알고리즘 스터디 7주차

- 백트랙킹 -

발표자 장수현

백준 14888 연산자 끼워넣기 https://www.acmicpc.net/problem/14888

문제

N개의 수로 이루어진 수열 A_1, A_2, ..., A_N이 주어진다. 또, 수와 수 사이에 끼워넣을 수 있는 N-1개의 연산자가 주어진다. 연산자는 덧셈(+), 뺄셈(-), 곱셈(×), 나눗셈(÷)으로만 이루어져 있다.

우리는 수와 수 사이에 연산자를 하나씩 넣어서, 수식을 하나 만들 수 있다. 이때, 주어진 수의 순서를 바꾸면 안 된다.

예를 들어, 6개의 수로 이루어진 수열이 1, 2, 3, 4, 5, 6이고, 주어진 연산자가 덧셈(+) 2개, 뺄셈(-) 1개, 곱셈(×) 1개, 나눗셈(÷) 1개인 경우에는 총 60가지의 식을 만들 수 있다. 예를 들어, 아래와 같은 식을 만들 수 있다.

- 1+2+3-4×5÷6
- 1÷2+3+4-5×6
- 1+2÷3×4-5+6
- 1÷2×3-4+5+6

식의 계산은 연산자 우선 순위를 무시하고 앞에서부터 진행해야 한다. 또, 나눗셈은 정수 나눗셈으로 몫만 취한다. 음수를 양수로 나눌 때는 C++14의 기준을 따른다. 즉, 양수로 바꾼 뒤 몫을 취하고, 그 몫을 음수로 바꾼 것과 같다. 이에 따라서, 위의 식 4개의 결과를 계산해보면 아래와 같다.

- 1+2+3-4×5÷6 = 1
- 1÷2+3+4-5×6 = 12
- 1+2÷3×4-5+6 = 5
- 1÷2×3-4+5+6 = 7

N개의 수와 N-1개의 연산자가 주어졌을 때, 만들 수 있는 식의 결과가 최대인 것과 최소인 것을 구하는 프로그램을 작성하시오.

접근방법

1. 주어진 연산자를 모두 사용하여 수식의 결과값을 구해야한다

   ➡️ 완전 탐색

2. 수식의 계산하는 과정이 반복된다

   ➡️ 재귀함수

3. 모든 숫자를 탐색하면 빠져나온다 (재귀함수의 깊이 = 숫자 리스트의 위치)

   ➡️ 재귀함수의 탈출 조건

입력받기

예제 입력 1 복사

```
2
5 6
0 0 1 0
```

예제 출력 1 복사

```
30
30
```

```python
n = int(input()) # 수의 개수
nums = list(map(int, input().split())) # n개의 수
op = list(map(int, input().split())) # 덧셈, 뺄셈, 곱셈, 나눗셈의 개수
ans_max, ans_min = -1e9, 1e9 # 최대값, 최소값
```

풀이

depth = 재귀함수의 깊이, 계산할 숫자 리스트 위치

result = 현재까지 계산한 수식의 값

plus, minus, multiply, divide = 남은 덧셈, 뺄샘, 곱셈, 나눗셈의 개수

```python
def solve(depth, result, plus, minus, multiply, divide):
    global ans_max, ans_min

    # 모든 숫자를 탐색했다면 최대값, 최소값을 구하고 리턴
    if depth == n:
        ans_max = max(ans_max, result)
        ans_min = min(ans_min, result)
        return

    # 연산자의 개수에 따른 재귀함수 호출
    if plus:
        solve(depth + 1, result + nums[depth], plus - 1, minus, multiply, divide)
    if minus:
        solve(depth + 1, result - nums[depth], plus, minus - 1, multiply, divide)
    if multiply:
        solve(depth + 1, result * nums[depth], plus, minus, multiply - 1, divide)
    if divide:
        solve(depth + 1, int(result / nums[depth]), plus, minus, multiply, divide - 1)
```

“ 또, 나눗셈은 정수 나눗셈으로 몫만 취한다. 음수를 양수로 나눌 때는 C++14의 기준을 따른다. 즉, 양수로 바꾼 뒤 몫을 취하고, 그 몫을 음수로 바꾼 것과 같다. ”

= -(-result // nums[depth])

= int(result / nums[depth])

전체코드

```python
n = int(input()) # 수의 개수
nums = list(map(int, input().split())) # n개의 수
op = list(map(int, input().split())) # 덧셈, 뺄셈, 곱셈, 나눗셈의 개수
ans_max, ans_min = -1e9, 1e9 # 최대값, 최소값

def solve(depth, result, plus, minus, multiply, divide):
    global ans_max, ans_min

    # 모든 숫자를 탐색했다면 최대값, 최소값을 구하고 리턴
    if depth == n:
        ans_max = max(ans_max, result)
        ans_min = min(ans_min, result)
        return

    # 연산자의 개수에 따른 재귀함수 호출
    if plus:
        solve(depth + 1, result + nums[depth], plus - 1, minus, multiply, divide)
    if minus:
        solve(depth + 1, result - nums[depth], plus, minus - 1, multiply, divide)
    if multiply:
        solve(depth + 1, result * nums[depth], plus, minus, multiply - 1, divide)
    if divide:
        solve(depth + 1, int(result / nums[depth]), plus, minus, multiply, divide - 1)

solve(1, nums[0], op[0], op[1], op[2], op[3])
print(ans_max)
print(ans_min)
```

백준 2580 스도쿠

https://www.acmicpc.net/problem/2580

문제

스도쿠는 18세기 스위스 수학자가 만든 '라틴 사각형'이랑 퍼즐에서 유래한 것으로 현재 많은 인기를 누리고 있다. 이 게임은 아래 그림과 같이 가로, 세로 각각 9개씩 총 81개의 작은 칸으로 이루어진 정사각형 판 위에서 이뤄지는데, 게임 시작 전 일부 칸에는 1부터 9까지의 숫자 중 하나가 쓰여 있다.

	3	5	4	6	9	2	7	8
7	8	2	1		5	6		9
	6		2	7	8	1	3	5
3	2	1		4	6	8	9	7
8		4	9	1	3	5		6
5	9	6	8	2		4	1	3
9	1	7	6	5	2		8	
6		3	7		1	9	5	2
2	5	8	3	9	4	7	6	

나머지 빈 칸을 채우는 방식은 다음과 같다.

1. 각각의 가로줄과 세로줄에는 1부터 9까지의 숫자가 한 번씩만 나타나야 한다.
2. 굵은 선으로 구분되어 있는 3x3 정사각형 안에도 1부터 9까지의 숫자가 한 번씩만 나타나야 한다.

위의 예의 경우, 첫째 줄에는 1을 제외한 나머지 2부터 9까지의 숫자들이 이미 나타나 있으므로 첫째 줄 빈칸에는 1이 들어가야 한다.

1	3	5	4	6	9	2	7	8

또한 위쪽 가운데 위치한 3x3 정사각형의 경우에는 3을 제외한 나머지 숫자들이 이미 쓰여있으므로 가운데 빈 칸에는 3이 들어가야 한다.

4	6	9
1	3	5
2	7	8

접근방법

1. 빈칸에 모든 숫자를 넣어보면서 스도쿠가 완성되는 경우를 찾아야한다

   ➡️ 완전 탐색

2. 빈칸에 숫자를 넣을 수 있는지 확인하고 넣는 과정이 반복된다

   ➡️ 재귀함수

3. 빈칸이 모두 채워지면 빠져나온다 (재귀함수의 깊이 = 빈칸 리스트의 위치)

   ➡️ 재귀함수의 탈출 조건

입력받기

예제 입력 1 복사

```
0 3 5 4 6 9 2 7 8
7 8 2 1 0 5 6 0 9
0 6 0 2 7 8 1 3 5
3 2 1 0 4 6 8 9 7
8 0 4 9 1 3 5 0 6
5 9 6 8 2 0 4 1 3
9 1 7 6 5 2 0 8 0
6 0 3 7 0 1 9 5 2
2 5 8 3 9 4 7 6 0
```

예제 출력 1 복사

```
1 3 5 4 6 9 2 7 8
7 8 2 1 3 5 6 4 9
4 6 9 2 7 8 1 3 5
3 2 1 5 4 6 8 9 7
8 7 4 9 1 3 5 2 6
5 9 6 8 2 7 4 1 3
9 1 7 6 5 2 3 8 4
6 4 3 7 8 1 9 5 2
2 5 8 3 9 4 7 6 1
```

```python
sudoku = [list(map(int, input().split())) for _ in range(9)]
blank = [(i, j) for i in range(9) for j in range(9) if sudoku[i][j] == 0] # 스도쿠의 빈칸 리스트
```

풀이

```python
# 행에 들어갈 수 있는지 체크
def check_row(x, k):
    for i in range(9):
        if sudoku[x][i] == k:
            return False
    return True

# 열에 들어갈 수 있는지 체크
def check_col(y, k):
    for i in range(9):
        if sudoku[i][y] == k:
            return False
    return True
```

```python
# 정사각형에 들어갈 수 있는지 체크
def check_rect(x, y, k):
    nx = x // 3 * 3
    ny = y // 3 * 3

    for i in range(3):
        for j in range(3):
            if sudoku[nx+i][ny+j] == k:
                return False
    return True
```

x = 행의 위치

y = 열의 위치

k = 확인할 숫자

nx, ny = 정사각형의 가장 왼쪽 위 위치

풀이

idx = 재귀함수의 깊이, 빈칸의

"스도쿠 판을 채우는 방법이 여럿인 경우는 그 중 하나만을 출력한다."

= exit(0) ⭕ return ❌

```
def dfs(idx):
    # 빈칸을 다 채웠다면 출력하고 종료
    if idx == len(blank):
        for row in sudoku: print(*row)
        exit(0)

    x, y = blank[idx] # 빈칸의 위치
    for num in range(1, 10):
        # 빈칸에 넣을 수 있는 숫자인 경우 다음 빈칸으로 넘어간다
        if check_row(x, num) and check_col(y, num) and check_rect(x, y, num):
            sudoku[x][y] = num
            dfs(idx + 1)
            sudoku[x][y] = 0 # 초기화
```

전체코드

```python
sudoku = [list(map(int, input().split())) for _ in range(9)]
blank = [(i, j) for i in range(9) for j in range(9) if sudoku[i][j] == 0] # 스도쿠의 빈칸 리스트

# 행에 들어갈 수 있는지 체크
def check_row(x, k):
    for i in range(9):
        if sudoku[x][i] == k:
            return False
    return True

# 열에 들어갈 수 있는지 체크
def check_col(y, k):
    for i in range(9):
        if sudoku[i][y] == k:
            return False
    return True

# 정사각형에 들어갈 수 있는지 체크
def check_rect(x, y, k):
    nx = x // 3 * 3
    ny = y // 3 * 3

    for i in range(3):
        for j in range(3):
            if sudoku[nx+i][ny+j] == k:
                return False
    return True
```

```python
def dfs(idx):
    # 빈칸을 다 채웠다면 출력하고 종료
    if idx == len(blank):
        for row in sudoku: print(*row)
        exit(0)

    x, y = blank[idx] # 빈칸의 위치
    for num in range(1, 10):
        # 빈칸에 넣을 수 있는 숫자인 경우 다음 빈칸으로 넘어간다
        if check_row(x, num) and check_col(y, num) and check_rect(x, y, num):
            sudoku[x][y] = num
            dfs(idx + 1)
            sudoku[x][y] = 0 # 초기화

dfs(0)
```

공통문제

- 백준 15684 사다리 조작
- 백준 1038 감소하는 수
- 백준 1062 가르침