Tree 형태 Solution Space 탐색

트리 탐색 과정과 유의사항에 익숙해 지기

<왜 **Tree**에 대해 배우는가?>

- Tree는 **여러 단계를 거쳐 하나의 해를 선택**하는 상황에서 **선택할 수 있는 모든 가능성**을 나타내는데 자연스러운 구조임
- 여러 가능한 해 중 **최적의 해를 선택**하기 위해서는 **Tree를 탐색**해야 함
- 이번 주에는 **효율적으로 Tree를 탐색하기 위해 고려할 것들**을 배워보겠음

01. 문제에서 사용하는 정의 이해: minimal product-sum number

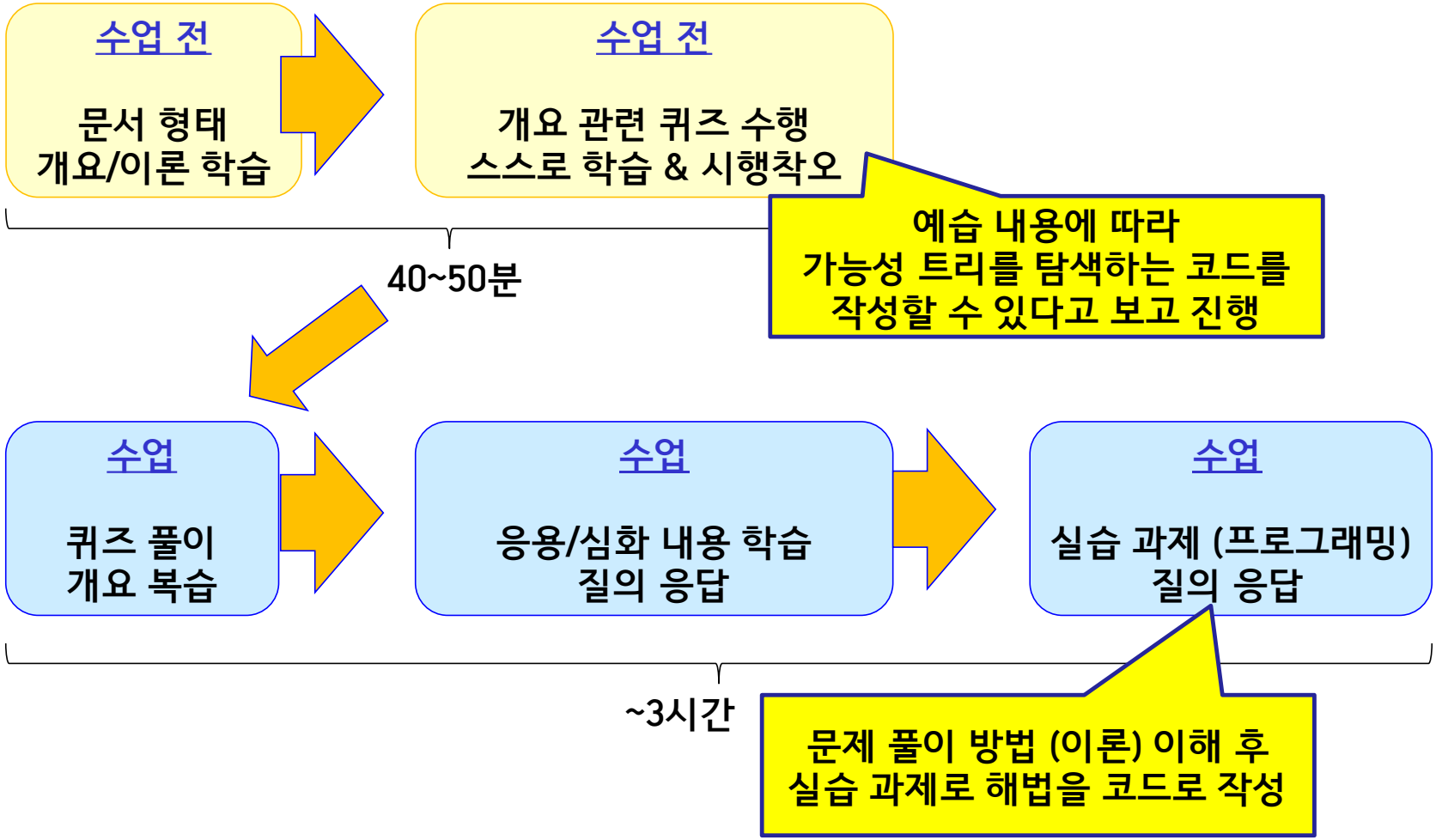
02. 첫 번째 알고리즘

03. 두 번째 알고리즘

04. 정리 문제 풀이

05. 실습 문제 풀이 & 질의 응답

수업 전 예습 → 문제풀이/실습/질의응답 (플립 러닝, 거꾸로 학습)



문제 정의: product-sum numbers

아래와 같이 **2개 이상의 자연수** $a_1, a_2, \ldots, a_k$의 **곱(product)**과 **합(sum)**으로 동시에 나타낼 수 있는 **자연수** P를 **product-sum** number라 하자.

$P = a_1 \times a_2 \times \ldots \times a_k = a_1 + a_2 + \ldots + a_k$

예를 들어, 6은 product-sum number이다. $6 = 1 \times 2 \times 3 = 1 + 2 + 3$ 로 나타낼 수 있기 때문이다.

자연수 입력 **N**이 주어졌을 때, $2 \leq k \leq N$ 범위에 속한 모든 k 값 각각에 대해

(i) minimal (ii) product-sum number 를 구하시오.

문제 풀이 과정

① **<u>이번 시간 문제를 잘 이해</u>**하기 위해 간단한 경우부터 시작해 문제 이해하고 답 이끌어내 보기

문제 명확히 이해하는데 도움

이 과정에서 문제에 대한 해법도 생각해 낼 수 있음

② 이번 시간 문제에 대한 **<u>풀이 방법 생각</u>**

③ 생각한 풀이 방법을 보다 **<u>효율적으로 개선</u>**

아래와 같이 **2개 이상의 자연수** $a_1, a_2, \ldots, a_k$의 **곱(product)**과 **합(sum)**으로 동시에 나타낼 수 있는 **자연수** P를 **product-sum** number라 하자.

$P = a_1 \times a_2 \times \ldots \times a_k = a_1 + a_2 + \ldots + a_k$

예를 들어, 6은 product-sum number이다. $6 = 1 \times 2 \times 3 = 1 + 2 + 3$ 로 나타낼 수 있기 때문이다.

(1) 1은 product-sum number 인가?

(2) 2는 product-sum number 인가?

아래와 같이 **2개 이상의 자연수** $a_1, a_2, \ldots, a_k$의 **곱(product)**과 **합(sum)**으로 동시에 나타낼 수 있는 **자연수** P를 **product-sum** number라 하자.

$P = a_1 \times a_2 \times \ldots \times a_k = a_1 + a_2 + \ldots + a_k$

예를 들어, 6은 product-sum number이다. $6 = 1 \times 2 \times 3 = 1 + 2 + 3$ 로 나타낼 수 있기 때문이다.

(3) 3은 product-sum number 인가?

(4) 4는 product-sum number 인가?

(5) 5는 product-sum number 인가?

(6) 6은 product-sum number 인가?

(7) 7은 product-sum number 인가?

(8) 8은 product-sum number 인가?

8은 둘 이상의 방법으로 인수 분해할 수 있다.

따라서 8을 product-sum으로 표현할 수 있는 모든 가능한 방법을 생각해 보자.

(9) 9는 product-sum number 인가?

(10) 10은 product-sum number 인가?

(11) 11은 product-sum number 인가?

(12) 12는 product-sum number 인가? 12는 둘 이상의 방법으로 인수 분해할 수 있다.

12를 product-sum으로 표현할 수 있는 모든 가능한 방법을 생각해 보자.

문제 정의: product-sum numbers

아래와 같이 **2개 이상의 자연수** $a_1, a_2, \ldots, a_k$의 **곱(product)**과 **합(sum)**으로 동시에 나타낼 수 있는 **자연수** P를 **product-sum** number라 하자.

$P = a_1 \times a_2 \times \ldots \times a_k = a_1 + a_2 + \ldots + a_k$

예를 들어, 6은 product-sum number이다. $6 = 1 \times 2 \times 3 = 1 + 2 + 3$ 로 나타낼 수 있기 때문이다.

자연수 입력 **N**이 주어졌을 때, $2 \le k \le N$ 범위에 속한 모든 k 값 각각에 대해

(i) minimal (ii) product-sum number 를 구하시오.

이제 minimal product-sum number에 대해 알아보자.

자연수 **k(≥2)가 주어졌을 때**, minimal product-sum number는

k개의 인수 $a_1, a_2, \ldots, a_k$의 곱(product)과 합(sum)으로 표현되는 **가장 작은 수** P이다.

$$P = a_1 \times a_2 \times \ldots \times a_k = a_1 + a_2 + \ldots + a_k$$

예:**k=5**에 대한 minimal product-sum number는 **5개**의 인수 $\{a_1, a_2, a_3, a_4, \text{and } a_5\}$로 분해되며 이들의 곱(product)이 합(sum)과 같은 가장 작은 수이다. 앞에서 풀이했던 문제의 예를 들면 **8, 9, 10이 5개의 인수를 가진 product-sum number**이다. 이보다 큰 수중에도 5개의 인수를 가진 product-sum number가 있을 수 있지만 그러한 가장 작은 수는 8이다. 따라서 k=5에 대한 minimal product-sum number는 8이다.

$8 = 1 \times 1 \times 2 \times 2 \times 2 = 1 + 1 + 2 + 2 + 2$

$9 = 1 \times 1 \times 1 \times 3 \times 3 = 1 + 1 + 1 + 3 + 3$

$10 = 1 \times 1 \times 1 \times 2 \times 5 = 1 + 1 + 1 + 2 + 5$

(13) k=2에 대한 minimal product-sum number (2개 인수의 product-sum으로 표현되는 가장 작은 수)는?

자연수 **k(≥2)가 주어졌을 때**, minimal product-sum number는

k개의 인수 $a_1, a_2, \ldots, a_k$의 곱(product)과 합(sum)으로 표현되는 **가장 작은 수** P이다.

$$P = a_1 \times a_2 \times \ldots \times a_k = a_1 + a_2 + \ldots + a_k$$

(14) k=3에 대한 minimal product-sum number (3개 인수의 product-sum으로 표현되는 가장 작은 수)는?

(15) k=4에 대한 minimal product-sum number는 무엇인가?

(16) k=5에 대한 minimal product-sum number는 무엇인가?

(17) k=6에 대한 minimal product-sum number는 무엇인가?

(18) k=7에 대한 minimal product-sum number는 무엇인가?

(19) k=8에 대한 minimal product-sum number는 무엇인가?

아래와 같이 **2개 이상의 자연수** $a_1, a_2, \ldots, a_k$의 **곱(product)**과 **합(sum)**으로 동시에 나타낼 수 있는 **자연수** P를 **product-sum** number라 하자.

$P = a_1 \times a_2 \times \ldots \times a_k = a_1 + a_2 + \ldots + a_k$

예를 들어, 6은 product-sum number이다. $6 = 1 \times 2 \times 3 = 1 + 2 + 3$ 로 나타낼 수 있기 때문이다.

자연수 입력 **N**이 주어졌을 때, $2 \le k \le N$ 범위에 속한 모든 k 값 각각에 대해

(i) minimal (ii) product-sum number 를 구하시오.

즉, 여러 개의 숫자를 답해야 함 (N-1개)

모든 k 값에 대해, k개 인자로 구성된 product-sum number는 반드시 하나 이상 존재한다고 보고 진행

(20) N=2일 때 답은 [4] 이다. 답이 [4]가 되는 이유를 이해 하시오.

(21) N=3일 때 답은 [4,6] 이다. 답이 [4,6]이 되는 이유를 이해 하시오.

(22) N=4일 때 답은 [4,6,8] 이다. 답이 [4,6,8]이 되는 이유를 이해 하시오.

(23) N=5일 때 답은 [

(24) N=6일 때 답은 [

(25) N=7일 때 답은 [

(26) N=8일 때 답은 [

첫 번째 알고리즘

for i=1에서 시작해 $2 \le k \le N$ 범위의 모든 k에 각각 대한 minimal product-sum number를 구할 때 까지

  for 자연수 i에 대한 서로 다른 인수분해 $i=a_1 \times a_2 \times \dots \times a_j$ 각각에 대해 (각 $a_p \ge 2$)

    if (product $\ge$ sum)

      (product – sum) 개 만큼의 1을 추가해 product-sum number 만들기

      이에 대한 인수의 개수 k = **j** + **(product – sum)** 이므로

      i가 k개 인수를 갖는 첫 product-sum number 라면 minimal로 기록

첫 번째 알고리즘

for i=1에서 시작해 $2 \le k \le N$ 범위의 모든 k에 각각 대한 minimal product-sum number를 구할 때 까지

  for 자연수 i에 대한 서로 다른 인수분해 $i=a_1 \times a_2 \times \dots \times a_j$ 각각에 대해 (각 $a_p \ge 2$)

    if (product $\ge$ sum)

      (product - sum) 개 만큼의 1을 추가해 product-sum number 만들기

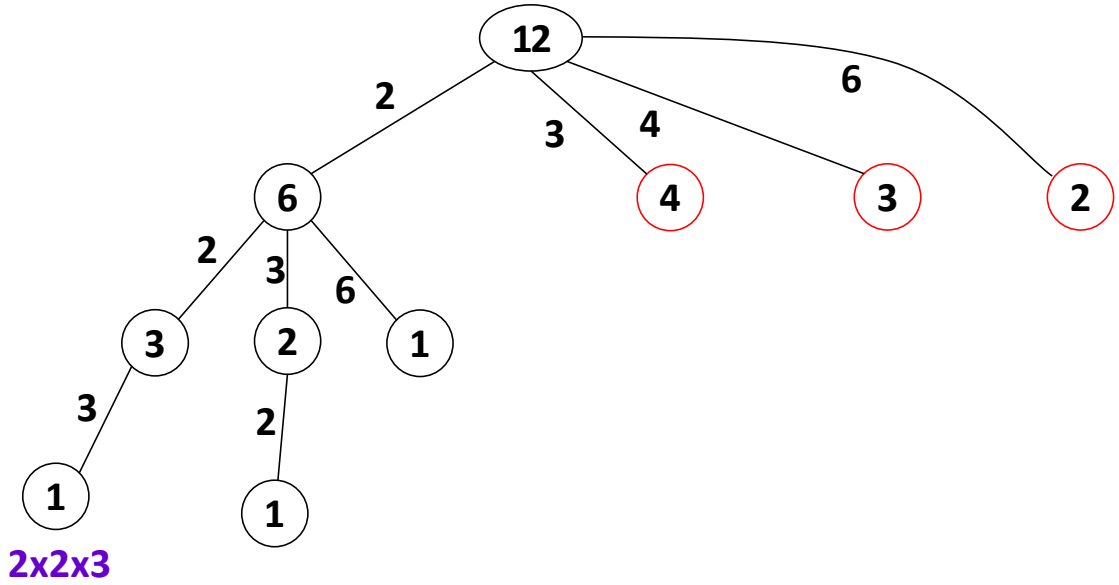
      이에 대한 인수의 개수 k = **j** + **(product - sum)** 이므로

      i가 k개 인수를 갖는 첫 product-sum number 라면 minimal로 기록

(27) 위 알고리즘에서 **가장 시간이 많이 소요되는 작업(또는 연산)**은 무엇인가?

속도를 개선하기 위해서는 이 작업을 좀 더 효율적으로 개선해야 할 것이다.

인수분해(나눗셈) Tree: 12를 **인수 분해**할 수 있는 **모든 가능한 방법** 찾는 과정 (**트리**)

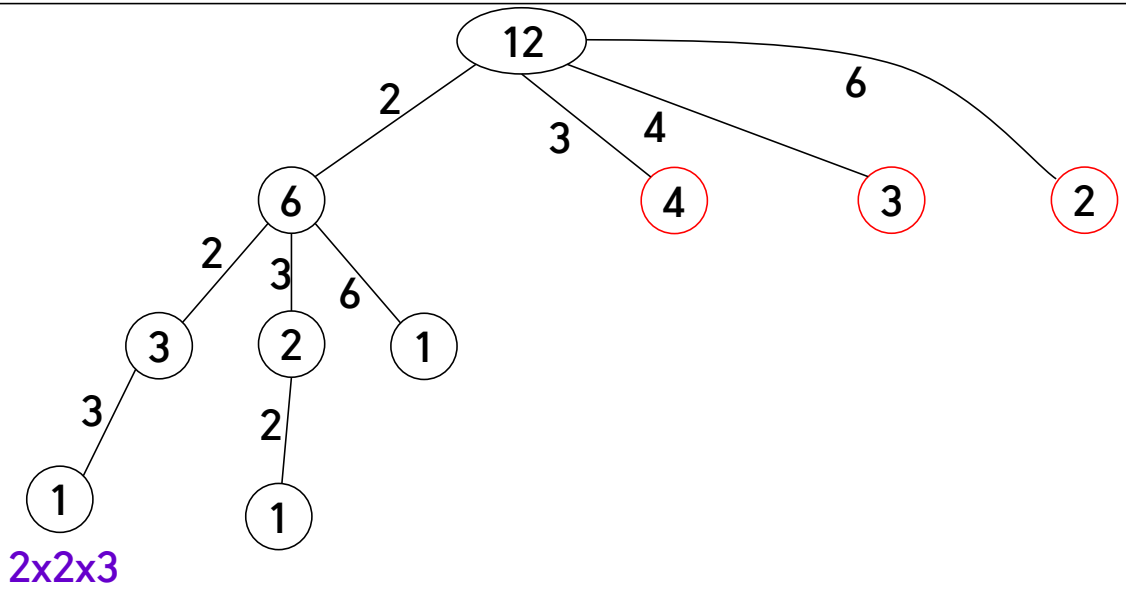


n : 인수 분해할(나눌) 숫자

a : 인수 (제수)

$a_1x...xa_j$: root에서 현재 정점까지의 숫자가 의미하는 인수 분해

인수분해(나눗셈) Tree: 12를 **인수 분해**할 수 있는 **모든 가능한 방법** 찾는 과정 **(트리)**



(28) Figure 1에서 붉은색 3개 정점(4, 3, 2) 아래의 가지를 그려 tree를 완성 하시오.

(29) 완성된 Tree에서 각 leaf가 어떤 인수분해를 나타내는지 표시하시오.

(30) 완성된 Tree에서 같은 인수 분해를 나타내는 leaf들(duplicate)을 모두 찾으시오.

(31) 지금까지 본 Tree 형태로 모든 가능한 인수분해를 탐색할 때,
중복되는 경우(duplicate)는 탐색하지 않으려면 어떤 조건에 따라 탐색해야 할까?

코드로 작성하는 방법은
예습자료 & 퀴즈에서 보았음

Tree 형태 Solution Space 탐색

트리 탐색 과정과 유의사항에 익숙해 지기

<왜 **Tree**에 대해 배우는가?>

- Tree는 **여러 단계를 거쳐 하나의 해를 선택**하는 상황에서 **선택할 수 있는 모든 가능성**을 나타내는데 자연스러운 구조임
- 여러 가능한 해 중 **최적의 해를 선택**하기 위해서는 **Tree를 탐색**해야 함
- 이번 주에는 **효율적으로 Tree를 탐색하기 위해 고려할 것들**을 배워보겠음

01. 문제에서 사용하는 정의 이해: minimal product-sum number

02. 첫 번째 알고리즘

03. 두 번째 알고리즘

04. 정리 문제 풀이

05. 실습 문제 풀이 & 질의 응답

두 번째 알고리즘

(32) 인수분해를 할 때 가장 많이 수행되는 연산은 무엇인가?

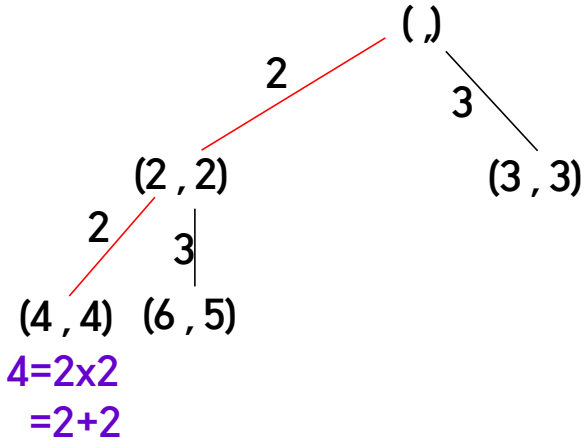
즉 인수분해를 할 때는 어떤 연산을 주로 수행하는가? (예: 덧셈, 뺄셈, 곱셈, 나눗셈, 합집합, 교집합 등)

※ 참조: 네 가지 기본 정수 연산을 수행 하는데 는 아래와 같은 CPU cycle이 소요됨
- 덧셈/뺄셈: 1 cycle
- 곱셈: 10 cycles
- 나눗셈: 66-80 cycles

※ 곱셈은 나눗셈보다 효율적으로 수행하는 방법이 존재하여 더 적은 수의 cycle 소요됨

현재 사용하는 연산과 반대되는 일을 하는 연산이 더 빠르다면, 더 빠른 연산을 사용해 같은 일을 할 수 있는지 생각해 보자.

곱셈 Tree: 여러 다른 수를 인수 분해할 수 있는 모든 가능한 방법을 한 번에 찾는 과정



(p, s): root에서 현재 정점까지의 숫자가 나타내는 곱 p와 합 s

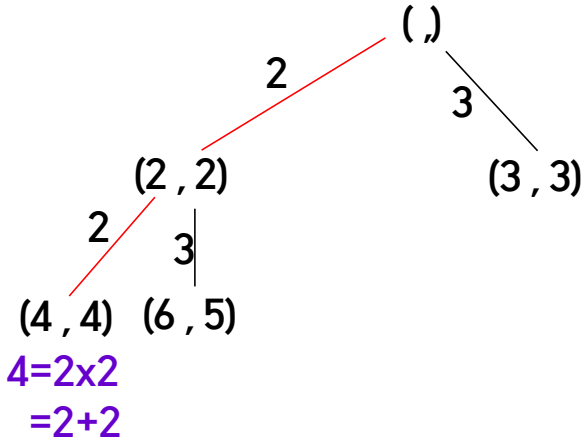
a : 인수 (곱하는 수)

간선에 있는 수는 나눗셈 트리에서와 마찬가지로 ≥ 2 조건을 만족하도록 함

$p=a_1 x \cdots x a_j$: root에서 현재 정점까지의 숫자가 나타내는 product-sum number

$=a_1+\cdots+a_j$

곱셈 Tree: 여러 다른 수를 인수 분해할 수 있는 모든 가능한 방법을 한 번에 찾는 과정



(33) 곱셈 tree를 좀 더 잘 이해하기 위해 위 트리에 가지를 더해 완성해 보시오.

특히 곱이 ≤ 9 조건을 만족하는 모든 경우를 탐색한다고 가정 하시오. (Hint: 총 6개의 가지를 추가할 수 있음)

인수 2개 이상이며, 각 인수 ≥ 2인 인수분해 찾아야 product-sum number 만들 수 있음

(34) 완성된 Tree에서 같은 인수 분해를 나타내는 leaf들(duplicate)을 모두 찾으시오.

(35) 탐색의 속도를 높이기 위해 **중복되는 경우(duplicate)는 탐색하지 않으려면 어떤 조건에 따라 탐색**해야 할까?

두 번째 알고리즘

for 탐색 과정에서 찾은 서로 다른 인수분해 $i=a_1 \times a_2 \times \dots \times a_j$ 각각에 대해 (각 $a_p \geq 2$)

if (product $\geq$ sum)

(product - sum) 개 만큼의 1을 추가해 product-sum number 만들기

이에 대한 인수의 개수 k = j + (product - sum) 이므로

i가 k개 인수를 갖는 가장 작은 수라면 기록

(36) 앞에서 그린 곱셈 tree에서 중복된 정점이 제거 되었다고 가정하자. 각 정점에서 어떤 product-sum number를 찾을 수 있으며, 이는 어떤 k 값에 대한 가능한 해인지 표기하시오. ※ depth$\geq$2 인 정점에 대해서만 답하시오. Product-sum number의 정의에 따라 '2개 이상의 자연수의 곱으로 표현되는 수'만을 포함하기 때문이다.

(37) 나눗셈 트리를 사용하는 경우와 비교해 곱셈 트리를 사용할 때의 장점은 무엇인가?

Tree 형태 Solution Space 탐색

트리 탐색 과정과 유의사항에 익숙해 지기

<왜 **Tree**에 대해 배우는가?>

- Tree는 **여러 단계를 거쳐 하나의 해를 선택**하는 상황에서 **선택할 수 있는 모든 가능성**을 나타내는데 자연스러운 구조임
- 여러 가능한 해 중 **최적의 해를 선택**하기 위해서는 **Tree를 탐색**해야 함
- 이번 주에는 **효율적으로 Tree를 탐색하기 위해 고려할 것들**을 배워보겠음

01. 문제에서 사용하는 정의 이해: minimal product-sum number

02. 첫 번째 알고리즘

03. 두 번째 알고리즘

04. 정리 문제 풀이

05. 실습 문제 풀이 & 질의 응답

(38) [소인수분해] 지금까지는 주어진 숫자를 2 이상의 모든 자연수로 분해하는 경우를 고려하였다.
이제 2 이상의 자연수 중 **소인수(prime number)로만 분해한다**는 조건이 주어졌다고 가정하고
(예: {2,3,5,7,11,13, ...}) 곱셈 tree를 그려보시오. 특히 **곱이 ≤9 조건을 만족하는 모든 경우를 탐색**한다고 가정하시오. Tree를 그릴 때 중복되는 인수분해는 탐색하지 않도록 하시오.
(Hint: 총 4개의 소인수분해를 얻을 수 있음)

(39) 앞 문제의 각 정점이 나타내는 인수분해 및 이로부터 찾을 수 있는 product-sum number를 각 정점에 표기하시오. Depth$\geq$2 인 정점에 대해서만 답하시오.

(40) k개의 인수로 인수 분해할 수 있는 **가장 작은 수**를 찾는 방법을 제시하시오.

각 인수는 $\geq$2인 정수여야 한다.

또한 제시한 방법을 사용해서 k=5인 경우의 해답을 보이시오.

(41) k개의 **서로 다른** 인수로 인수 분해할 수 있는 **가장 작은 수**를 찾는 방법을 제시하시오.

각 인수는 ≥ 2인 정수여야 한다.

또한 제시한 방법을 사용해서 k=5인 경우의 해답을 보이시오.

<정리>

- 새로운 문제를 풀 때는 **간단한 경우에서 시작**해 **손으로** 풀어가 보며 **문제를 충분히 이해할 수 있는 시간**을 갖자. 이러한 과정 중에 문제에 대한 첫 번째 해결방법을 떠올릴 수 있을 것이고, 그 후에는 이를 보다 효율적으로 만드는 것에 집중하면 된다.

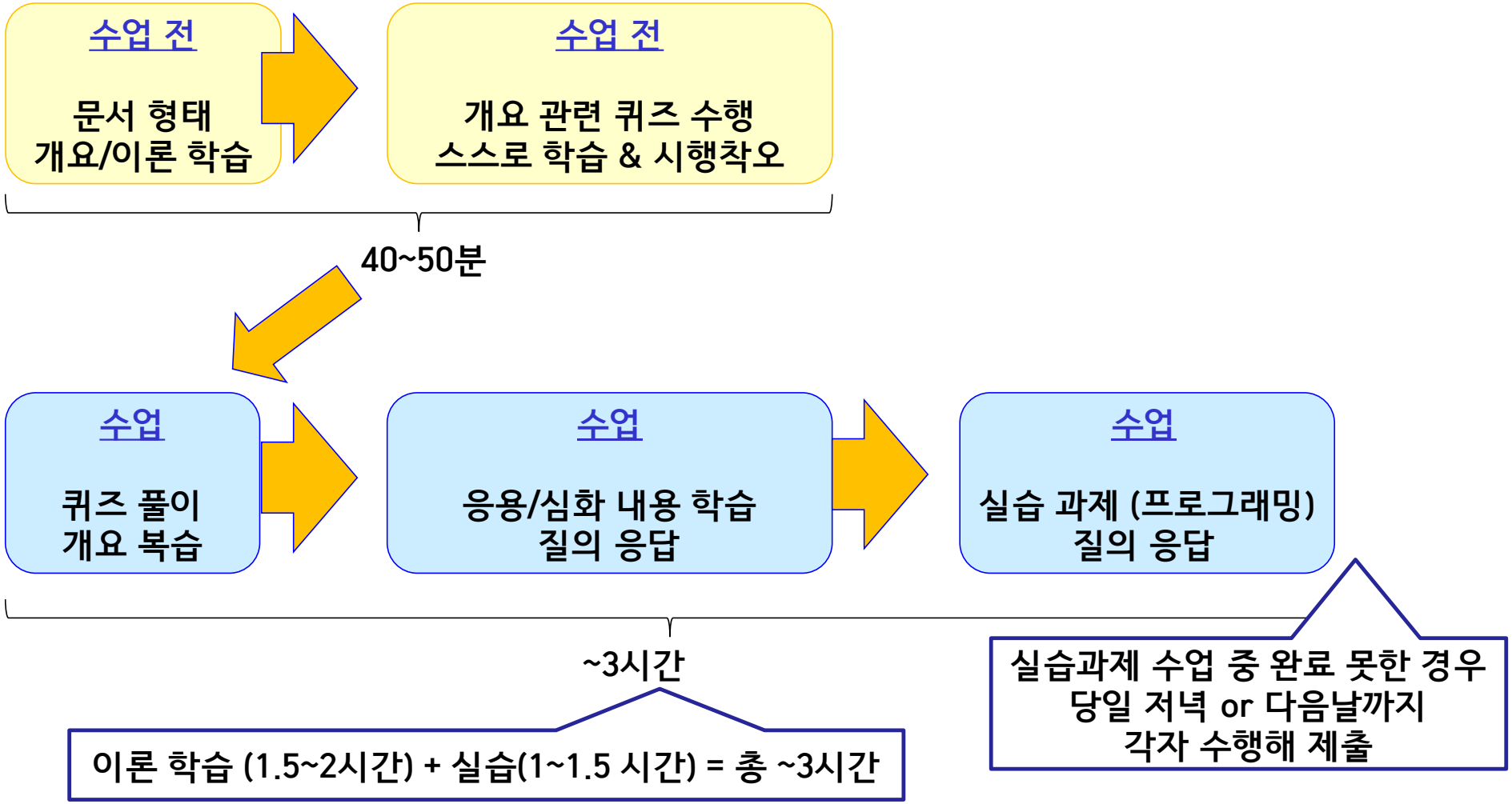
- **처음에는 비효율적인 방법(예: 전 범위를 탐색하는 brute-force algorithm)에서 시작**하여도 괜찮다. 이로부터 **중복된 경우**를 탐색하는 경우를 발견하며 탐색 범위를 **조금씩 줄여 나가**다 보면 훨씬 더 효율적인 방법에 도달할 수 있을 것이다.

- 문제에 대한 해법을 더 효율적으로 개선하기에 앞서 반드시 **어느 부분이 가장 시간이 많이 소요되는 부분(major performance bottleneck)인지를 생각**해 보고 이 부분을 개선해 나가자. 그 외의 중요하지 않은 부분을 개선한다면 전체적인 성능이 크게 개선되지 않으므로 대신 중요한 부분을 개선하는데 더 많은 시간을 투자하자.

- 때때로 처음 생각한 **방법을 완전히 뒤집어 생각**함으로써 더 나은 방법에 도달하기도 한다. 항상 여러 다른 해법을 생각해 보고 어느 방법이 나은지 비교해 보자.

- **연산의 종류에 따라 수행하는 시간이 다름**을 기억하자.

스마트 출결



05. 실습문제풀이

- **이번 시간에 배운 내용에 대한 실습 문제 풀이 & 질의 응답**
- **채점 방식은 지난 시간 문제 풀이때와 유사함**

- **왜 중요한가?**
- 이번 주 배운 내용을 총괄하는 문제 풀이 통해 배운 내용 활용 & 복습
- 문제 풀이 점수는 이번 주 **과제 점수에 포함**됨

findProductSum() 함수 구현 조건: product-sum number 탐색 코드 작성

- 자연수 **n**이 입력으로 주어졌을 때, 2~**n** 범위에 속하는 모든 product-sum number를 찾아 반환

  def findProductSum (**n**):

- 입력 **n**: 2≤**n**≤100 범위의 정수
  - 위 범위를 벗어나는 값은 입력으로 들어오지 않는다고 가정 (즉 오류 처리 하지 않아도 됨)

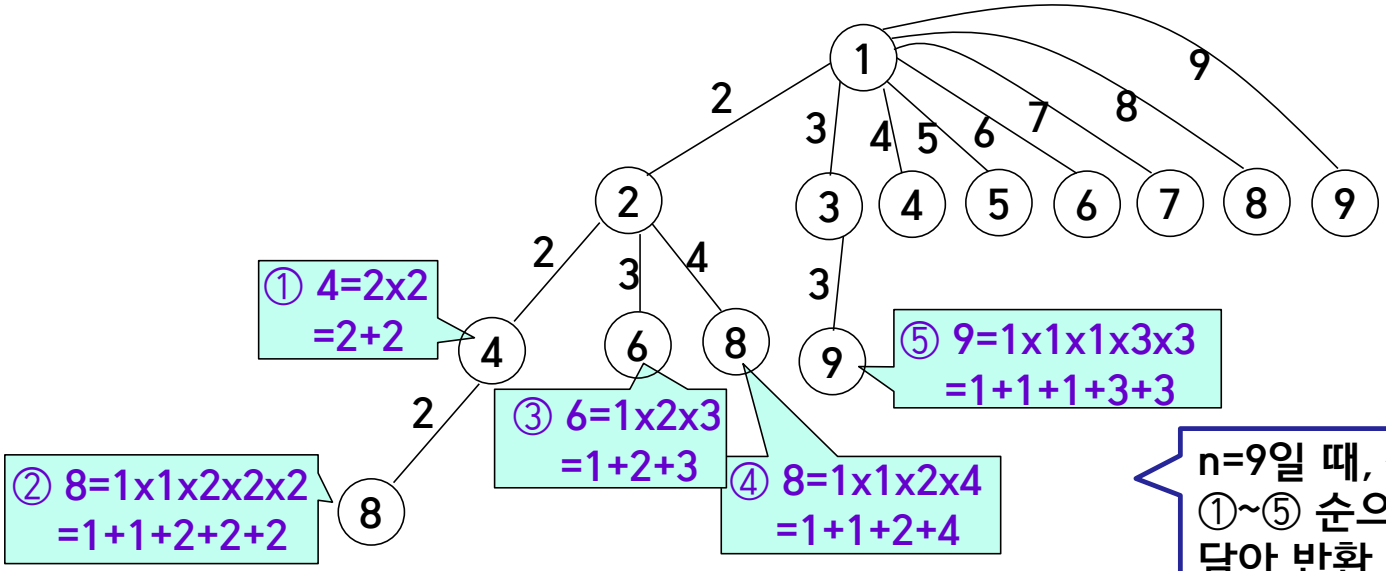
- 반환 값: 2~**n** 범위의 모든 product-sum number를 문자열 형식으로 담은 리스트
  - 한 숫자가 여러 다른 방식으로 표현된다면 이들을 모두 리스트에 담아야 함
  - 같은 문자열을 두 번 이상 중복으로 담으면 안 됨
  - 문자열을 담는 순서는 중요하지 않으며, 필요한 문자열이 빠짐없이 담겨 있으면 됨
  - 예: **n**=9일 때, 반환 값은 [“4=2*2=2+2”, “8=1*1*2*2*2=1+1+2+2+2”, “6=1*2*3=1+2+3”, “8=1*1*2*4=1+1+2+4”, “9=1*1*1*3*3=1+1+1+3+3”]

- 이번 시간에 제공한 코드 ProductSum.py의 findProductSum() 함수 내부에 코드 작성해 제출
- 재귀호출을 사용해도 괜찮음 (2≤**n**≤100 범위에서는 문제 없음)

입출력 예 (2~n 범위의 모든 product-sum number 반환)

```
print(findProductSum(9))
```

```
['4=2*2=2+2', '8=1*1*2*2*2=1+1+2+2+2', '6=1*2*3=1+2+3', '8=1*1*2*4=1+1+2+4', '9=1*1*1*3*3=1+1+1+3+3']
```



n=9일 때, 작은 인수부터 DFS 방식으로 탐색하면 ①~⑤ 순으로 찾게 되는데, 이 순서대로 리스트에 담아 반환

그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

입출력 예 (2~n 범위의 모든 product-sum number 반환)

```
print(findProductSum(4))
```

```
['4=2*2=2+2']
```

```
print(findProductSum(6))
```

```
['4=2*2=2+2', '6=1*2*3=1+2+3']
```

```
print(findProductSum(9))
```

```
['4=2*2=2+2', '8=1*1*2*2*2=1+1+2+2+2', '6=1*2*3=1+2+3', '8=1*1*2*4=1+1+2+4', '9=1*1*1*3*3=1+1+1+3+3']
```

```
print(findProductSum(12))
```

```
['4=2*2=2+2', '8=1*1*2*2*2=1+1+2+2+2', '12=1*1*1*1*1*2*2*3=1+1+1+1+1+2+2+3', '6=1*2*3=1+2+3',
'8=1*1*2*4=1+1+2+4', '10=1*1*1*2*5=1+1+1+2+5', '12=1*1*1*1*2*6=1+1+1+1+2+6', '9=1*1*1*3*3=1+1+1+3+3',
'12=1*1*1*1*1*3*4=1+1+1+1+1+3+4']
```

그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

findMinimalProductSum() 함수 구현 조건: minimal product-sum number 탐색 코드 작성

- 자연수 **n**이 입력으로 주어졌을 때, 2~**n** 범위에 속한 값 각각에 대한 minimal product-sum number 찾아 반환

  def findMinimalProductSum (**n**):

- 입력 **n**: 2≤**n**≤100 범위의 정수
  - 위 범위를 벗어나는 값은 입력으로 들어오지 않는다고 가정 (즉 오류 처리 하지 않아도 됨)

- 반환 값: 2~**n** 범위에 속한 k값 각각에 대한 minimal product-sum number를 순서대로 담은 리스트
  - 예: n=**8**일 때, 반환 값은 [4,6,8,8,12,12,12] 이며, 이들은 각각 2~**8**에 대한 minimal product-sum number임. 이번시간 문제 (20)~(26) 참조

- 이번 시간에 제공한 코드 ProductSum.py의 findMinimalProductSum() 함수 내부에 코드 작성해 제출
- 재귀호출을 사용해도 괜찮음 (2≤**n**≤100 범위에서는 문제 없음)
- 유의사항: 곱셈 트리로 탐색할 때, 곱이 n이 될 때가지만 탐색하면 곱이 n^2이 될 때까지 탐색하면 인자 수가 2~**n**개인 minimal product-sum number를 다 찾을 수 있음

입출력 예 (2~n 범위의 값 각각에 대한 product-sum number 반환)

```
print(findMinimalProductSum(2))
[4]
```

```
print(findMinimalProductSum(3))
[4,6]
```

```
print(findMinimalProductSum(4))
[4,6,8]
```

```
print(findMinimalProductSum(5))
[4,6,8,8]
```

```
print(findMinimalProductSum(6))
[4,6,8,8,12]
```

```
print(findMinimalProductSum(7))
[4,6,8,8,12,12]
```

```
print(findMinimalProductSum(8))
[4,6,8,8,12,12,12]
```

그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

유의사항

- 유의사항: 곱셈 트리로 탐색할 때, 곱이 **n**이 될 때 까지만 탐색하면 인자 수가 2~**n**개인 minimal product-sum number를 다 찾을 수 없음. 예를 들어 n=8일 때 답은 [4,6,8,8,12,12,12] 이므로, 8보다 큰 수인 12까지 탐색해야 함

- 일반적으로 곱이 $\mathbf{n}^2$이 될 때까지 탐색하면 인자 수가 2~**n**개인 minimal product-sum number를 빠뜨리지 않고 다 찾을 수 있음

그 외 프로그램 구현 조건

- 최종 결과물로 ProductSum.py 파일 하나만 제출하며, 이 파일만으로 코드가 동작해야 함

- import는 사용할 수 없음

- __main__ 아래의 코드는 작성한 함수가 올바름을 확인하는 코드로
- 코드 작성 후 스스로 채점해 보는데 활용하세요.
- 각 테스트 케이스에 대해 P(or Pass) 혹은 F(or Fail)이 출력됩니다.
- 코드를 제출할 때는 __main__ 아래 코드는 제거하거나 수정하지 말고 제출합니다. (채점에 사용되기 때문)

이번 시간 제공 코드 함께 보기

- ProductSum.py

실습 문제 풀이 & 질의 응답

- 2개 문제는 연속되는 문제이므로 **문제1→문제2 순서대로 풀이** 하세요.

- 종료 시간(17:00) 이전에 **일찍 모든 문제를 통과**한 경우 각자 **퇴실 가능**
- 실습 문제에 대한 코드는 lms 과제함에 **다음 날 23:59까지 제출** 가능합니다.
- 마감 시간까지 작성을 다 못한 경우는 (제출하지 않으면 0점이므로) 그때까지 작성한 코드를 꼭 제출해 부분점수를 받으세요.

- 실습 문제는 **개별 평가**입니다. 제출한 코드에 대한 유사도 검사를 실습 문제마다 진행하며, 코드를 건내 준 사례가 발견되면 0점 처리됩니다.
- 로직에 대해 서로 의견을 나누는 것은 괜찮지만, 코드를 직접 건내 주지는 마세요.
- 본인 실력 향상을 위해서도 **코드는 꼭 각자 직접 작성**해 주세요.