# 1주차 문제 리뷰

| | |
|---|---|
| 생성일 | @2025년 7월 10일 오후 2:44 |
| 태그 | |

짧은 리뷰 / 대체 리뷰 예시

## #1018번

### 류하경

**소요 시간:**

**리뷰:**

### 박재현

**소요 시간: 30 min**

**리뷰:**

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int countCheck (int colStart, int rowStart, char cmp, const vector<string>& p);

int main() {
    int depth, width;
    cin >> depth >> width;

    vector<string> p(depth);

    for (int i = 0; i < depth; i++) {
        cin >> p[i];
    }
```

```
    int count = 64;


    for (int i = 0; i < depth - 7 ; i++){
        for( int j = 0; j < width - 7; j++ ){
            int count1 = countCheck( i,j, 'W', p);
            int count2 = countCheck( i,j, 'B', p);
            int TempCount = min(count1,count2   );
            count = min (count, TempCount);
        }
    }


    cout << count;

}



int countCheck (int colStart, int rowStart, char cmp, const vector<string>& p)
{
    int count = 0;

    for (int i = colStart; i < colStart + 8; i++) {
        for ( int j = rowStart; j < rowStart + 8; j++){

            if (i % 2 == 0 ){
                if ( j % 2 != 0 && p[i][j] == cmp) count++;

                else if ( j % 2 == 0 && p[i][j] != cmp) count++;
            }

            else {
                if ( j % 2 != 0 && p[i][j] != cmp) count++;

                else if ( j % 2 == 0 && p[i][j] == cmp) count++;
```

```
            }
        }
    }
    return count;
}
```

1. 2차원 배열로 구현하려 했으나, 사용중인 XCode가 2차원 배열 코드에서 계속 문제를 일으켜,

어쩔 수 없이 벡터를 만든후, 벡터 각 칸에 스트링을 넣는 방식으로 우회적으로 체스판을 받을 공간을 구현했습니다.

2. 그 후에 countChecke 라는 함수를 만들었는데, 이는 우리가 시작지점을 잡은 곳으로부터 8x8의 체스판을 확보한 후에,

그 안에 있는 모든 칸들을 전부 순회하면서 고쳐야 할 칸들을 세는 것으로 구현했습니다. 저는 짝홀 기준 행과 열로 나누어 계산했는데, 다 구하고 gpt한테 물어보니, 행과열의 합의 짝, 홀 여부로 더 쉽게 한꺼번에 처리가 가능하다는 답을 들었는데, 해당 방안이 훨씬 더 쉽겠구나 라고 생각되었습니다. 이 경우엔,

ㄱ. i + j 가 홀수이고 p[i][j] = = cmp인 경우

ㄴ. i + j 가 짝수이고 p[i][j] ≠ cmp인 경우

단 두가지로 경우아 나뉘어 더 효율적임을 알 수 있습니다.

3. 그렇게 각 경우 바뀌어야 하는 경우의수를 구한다음에, 최댓값인 64와 비교하면서 최솟값을 누적비교해 나가게 두었습니다.

## 안준수

**소요 시간:** 30분 고민 후 답참고

**리뷰:**

## 이동원

**소요 시간:** 2시간

**리뷰:**

```
def count_repaints(board, start_row, start_col):
    repaint_W_start = 0  # W로 시작
    repaint_B_start = 0  # B로 시작
```

```
    for i in range(8):
        for j in range(8):
            current = board[start_row + i][start_col + j]
            if (i + j) % 2 == 0:
                if current != 'W':
                    repaint_W_start += 1
                if current != 'B':
                    repaint_B_start += 1
            else:
                if current != 'B':
                    repaint_W_start += 1
                if current != 'W':
                    repaint_B_start += 1

    return min(repaint_W_start, repaint_B_start)

N, M = map(int, input().split())
board = [input() for _ in range(N)]

min_repaint = float('inf')

# 모든 가능한 8x8 구간에 대해 확인
for i in range(N - 7):
    for j in range(M - 7):
        repaints = count_repaints(board, i, j)
        min_repaint = min(min_repaint, repaints)

print(min_repaint)
```

1. 처음 생각했던 스터디와 다르게 바로 문제 풀이를 들어간다는 점에서 당황했지만 최우선적으로 브루트포스에 대한 개념 복습을 우선적으로 진행 (약 1시간 정도)
2. 문제를 읽고 예제 입력과 출력을 보고 어떤 방식으로 입력 후 출력이 되는 지에 대해서 분석 진행
3. 체스판의 크기를 고정하고 전체 보드에서 가능한 모든 영역을 전부 순회하도록 코드 작성
4. 각 후보에 대해 다시 칠해야 하는 칸 수 계산

5. 그 중 최소값을 정답으로 출력하는 방식 설정
6. 결론 : 각 주차 주제에 대한 복습 진행 후 간단하게 쉬운 파이썬 문제를 병행하면서 숙제를 진행해야 된다는 것을 판단함 (단순 개념만 학습 후 실제 코딩을 진행한 지 오래 되어 어려움을 많이 느낌)

## 이현진

**소요 시간:**

```
def repaint_count(si, sj, board):
    """
    (si, sj)를 왼쪽 위 코너로 하는 8*8 블록을 두 가지 체스 패턴과 비교하여 최소 재칠 횟수
    """
    cnt1 = 0 #기준 A: (0, 0)이 'W'
    cnt2 = 0 #기준 B: (0, 0)이 'B'
    for dx in range(8):
        for dy in range(8):
            #기대 색상 계산: (dx + dy) 짝수면 시작색, 홀수면 반대색
            expected1 = 'W' if (dx + dy) % 2 == 0 else 'B'
            expected2 = 'B' if (dx + dy) %2 == 0 else 'W'
            actual = board[si + dx][sj + dy]
            #실제 색상이 패턴 A의 기대와 다르면 cnt1 증가
            if actual != expected1:
                cnt1 += 1
            #실제 색상이 패턴 B의 기대와 다르면 cnt2 증가
            if actual != expected2:
                cnt2 += 1

            # 두 패턴 중 더 적게 칠해야 하는 경우를 반환
    return min(cnt1, cnt2)

def main():
    import sys
    input = sys.stdin.readline  # 빠른 입력을 위해 readline 사용

    # n(행 수), m(열 수) 입력
    n, m = map(int, input().split())
```

```
    # 보드 정보를 문자열 리스트로 저장
    board = [input().rstrip() for _ in range(n)]

    ans = float('inf')  # 최소값을 찾기 위해 무한대로 초기화

    # 가능한 모든 8×8 블록의 시작 위치를 순회
    for i in range(n - 7):          #(0부터 n-8까지)
        for j in range(m - 7):      #(0부터 m-8까지)
            # 현재 블록의 칠해야 하는 최소 횟수 계산
            cnt = repaint_count(i, j, board)
            # 최소값 갱신
            if cnt < ans:
                ans = cnt

    # 결과 출력

    print(ans)

    if __name__ == "__main__":
        main()
```

**리뷰:**

- `repaint_count(si, sj, board)` **함수**
  - `(si, sj)`를 왼쪽 위로 하는 8×8 영역을 두 가지 체스판 패턴과 비교.
  - `expected1`, `expected2` 로 각각 “W 시작” 패턴과 “B 시작” 패턴의 기대값을 계산.
  - 실제 색(`actual`)과 비교해 다르면 카운트 증가.
  - 두 패턴 중 더 작은 재칠 횟수를 반환.
- `main()` **함수**
  - `sys.stdin.readline` 으로 빠르게 입력을 읽음.
  - `n, m` 크기의 보드를 리스트로 저장.
  - 모든 가능한 8×8 블록의 시작 좌표 `(i, j)`에 대해 `repaint_count` 호출.
  - 최솟값을 갱신하고 최종적으로 출력.

## 최수철

**소요 시간:**

**리뷰:**

#1018

-체스판 문제 에서 for문 등의 반복문을 돌 때 범위를 넘어가는 것을 애초에 막기 위해for문 설정에서 끝 항이 n이 아닌 n-8 등으로 만드는 것 익숙해지기

-시작 point 설정할 때 ?:조건부 연산 사용해보기/코드 간략화 및 시각화 용이함.

예시:char expected1 = ((x + y) % 2 == 0) ? 'W' : 'B'; 등

## 김주영

```
import java.util.Scanner;
class BOJ1018 {
    public static int getSolution(int startRow, int startCol, String[] board) {
        String[] orgBoard = { "WBWBWBWB", "BWBWBWBW" };
        int whiteSol = 0;
        for (int i = 0; i < 8; i++) {
            int row = startRow + i;
            for (int j = 0; j < 8; j++) {
                int col = startCol + j;
                if (board[row].charAt(col) != orgBoard[row % 2].charAt(j)) whiteSol+
            }
        }
        return Math.min(whiteSol, 64 - whiteSol);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int row = sc.nextInt();
        int col = sc.nextInt();
        sc.nextLine();
        String[] board = new String[row];
        for (int i = 0; i < row; i++) board[i] = sc.nextLine();
        int sol = Integer.MAX_VALUE;
        for (int i = 0; i <= row - 8; i++) {
            for (int j = 0; j <= col - 8; j++) {
                int curSol = getSolution(i, j, board);
                if (sol > curSol) sol = curSol;
            }
```

```
        }
        System.out.println(sol);
        sc.close();
    }
}
```

**소요 시간 : 30분**

**리뷰**

- getsoultion() : 특정 8×8 영역의 최소 칠하기 횟수 계산
- main() : 전체 영역 순회 및 최솟값 탐색

1. 모든 8x8  체스판 영역을 완전 탐색 검사(브루트포스)
2. W로 시작하는 체스판 패턴과 몇 칸 다른지 비교하고, B로 시작하는 경우는 64에서 빼서 계산

   (몇 칸 다른지 whiteSol 세고,나머지는 **64 - whiteSol** 로 바로 계산 가능)
3. 모든 경우 중 가장 적게 칠해야 하는 칸 수를 출력

# #1436번

## 류하경

**소요 시간:** 30분

**리뷰:** 브루트포스 알고리즘 문제라는 걸 알았지만, 패턴이 있는 것 같아 처음에는 패턴을 찾아봤다. 결국엔 단순히 숫자를 하나씩 더하며 조건문을 거치는 코드를 작성해 내봤는데, 생각보다 코드도 단순하고 런타임 에러도 나지 않았다. 브루트포스 알고리즘이 어떤 건지 경험적으로 알게 된 것 같다.

## 박재현

**소요 시간: 1h**

**리뷰:**

```
#include <iostream>
using namespace std;
```

```
bool hasThreeConsecutiveSix(int num) {
    int count = 0;

    while(num != 0) {
        if ( num % 10 == 6 ){
            count++;
            if ( count == 3) {
                return true;
            }
        }

        else count = 0;

        num /= 10;
    }

    return false;
}
ㅅㅇ
int main() {
    int n;
    cin >> n;

    int number = 666;
    int count = 0;

    while(true)
    {
        if (hasThreeConsecutiveSix(number)){
            count++;
        }

        if (count == n) break;

        number++;

    }
```

```
    cout << number;

    return 0;
}
```

1. 처음에는 브루트포스라는 단원명을 고려안하고 안에 나름 규칙이 있을것이라 생각하고 풀이를 해 보았으나,

패턴화 하기엔 너무 복잡한 양상에, 포기하고 666부터 숫자를 하나하나 늘려가고 거기서 나오는 숫자들이 666이 연속하는 것이 과연 들어있는지, 확인하는 것으로 방향을 바꿨습니다.

2. 거기서 3연속숫자라는 함수를 만들었는데, 6이 연속 3번 카운트되면  true를 리턴하고 그걸로 count를 늘려서 갯수를 세는 것으로 갔습니다. 6이 연속되지 않으면 count = 0으로 초기화되게 두었고, 숫자를 10으로 계속 나눈 몫을 대상으로 계산하기에, 일의 자리 수를 하나하나 지워가면서 모든 자리의 수를 다 판단하는 로직을 작성할 수 있었습니다.

## 안준수

**소요 시간:**  40분

**리뷰:**

작은 수부터 시작해 666이라는 문자열이 포함되어 있는지 확인하고, 포함되어 있다면 카운트 증가시키며, N번째가 되면 출력하게 만들었습니다.

## 이동원

**소요 시간:** 1시간 30분

**리뷰:**

```
n = int(input())
count = 0
num = 666

# 숫자를 문자열로 변환해서 "666" 포함 여부 확인
while True:
    # 문자열에 "666"이 포함되어 있는 지 확인
```

```
    if '666' in str(num):
        count += 1
        # n번째 종말의 수에 도달했는 지 확인
        if count == n:
            print(num)
            break
    # 숫자를 하나씩 증가시키면서 브루트포스 방식으로 탐색
    num += 1
```

1. 핵심 아이디어 : 자연수 1부터 하나씩 증가시키면서 “666”이 포함되어 있는 지 문자열로 검사 후 N번째 종말의 수가 나올 때까지 하나씩 세면서 찾는 브루트포스 패턴 파악
2. 숫자 변수를 1씩 증가시키면서 “666”이 포함되어 있는 지 확인 후 있을 경우 count를 1씩 증가시키면서 count == N이 되는 순간 탐색을 멈추고 숫자 변수를 출력하는 방식으로 구현
3. 개념 학습을 진행했었던 경험이 있었기에 알고리즘 및 구현 방식을 구상하는 데에는 많은 시간을 소요하지 않지만 기초 파이썬 코딩 지식 및 경험이 부족한 지 해당 부분에서 오랜 시간 소요

## 이현진

**소요 시간:** 40분

**리뷰:** 1. c++로 할 때 모두 문자열로 생각해서 6 && 6 && 6을 검사하는 식으로 풀었다. (6의 연속성)

```
if (num[i] == '6' && num[i + 1] == '6' && num[i + 2] == '6') { cnt++; break; }
```

2. python이 아직 익숙치 않은데, 더 간결하기도 하고, 어렵기도 한 것 같다.

   위의 코드를 파이썬으로 바꾸면

   ```
   if '666' in str(num):
           count += 1
   ```

   이게 끝이다.  짧고 간결하다.

   그 다음에 배운 건 파이썬 입출력 속도 높이기

   들어가는 입출력이 많을 때,

c++에서는 ios_base::sync_with_stdio(false);  cin.tie(0); cout.tie(0);를 적어주는데,
파이썬 에서도 대용량 입력시 필요한 메소드가 있고 패키지는 sys라고 한다.

```
import sys
 n = int(sys.stdin.readline())   #\n까지 포함하기 때문에 몇줄을 받아올 수 있다.
                                 #EOF 시 ""를 반환한다.
```

### 최수철

**소요 시간:** 30분 고민 후 답 참고

**리뷰:**

#1436

-1씩 증가 시키며 찾아야 한다는 알고리즘은 생각했음
그러나 String헤더파일 기능을 몰라 답지 참고하여 살짝 허탈하였음. 이를 알고 있을 때 너무 쉽게 풀리는 문제
To find, string::npos 라는 기능 알고 있기
-“666”을 찾아야 하는데 ‘666’을 찾는 거로 하여 오류가 나왔다. 웬만하면 큰따움표를 쓰기!

## #2839번

### 류하경

**소요 시간:**

**리뷰:**

### 박재현

**소요 시간: 40분**

**리뷰:**

```cpp
#include <iostream>
#include <vector>
using namespace std;

int ThreeMaker (int numFromFive, int weight);
int judge(int weight, int numberFive);

int main() {
    int weight;
    cin >> weight;


    int numOfFive;
    numOfFive = weight / 5;
    int answer = judge(weight, numOfFive); // 5를 일단 최대치로 묶어보고 보냅니다.

    cout << answer;

}



int judge(int weight, int numberFive)
{
    int answer = -1;
    int numberThree = (weight - 5 * numberFive) / 3;
    if (weight - 5 * numberFive - 3 * numberThree != 0 ){

        if (numberFive == 0) return -1; // 5가 이미 0인데 합도 못 만들면 그건 불능인것
        else{
            numberFive--; // 5로 묶고 나머지를 3으로 처리가 안되면, 5를 하나더 풀어서 다
            return judge(weight, numberFive);
        }
    }
    else {
        answer = numberFive + numberThree;
    }
```

```
    return answer;
}
```

케이스가 굉장히 복잡할것 같았으나 막상 나누고 나니 생각보단 간단했습니다.

1. 5로 일단 최대한 담아보고 judge함수에 전체 무게와 5로 나눠본 봉지수를 보냅니다.
2. 5와 3으로 합을 만들어 낼 수 없는 경우 두 가지 경우로 나누는데,

ㄱ. 5로 애초에 1봉지도 못만드는 경우 → 이제 3으로도 못만드니까 이건 불능입니다.

ㄴ. 5로 1봉지 이상 만든 경우 → 이제 5의 봉지르 하나씩 풀면서 3으로 처리해볼 수 있는지를 보아야 합니다.

따라서 이건 5의 봉지수를 하나 줄이고, 재귀적함수로 다시 judge에 넣습니다. 그러면 줄어든 5의 봉지에 해당되는 부분을

이제 3으로 처리 가능한지 보게 됩니다.

3. 5와 3으로 합을 만들어낼 수 있는경우 그대로 답이 됩니다.
4. 이러한 구조로 작성하였습니다!

## 안준수

**소요 시간:** 30~40분

**리뷰:** 5kg 봉지를 최대한 많이 사용하고, 남은 무게를 3kg 봉지로 나눌 수 있는지 확인하였습니다.

## 이동원

**소요 시간: 1시간 20~30분**

**리뷰:**

```
n = int(input())
min_bags = -1  # 기본값은 -1 (불가능한 경우)

# 5kg 봉지를 가능한 최대한 많이 쓰는 경우부터 줄여가면서 탐색하는 방식
for five_kg in range(n // 5, -1, -1):
    remaining = n - (five_kg * 5)
```

```
    # 남은 무게가 3kg으로 나누어 떨어지는 경우에만 유효한 조합
    if remaining % 3 == 0:
        three_kg = remaining // 3

        # 현재 조합에서 사용한 봉지 수의 총합 구하기
        min_bags = five_kg + three_kg

        # 가장 먼저 찾은 유효한 조합이 최소 봉지 수이기 때문에 바로 종료
        break
print(min_bags)
```

1. 전체 경우의 수를 전부 만드는 대신 최대한 많이 사용한 경우부터 차례대로 줄이면서 유효한 조합을 찾는 방식의 알고리즘 문제임을 파악
2. 조합의 수가 제한되어 있고 각 조합의 유효성을 검사하는 조건이 단순한 경우에는 코드 역시 짧은 코드를 나타날 수 있다는 것을 확인함

## 이현진

**소요 시간:** 1시간

**리뷰:** c++

- 반복문을 while로 시작했다면 더 가독성있는 코드로 바꿀 수 있지 않았을까 생각한다.
- -1같은 세부적인 정보를 놓쳤다.
- idea

1. 먼저 5kg 봉지를 최대한 사용합니다.
2. 남은 무게가 3kg로 나눠 떨어지면 바로 정답을 출력합니다.
3. 그렇지 않으면, 5kg 봉지 개수를 줄여가며 남은 무게가 3kg로 나눠 떨어지는 경우를 찾습니다.
4. 정확히 나눠 떨어지지 않으면 -1을 출력한다.

## 최수철

**소요 시간:** 3~40분

**리뷰:**

#2839

-이중 포문으로 i,j로 n=5*i+3*j설정후 접근했는데 다행히 답은 맞았지만 이중 포문은 n제곱으로 시간 복잡도에서 오랜 시간이 걸리는 거로 알고 있어서 다음엔 더 쉬운 풀이를 찾아봐야겠다 생각했다.

# #1316번

## 김세람

**소요 시간:**

**리뷰:**

## 김희재

**소요 시간:**

**리뷰:**

## 류하경

**소요 시간:**

**리뷰:**

## 박재현

**소요 시간: 1h 30min**

**리뷰:**

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

bool groupChecker ( string str );
```

```
int main() {
    int n;
    cin >> n;

    int countOfGroup = 0;

    string str;

    for (int i = 0 ; i < n ; i++ ){
        cin >> str;
        countOfGroup += groupChecker(str);
    }

    cout << countOfGroup;
}

bool groupChecker ( string str ){
    vector<int> v(26,-1);
    for (int i = 0; i < str.length() ; i++){

        if ( v[str[i] - 'a'] == -1 || i - v[str[i] - 'a'] <= 1 ){
            v[str[i] - 'a'] = i;
        }

        else return false;
    }

    return true;


}
```

문제는 간단한 데에 반해, 조건이 까다로워서 풀이가 쉽지 않아 시간이 오래 걸렸습니다.

1. 알파벳 소문자 갯수  만큼의 벡터를 -1로 초기화 후  할당합니다. 이 벡터는 문자열을 가져와 문자를 앞에서 부터 읽을 때, 알파벳 별로 가장 마지막에 나온 위치를 저장하기 위함입니다.

2. 문자열을 가져와 앞에서부터 읽는데,

    ㄱ. 해당 문자가 처음 나오는(벡터값 -1)경우

    ㄴ. 혹은 이전에 나왔던 경우라면 그 마지막 위치와 위치값 차이가 1이하인 경우 (떨어지지 않고 이웃함을 뜻함)

해당 지점의 위치 값을 벡터값으로 할당하는 함수 groupChecker를 만들어, 그룹단어 갯수를 세는 코드를 작성했습니다.

## 안준수

**소요 시간:**

**리뷰:**

## 이동원

**소요 시간:**

**리뷰:**

## 이현진

**소요 시간:**

**리뷰:**

## 최수철

**소요 시간:**

**리뷰:**

## 김주영

**소요시간** : 30분

**리뷰**

```java
import java.util.Scanner;

public class BOJ1436 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        int count = 0;
        int index = 666;
        while(N>count){
            String temp = String.valueOf(index);
            if(temp.contains("666")){
                count++;
            }
            index++;
        }
        System.out.println(index-1);
    }
}
```

1. 숫자 666부터 하나씩 증가시키며 "666"이 포함된 숫자를 찾는다.
2. 문자열로 변환한 뒤 `contains("666")`로 종말의 수 여부를 검사한다.
3. N번째 종말의 수를 찾으면 출력하고 종료한다.