<[파이썬 알고리즘 탐험] 제5장. 숫자들의 만남의 광장, 최소공배수 찾기!>

선생님: (두 개의 버스 노선도를 보여주며) 얘들아, 안녕! 여기 A번 버스는 20분마다 출발하고, B번 버스는 75분마다 출발한다고 해보자. 두 버스가 지금 동시에 땡! 하고 출발했다면, 다음번에 또 **동시에 출발하는 가장 빠른 시간**은 언제일까?

코딩새싹 🌱: 으음… A번 버스는 20분, 40분, 60분… 이렇게 가고, B번 버스는 75분, 150분… 이렇게 가니까… 언젠가는 만나겠죠?

선생님: 맞아! 이렇게 두 개 이상의 숫자들이 공통으로 가지는 배수 중에서 가장 작은 것을 바로 **최소공배수(Least Common Multiple, LCM)**라고 한단다! 방금 그 버스 문제의 답이 바로 20과 75의 최소공배수가 되는 거지.

<1. 방법 1: 배수 목록에서 처음 만나는 친구 찾기!>

선생님: 최소공배수를 찾는 가장 간단한 방법은, 각 숫자의 배수들을 쭉~ 적어놓고, 두 목록에 공통으로 등장하는 첫 번째 숫자를 찾는 거야!

```
def lcm(a, b):
  # a의 배수들을 a * b 까지 만들어보자 (a*b는 반드시 a와 b의 공배수니까!)
  a_multiples = range(a, a * b + 1, a)
  # b의 배수들도 똑같이!
  b_multiples = range(b, a * b + 1, b)

  # print(f"{a}의 배수 (일부): {list(a_multiples)[:15]}") # 너무 많으니 앞 15개만!
  # print(f"{b}의 배수 (일부): {list(b_multiples)[:15]}")

  # a의 배수 목록을 하나씩 보면서,
  for i in a_multiples:
    if i in b_multiples: # 만약 그 숫자가 b의 배수 목록에도 있다면,
      return i # 그게 바로 우리가 찾는 가장 작은 공통 배수(최소공배수)!

  # (실제로는 a*b 안에 반드시 공배수가 있으므로 이 부분은 실행될 일은 거의 없어)
  return a * b # 만약을 위한 대비
```

```
result_lcm = lcm(20, 75)
print(f"20과 75의 최소공배수 (방법1): {result_lcm}") # 300이 나오겠지?
```

궁금이 💡: `range(a, a*b + 1, a)` 이렇게 하면 a부터 시작해서 a씩 커지는 숫자들, 즉 a의 배수들이 만들어지네요! 그리고 a*b는 항상 a와 b의 공배수가 되니까, 그 안에서 찾으면 되겠군요!

선생님: 정확해! 이 방법은 이해하기는 쉽지만, 숫자가 아주 커지면 배수 목록도 엄청나게 길어져서 조금 느릴 수도 있단다.

<2. 방법 2: 소인수분해로 최소공배수 조립하기!>

선생님: 더 똑똑한 방법은 우리가 지난 시간에 배운 '소인수분해' 마법을 이용하는 거야! 각 숫자를 소인수라는 '기본 벽돌'로 분해한 다음, 그 벽돌들을 잘 조합해서 최소공배수라는 '공동의 탑'을 쌓는 거지!

코딩새싹 🌱: 소인수분해요? 12 = 2*2*3 이렇게 나누는 거요? 그걸로 어떻게 최소공배수를 만들어요?

선생님: 아주 좋은 질문이야! 두 숫자 A와 B의 최소공배수를 만들려면,

1. A와 B를 각각 소인수분해한다.
2. 두 분해 결과에 등장하는 **모든 종류의 소인수 벽돌**을 모은다.
3. 각 종류의 소인수 벽돌에 대해, A와 B 중에서 그 벽돌을 **더 많이 가지고 있는 쪽의 개수만큼** 챙긴다!
4. 그렇게 챙긴 모든 벽돌들을 곱하면 바로 최소공배수가 된단다!

자, 이걸 코드로 옮겨볼까? (우리가 예전에 만든 factorization 함수와 union 함수가 필요해!)

```
# (지난 시간에 만든 factorization 함수와 union 함수가 있다고 가정!)
def factorization(n): # 숫자를 소인수분해해서 리스트로 돌려주는 함수
  result = []; a = 2
  while a <= n:
    if n % a == 0: result.append(a); n = n / a
    else: a += 1
  return result

def union(list1, list2): # 두 리스트의 모든 종류의 원소를 모아주는 함수 (중복 없이)
  c = list1 + list2; d = []
  for i in c:
```

```
    if i not in d: d.append(i)
  d.sort(); return d

# 소인수분해를 이용한 최소공배수 함수
def lcm1(num1, num2):
  factors1 = factorization(num1) # 첫 번째 숫자를 소인수분해
  factors2 = factorization(num2) # 두 번째 숫자를 소인수분해

  print(f"\n{num1}의 소인수분해: {factors1}")
  print(f"{num2}의 소인수분해: {factors2}")

  # 1. 두 분해 결과에 등장하는 모든 '종류'의 소인수 벽돌 모으기
  unique_factors = union(factors1, factors2)
  print(f"등장한 모든 종류의 소인수: {unique_factors}")

  lcm_factors = [] # 최소공배수를 만들 소인수 벽돌들을 담을 주머니
  # 2. 각 종류의 소인수 벽돌에 대해, 더 많이 가진 쪽의 개수만큼 챙기기
  for factor in unique_factors:
    count1 = factors1.count(factor) # num1이 가진 해당 소인수의 개수
    count2 = factors2.count(factor) # num2가 가진 해당 소인수의 개수

    # 더 많은 개수만큼 lcm_factors 주머니에 그 소인수를 넣어준다!
    # [factor] * 개수  하면 [factor, factor, ...] 이런 리스트가 만들어져.
    if count1 >= count2:
      lcm_factors += [factor] * count1
    else:
      lcm_factors += [factor] * count2

  print(f"최소공배수를 만들 소인수 벽돌들: {lcm_factors}")
```

```
  # 3. 모은 모든 소인수 벽돌들을 곱해서 최소공배수 완성!
  final_lcm = 1
  for f in lcm_factors:
    final_lcm *= f
  return final_lcm

result_lcm1 = lcm1(12, 18) # 12 = 2*2*3, 18 = 2*3*3
# unique_factors = [2, 3]
# 2의 경우: 12는 2개, 18은 1개 -> 2개 챙김 ([2,2])
# 3의 경우: 12는 1개, 18은 2개 -> 2개 챙김 ([3,3])
# lcm_factors = [2,2,3,3]
# final_lcm = 2*2*3*3 = 36
print(f"12와 18의 최소공배수 (방법2): {result_lcm1}")

result_lcm_large = lcm1(20, 75) # 20 = [2,2,5], 75 = [3,5,5]
# unique_factors = [2,3,5]
# 2: 20이 2개 -> [2,2]
# 3: 75가 1개 -> [3]
# 5: 75가 2개 -> [5,5]
# lcm_factors = [2,2,3,5,5]
# final_lcm = 2*2*3*5*5 = 300
print(f"20과 75의 최소공배수 (방법2): {result_lcm_large}")
```

궁금이 💡: 와! `lcm_factors += [factor] * count1` 이 부분, 정말 신기해요! `[2] * 3` 하면 `[2, 2, 2]` 이렇게 리스트가 만들어지는 걸 이용해서 필요한 만큼 소인수 벽돌을 한 번에 넣는 거군요!

코딩새싹 🌱: 그리고 마지막에 `final_lcm = 1` 로 시작해서 `lcm_factors`에 있는 모든 소인수들을 곱해주니까 정말 최소공배수가 딱 나오네요!

<선생님의 프로그래머 꿀팁! 🍯 - 최대공약수와 최소공배수의 비밀 관계>

선생님: (윙크하며) 사실 말이야, 최소공배수를 구하는 아주아주 비밀스럽고 빠른 방법이 하나 더 있단다! 그건 바로 우리가 예전에 배운 '최대공약수(GCD)'를 이용하는 거야!

두 숫자 a와 b의 최소공배수(LCM)는 (a * b) / 최대공약수(GCD) 와 똑같다!

만약 우리가 지난 시간에 배운 유클리드 호제법으로 최대공약수를 빠르게 구할 수 있다면, 최소공배수도 눈 깜짝할 사이에 구할 수 있겠지?

```
# (지난 시간에 배운 gcd 함수가 있다고 가정!)
def gcd(a, b):
  while b != 0: a, b = b, a % b
  return a

def lcm_with_gcd(a, b):
  return (a * b) // gcd(a, b) # 정수 결과를 얻기 위해 // 사용!

print(f"\n--- GCD를 이용한 초고속 LCM 계산! ---")
print(f"20과 75의 최소공배수 (GCD 활용): {lcm_with_gcd(20, 75)}")
```

코딩새싹 🌱: 헉! 정말 간단하게 나오네요! 수학은 정말 신기한 것 같아요!

선생님: (흐뭇하게) 그렇지? 오늘 우리는 최소공배수라는 '만남의 광장'을 찾는 두 가지 방법 - 배수 목록 비교와 소인수분해 활용 - 을 탐구했고, 마지막에는 최대공약수와의 비밀스러운 관계까지 알게 되었어. 이렇게 하나의 문제를 여러 가지 방법으로 해결해보고, 그 원리를 깊이 이해하는 것이 바로 진정한 알고리즘 탐험가란다! 오늘 정말 수고 많았어, 우리 똑똑이 탐험가들! 💯