2023 여름학기 | 동국대학교 SW 역량강화 캠프

8일차. 중간고사

문제

- 분해합

문제

어떤 자연수의 분해합은 그 자연수와 자연수를 이루는 각 자리수의 합이다. 예를 들어, 123의 분해합은 129(123+1+2+3)가 된다.

또, 어떤 자연수 M의 분해합이 N인 경우, M을 N의 생성자라 한다. 예를 들어 123은 129의 생성자가 된다.

자연수가 주어졌을 때, 그 자연수의 가장 작은 생성자를 구하는 프로그램을 작성해보자.

입력

첫째 줄에 자연수 N(1≤N≤1,000,000)이 주어진다.

출력

첫째 줄에 답을 출력한다. 생성자가 없는 경우에는 0을 출력한다.

풀이

분해합의 생성자를 찾는 것보다, 생성자의 분해합을 구하는 것이 더 쉽다.

또한, 생성자의 분해합은 (생성자 + 생성자의 각 자리 수의 합)이므로 항상 생성자 < 분해합을 만족한다.

즉, 입력으로 받은 N 이하의 모든 i에 대하여 i의 분해합을 구하고, i의 분해합이 N이면 i를 출력해주면 된다.

각 자리수의 합 구하는 방법:

10으로 나눈 나머지를 구해주면 일의 자리 값을 알 수 있다.
10으로 나눈 몫을 구해주면 일의 자리 값을 없앨 수 있다.

즉, 어떤 수 X의 각 자리 수를 알고 싶다면, 0이 될 때까지 10으로 나눠주면서 10으로 나눈 나머지를 더해주면 된다.

코드

```java
boolean flag=false;
for(int i=1; i<N; i++){
    int sum=0, j=i;
    while(j > 0){
        sum += j%10;
        j /= 10;
    }

    if(sum+i == N){
        System.out.print(i);
        flag = true;
        break;
    }
}

if(!flag) System.out.print(0);
```

```python
N = int(input())

flag = False
for i in range(N):
    j = i
    s = 0
    while j > 0:
        s += j%10
        j //= 10

    if s+i == N:
        print(i)
        flag = True
        break

if not flag: print(0)
```

문제

● 랜덤 만들기

문제

코딩 문제를 만들고 있는 윌리는 테스트케이스를 만들기 위해 무작위 수를 만들고 있다.
직접 랜덤한 수를 만드는데 지친 윌리는 당신에게 무작위 수를 만들어주는 프로그램을 만들어 달라는 부탁을 하였다.

랜덤한 수를 만드는 방법은 아래와 같다.

1. 9999이하의 양의 정수 하나를 고른다.
2. 그 정수의 가운데 두자리 (백의 자리와 십의 자리)를 고른다.
3. 2번에서 고른 수를 제곱하면, 제곱한 수가 새로운 수가 된다.
4. 2번과 3번을 반복하여 계속 수를 만들어간다.

문제

- 랜덤 만들기

7339의 경우에는 1089, 64, 36, 9, 0, 0처럼 6번 반복해야 한번 나왔던 수가 다시 나왔다.

입력

4자리 이하의 양의 정수 N이 주어진다.

출력

몇 번 위 동작을 반복 해야 한 번 나왔던 수가 다시 나오는지 그 횟수를 출력한다.

풀이

문제에서 주어진 방법대로 새로운 수를 만드는 while문 혹은 재귀함수를 작성하고,
chk배열을 만들어 이전에 만든 숫자가 또 다시 등장하는지 판별해주면 된다.

다음 수 만드는 방법:

1. 가운데 두자리 가져오기:
    1. 10으로 나눈 몫(일의 자리 없애기)을 100으로 나눈 나머지(뒤에 2자리 가져오기)
    2. 1000으로 나눈 나머지(천의 자리 없애기)를 10으로 나눈 몫(일의 자리 없애기)
2. 1에서 가져온 두자리 수를 제곱한다.

코드

```
static boolean[] chk = new boolean[10000];

public static void main(String[] args) throws IOException{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    int N = Integer.parseInt(br.readLine());

    System.out.print(dfs(N, 0));
}

static int dfs(int x, int d){
    if(chk[x]) return d;
    chk[x] = true;

    int middle = (x/10)%100;
    return dfs(middle * middle, d+1);
}
```

문제

● 알파벳

문제

세로 R칸, 가로 C칸으로 된 표 모양의 보드가 있다. 보드의 각 칸에는 대문자 알파벳이 하나씩 적혀 있고, 좌측 상단 칸 (1행 1열) 에는 말이 놓여 있다.

말은 상하좌우로 인접한 네 칸 중의 한 칸으로 이동할 수 있는데, 새로 이동한 칸에 적혀 있는 알파벳은 지금까지 지나온 모든 칸에 적혀 있는 알파벳과는 달라야 한다. 즉, 같은 알파벳이 적힌 칸을 두 번 지날 수 없다.

좌측 상단에서 시작해서, 말이 최대한 몇 칸을 지날 수 있는지를 구하는 프로그램을 작성하시오. 말이 지나는 칸은 좌측 상단의 칸도 포함된다

입력

첫째 줄에 R과 C가 빈칸을 사이에 두고 주어진다. (1<=R,C<=20) 둘째 줄부터 R개의 줄에 걸쳐서 보드에 적혀 있는 C개의 대문자 알파벳들이 빈칸 없이 주어진다.

출력

첫째 줄에 말이 지날 수 있는 최대의 칸 수를 출력한다.

풀이

시작점이 (1,1)인 dfs를 돌려주면 된다.

단, 이때 각 칸을 방문했는지가 아닌 특정 알파벳을 방문했는지에 대한 여부를 기록해 둔다면 문제를 해결할 수 있다.

문자에 대한 chk배열 만드는 방법:

문자의 아스키코드 값을 활용한다.
C++과 Java는 문자를 그대로 써도 아스키코드 값으로 인식하며, Python은 ord 함수를 사용해 문자의 아스키코드 값을 알아낼 수 있다.

- C++, Java: chk[‘A’] = true;
- Python: chk[ord(‘A’)] = True

chk배열 생성:

- C++: bool chk[‘Z’+1];
- Java: boolean[] chk = new boolean[‘Z’+1];
- Python: chk = [False] * (ord(‘Z’) + 1)

코드

```java
static void dfs(int x, int y, int d){
    if(chk[arr[x][y]]) return;

    ans = Math.max(ans, d);

    chk[arr[x][y]] = true;
    for(int i=0; i<4; i++) dfs(x+dx[i], y+dy[i], d+1);
    chk[arr[x][y]] = false;
}
```

```python
def dfs(x, y, d):
    global R, C, chk, ans

    if x < 0 or x >= R or y < 0 or y >= C: return
    if chk[ord(arr[x][y])]: return
    ans = max(ans, d)

    chk[ord(arr[x][y])] = True
    for i in range(4): dfs(x+dx[i], y+dy[i], d+1)
    chk[ord(arr[x][y])] = False
```

문제

- 고양이를 구해라

문제

민기는 사라진 고양이를 구하기 위해 숲 속으로 들어갔다.
숲은 (N, M) 크기의 격자로 표현되며, 입구는 (1, 1)에 있고 고양이는 (N, M)에 위치해있다.
민기는 상하좌우로 이동할 수 있고, 한 칸을 이동하는데 1분이 걸린다.

숲에는 나무로 이루어진 격자가 있고, 민기는 이를 통과할 수 없다.
하지만, 숲 속에 도끼를 주으면 앞을 가로막는 나무를 단숨에 베어낼 수 있다.
도끼는 숲 속에 단 하나 존재한다.

고양이는 인내심에 한계가 있기 때문에, 주인을 T시간만 기다린다.
만약 T시간 동안 고양이가 있는 칸에 도달하지 못하면 고양이는 영영 도망쳐버린다.

민기가 고양이를 구할 있는지, 구하려면 얼마의 시간이 필요한지 구하여라

출력

고양이를 구하기 위해서는 최소 몇 분이 걸리는지 출력하여라.
만약 고양이를 구할 수 없다면 Fail을 출력하여라

풀이

좌표와 거리와 더불어 도끼를 찾았는지 여부에 대한 정보와 함께 bfs를 돌려주면 된다.

bfs(x, y, z, d):
- z = 0: 중간에 도끼를 줍지 않고, (x, y)를 d 시간만에 방문한 경우
- z = 1: 중간에 도끼를 줍고, (x, y)를 d 시간만에 방문한 경우

(함수와 같은 형태로 작성했지만, 실제로 함수로 작성하지는 않는다.)

dist[x][y][z]:
- z = 0: 중간에 도끼를 줍지 않고 (x, y)까지 방문하는 최단 거리
- z = 1: 중간에 도끼를 줍고, (x, y)까지 방문하는 최단 거리

z = 0 인 경우에는 arr[x][y] = 1인 곳을 가지 못하지만,
z = 1 인 경우에는 배열 범위를 벗어나지만 않는다면 어디든 갈 수 있다.

(N, M)까지 가지 못하거나, (N, M)까지 가는 최단 거리가 T보다 크면 “Fail”, 그렇지 않으면 최단 거리 출력

코드

```
que.offer(0); que.offer(0); que.offer(0); que.offer(1);
while(!que.isEmpty()){
    int x = que.poll();
    int y = que.poll();
    int z = que.poll();
    int d = que.poll();

    if(x < 0 || x >= N || y < 0 || y >= M) continue;
    if(arr[x][y] == 2) z = 1;
    if(arr[x][y] == 1 && z == 0) continue;
    if(dist[x][y][z] > 0) continue;
    dist[x][y][z] = d;

    for(int i=0; i<4; i++){
        que.offer(x+dx[i]);
        que.offer(y+dy[i]);
        que.offer(z);
        que.offer(d+1);
    }
}
```

```
int ans = N*M*3+1;
if(dist[N-1][M-1][0] > 0) ans = Math.min(ans, dist[N-1][M-1][0] - 1);
if(dist[N-1][M-1][1] > 0) ans = Math.min(ans, dist[N-1][M-1][1] - 1);

if(ans == N*M*3+1 || ans > T) System.out.print("Fail");
else System.out.print(ans);
```