고 급 문 제 해 결

문제 16.1

5주차 발표 자료

2018117543 한재성

Chapter 16

Dynamic Programming

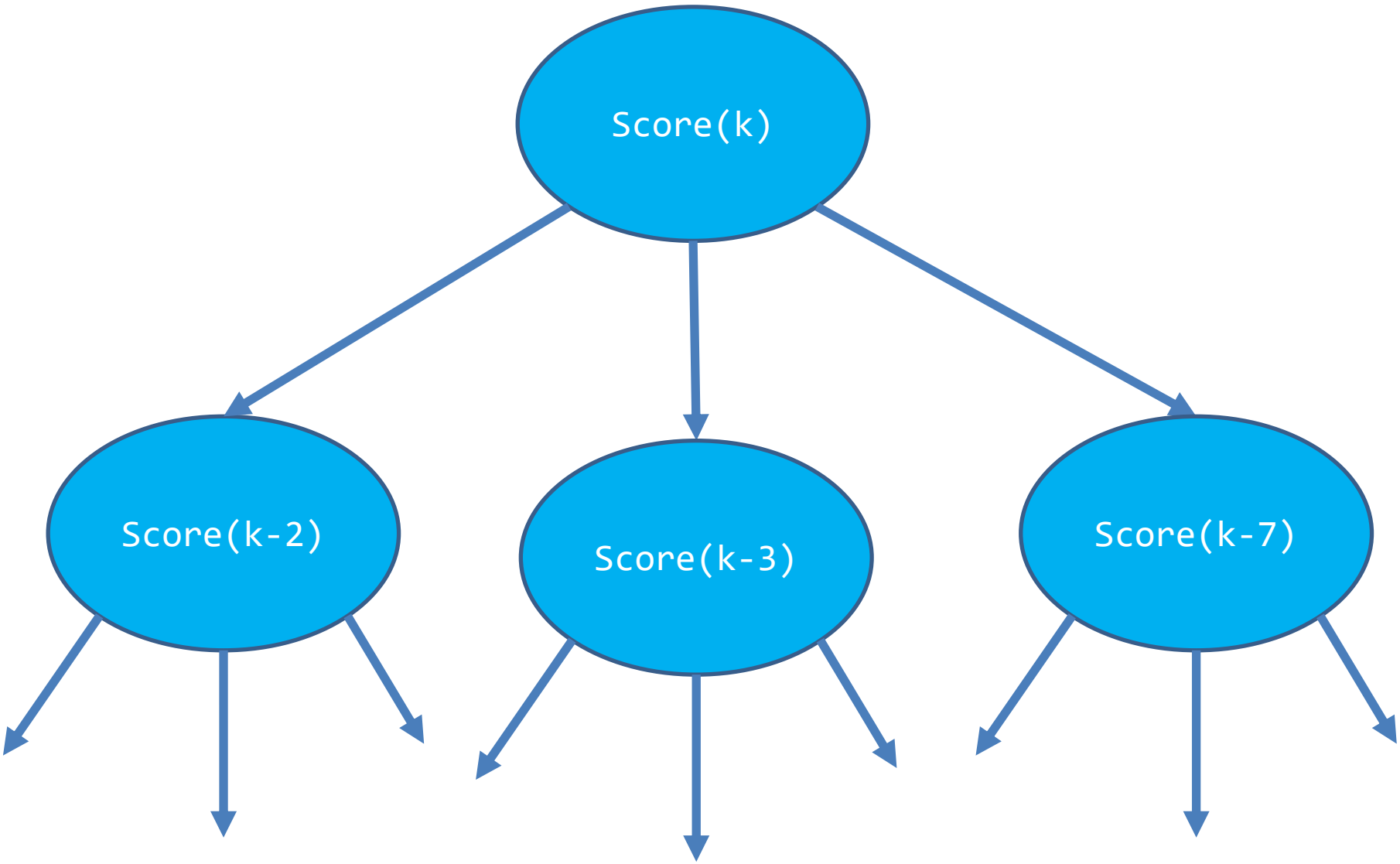
16.1 가능한 점수가 몇 개인지 구하기

In an American football game, a play can lead to 2 points (safety), 3 points (field goal), or 7 points (touchdown, assuming the extra point). Many different combinations of 2, 3, and 7 point plays can make up a final score. For example, four combinations of plays yield a score of 12:

- 6 safeties ($2 \times 6 = 12$),
- 3 safeties and 2 field goals ($2 \times 3 + 3 \times 2 = 12$),
- 1 safety, 1 field goal and 1 touchdown ($2 \times 1 + 3 \times 1 + 7 \times 1 = 12$), and
- 4 field goals ($3 \times 4 = 12$).

Write a program that takes a final score and scores for individual plays, and returns the number of combinations of plays that result in the final score.

Recursion



Score() <- Recursion

Recursion

```
def score(k):
    if k < 0:
        return 0
    if k == 2 or k == 3 or k == 7:
        return 1
    return score(k - 2) + score(k - 3) + score(k - 7)
```

```
score(9) =  4
score(7) =  1
score(6) =  2
score(5) =  2
```

Score() <- memoization

Recursion + Caching => D.P. (Memoization)

```
def caching_score(k):
    if k < 0:
        return 0
    if SCORE[k] != -1:
        return SCORE[k]
    SCORE[k] = score(k - 2) + score(k - 3) + score(k - 7)
    return SCORE[k]
```

Easy!

Path 찍어보기

Path를 찍어본 결과

```
score(9) =  4
[[7, 2], [2, 3, 2, 2], [3, 3, 3], [2, 7]]
score(7) =  1
[[7]]
score(6) =  2
[[2, 2, 2], [3, 3]]
score(5) =  2
[[3, 2], [2, 3]]
```

Path 찍어보기

```
score(12) =  12
[[2, 2, 2, 2, 2, 2], [3, 2, 2, 2, 3]]
[[3, 2, 2, 3, 2], [2, 2, 2, 3, 3]]
[[7, 2, 3], [3, 2, 7]]
[[7, 3, 2], [2, 3, 3, 2, 2]]
[[3, 3, 3, 3], [2, 3, 7]]
[[3, 7, 2], [2, 7, 3]]
```

plays yield a score of 12:

- 6 safeties ($2 \times 6 = 12$),
- 3 safeties and 2 field goals ($2 \times 3 + 3 \times 2 = 12$),
- 1 safety, 1 field goal and 1 touchdown ($2 \times 1 + 3 \times 1 + 7 \times 1 = 12$), and
- 4 field goals ($3 \times 4 = 12$).

Sort() & Hasing -> 해결

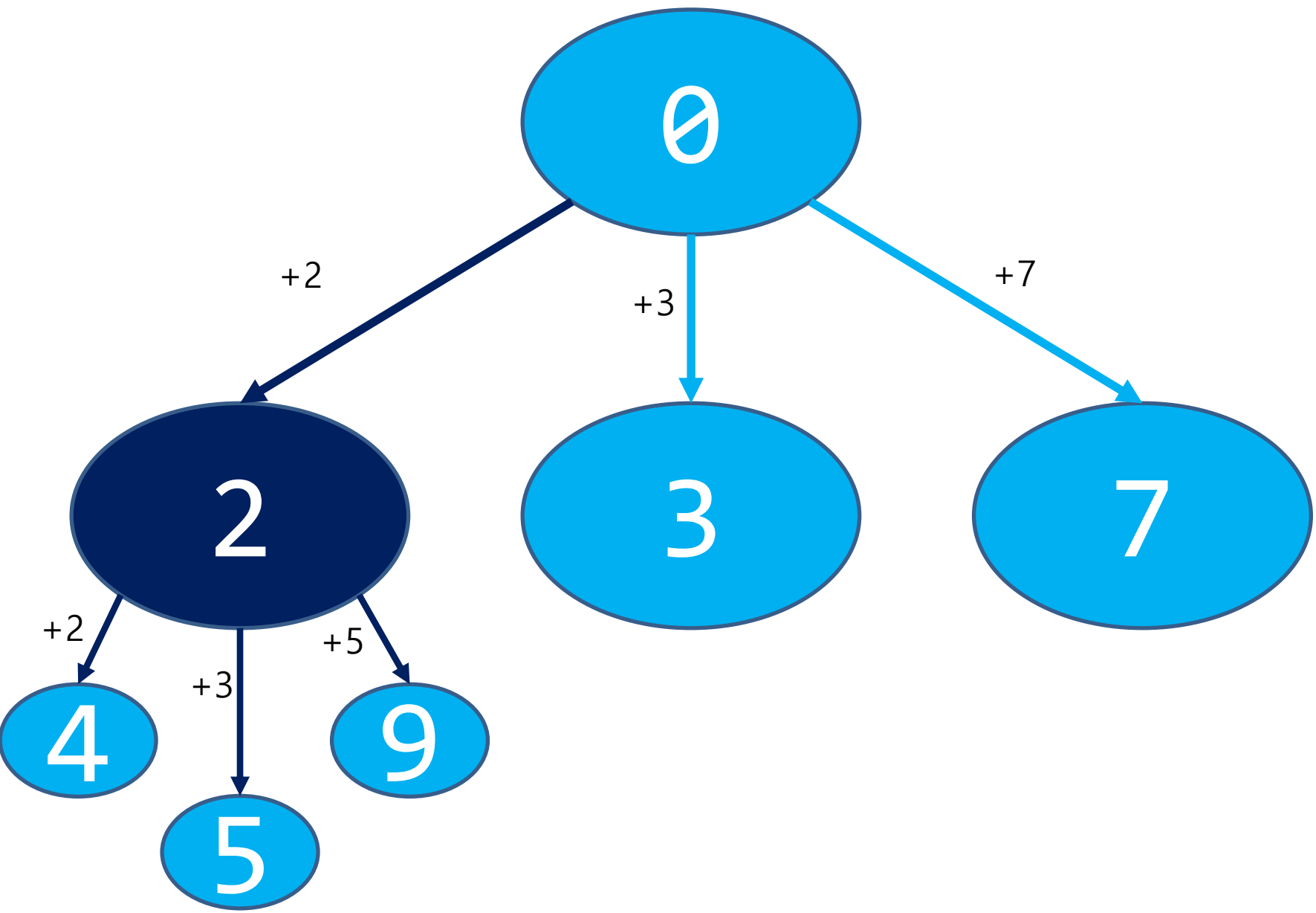
```
S = set()
for path in res:
    path = tuple(sorted(path))
    if path not in S:
        S.add(path)

print('score(12) = ', len(S))
print('path = ', S)
```

```
score(12) =  4
path =  {(3, 3, 3, 3), (2, 2, 2, 3, 3), (2, 3, 7), (2, 2, 2, 2, 2, 2)}
```

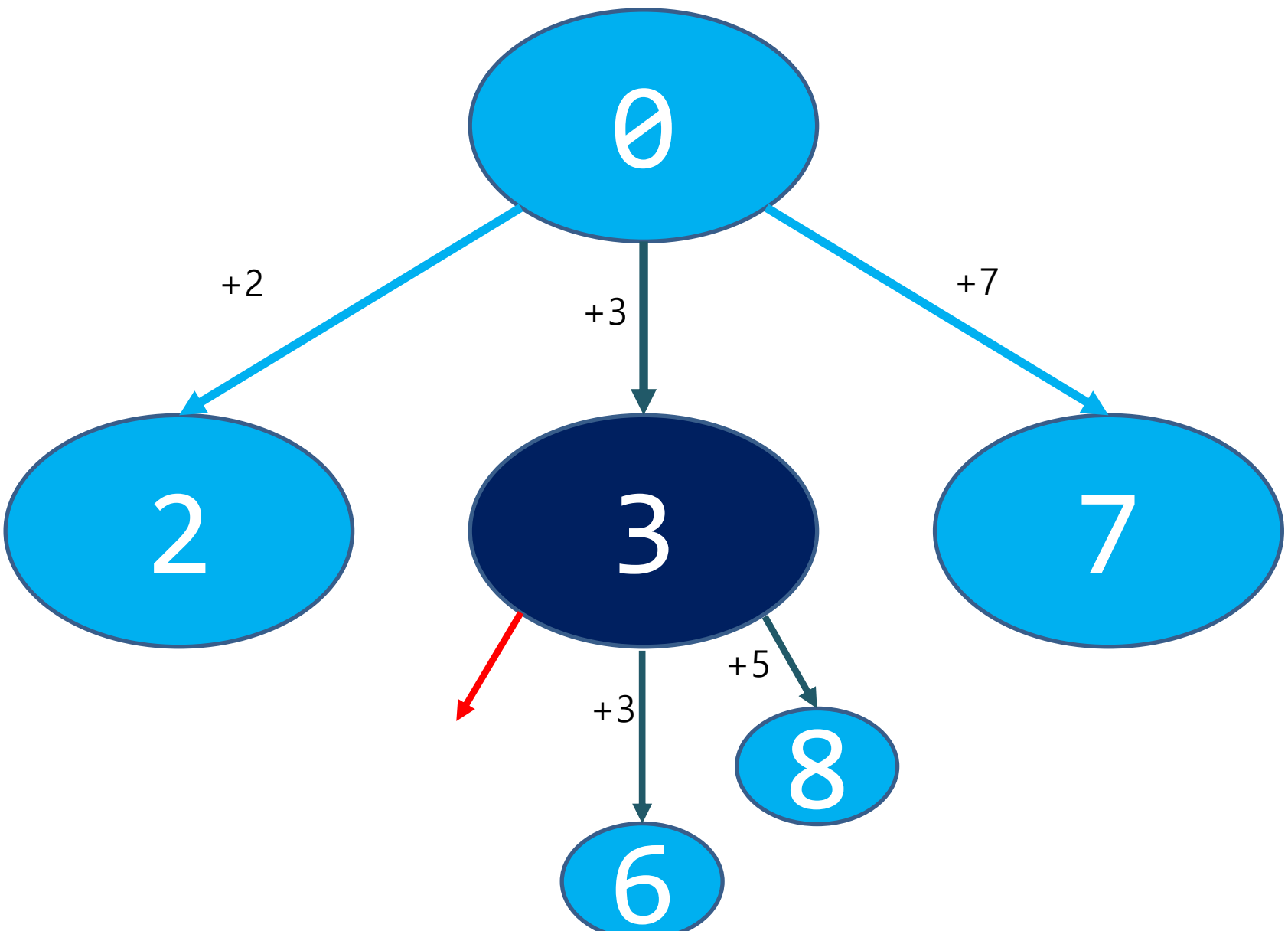
조합 만들기

- 중복을 제거하는 것이 가장 중요
- 우선, 조합을 만들고자 하는 배열이 Sort 되어야 한다.
- Recursion 하듯이 경우를 따지지 않고,
  - ✓ 한쪽 방향으로만 경우를 따져봐야 한다. (중복 방지)
  - ✓ 오름차순 정렬이므로, 증가하는 쪽으로만 탐색해야 함.

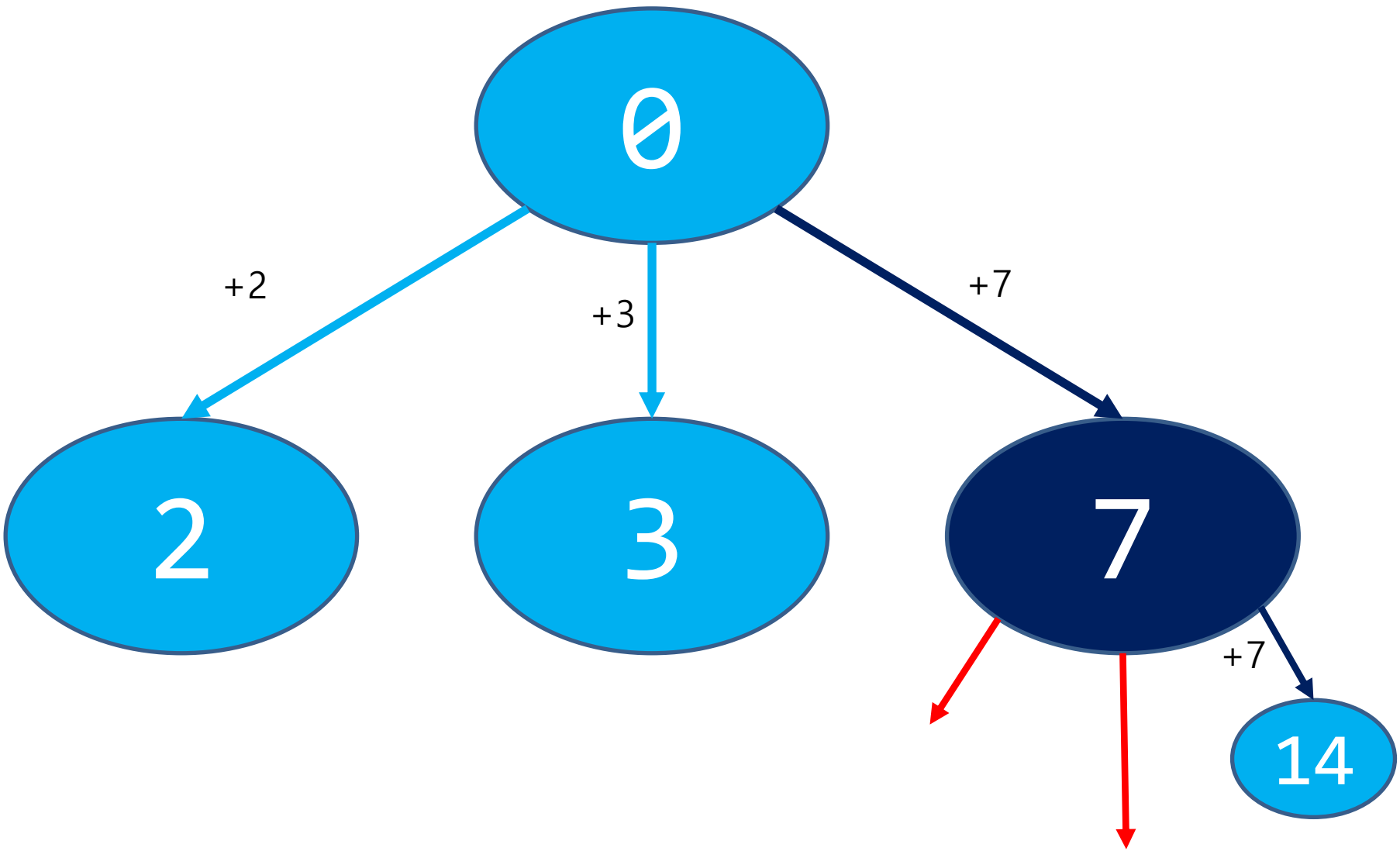
조합 만들기



조합 만들기



조합 만들기



조합 만들기

```python
points = [2, 3, 7]
res = []
path = []
def score( k:int, p:int, target:int):
    if k == target:
        res.append(list(path))
        return

    for i in range(p, len(points)):

        if k + points[i] > target:
            break

        path.append(points[i])
        score(k + points[i], i, target)
        path.pop()

points.sort()
score(0, 0, 12)
```

결과

Recursion + Sort() + Hasing

```
score(12) =  4
path =  {(3, 3, 3, 3), (2, 2, 2, 3, 3), (2, 3, 7), (2, 2, 2, 2, 2, 2)}
```

Recursion + Sort()

```
[[2, 2, 2, 2, 2, 2], [2, 2, 2, 3, 3], [2, 3, 7], [3, 3, 3, 3]]
```

Dynamic Programming

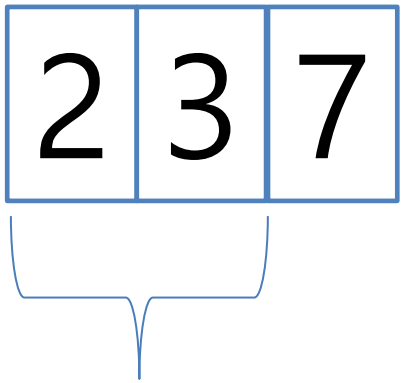
- A[i][j] := 얻을 수 있는 score 배열의 ~i번째 index까지만 사용해서 j점을 만드는 조합의 개수

- A[i][j] = A[i-1][j] + A[i][j – score[i]]

Point index ~ (i-1) 까지만 넣어서 j점을 만드는 경우의 수

Point index[i] 까지 포함하여 j – score[i]점을 만드는 경우의 수

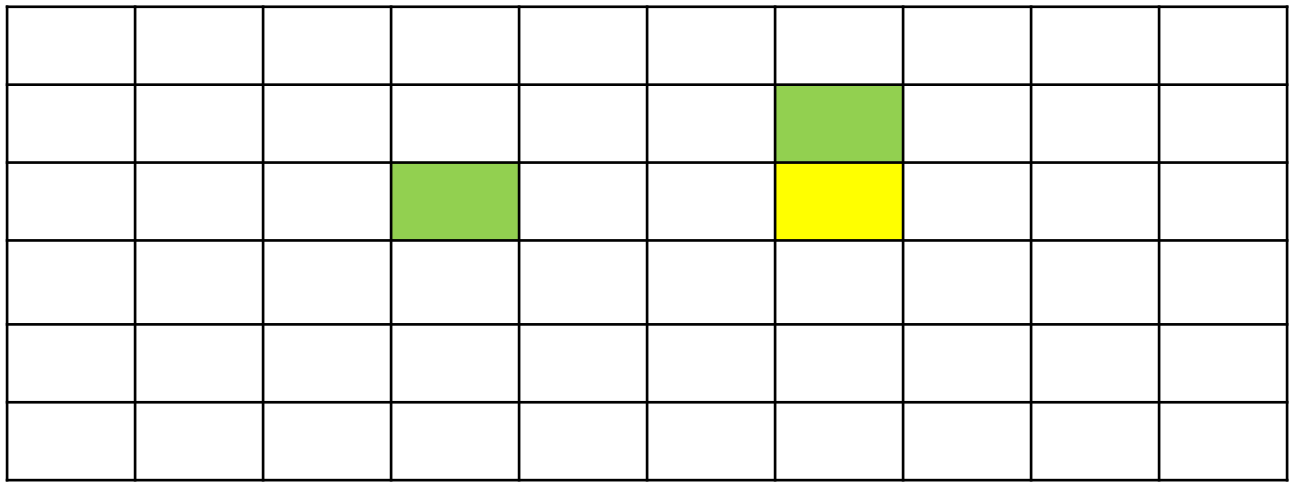
score



A[i][j] 관계

원하는 것 : combination sum 의 가짓수

	0	1	2	3	4	5	6	7	8	9	10	11	12
2	1	0	1	0	1	0	1	0	1	0	1	0	1
2,3	1	0	1	1	1	1	2	1	2	2	2	2	3
2,3,7	1	0	1	1	1	1	2	2	2	3	3	3	4



Dynamic Programming

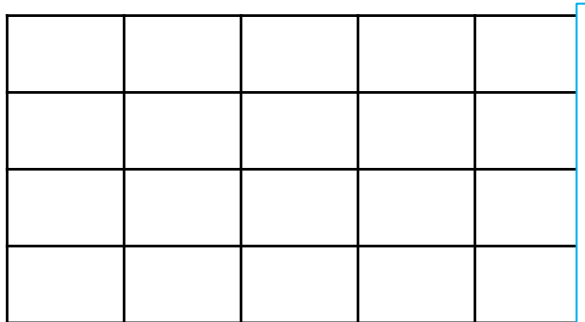
```python
def numCombinationsForFinalScore(finalScore, individualPlayScores):
    numCombinationsForScore = [[0] * (finalScore + 1) for _ in range(len(individualPlayScores))]
    for i in range(len(individualPlayScores)):
        numCombinationsForScore[i][0] = 1
        for j in range(1, finalScore + 1):
            if i - 1 >= 0:
                withoutThisPlay = numCombinationsForScore[i-1][j]
            else:
                withoutThisPlay = 0
            if j >= individualPlayScores[i]:
                withThisPlay = numCombinationsForScore[i][j - individualPlayScores[i]]
            else:
                withThisPlay = 0
            numCombinationsForScore[i][j] = withoutThisPlay + withThisPlay
    return numCombinationsForScore[len(individualPlayScores) - 1][finalScore]
```

Dynamic Programming

```python
def dp_score(W, score):
    dp = [[0] * (W + 1) for _ in range(len(score))]
    for i in range(len(score)):
        dp[i][0] = 1
        for j in range(1, W + 1):
            if i - 1 >= 0:
                prev = dp[i-1][j]       # score[:index] 까지만 이용해서 j점 만들기
            else:
                prev = 0                # outOfBound

            if j - score[i] >= 0:
                curr = dp[i][j-score[i]]    # score[:index + 1] 까지 이용해서 j - score[i]점 만들기
            else:
                curr = 0                    # outOfBound

            dp[i][j] = prev + curr
    return dp[len(score) - 1][W]
```

W(TARGET)

SCORE



	0	1	2	3	4	5	6	7	8	9	10	11	12
2	1	0	1	0	1	0	1	0	1	0	1	0	1
2,3	1	0	1	1	1	1	2	1	2	2	2	2	3
2,3,7	1	0	1	1	1	1	2	2	2	3	3	3	4

Summary

- **문제 16.1 : 가능한 점수가 몇 개인지 구하기**
  - ✓ Combination Sum
- Recursion + Sort() + Hasing
- Sort() + recursion
- Dynamic programming(Tabulation)

들어 주셔서 감사합니다