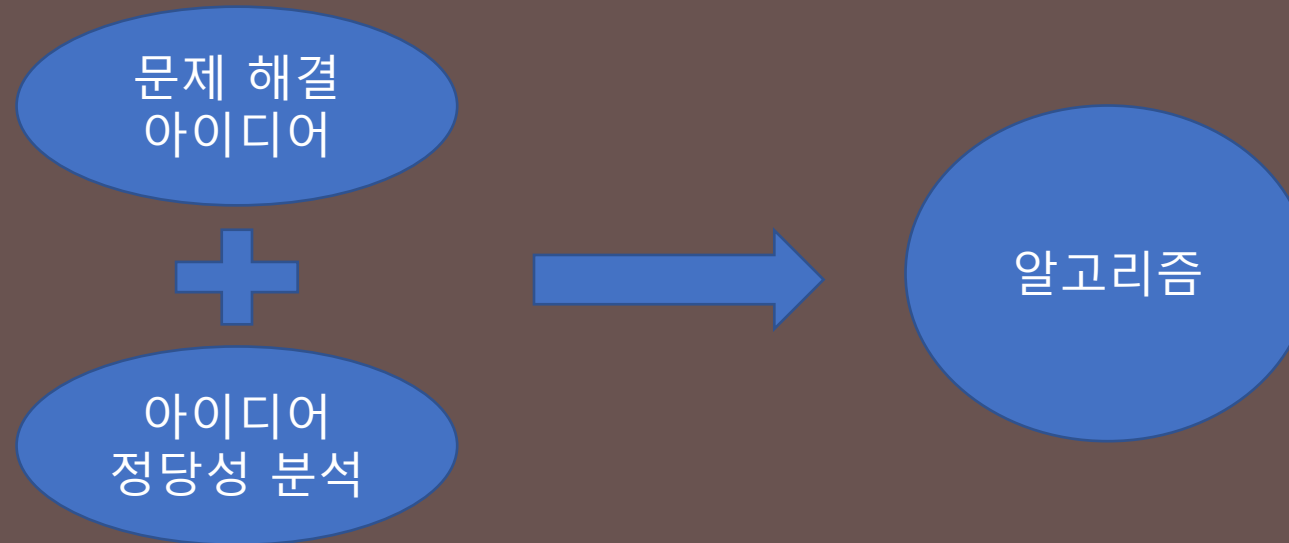


그리디 알고리즘

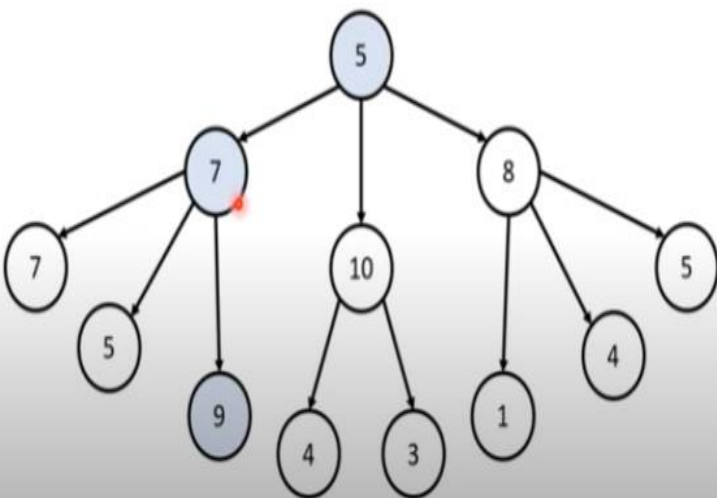
- 그리디 알고리즘(탐욕법)은 현재 상황에서 지금 당장 좋은 것만 고르는 방법을 의미합니다.
- 일반적인 그리디 알고리즘은 문제를 풀기 위한 최소한의 아이디어를 떠올릴 수 있는 능력을 요구합니다.
- 그리디 해법은 그 정당성 분석이 중요합니다.
 - 단순히 가장 좋아 보이는 것을 반복적으로 선택해도 최적의 해를 구할 수 있는지 검토합니다.



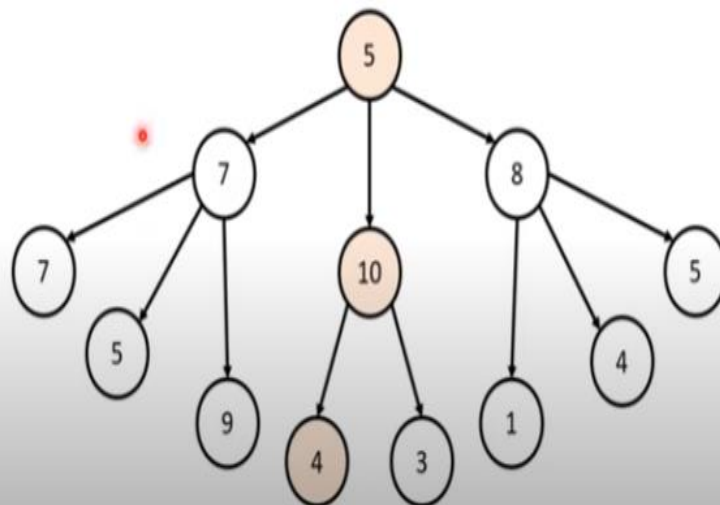
그리디 알고리즘

- 일반적인 상황에서 그리디 알고리즘은 최적의 해를 보장할 수 없을 때가 많습니다.
- 하지만 코딩 테스트에서의 대부분의 그리디 문제는 탐욕법으로 얻은 해가 최적의 해가 되는 상황에서, 이를 추론할 수 있어야 풀리도록 출제됩니다.

- [문제 상황] 루트 노드부터 시작하여 거쳐 가는 노드 값의 합을 최대로 만들고 싶습니다.
- Q. 최적의 해는 무엇인가요?



- [문제 상황] 루트 노드부터 시작하여 거쳐 가는 노드 값의 합을 최대로 만들고 싶습니다.
- Q. 단순히 매 상황에서 가장 큰 값만 고른다면 어떻게 될까요?



〈문제〉 거스름 돈: 문제 설명

- 당신은 음식점의 계산을 도와주는 점원입니다. 카운터에는 거스름돈으로 사용할 500원, 100원, 50원, 10원짜리 동전이 무한히 존재한다고 가정합니다. 손님에게 거슬러 주어야 할 돈이 N원일 때 거슬러 주어야 할 동전의 최소 개수를 구하세요. 단, 거슬러 줘야 할 돈 N은 항상 10의 배수입니다.



〈문제〉 거스름 돈: 문제 해결 아이디어

- 최적의 해를 빠르게 구하기 위해서는 가장 큰 화폐 단위부터 돈을 거슬러 주면 됩니다.
- N원을 거슬러 줘야 할 때, 가장 먼저 500원으로 거슬러 줄 수 있을 만큼 거슬러 줍니다.
 - 이후에 100원, 50원, 10원짜리 동전을 차례대로 거슬러 줄 수 있을 만큼 거슬러 주면 됩니다.
- $N = 1,260$ 일 때의 예시를 확인해 봅시다.

〈문제〉 거스름 돈: 문제 해결 아이디어

- [Step 4] 남은 돈: 0원



화폐 단위	500	100	50	10
손님이 받은 개수	2	2	1	1

〈문제〉 거스름 돈: 정당성 분석

- 가장 큰 화폐 단위부터 돈을 거슬러 주는 것이 최적의 해를 보장하는 이유는 무엇일까요?
 - 가지고 있는 동전 중에서 큰 단위가 항상 작은 단위의 배수이므로 작은 단위의 동전들을 종합해 다른 해가 나올 수 없기 때문입니다.
- 만약에 800원을 거슬러 주어야 하는데 화폐 단위가 500원, 400원, 100원이라면 어떻게 될까요?
- 그리디 알고리즘 문제에서는 이처럼 문제 풀이를 위한 최소한의 아이디어를 떠올리고 이것이 정당한지 검토할 수 있어야 합니다.

〈문제〉 거스름 돈: 답안 예시 (Python)

```
n = 1260
count = 0

# 큰 단위의 화폐부터 차례대로 확인하기
array = [500, 100, 50, 10]

for coin in array:
    count += n // coin # 해당 화폐로 거슬러 줄 수 있는 동전의 개수 세기
    n %= coin

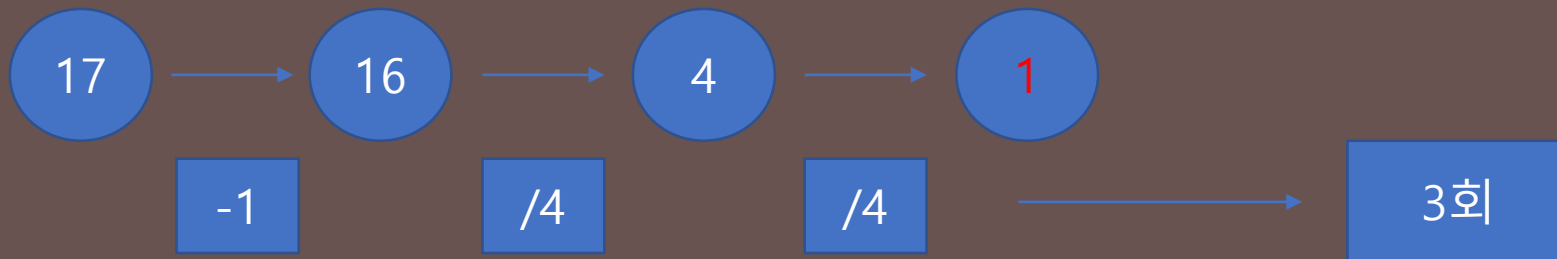
print(count)
```

〈문제〉 1이 될 때까지: 문제 설명

- 어떠한 수 N 이 1이 될 때까지 다음의 두 과정 중 하나를 반복적으로 선택하여 수행하려고 합니다. 단, 두 번째 연산은 N 이 K 로 나누어 떨어질 때만 선택할 수 있습니다.
 - N 에서 1을 뺍니다.
 - N 을 K 로 나눕니다.
- 예를 들어 N 이 17, K 가 4라고 가정합니다. 이때 1번의 과정을 한 번 수행하면 N 은 16이 됩니다. 이후에 2번의 과정을 두 번 수행하면 N 은 1이 됩니다. 결과적으로 이 경우 전체 과정을 실행한 횟수는 3이 됩니다. 이는 N 을 1로 만드는 최소 횟수입니다.
- N 과 K 가 주어질 때 N 이 1이 될 때까지 1번 혹은 2번의 과정을 수행해야 하는 최소 횟수를 구하는 프로그램을 작성하세요.

$N = 17$

$K = 4$



〈문제〉 1이 될 때까지: 답안 예시 (Python)

```
# N, K을 공백을 기준으로 구분하여 입력 받기
n, k = map(int, input().split())

result = 0

while True:
    # N이 K로 나누어 떨어지는 수가 될 때까지 빼기
    target = (n // k) * k
    result += (n - target)
    n = target
    # N이 K보다 작을 때 (더 이상 나눌 수 없을 때) 반복문 탈출
    if n < k:
        break
    # K로 나누기
    result += 1
    n //= k

# 마지막으로 남은 수에 대하여 1씩 빼기
result += (n - 1)
print(result)
```


〈문제〉 곱하기 혹은 더하기: 문제 설명

- 각 자리가 숫자(0부터 9)로만 이루어진 문자열 S 가 주어졌을 때, 왼쪽부터 오른쪽으로 하나씩 모든 숫자를 확인하며 숫자 사이에 '×' 혹은 '+' 연산자를 넣어 결과적으로 만들어질 수 있는 가장 큰 수를 구하는 프로그램을 작성하세요. 단, +보다 ×를 먼저 계산하는 일반적인 방식과는 달리, 모든 연산은 왼쪽에서부터 순서대로 이루어진다고 가정합니다.
- 예를 들어 02984라는 문자열로 만들 수 있는 가장 큰 수는 $(((((0 + 2) \times 9) \times 8) \times 4) = 576$ 입니다. 또한 만들어질 수 있는 가장 큰 수는 항상 20억 이하의 정수가 되도록 입력이 주어집니다.

〈문제〉 곱하기 혹은 더하기: 문제 해결 아이디어

- 대부분의 경우 '+'보다는 '×'가 더 값을 크게 만듭니다.
 - 예를 들어 $5 + 6 = 11$ 이고, $5 \times 6 = 30$ 입니다.
- 다만 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기를 수행하는 것이 효율적입니다.
- 따라서 두 수에 대하여 연산을 수행할 때, 두 수 중에서 하나라도 1 이하인 경우에는 더하며, 두 수가 모두 2 이상인 경우에는 곱하면 정답입니다.

〈문제〉 곱하기 혹은 더하기: 답안 예시 (Python)

```
data = input()

# 첫 번째 문자를 숫자로 변경하여 대입
result = int(data[0])

for i in range(1, len(data)):
    # 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기 수행
    num = int(data[i])
    if num <= 1 or result <= 1:
        result += num
    else:
        result *= num

print(result)
```

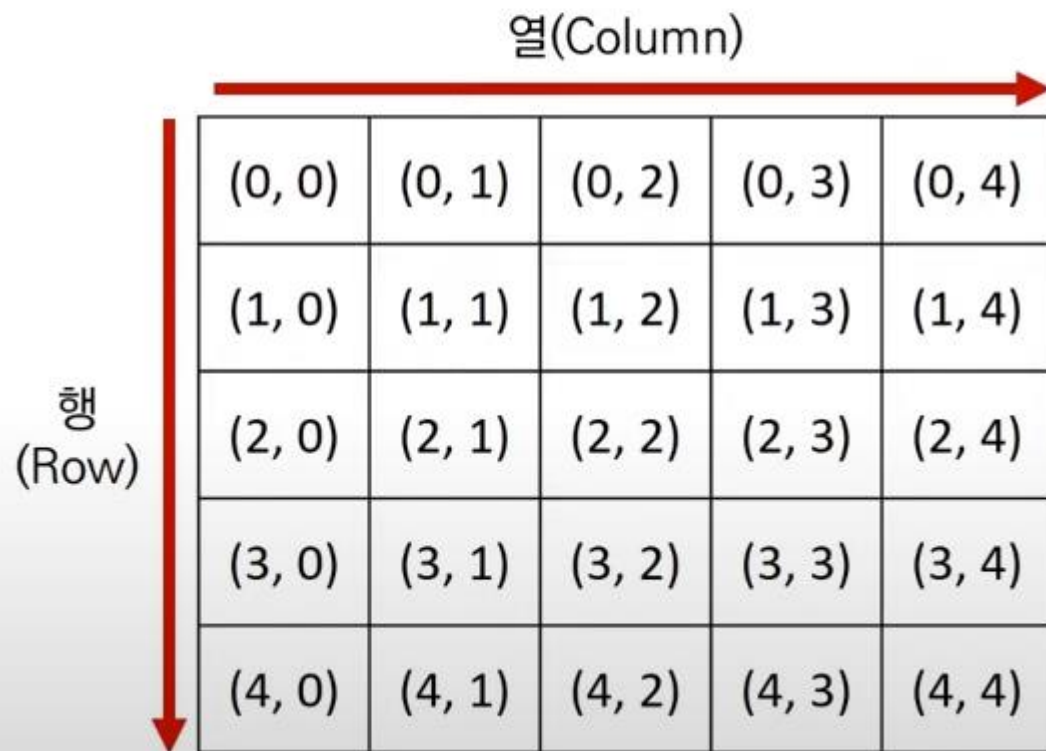
구현(Implementation)

- 구현이란, 머릿속에 있는 알고리즘을 소스코드로 바꾸는 과정입니다.



구현(Implementation)

- 일반적으로 알고리즘 문제에서의 2차원 공간은 행렬(Matrix)의 의미로 사용됩니다.



The diagram shows a 5x5 matrix with rows and columns indexed from 0 to 4. A horizontal red arrow above the matrix points to the right, labeled '열(Column)'. A vertical red arrow to the left of the matrix points downwards, labeled '행 (Row)'.


(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)

```
for i in range(5):  
    for j in range(5):  
        print('(', i, ', ', j, ')', end=' ')  
    print()
```

구현(Implementation)

- 시뮬레이션 및 완전 탐색 문제에서는 2차원 공간에서의 **방향 벡터**가 자주 활용됩니다.

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)



```
# 동, 북, 서, 남  
dx = [0, -1, 0, 1]  
dy = [1, 0, -1, 0]
```

```
# 현재 위치  
x, y = 2, 2
```

```
for i in range(4):  
    # 다음 위치  
    nx = x + dx[i]  
    ny = y + dy[i]  
    print(nx, ny)
```

〈문제〉 상하좌우: 문제 설명

- 여행가 A는 $N \times N$ 크기의 정사각형 공간 위에 서 있습니다. 이 공간은 1×1 크기의 정사각형으로 나누어져 있습니다. 가장 왼쪽 위 좌표는 (1, 1)이며, 가장 오른쪽 아래 좌표는 (N, N)에 해당합니다. 여행가 A는 상, 하, 좌, 우 방향으로 이동할 수 있으며, 시작 좌표는 항상 (1, 1)입니다. 우리 앞에는 여행가 A가 이동할 계획이 적힌 계획서가 놓여 있습니다.
- 계획서에는 하나의 줄에 띄어쓰기를 기준으로 하여 L, R, U, D 중 하나의 문자가 반복적으로 적혀 있습니다. 각 문자의 의미는 다음과 같습니다.
 - L: 왼쪽으로 한 칸 이동
 - R: 오른쪽으로 한 칸 이동
 - U: 위로 한 칸 이동
 - D: 아래로 한 칸 이동

〈문제〉 상하좌우: 답안 예시 (Python)

```
# N 입력 받기
n = int(input())
x, y = 1, 1
plans = input().split()

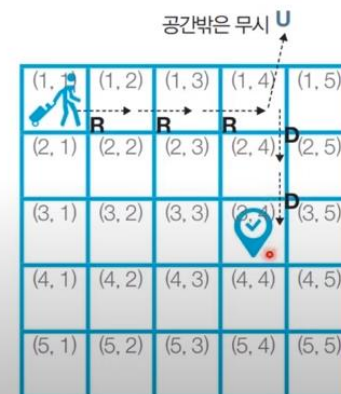
# L, R, U, D에 따른 이동 방향
dx = [0, 0, -1, 1]
dy = [-1, 1, 0, 0]
move_types = ['L', 'R', 'U', 'D']

# 이동 계획을 하나씩 확인하기
for plan in plans:
    # 이동 후 좌표 구하기
    for i in range(len(move_types)):
        if plan == move_types[i]:
            nx = x + dx[i]
            ny = y + dy[i]
            # 공간을 벗어나는 경우 무시
            if nx < 1 or ny < 1 or nx > n or ny > n:
                continue
            # 이동 수행
            x, y = nx, ny

print(x, y)
```

〈문제〉 상하좌우: 문제 설명

- 이때 여행가 A가 $N \times N$ 크기의 정사각형 공간을 벗어나는 움직임은 무시됩니다. 예를 들어 (1, 1)의 위치에서 L 혹은 U를 만나면 무시됩니다. 다음은 $N = 5$ 인 지도와 계획서입니다.



〈문제〉 시각: 문제 설명

- 정수 N이 입력되면 00시 00분 00초부터 N시 59분 59초까지의 모든 시각 중에서 3이 하나라도 포함되는 모든 경우의 수를 구하는 프로그램을 작성하세요. 예를 들어 1을 입력했을 때 다음은 3이 하나라도 포함되어 있으므로 세어야 하는 시각입니다.
 - 00시 00분 03초
 - 00시 13분 30초
- 반면에 다음은 3이 하나도 포함되어 있지 않으므로 세면 안 되는 시각입니다.
 - 00시 02분 55초
 - 01시 27분 45초

〈문제〉 시각: 문제 해결 아이디어

- 이 문제는 가능한 모든 시각의 경우를 하나씩 모두 세서 풀 수 있는 문제입니다.
- 하루는 86,400초이므로, 00시 00분 00초부터 23시 59분 59초까지의 모든 경우는 86,400가지 입니다.
 - $24 * 60 * 60 = 86,400$
- 따라서 단순히 시각을 1씩 증가시키면서 3이 하나라도 포함되어 있는지를 확인하면 됩니다.
- 이러한 유형은 완전 탐색(Brute Forcing) 문제 유형이라고 불립니다.
 - 가능한 경우의 수를 모두 검사해보는 탐색 방법을 의미합니다.

〈문제〉 시각: 답안 예시 (Python)

```
# H 입력 받기
h = int(input())

count = 0
for i in range(h + 1):
    for j in range(60):
        for k in range(60):
            # 매 시각 안에 '3'이 포함되어 있다면
            if '3' in str(i) + str(j) + str(k):
                count += 1

print(count)
```

```
h = int(input())

count = 0
for i in range(h+1):
    for j in range(60):
        for k in range(60):
            if '3' in str(i) + str(j) + str(k):
                count += 1
            print(str(i) + str(j) + str(k))

print(count)
```

```
1
003
0013
0023
0030
0031
0032
0033
0034
0035
0036
0037
0038
0039
0043
0053
013
0113
0123
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0143
```