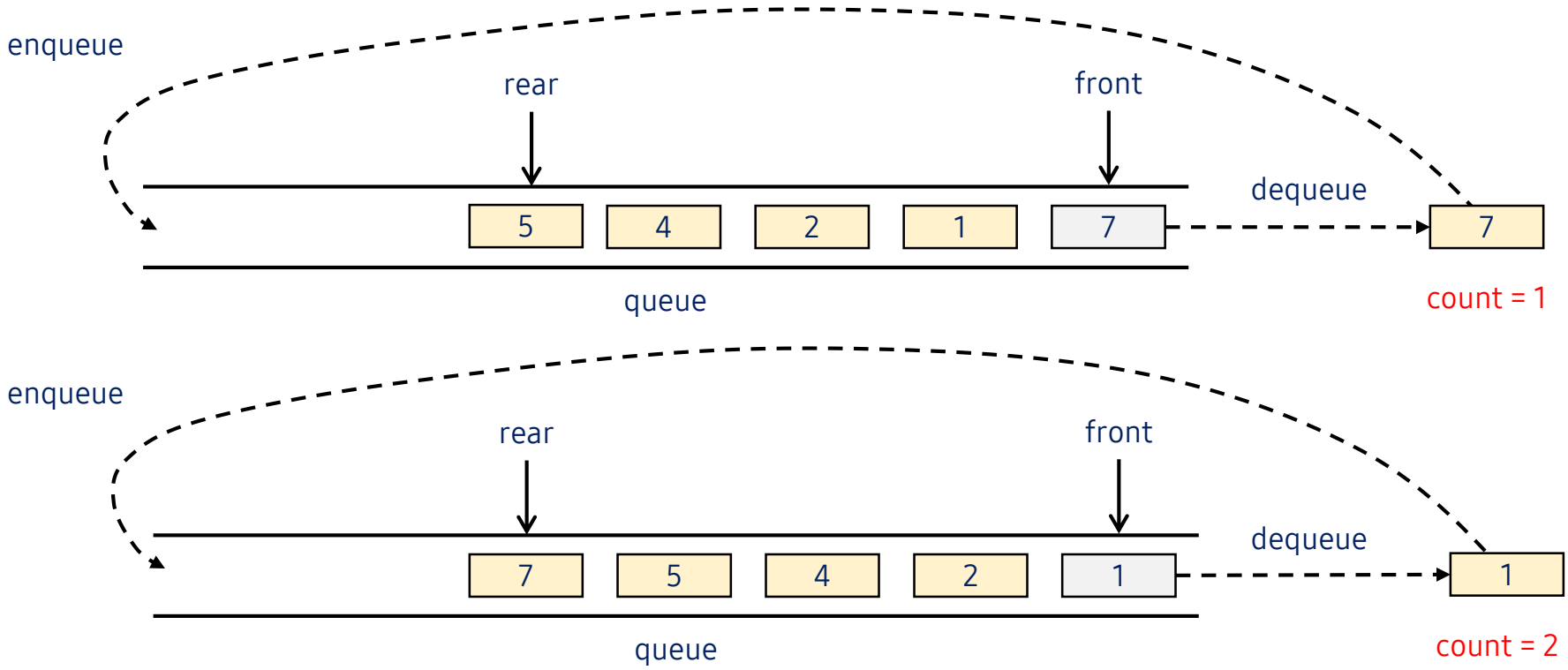
ໝາຍເລກ 6 ແມ່ນອົງປະກອບຕົວທີ 3, ສະນັ້ນໃຫ້ເພີ່ມມັນເຂົ້າໄປໃນ list ຕາມອັນດັບ ໂດຍບໍ່ຈຳເປັນ ຕ້ອງເພີ່ມມັນກັບຄືນເຂົ້າໄປໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດ້ວຍ່າວໂດຍໃຊ້ queue

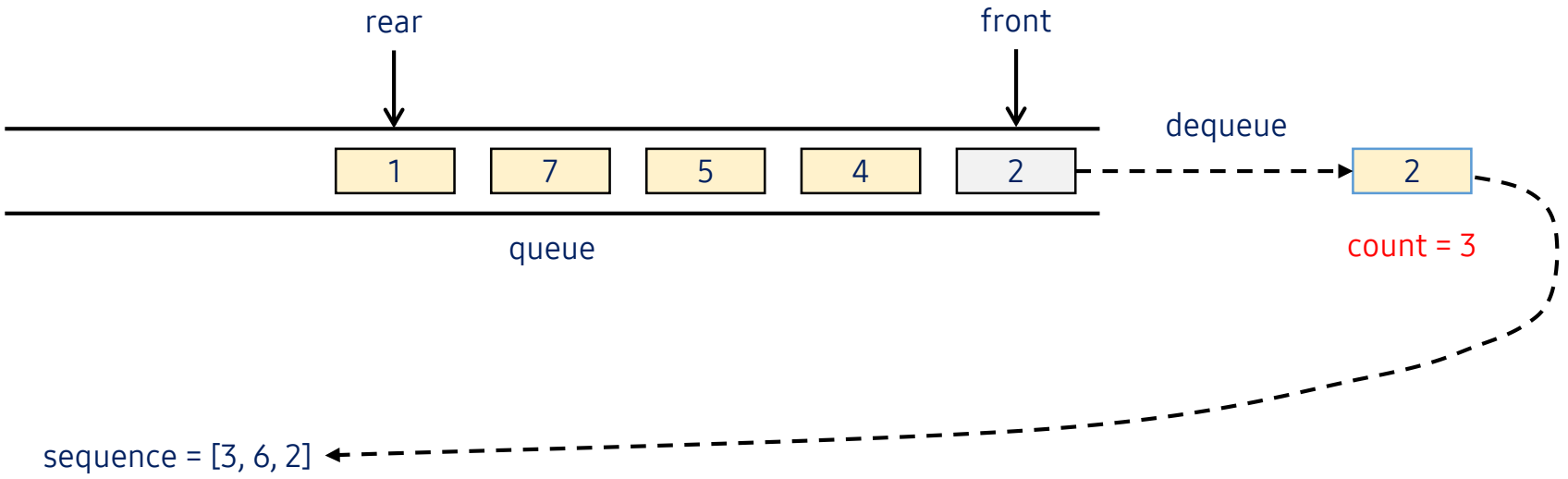
ຫຼັງຈາກເອົາໝາຍເລກ 7 ແລະ 1 ອອກມາ, ແລ້ວເພີ່ມພວກມັນກັບຄືນເຂົ້າໄປໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

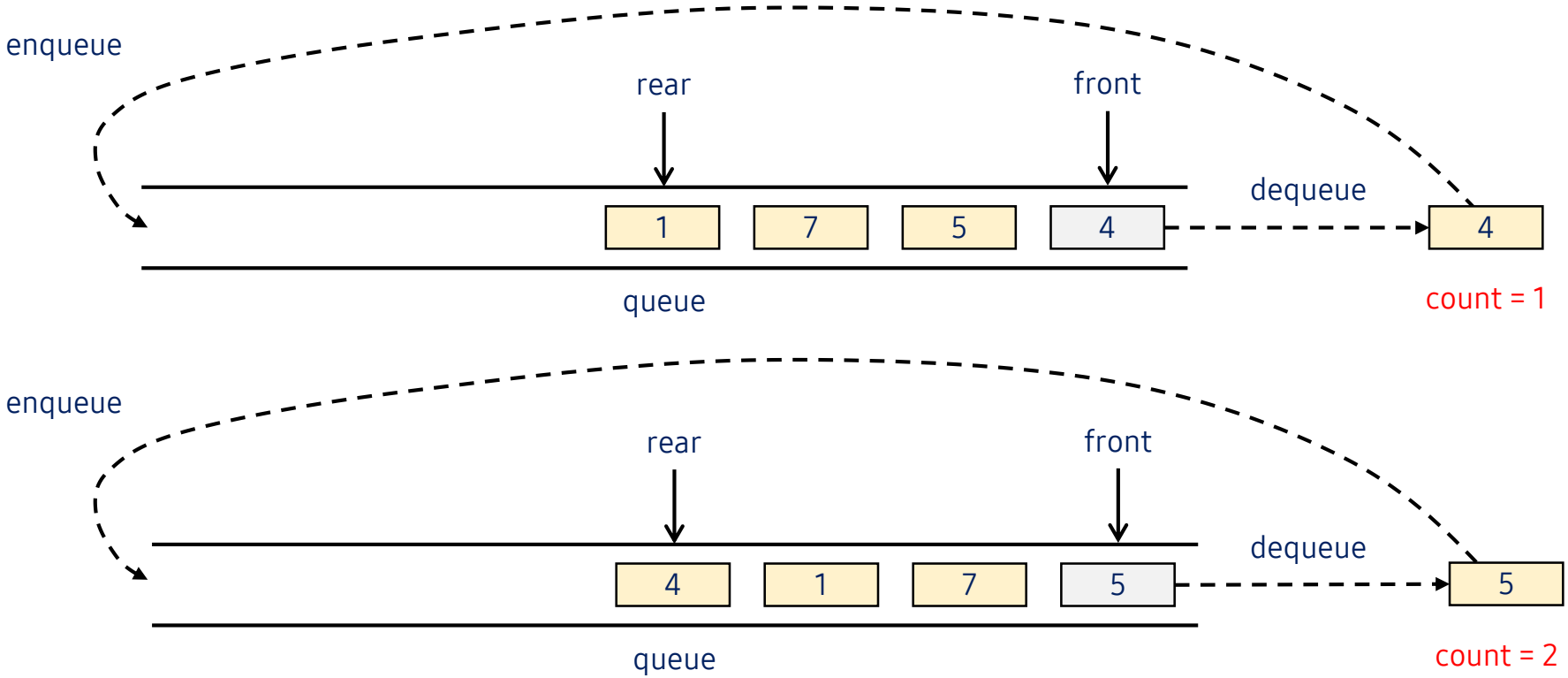
- ໝາຍເລກ 2 ແມ່ນອົງປະກອບທີ່ K=3, ສະນັ້ນເພີ່ມມັນໃສ່ໃນ list ຕາມອັນດັບ, ບໍ່ເພີ່ມມັນເຂົ້າໄປໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

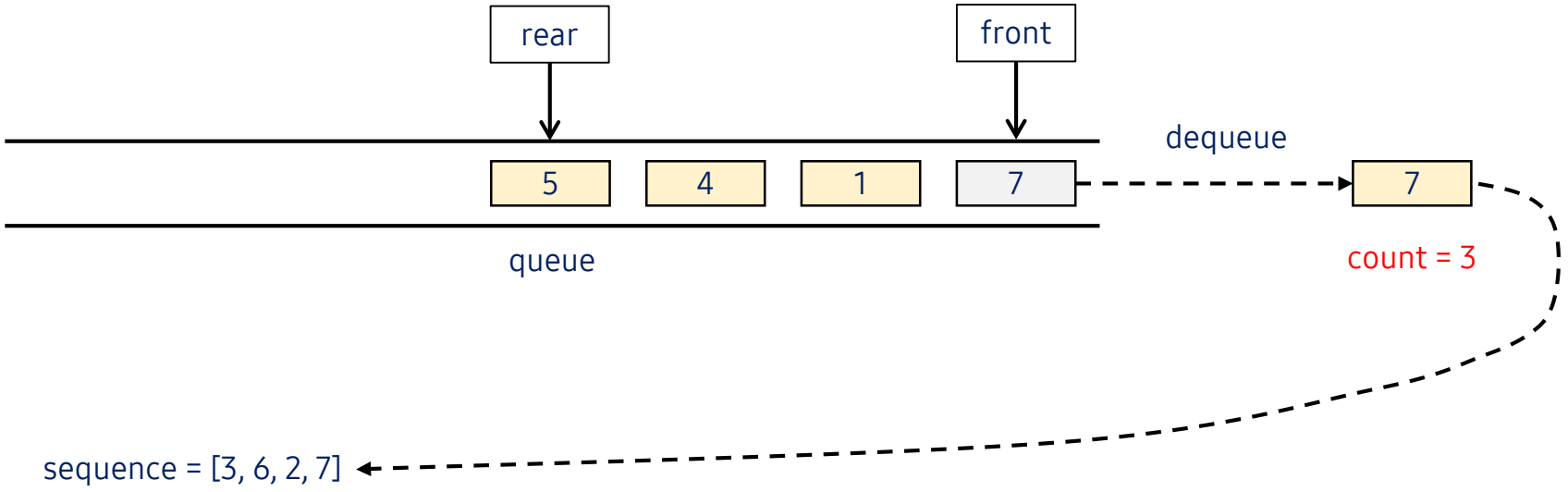
ເຮັດຊ້ຳຂະບວນການແບບດຽວກັນຈົນກວ່າຍັງເຫຼືອອົງປະກອບຫນຶ່ງຢູ່ໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

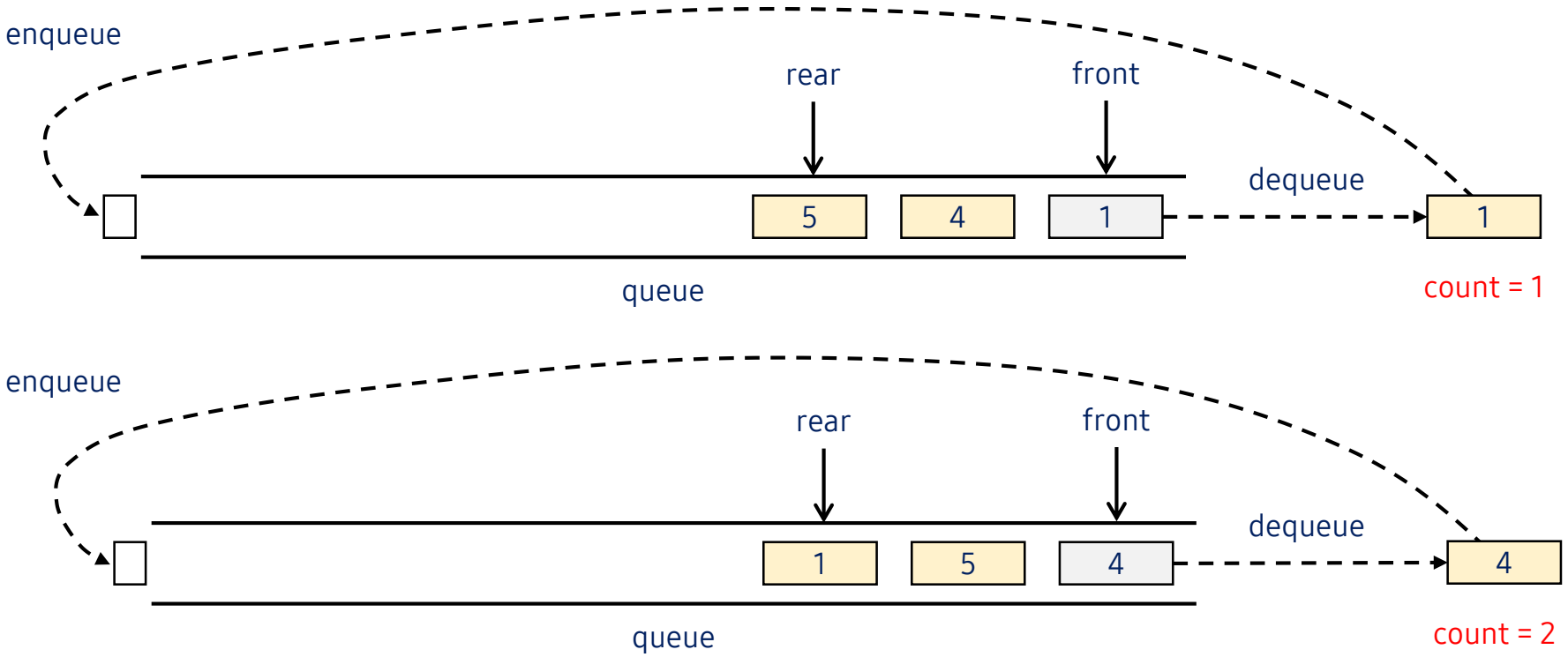
ເຮັດຊ້ຳຂະບວນການແບບດຽວກັນຈົນກວ່າຍັງເຫຼືອອົງປະກອບຫນຶ່ງຢູ່ໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

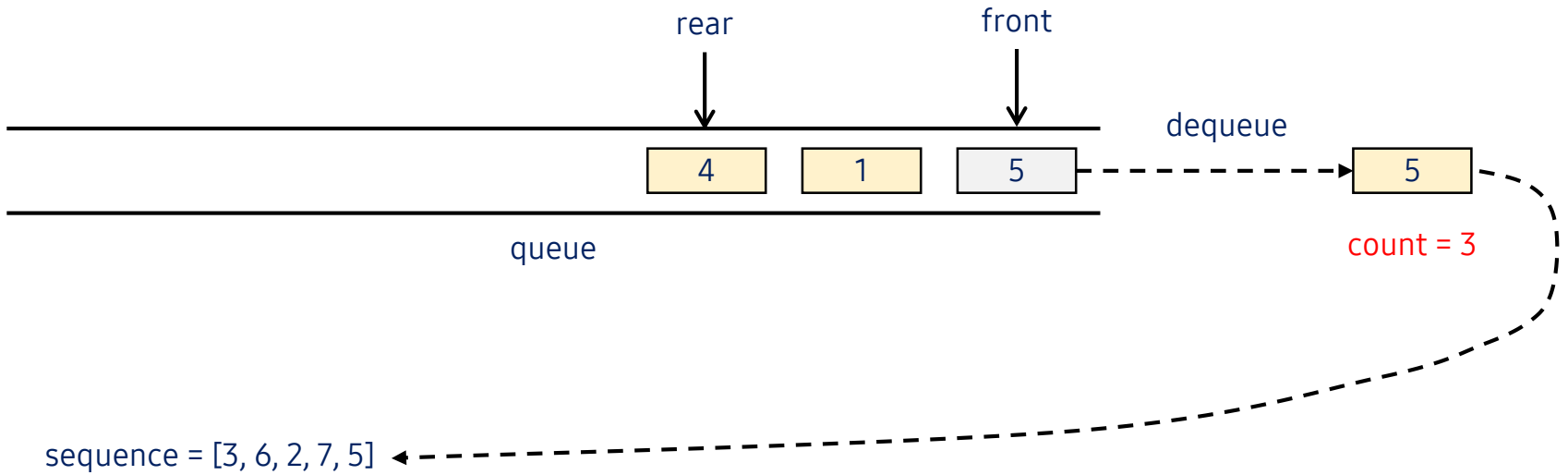
ເຮັດຊ້ຳຂະບວນການແບບດຽວກັນຈົນກວ່າຍັງເຫຼືອອົງປະກອບຫນຶ່ງຢູ່ໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດ້ວຍ່າອໂດຍໃຊ້ queue

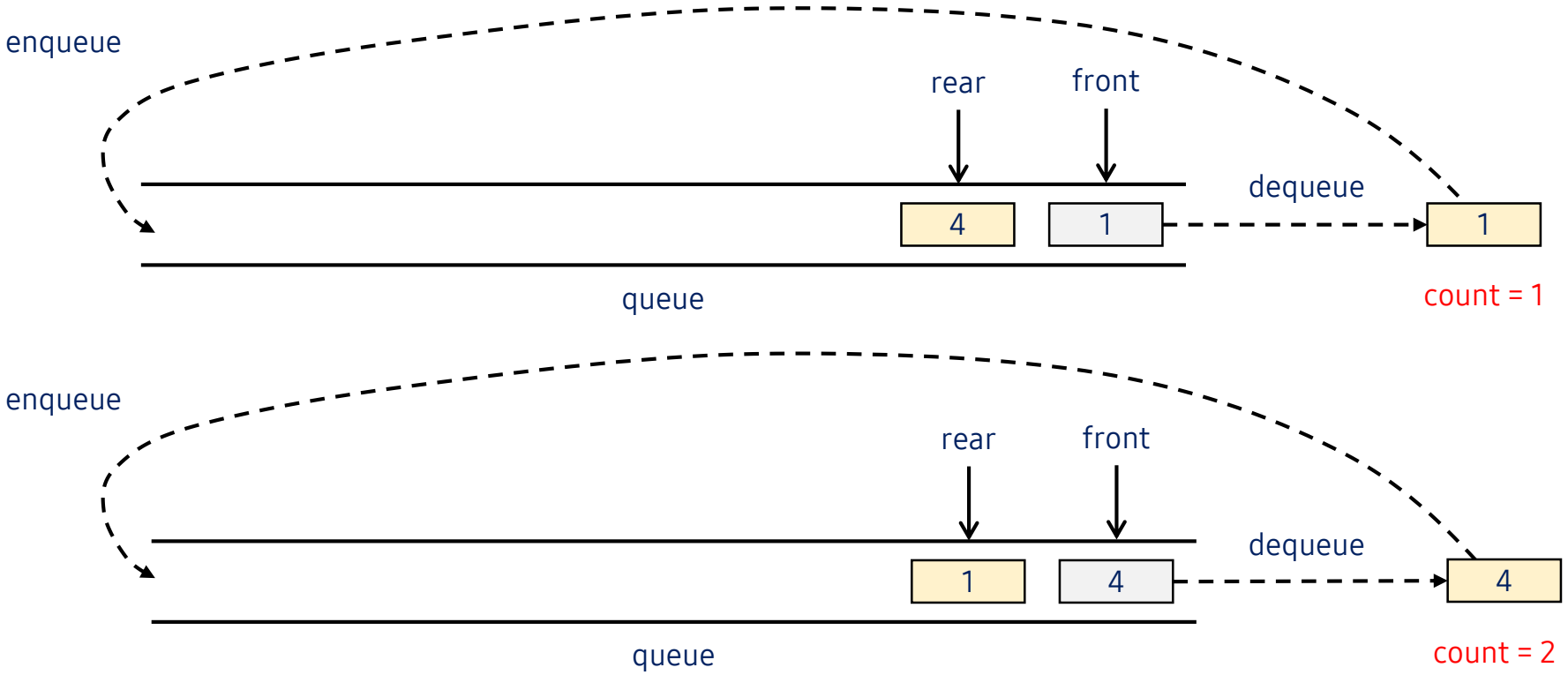
ເຮັດຊ້ຳຂະບວນການແບບດຽວກັນຈົນກວ່າຍັງເຫຼືອອົງປະກອບຫນຶ່ງຢູ່ໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

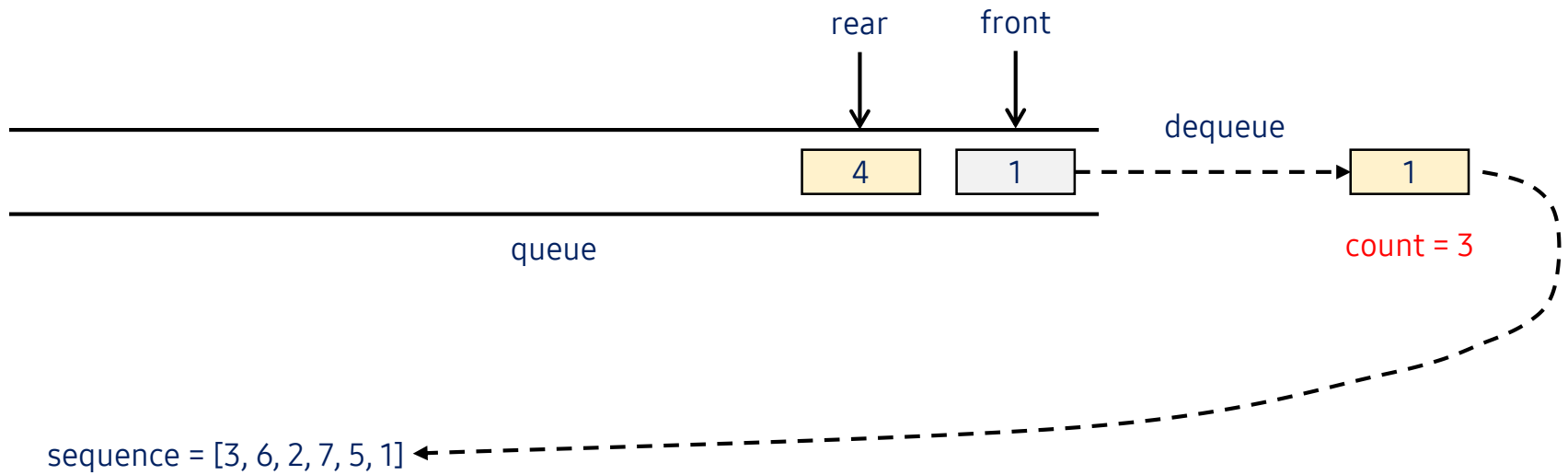
ເຮັດຊ້ຳຂະບວນການແບບດຽວກັນຈົນກວ່າຍັງເຫຼືອອົງປະກອບຫນຶ່ງຢູ່ໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດ້ວຍ່າອໂດຍໃຊ້ queue

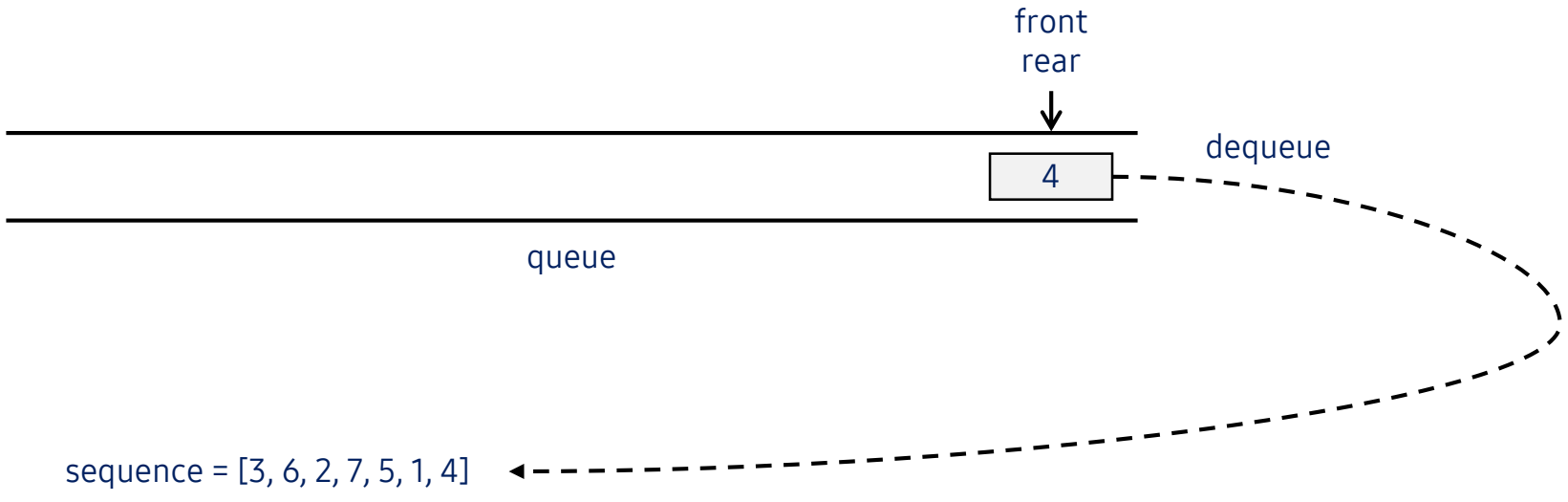
ເຮັດຊ້ຳຂະບວນການແບບດຽວກັນຈົນກວ່າຍັງເຫຼືອອົງປະກອບຫນຶ່ງຢູ່ໃນ Queue.



1. ບັນຫາ Josephus

1.2. ແກ້ໄຂບັນຫາດັ່ງກ່າວໂດຍໃຊ້ queue

- ຖ້າຍັງເຫຼືອອົງປະກອບດຽວຢູ່ໃນ Queue, dequeue ມັນອອກມາ ແລະ ເພີ່ມມັນເປັນອົງປະກອບສຸດທ້າຍຂອງ list ຕາມອັນດັບ.



1. ບັນຫາ Josephus

1.3. ສ້າງ ແລະ ຂຽນ code

ປະຕິບັດຕາມຂັ້ນຕອນທີ່ຜ່ານມາ ໂດຍຂຽນເປັນຊອດໂຄດພາສາ Python. ທຳອິດ, ສ້າງຟັງຊັນ josepush() ທີ່ຮັບຈຳນວນຖ້ວນ n ແລະ k ເປັນພາຣາມິເຕີປ້ອນເຂົ້າ.

```
1  def josephus(n, k):
2      sequence = []
3      queue = Queue()
4      for i in range(1, n + 1):
5          queue.enqueue(i)
6      j = 1
7      while queue.size() > 1:
8          item = queue.dequeue()
9          if j % k == 0:
10             sequence.append(item)
11         else:
12             queue.enqueue(item)
13         j += 1
14     sequence.append(queue.dequeue())
15     return sequence
```

Line 1-3

- ຟັງຊັນ josephus() ສົ່ງຄືນຄ່າການປ່ຽນແປງຂອງ Josephus ເຂົ້າໄປໃນ list ໂດຍຜ່ານພາຣາມິເຕີປ້ອນເຂົ້າ n ແລະ k.
- ລຳດັບທີ່ສົ່ງກັບຄືນມາແມ່ນເລີ່ມຕົ້ນດ້ວຍ list ຫວ່າງເປົ່າ, ໃນຂະນະທີ່ຄິວຖືກເລີ່ມຕົ້ນໃນຖານະທີ່ເປັນ object ໜຶ່ງຂອງ queue class ທີ່ໄດ້ກຳນົດໄວ້ກ່ອນຫນ້ານີ້.

1. ปัญหา Josephus

1.3. สร้าง และ ຂຽນ code

```
1  def josephus(n, k):
2      sequence = []
3      queue = Queue()
4      for i in range(1, n + 1):
5          queue.enqueue(i)
6      j = 1
7      while queue.size() > 1:
8          item = queue.dequeue()
9          if j % k == 0:
10             sequence.append(item)
11         else:
12             queue.enqueue(item)
13         j += 1
14     sequence.append(queue.dequeue())
15     return sequence
```

Line 4-5

- Enqueue ຕົວເລກຖ້ວນຈາກ 1 ຫາ N ເຂົ້າໄປໃນ Queue ກ່ອນ.

1. ບັນຫາ Josephus

1.3. ສ້າງ ແລະ ຂຽນ code

```
1  def josephus(n, k):
2      sequence = []
3      queue = Queue()
4      for i in range(1, n + 1):
5          queue.enqueue(i)
6      j = 1
7      while queue.size() > 1:
8          item = queue.dequeue()
9          if j % k == 0:
10             sequence.append(item)
11         else:
12             queue.enqueue(item)
13         j += 1
14     sequence.append(queue.dequeue())
15     return sequence
```

Line 6-7, 13

- 'j' ເປັນຕົວນັບເມື່ອເອົາຂໍ້ມູນອອກຈາກຄິວ. ດັ່ງນັ້ນ, ມັນຈະເລີ່ມຕົ້ນເຮັດຈາກໝາຍເລກ 1.
- ຈາກນັ້ນ, ເພີ່ມຄ່າ j ຂຶ້ນເທື່ອລະ 1 ໂດຍການເຮັດລົ້ມຄືນການ enqueue ແລະ dequeue ຈົນກ່ວາຍັງເຫຼືອພຽງແຕ່ຫນຶ່ງ ລາຍການຢູ່ໃນ Queue.
- ຖ້າສ່ວນທີ່ເຫຼືອຂອງການແບ່ງດ້ວຍ j ກັບ k ເທົ່າ 0, ໝາຍຄວາມລາຍການນີ້ແມ່ນຕົວທີ k ຕໍ່ໄປ ທີ່ຈະຖືກ dequeued.

1. ບັນຫາ Josephus

1.3. ສ້າງ ແລະ ຂຽນ code

```
1  def josephus(n, k):
2      sequence = []
3      queue = Queue()
4      for i in range(1, n + 1):
5          queue.enqueue(i)
6      j = 1
7      while queue.size() > 1:
8          item = queue.dequeue()
9          if j % k == 0:
10             sequence.append(item)
11         else:
12             queue.enqueue(item)
13         j += 1
14     sequence.append(queue.dequeue())
15     return sequence
```

Line 8-12

- Dequeue ຫນຶ່ງລາຍການຈາກຄິວ.
- ຖ້າ j%k == 0, ນັ້ນແມ່ນເປັນລາຍການທີ k, ໃຫ້ເພີ່ມມັນເຂົ້າໄປຕໍ່ທ້າຍຂອງ list ອັນດັບ.
- ຖ້າ j%k != 0, ລາຍການນີ້ຈະບໍ່ຖືກ dequeued, ສະນັ້ນເພີ່ມມັນກັບຄືນເຂົ້າໄປໃນ Queue ບ່ອນເກົ່າ.
- ເຮັດຊ້ຳຂະບວນການຈົນກ່ວາຍັງເຫຼືອພຽງແຕ່ຫນຶ່ງອົງປະກອບໃນຄິວ ແລະ ອົງປະກອບນັ້ນຈະຖືກເອົາອອກມາແລ້ວຖືກບັນທຶກເຂົ້າໄປໃນ list ອັນດັບ.

1. ບັນຫາ Josephus

1.3. ສ້າງ ແລະ ຂຽນ code

```
1  def josephus(n, k):
2      sequence = []
3      queue = Queue()
4      for i in range(1, n + 1):
5          queue.enqueue(i)
6      j = 1
7      while queue.size() > 1:
8          item = queue.dequeue()
9          if j % k == 0:
10             sequence.append(item)
11         else:
12             queue.enqueue(item)
13         j += 1
14     sequence.append(queue.dequeue())
15     return sequence
```

Line 14-15

- ເມື່ອປະຕິບັດ while loop ສຳເລັດ, ມີພຽງແຕ່ອົງປະກອບດຽວຢູ່ໃນ Queue, ເຊິ່ງຈະຖືກ dequeued ເປັນຕົວສຸດທ້າຍ.
- ດັ່ງນັ້ນ, ມັນເປັນໄປໄດ້ທີ່ຈະພິມ list ອັນດັບ ໂດຍການເອົາອົງປະກອບອອກຈາກຄິວ ແລະ ເພີ່ມມັນເຂົ້າໄປໃນຕອນທ້າຍຂອງ list ອັນດັບ.

1. ບັນຫາ Josephus

1.3. ສ້າງ ແລະ ຂຽນ code

ນຳໃຊ້ຕົວຢ່າງທີ່ຜ່ານມາກັບຟັງຊັນ josephus() ເພື່ອໃຫ້ໄດ້ຜົນຕໍ່ໄປນີ້.

```
1 N = int(input("Input the number of people(N): "))
2 K = int(input("Input the number to be skipped(K): "))
3 print(josephus(N, K))
```

```
Input the number of people(N): 7
Input the number to be skipped(K): 3
[3, 6, 2, 7, 5, 1, 4]
```

```
1 N = int(input("Input the number of people(N): "))
2 K = int(input("Input the number to be skipped(K): "))
3 print(josephus(N, K))
```

```
Input the number of people(N): 41
Input the number to be skipped(K): 3
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 1, 5, 10, 14, 19, 23, 28, 32, 37, 41, 7, 13, 20, 26, 34, 40,
8, 17, 29, 38, 11, 25, 2, 22, 4, 35, 16, 31]
```

Pop quiz

Quiz. #1

ຂຽນຜົນໄດ້ຮັບຂອງ codes ຈາກຕົວຢ່າງທີ່ໃຊ້ Queue.

```
queue = Queue()
queue.enqueue("Banana")
queue.enqueue("Apple")
queue.enqueue("Tomato")
queue.dequeue()
queue.enqueue("Strawberry")
queue.enqueue("Grapes")
queue.dequeue()
print(queue.queue)
```

Quiz. #2

ຂຽນຜົນໄດ້ຮັບຂອງ codes ຈາກຕົວຢ່າງທີ່ໃຊ້ Queue.

```
queue = Queue()
items = [10 * i for i in range(1, 11)]
for item in items:
    queue.enqueue(item)
    if (item // 10) % 2 == 0:
        queue.dequeue()
print(queue.queue)
```

Pair programming

Pair Programming Practice

ແນວທາງ, ກົນໄກ ແລະ ແຜນສຸກເສີນ

ການກະກຽມການສ້າງໂປຣແກຣມຮ່ວມກັນເປັນຄູ່ກ່ຽວຂ້ອງກັບການໃຫ້ຄຳແນະນຳແລະກົນໄກເພື່ອຊ່ວຍໃຫ້ນັກຮຽນຈັບຄູ່ຢ່າງຖືກຕ້ອງແລະ ໃຫ້ເຂົາເຈົ້າເຮັດວຽກເປັນຄູ່. ຕົວຢ່າງ, ນັກຮຽນຄວນປ່ຽນ "ເຮັດ." ການກະກຽມທີ່ມີປະສິດຕິຜົນຕ້ອງໃຫ້ມີແຜນການສຸກເສີນໃນກໍລະນີທີ່ຄູ່ຮ່ວມງານຫນຶ່ງບໍ່ຢູ່ຫຼືຕັດສິນໃຈທີ່ຈະບໍ່ເຂົ້າຮ່ວມດ້ວຍເຫດຜົນໃດຫນຶ່ງ ຫຼືດ້ວຍເຫດຜົນອື່ນ. ໃນກໍລະນີເຫຼົ່ານີ້, ມັນເປັນສິ່ງສໍາຄັນທີ່ຈະເຮັດໃຫ້ມັນຊັດເຈນວ່ານັກຮຽນທີ່ມີປະຕິບັດໜ້າທີ່ຢ່າງຫ້າວຫັນຈະບໍ່ຖືກລົງໂທດ ຍ້ອນວ່າການຈັບຄູ່ບໍ່ໄດ້ຜົນດີ.

ການຈັບຄູ່ທີ່ຄ້າຍຄືກັນ, ບໍ່ຈຳເປັນຕ້ອງເທົ່າທຽມກັນ, ຄວາມສາມາດເປັນຄູ່ຮ່ວມງານ

ການຂຽນໂປຣແກຣມຄູ່ຈະມີປະສິດທິພາບເມື່ອນັກຮຽນຕັ້ງໃຈຮ່ວມກັນເຮັດວຽກ, ຊຶ່ງວ່າບໍ່ຈຳເປັນຕ້ອງມີຄວາມຮູ້ເທົ່າທຽມກັນ, ແຕ່ຕ້ອງມີຄວາມສາມາດເຮັດວຽກເປັນຄູ່ຮ່ວມງານ. ການຈັບຄູ່ນັກຮຽນທີ່ບໍ່ສາມາດເຂົ້າກັນໄດ້ມັກຈະເຮັດໃຫ້ການມີສ່ວນຮ່ວມທີ່ບໍ່ສົມດຸນກັນ. ຄູສອນຕ້ອງເນັ້ນຫນັກວ່າການຂຽນໂປຣແກຣມຄູ່ບໍ່ແມ່ນຍຸດທະສາດ “divide-and-conque”, ແຕ່ຈະເປັນຄວາມພະຍາຍາມຮ່ວມມືເຮັດວຽກທີ່ແທ້ຈິງໃນທຸກໆດ້ານສໍາລັບໂຄງການທັງຫມົດ. ຄູຄວນຫຼີກເວັ້ນການຈັບຄູ່ນັກຮຽນທີ່ອ່ອນຫຼາຍກັບນັກຮຽນທີ່ເກັ່ງຫຼາຍ.

ກະຕຸ້ນນັກຮຽນໂດຍການໃຫ້ສິ່ງຈູງໃຈພິເສດ

ການສະເໜີແຮງຈູງໃຈພິເສດສາມາດຊ່ວຍກະຕຸ້ນນັກຮຽນໃຫ້ຈັບຄູ່, ໂດຍສະເພາະກັບນັກຮຽນທີ່ມີຄວາມສາມາດຫຼາຍ. ຈະເຫັນວ່າມັນເປັນປະໂຫຍດທີ່ຈະໃຫ້ນັກຮຽນຈັບຄູ່ເຮັດວຽກຮ່ວມກັນພຽງແຕ່ຫນຶ່ງຫຼືສອງວຽກເທົ່ານັ້ນ.

Pair Programming Practice

ປ້ອງກັນການໂກງໃນການເຮັດວຽກຮ່ວມກັນ

ສິ່ງທ້າທາຍສຳລັບຄູແມ່ນເພື່ອຊອກຫາວິທີທີ່ຈະປະເມີນຜົນໄດ້ຮັບຂອງບຸກຄົນ, ໃນຂະນະທີ່ນຳໃຊ້ຜົນປະໂຫຍດຂອງການຮ່ວມມື. ຈະຮູ້ໄດ້ແນວໃດວ່ານັກຮຽນຕັ້ງໃຈເຮັດວຽກ ຫຼື ກົ້ງແຮງງານຜູ້ຮ່ວມງານ? ຜູ້ຊ່ຽວຊານແນະນຳໃຫ້ທົບທວນຄືນການອອກແບບຫຼັກສູດ ແລະ ການປະເມີນ ພ້ອມທັງປຶກສາຫາລືຢ່າງຈະແຈ້ງ ແລະ ຊັດເຈນກ່ຽວກັບພຶດຕິກຳຂອງນັກຮຽນທີ່ຈະຖືກຕີຄວາມວ່າຂີ້ຕົວະ. ຜູ້ຊ່ຽວຊານເນັ້ນໜັກໃຫ້ຄູເຮັດການມອບໝາຍໃຫ້ມີຄວາມໝາຍຕໍ່ນັກຮຽນ ແລະ ອະທິບາຍຄຸນຄ່າຂອງສິ່ງທີ່ນັກຮຽນຈະຮຽນຮູ້ໂດຍການເຮັດສຳເລັດ.

ສະພາບແວດລ້ອມການຮຽນຮູ້ໃນການຮ່ວມມື

ສະພາບແວດລ້ອມການຮຽນຮູ້ໃນຮ່ວມກັນເກີດຂຶ້ນໄດ້ທຸກເວລາທີ່ຜູ້ສອນຮຽກຮ້ອງໃຫ້ນັກຮຽນເຮັດວຽກຮ່ວມກັນໃນກິດຈະກຳການຮຽນຮູ້. ສະພາບແວດລ້ອມການຮຽນຮູ້ຮ່ວມກັນສາມາດມີສ່ວນຮ່ວມທັງກິດຈະກຳທີ່ເປັນທາງການ ແລະ ບໍ່ເປັນທາງການ ແລະ ອາດຈະບໍ່ລວມເຖິງການປະເມີນໂດຍກົງ. ຕົວຢ່າງ, ນັກສຶກສາຄູ່ເຮັດວຽກມອບໝາຍຮ່ວມກັນໃນການຂຽນໂປຣແກຣມ; ນັກສຶກສາກຸ່ມນ້ອຍໆສົນທະນາຄຳຕອບທີ່ເປັນໄປໄດ້ຕໍ່ກັບຄຳຖາມຂອງອາຈານໃນລະຫວ່າງການບັນຍາຍ; ແລະ ນັກຮຽນເຮັດວຽກຮ່ວມກັນນອກຫ້ອງຮຽນເພື່ອຮຽນຮູ້ແນວຄວາມຄິດໃໝ່. ການຮຽນຮູ້ການຮ່ວມມືແມ່ນແຕກຕ່າງຈາກໂຄງການທີ່ນັກຮຽນ “divide and conquer.” ເມື່ອນັກຮຽນແບ່ງວຽກກັນ, ແຕ່ລະຄົນຮັບຜິດຊອບພຽງແຕ່ສ່ວນໜຶ່ງຂອງການແກ້ໄຂບັນຫາ ແລະ ຈະບໍ່ຄ່ອຍມີບັນຫາຫຍັງໃນການເຮັດວຽກຮ່ວມກັບຄົນອື່ນໃນທີມ. ໃນສະພາບແວດລ້ອມການເຮັດວຽກຮ່ວມກັນ, ນັກຮຽນມີສ່ວນຮ່ວມໃນການສົນທະນາປຶກສາຫາລືເຊິ່ງກັນແລະກັນ.

Q1. ໂດຍອ້າງອີງຕາມຄຳນິຍາມຂອງ class ຕໍ່ໄປນີ້ເພື່ອເຮັດໃຫ້ສຳເລັດ ແລະ ທົດສອບ class Deque.

```
class Deque:

    def __init__(self):
        self.queue = []

    def add_first(self, item):
        ''' Add an item to the front of the queue '''

    def remove_first(self):
        ''' Remove and return the item from the front '''

    def add_first(self, item):
        ''' Add an item to the rear of the queue '''

    def remove_first(self):
        ''' Remove and return the item from the rear '''
```

TIP

Queue ແມ່ນປະເພດຂໍ້ມູນນາມມະທຳທີ່ມີທາງເຂົ້າ ແລະ ທາງອອກຂອງຂໍ້ມູນແຍກຈາກກັນ. ສາມາດເວົ້າໄດ້ອີກວ່າ, Double Ended Queue (Deque) ແມ່ນໂຄງສ້າງຂໍ້ມູນທີ່ບ່ອນເອົາຂໍ້ມູນເຂົ້າ ແລະ ບ່ອນເອົາຂໍ້ມູນອອກແມ່ນເຮັດໄດ້ຢູ່ທັງສອງດ້ານ.