오픈채팅방 (난이도: 하)

[핵심 포인트]

- 닉네임을 출력하는 경우는 오직 두 가지입니다.

  1) Enter 2) Leave
- 사용자의 ID에 따른 닉네임에 영향을 미치는 명령은 오직 두 가지입니다.

  1) Enter 2) Change

Key	Nickname
uid1234	Muzi
uid4567	Ryan

[프로그램 작성]

- 리스트를 돌며 Enter와 Change를 만나면 닉네임 정보를 저장합니다.
- 다시 리스트를 돌며 Enter와 Leave를 만나면 특정 사용자 ID의 닉네임을 출력합니다.

나동빈 (dongbinna@postech.ac.kr)

오픈채팅방 (난이도: 하)

```
def solution(record):
  answer = []
  user_list = {}
  for i in record:
    if i.split(' ')[0] == 'Enter' or i.split(' ')[0] == 'Change':
      user_list[i.split(' ')[1]] = i.split(' ')[2]
  for i in record:
    if i.split(' ')[0] == 'Enter':
      answer.append(user_list[i.split(' ')[1]] + '님이 들어왔습니다.')
    elif i.split(' ')[0] == 'Leave':
      answer.append(user_list[i.split(' ')[1]] + '님이 나갔습니다.')
  return answer
```

나동빈 (dongbinna@postech.ac.kr)

실패율 (난이도: 하)

[핵심 포인트]

- 말 그대로 실패율의 정의에 따라서 구현만 해주면 됩니다.

  실패율: 스테이지에 도달했으나 아직 클리어하지 못한 플레이어의 수 / 스테이지에 도달한 플레이어 수

[프로그램 작성]

- 스테이지 번호(i)를 1부터 N까지 증가시키며 해당 단계에 머물러 있는 플레이어들의 수(count)를 계산합니다.
  - 스테이지 번호에 따른 실패율을 저장합니다.
- 저장된 내용을 확인하며 실패율이 높은 순서대로 스테이지 번호를 출력합니다.

나동빈 (dongbinna@postech.ac.kr)

실패율 (난이도: 하)

```python
def solution(N, stages):
  answer = []
  length = len(stages)
  for i in range(1, N + 1):
    count = stages.count(i)
    if length == 0:
      fail = 0
    else:
      fail = count / length
    answer.append((i, fail))
    length -= count
  answer = sorted(answer, key=lambda t: t[1], reverse=True)
  answer = [i[0] for i in answer]
  return answer
```

나동빈 (dongbinna@postech.ac.kr)

후보키 (난이도: 중하)

[핵심 포인트]

- '부분집합 계산 함수'를 이용해 가능한 모든 열(Column)의 쌍을 계산합니다.
- 모든 열의 쌍에 대하여 유일성을 만족하는 경우만을 계산합니다.
- 유일성을 만족하는 경우 중에서 최소성을 만족하는 경우만을 남깁니다.

[프로그램 작성]

- 세 가지 파트로 나누어서 프로그램을 작성합니다.

1) 모든 부분집합을 구하기
2) 부분집합 중에서 유일성을 만족하는 부분집합을 구하기
3) 유일성을 만족하는 부분집합 중에서 최소성을 만족하는 부분집합만을 남기기

학번	이름	전공	학년
100	ryan	music	2
200	apeach	math	2
300	tube	computer	3
400	con	computer	1
500	muzi	music	3
600	apeach	music	2

나동빈 (dongbinna@postech.ac.kr)

후보키 (난이도: 중하)

```python
from itertools import chain, combinations

# 모든 부분집합(열의 쌍)을 구하는 함수
def get_all_subset(iterable):
  s = list(iterable)
  return chain.from_iterable(combinations(s, r) for r in range(len(s) + 1))

# 부분집합 중에서 유일성을 만족하는 부분집합(열의 쌍)을 구하는 함수
def get_all_unique_subset(relation):
  subset_list = get_all_subset(list(range(0, len(relation[0]))))
  unique_list = []
  for subset in subset_list:
    unique = True
    row_set = set()
    for i in range(len(relation)):
      row = ''
      for j in subset:
        row += relation[i][j] + '.'
      if row in row_set:
        unique = False
        break
      row_set.add(row)
    if unique:
      unique_list.append(subset)
  return unique_list
```

나동빈 (dongbinna@postech.ac.kr)

후보키 (난이도: 중하)

```
def solution(relation):
  unique_list = get_all_unique_subset(relation)
  unique_list = sorted(unique_list, key=lambda x: len(x))
  # 부분집합 중에서 최소성을 만족하는 부분집합(열의 쌍)을 구하기
  answer_list = []
  for subset in unique_list:
    subset = set(subset)
    check = True
    for j in answer_list:
      if j.issubset(subset):
        check = False
    if check == True:
      answer_list.append(subset)
  return len(answer_list)
```

나동빈 (dongbinna@postech.ac.kr)

무지의 먹방 라이브 (난이도: 중)

[핵심 포인트]

- 시간이 적게 걸리는 음식부터 확인하는 그리디(Greedy)한 접근 방식으로 문제를 풀 수 있습니다.
- 음식의 수를 최대한 줄인 뒤에는 K 시간 이후에 어떤 음식을 먹으면 되는지 확인하면 됩니다.

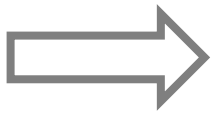
[프로그램 작성]

- 굉장히 다양한 접근 방식이 있지만, 가장 이해가 쉬운 방식인 우선순위 큐를 이용하는 방식을 소개합니다.
- 그림으로 이해해 봅시다!

나동빈 (dongbinna@postech.ac.kr)

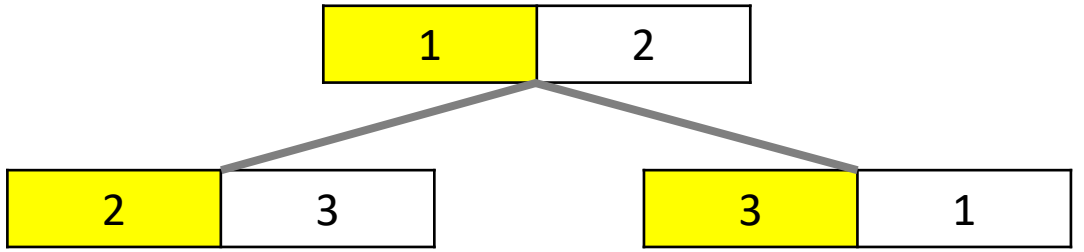
무지의 먹방 라이브 (난이도: 중)

Time	#
3	1
1	2
2	3



남은 시간: 5

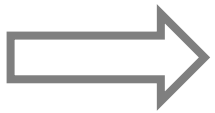
남은 음식: 3개



나동빈 (dongbinna@postech.ac.kr)

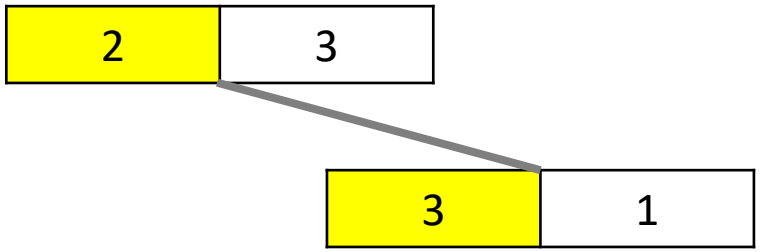
무지의 먹방 라이브 (난이도: 중)

Time	#
3	1
1	2
2	3



남은 시간: 2

남은 음식: 2개



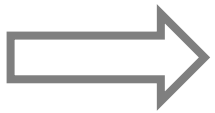
먹은 음식들:

1	2

나동빈 (dongbinna@postech.ac.kr)

무지의 먹방 라이브 (난이도: 중)

Time	#
3	1
1	2
2	3



남은 시간: 0

남은 음식: 1개

3	1

먹은 음식들:

1	2

2	3

나동빈 (dongbinna@postech.ac.kr)

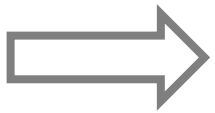
무지의 먹방 라이브 (난이도: 중)

조금 더 자세한 예시를 봅시다!

나동빈 (dongbinna@postech.ac.kr)

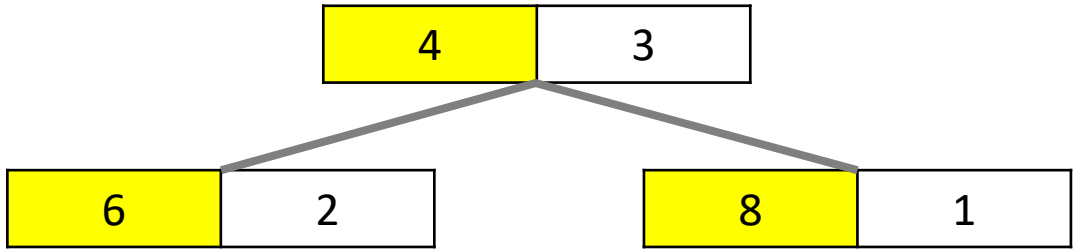
무지의 먹방 라이브 (난이도: 중)

Time	#
8	1
6	2
4	3



남은 시간: 15

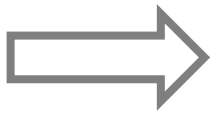
남은 음식: 3개



나동빈 (dongbinna@postech.ac.kr)

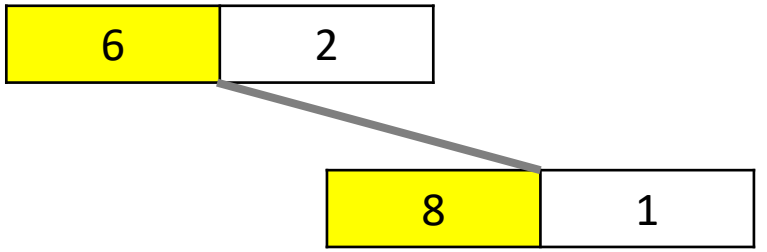
무지의 먹방 라이브 (난이도: 중)

Time	#
8	1
6	2
4	3



남은 시간: 3

남은 음식: 2개



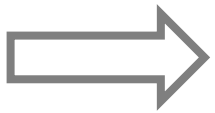
먹은 음식들:

4	3

나동빈 (dongbinna@postech.ac.kr)

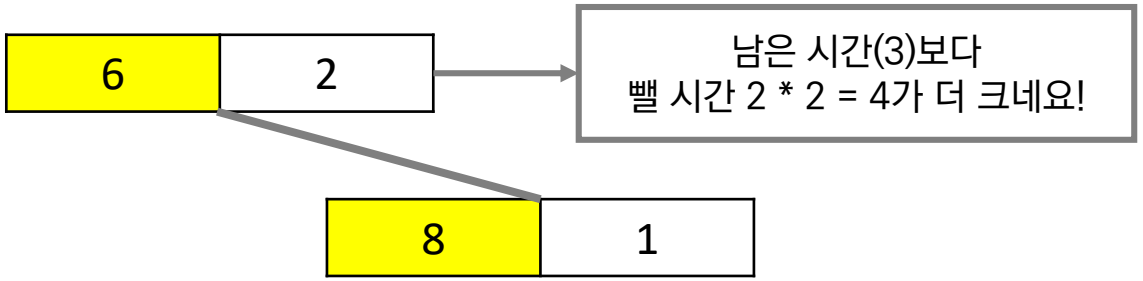
무지의 먹방 라이브 (난이도: 중)

Time	#
8	1
6	2
4	3



남은 시간: 3

남은 음식: 2개



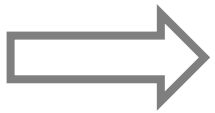
먹은 음식들:

4	3

나동빈 (dongbinna@postech.ac.kr)

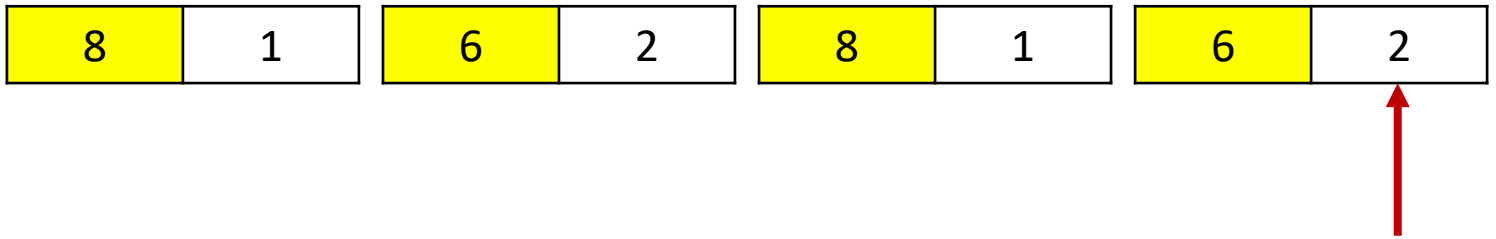
무지의 먹방 라이브 (난이도: 중)

Time	#
8	1
6	2
4	3



남은 시간: 3

남은 음식: 2개



먹은 음식들:

4	3

나동빈 (dongbinna@postech.ac.kr)

무지의 먹방 라이브 (난이도: 중)

```
from queue import PriorityQueue

def solution(food_times, k):
  if sum(food_times) <= k:
    return -1
  answer = 0
  q = PriorityQueue()
  for i in range(len(food_times)):
    q.put((food_times[i], i + 1))
  sum_value = 0
  previous = 0
  length = len(food_times)
  while sum_value + ((q.queue[0][0] - previous) * length) <= k:
    now = q.get()[0]
    sum_value += (now - previous) * length
    length -= 1
    previous = now
  # 남은 음식 중에서 몇 번째 음식인지 확인
  target = k - sum_value + 1
  length = len(q.queue)
  temp = (target - 1) // length
  result = sorted(q.queue, key=lambda x: x[1])
  target -= temp * length
  return result[target - 1][1]
```

나동빈 (dongbinna@postech.ac.kr)

길 찾기 게임 (난이도: 중)

[핵심 포인트]

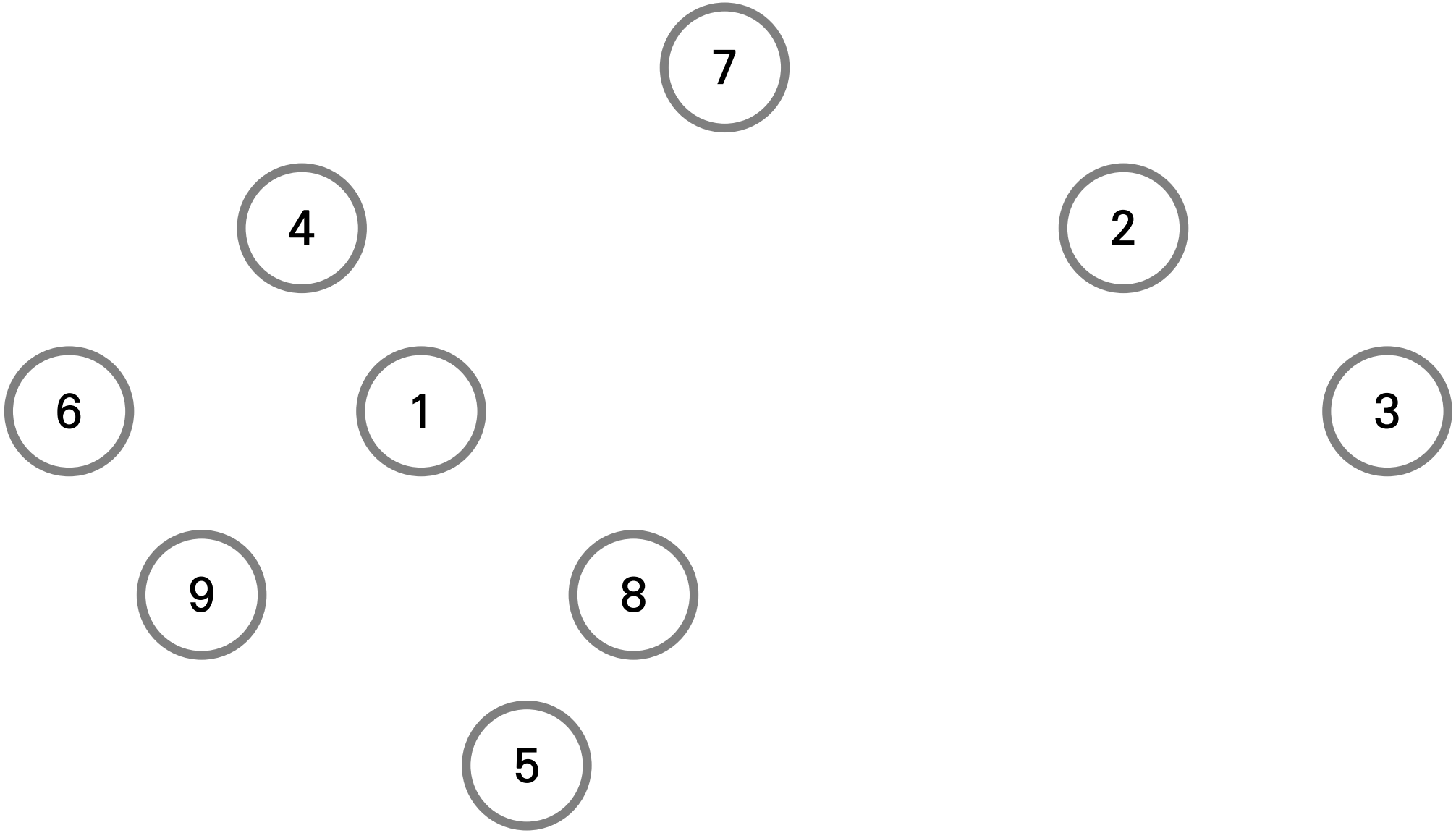
- 알고리즘 자체는 어렵지 않으므로 이진 트리를 실수 없이 구현할 수 있으면 풀 수 있습니다.
- 각 노드마다 자식으로 가질 수 있는 X 좌표 범위를 정해두면 훨씬 간결해집니다.

[프로그램 작성]

- 트리의 노드들을 Y 좌표를 기준으로 내림차순 정렬합니다. (Level에 따른 정렬)
- Level을 하나씩 내려가며 매 Level 안에 포함되어 있는 각 노드를 부모 노드(이전 Level)와 연결합니다.

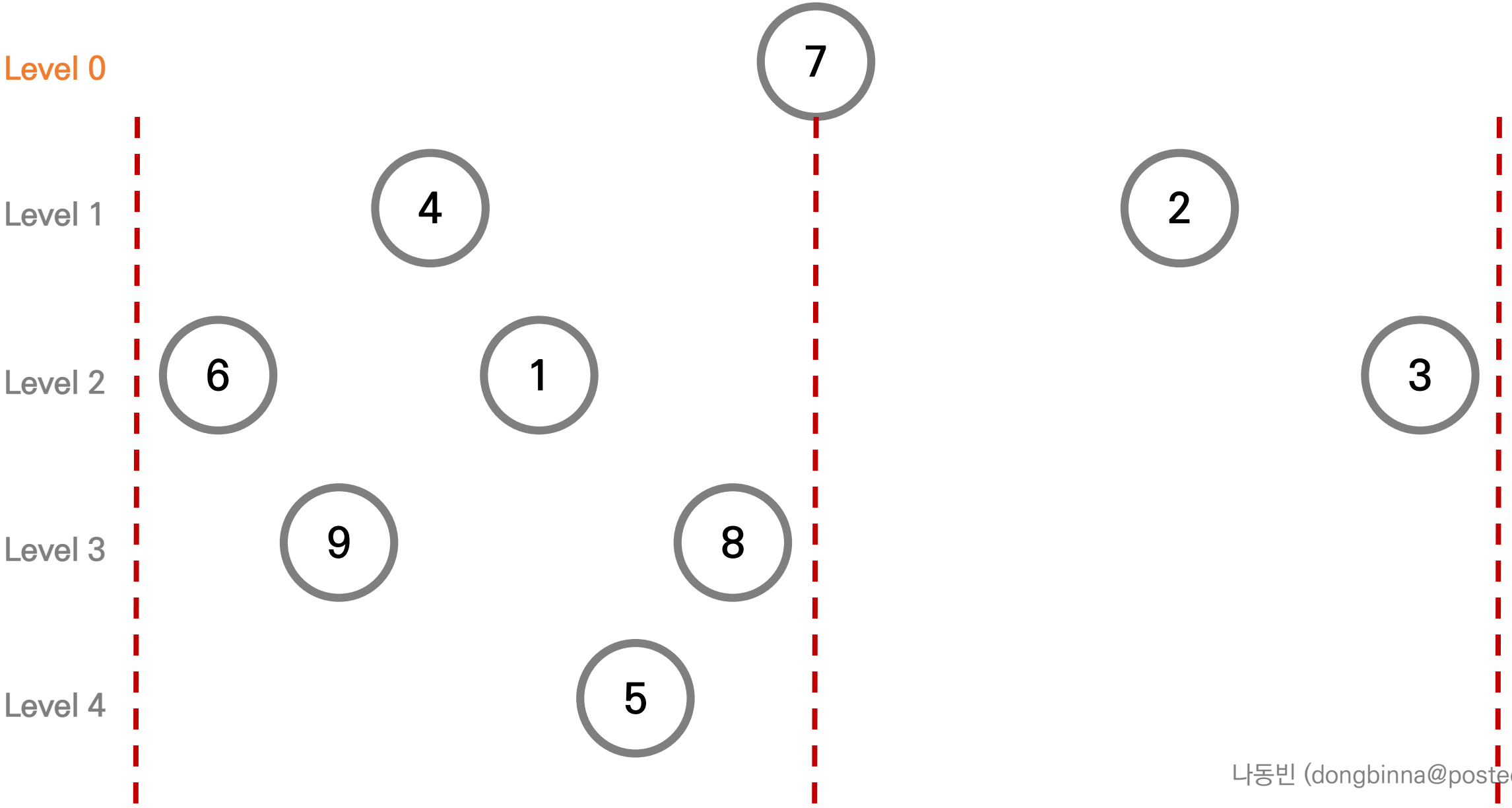
나동빈 (dongbinna@postech.ac.kr)

길 찾기 게임 (난이도: 중)



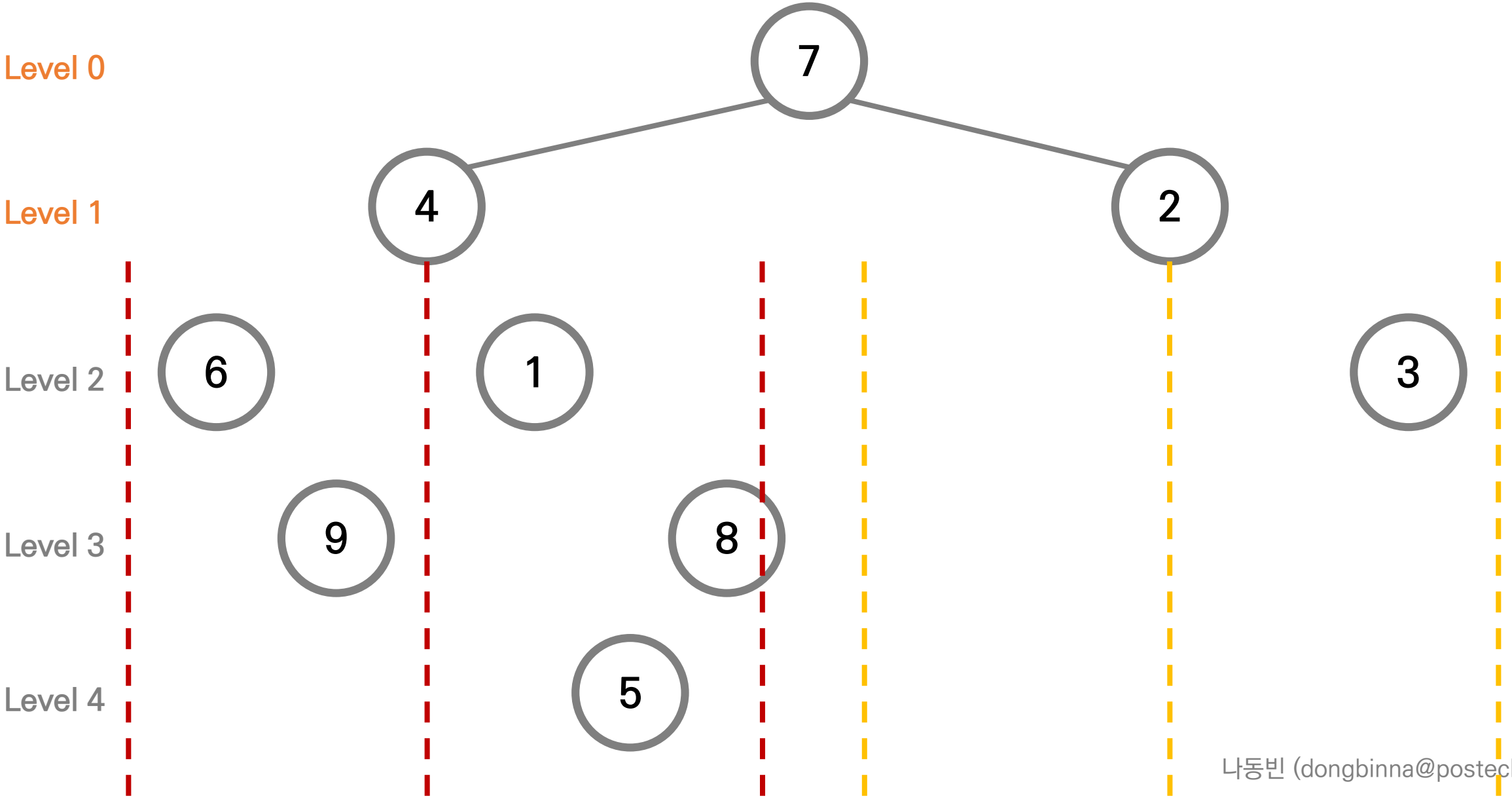
나동빈 (dongbinna@postech.ac.kr)

길 찾기 게임 (난이도: 중)



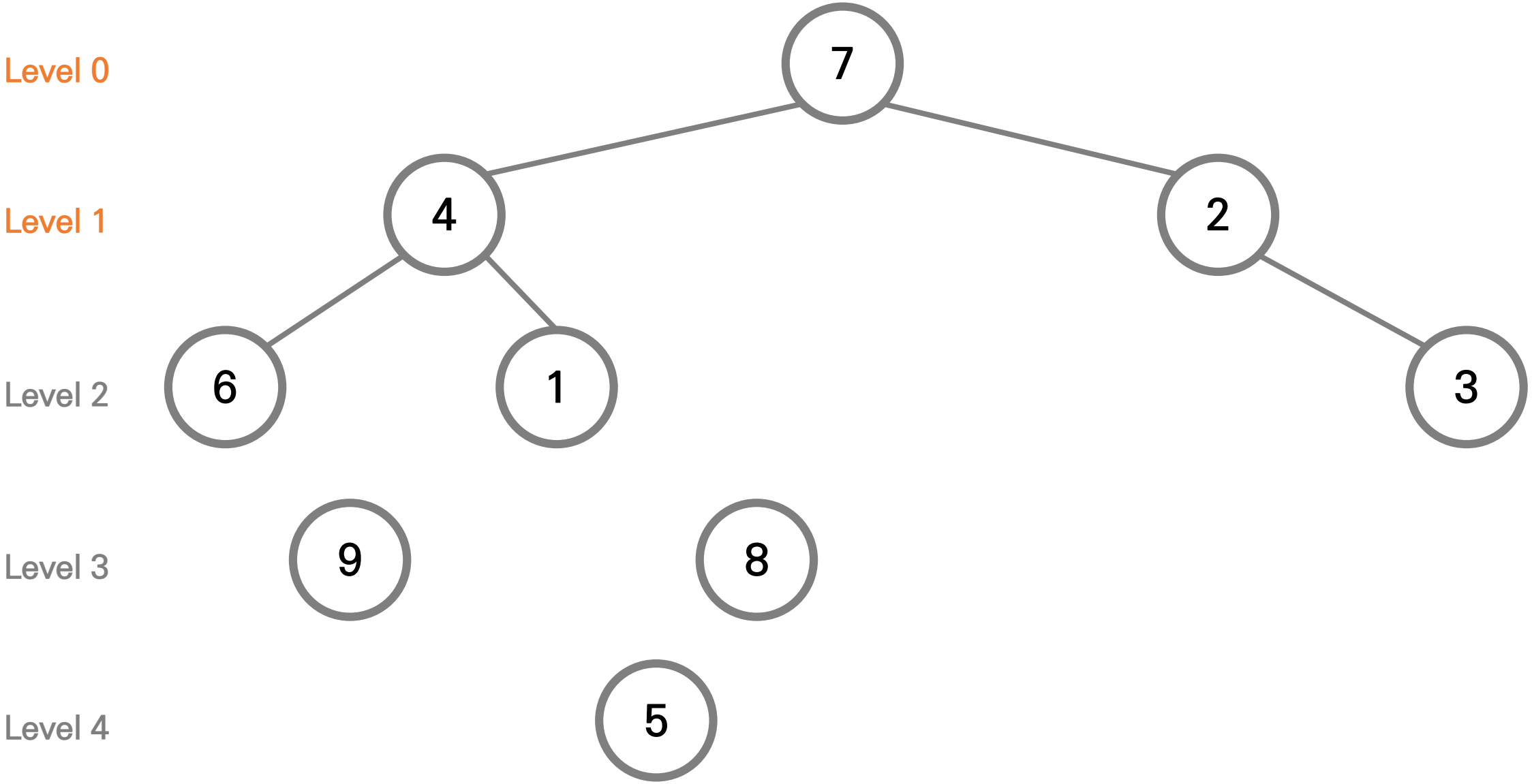
나동빈 (dongbinna@postech.ac.kr)

길 찾기 게임 (난이도: 중)



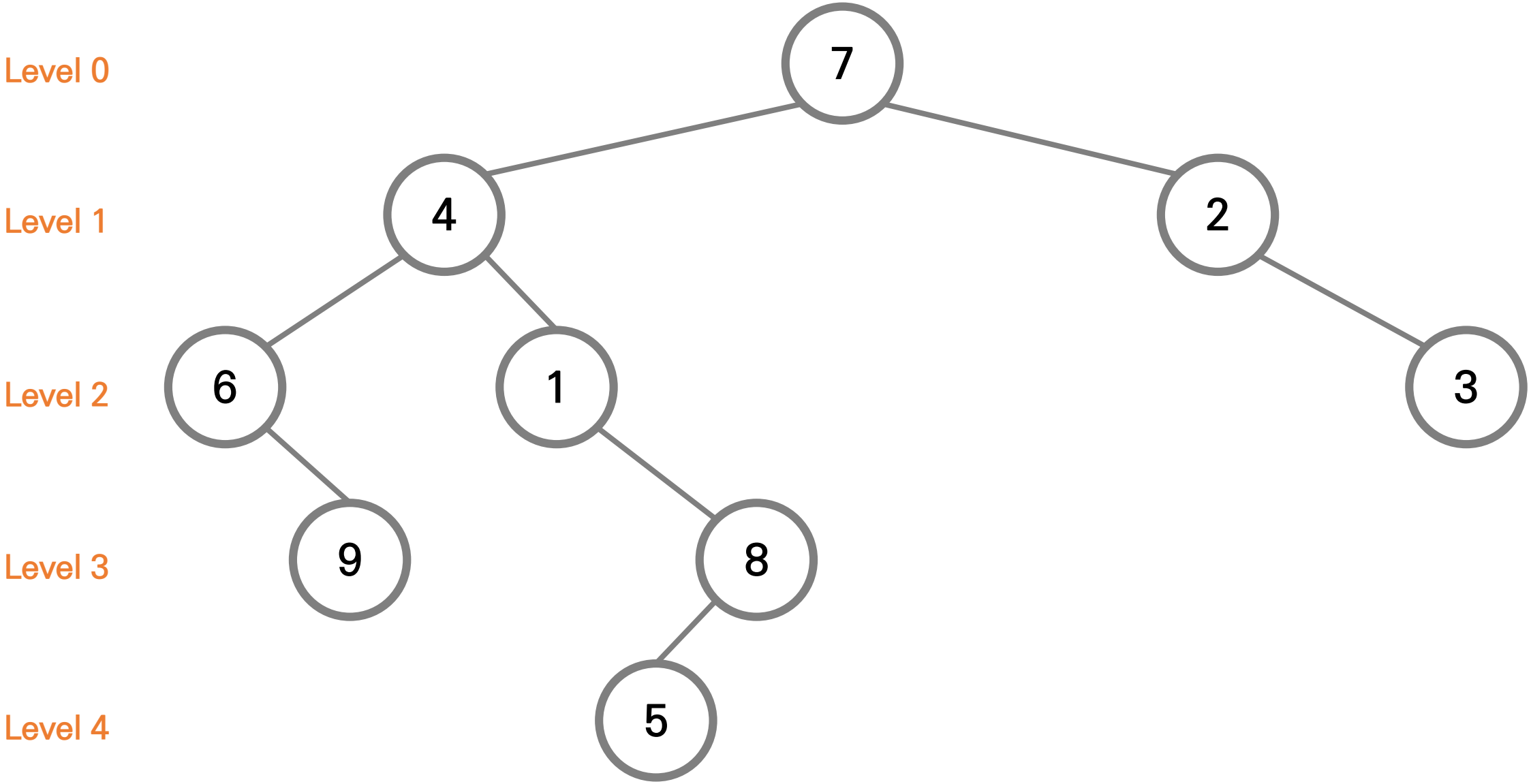
나동빈 (dongbinna@postech.ac.kr)

길 찾기 게임 (난이도: 중)



나동빈 (dongbinna@postech.ac.kr)

길 찾기 게임 (난이도: 중)



나동빈 (dongbinna@postech.ac.kr)

길 찾기 게임 (난이도: 중)

```
import sys
sys.setrecursionlimit(10 ** 6)

class Node:
  def __init__(self, x, id, left_bound, right_bound):
    self.x = x
    self.id = id
    self.left_bound = left_bound
    self.right_bound = right_bound
    self.left_node = None
    self.right_node = None

preorder_result = []
postorder_result = []

def preorder(node):
  preorder_result.append(node.id)
  if node.left_node is not None:
    preorder(node.left_node)
  if node.right_node is not None:
    preorder(node.right_node)

def postorder(node):
  if node.left_node is not None:
    postorder(node.left_node)
  if node.right_node is not None:
    postorder(node.right_node)
  postorder_result.append(node.id)
```

나동빈 (dongbinna@postech.ac.kr)

길 찾기 게임 (난이도: 중)

```
def solution(nodeinfo):
  nodeinfo = [i + [idx + 1] for idx, i in enumerate(nodeinfo)] # 세 번째 원소로 인덱스 넣기
  nodeinfo = sorted(nodeinfo, key=lambda x: x[1], reverse=True) # Y 좌표를 기준으로 내림차순 정렬
  array = [] # 같은 Y 좌표에 여러 개의 X 좌표가 들어가도록 리스트 구성
  now = -1
  for i in nodeinfo:
    y = i[1]
    if y != now:
      array.append([])
      now = y
    array[len(array) - 1].append((i[0], i[2])) # X 값과 인덱스만 삽입
  for i in range(len(array)):
    array[i] = sorted(array[i])
  root = Node(array[0][0][0], array[0][0][1], 0, 100000) # X 루트 노드 설정
  node_list = [[]]
  node_list[0].append(root)
  for level in range(1, len(array)): # 두 번째 Level 차례대로 확인
    node_list.append([])
    for data in array[level]:
      x = data[0]
      id = data[1]
      for parent_node in node_list[level - 1]:
        if parent_node.left_bound <= x and parent_node.x > x:
          now_node = Node(x, id, parent_node.left_bound, parent_node.x)
          parent_node.left_node = now_node
          node_list[level].append(now_node)
          break
        elif parent_node.right_bound >= x and parent_node.x < x:
          now_node = Node(x, id, parent_node.x, parent_node.right_bound)
          parent_node.right_node = now_node
          node_list[level].append(now_node)
          break
  preorder(root)
  postorder(root)
  return [preorder_result, postorder_result]
```

나동빈 (dongbinna@postech.ac.kr)

매칭 점수 (난이도: 중)

[핵심 포인트]

- 철저한 구현 문제입니다.
- 파이썬이 상대적으로 유리한 문제입니다.

[프로그램 작성]

- HTML 문서에서 정보를 크롤링하여 모든 페이지에 대한 기본 점수 및 외부 링크 개수를 구합니다.
- 링크 점수를 계산한 뒤에, 이를 기본 점수에 더해 최종 점수를 계산합니다.
- 최종 점수가 제일 높은 페이지의 인덱스를 출력합니다.

나동빈 (dongbinna@postech.ac.kr)

매칭 점수 (난이도: 중)

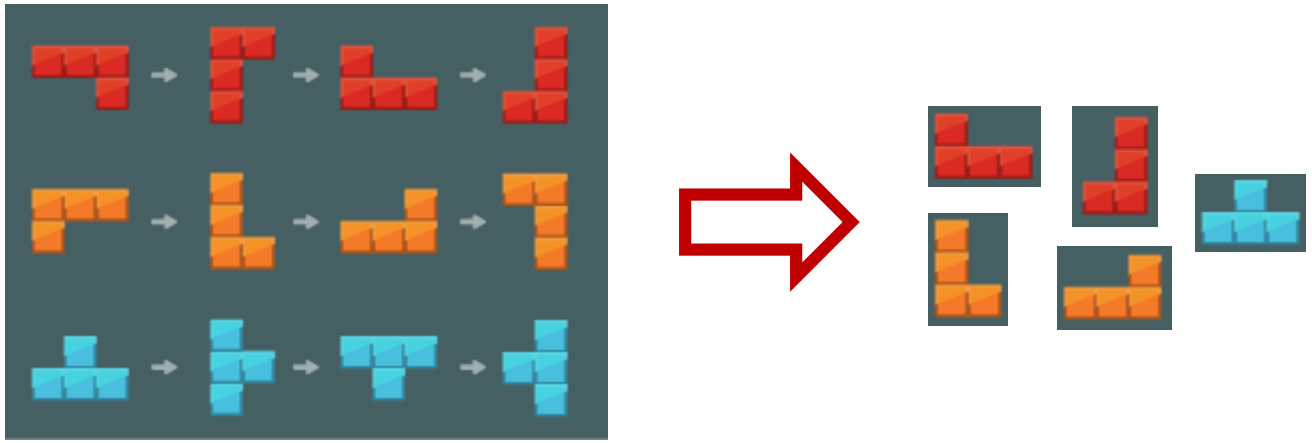
```python
import re

def solution(word, pages):
  answer = 0
  basic_score = {}
  all_link_list = []
  link_count = {}
  original_page = []
  for page in pages:
  # 현재 페이지 URL 저장
    url = re.findall('<meta property="og:url" content="https://(.*)"/>', page)[0]
    original_page.append(url)
    # 외부 링크 저장: (From, To)
    links = page.split('a href="https://')[1:]
    for link in links:
      all_link_list.append((url, link.split('"')[0]))
    # 기본 점수 및 외부 링크 개수 저장
    link_count[url] = len(links)
    basic_score[url] = re.sub('[^a-z]+', '.', page.lower()).split('.').count(word.lower())
  # 링크 점수 계산
  link_score = {}
  for url, link in all_link_list:
    if link in link_score:
      link_score[link] += basic_score[url] / link_count[url]
    else:
      link_score[link] = basic_score[url] / link_count[url]
  # 최종 점수 계산
  score = {}
  for url in basic_score:
    score[url] = basic_score[url]
  for url in link_score:
    if url in score:
      score[url] += link_score[url]
  # 최종 점수가 제일 높은 페이지 인덱스 반환
  max_value = -1
  index = -1
  count = 0
  for url in original_page:
    if score[url] > max_value:
      max_value = score[url]
      index = count
    count += 1
  return index
```

나동빈 (dongbinna@postech.ac.kr)

블록 게임 (난이도: 중)

[핵심 포인트]

- 블록 중에서 지워질 수 있는 블록은 5개 뿐이라는 점을 이해합니다.
- 지워질 수 있는 블록에 대하여 위에서부터 검은 블록을 떨어뜨리는 과정을 반복합니다.



나동빈 (dongbinna@postech.ac.kr)

블록 게임 (난이도: 중)

```
checked = set()

def get_possible_block(board, i, j):
  try:
    if (board[i + 1][j] == board[i][j] and
      board[i + 1][j + 1] == board[i][j] and
      board[i + 1][j + 2] == board[i][j]):
      return ((i, j + 1), (i, j + 2), board[i][j])
  except:
    pass
  try:
    if (board[i + 1][j] == board[i][j] and
      board[i + 2][j] == board[i][j] and
      board[i + 2][j - 1] == board[i][j]):
      return ((i, j - 1), (i + 1, j - 1), board[i][j])
    except:
      pass
  try:
    if (board[i + 1][j] == board[i][j] and
      board[i + 2][j] == board[i][j] and
      board[i + 2][j + 1] == board[i][j]):
      return ((i, j + 1), (i + 1, j + 1), board[i][j])
    except:
      pass
  try:
    if (board[i + 1][j] == board[i][j] and
      board[i + 1][j - 1] == board[i][j] and
      board[i + 1][j - 2] == board[i][j]):
      return ((i, j - 1), (i, j - 2), board[i][j])
  except:
    pass
  try:
    if (board[i + 1][j - 1] == board[i][j] and
      board[i + 1][j] == board[i][j] and
      board[i + 1][j + 1] == board[i][j]):
      return ((i, j - 1), (i, j + 1), board[i][j])
  except:
    pass
  return None
```

나동빈 (dongbinna@postech.ac.kr)

블록 게임 (난이도: 중)

```
def preprocess(board):
  list = []
  for i in range(len(board)):
    for j in range(len(board[0])):
      if board[i][j] != 0 and board[i][j] not in checked:
        checked.add(board[i][j])
        type = get_possible_block(board, i, j)
        if type != None:
          list.append(type)
  return list

def over_head(board, i, j):
  for a in range(i + 1):
    if board[a][j] != 0:
      return True
  return False

def remove(board, k):
  for i in range(len(board)):
    for j in range(len(board[0])):
      if board[i][j] == k:
        board[i][j] = 0

removed = set()
```

```
def solution(board):
  answer = 0
  list = preprocess(board) # 지워질 가능성이 있는 블럭만 남기기
  while True:
    check = False
    for i in list:
      a1, b1 = i[0]
      a2, b2 = i[1]
      k = i[2]
      if k in removed: # 이미 지워진 블럭이면 무시
        continue
      # 남은 두 공간의 위에 아무 블럭도 없다면 해당 블럭 지움 처리
      if not over_head(board, a1, b1) and not over_head(board, a2, b2):
        removed.add(k)
        remove(board, k)
        check = True
        answer += 1
    if check is False:
      break
  return answer
```

나동빈 (dongbinna@postech.ac.kr)