자료구조

과제 1. 미로 저장하기

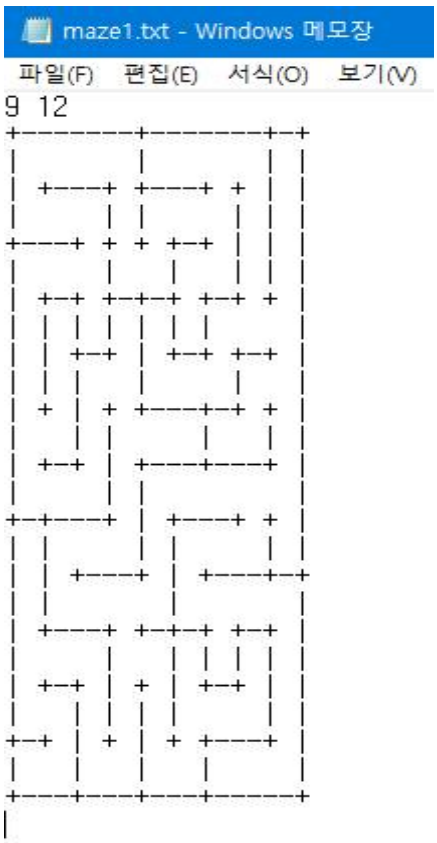
과목명 : 자료구조
담당교수 : 김승태
소속학과 : 응용통계학과
학번 : 20193011
이름 : 유승희

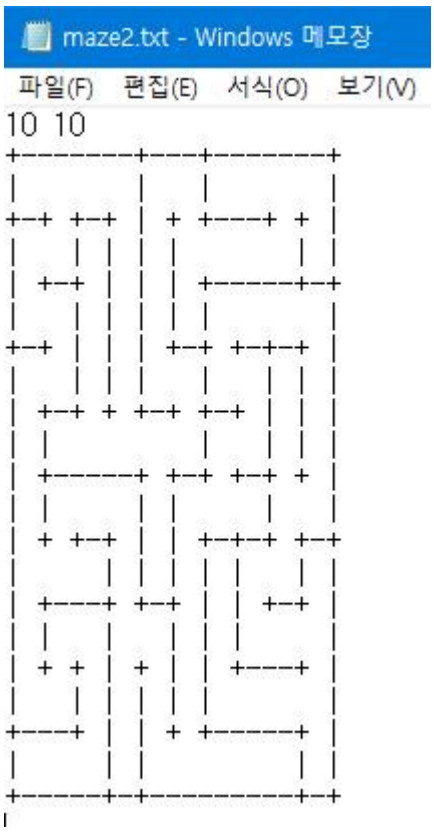
1. 파일 읽고 화면에 표시하기

1. 문제 내용

'maze1.txt' , 'maze2.txt' 파일을 읽어 화면에 표시한다.

각각의 파일은 다음과 같다.





2. 문제해결방안

txt 파일을 파이썬으로 읽기 위해 반복문을 통해 한줄씩 읽는다. 파일에서 줄바꿈이 생길 때까지 한 str로 append 해주고 줄바꿈이 생기면 data에 append 한 후 반복해서 요소를 만들어준다. 줄이 끝나면 반복문을 나온다. 파일 내용이 담긴 data는 1차원 리스트가 된다.

- 코드

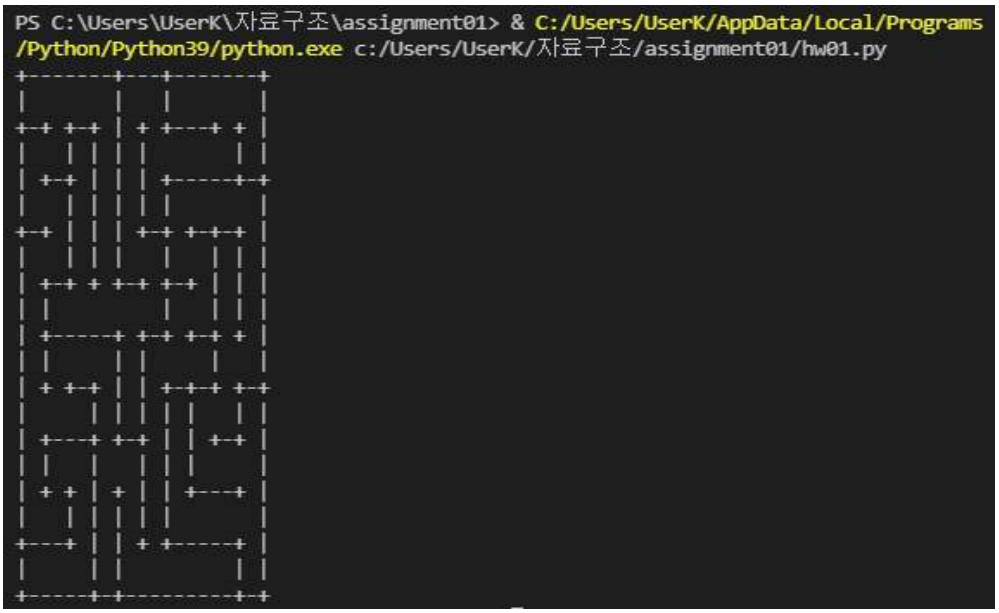
```
data = []
# 파일에서 불러 오는 부분
with open('maze1.txt') as f:
    size = f.readline()[:-1]
    while True:
        line = f.readline()
        if not line: break
        if '\n' in line:
            if '\n' != line:
                data.append(line[:-1])
        else:
            data.append(line)
```

maze1 불러오는 코드이고 maze2를 불러오기 위해선 open('maze2.txt')를 입력하면 된다.

3. 결과

- maze1 결과

- maze2 결과



2. 미로 저장하기

1. 문제내용

불러온 파일의 미로를 효과적인 저장 방법에 의해 저장해야 한다. 미로의 내용은 ' ', '-' , '|' ,'+'를 이용해 미로가 구성되어 있다.

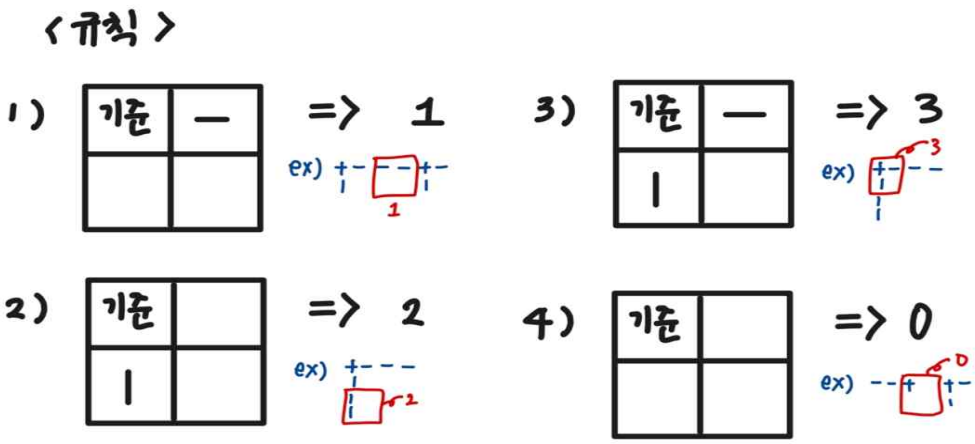
2. 문제해결방안

데이터 파일에는 미로의 벽과 미로의 길을 모두 담고 있지만 효과적으로 저장하기 위해 미로의 벽을 저장하였다. 미로의 벽을 0,1,2,3 의 집합으로 표현하였습니다. 또한 중복된 정보 저장을 막기 위해 짝수번째 줄과 짝수번째 칸 정보는 생략하여 표현하였습니다. 즉, 홀수번째 줄에서 홀수번째 칸이 바로 옆 칸(짝수번째 칸)과 아랫줄 (짝수번째 줄) 정보를 포함합니다. 0,1,2,3 의 숫자의 집합으로 표현하기 위해서는 다음과 같은 규칙을 사용하였습니다.

[규칙]

- 자신의 위치에서 오른쪽으로 가는 선이 있으면 1로 표현한다.
- 자신의 위치에서 아래로 가는 선이 있으면 2로 표현한다.
- 자신의 위치에서 오른쪽으로 가는 선, 아래로 가는 선 둘다 있으면 3으로 표현한다.
- 자신의 위치에서 오른쪽으로 가는 선, 아래로 가는 선 둘다 없으면 0으로 표현한다.

 * 왼쪽과 위로 가는 선을 고려하지 않는 이유는 앞전 위치에서 정보가 저장이 되었기 때문이다.



위 규칙을 코드로 나타내기 위해 우선 읽어온 데이터 파일의 길이를 높이(height)로, 모든 요소의 길이가 같으므로 요소의 길이를 폭(width)로 저장했다.
그리고 데이터 파일의 각 줄을 행으로, 각 요소의 한 칸을 열로 취급하여 행은 높이만큼 열은 폭만큼의 반복문을 이용해 숫자로 표현했다. 조건에 맞는 숫자를 할당하는 데에는 조건문을 사용했다. 여기서 주의해야할 점은 맨 마지막 열과 맨 마지막 행이다. 맨 마지막 열은 확인할 다음 칸이 없고, 맨 마지막 줄에서는 확인할 아랫 줄이 없어서 따로 지정해서 숫자로 표현해주었다.
결과는 2차원 리스트로 표현된다.

- 코드

```
width=len(data[0])
height=len(data)
```

```
def make_maze(maze, width, height):
    newmaze = []
    for row_idx in range(0, height-1, 2):
        row_data = []
        for col_idx in range(0, width-1, 2):
            if maze[row_idx][col_idx+1] == '-' and maze[row_idx+1][col_idx] == '|' :
                row_data.append(3)
            elif maze[row_idx][col_idx+1] == '-' and maze[row_idx+1][col_idx] == ' ':
                row_data.append(1)
            elif maze[row_idx][col_idx+1] == ' ' and maze[row_idx+1][col_idx] == '|':
                row_data.append(2)
            elif maze[row_idx][col_idx+1] == ' ' and maze[row_idx+1][col_idx] == ' ':
                row_data.append(0)
        row_data.append(2)
        newmaze.append(row_data)
    newmaze.append([1]*int((len(data[0])-1)/2) +[0])
    return newmaze
```

3. 결과

- maze1

```
PS C:\Users\UserK\자료구조\assignment01> & C:/Users/UserK/AppData/Local/Programs
/Python/Python39/python.exe c:/Users/UserK/자료구조/assignment01/hw01.py
[3, 1, 1, 1, 3, 1, 1, 1, 3, 2]
[2, 1, 1, 2, 3, 1, 0, 2, 2, 2]
[3, 1, 0, 2, 0, 3, 0, 2, 2, 2]
[2, 3, 2, 3, 3, 2, 3, 0, 0, 2]
[2, 2, 3, 0, 2, 1, 0, 3, 0, 2]
[2, 0, 2, 2, 1, 1, 3, 0, 2, 2]
[2, 1, 0, 2, 3, 1, 1, 1, 0, 2]
[3, 3, 1, 0, 2, 3, 1, 0, 2, 2]
[2, 2, 1, 1, 0, 2, 1, 1, 1, 2]
[2, 1, 1, 2, 1, 3, 2, 3, 2, 2]
[2, 1, 2, 2, 2, 2, 1, 0, 2, 2]
[3, 0, 2, 0, 2, 0, 3, 1, 0, 2]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
```

- maze2

```
PS C:\Users\UserK\자료구조\assignment01> & C:/Users/UserK/AppData/Local/Programs
/Python/Python39/python.exe c:/Users/UserK/자료구조/assignment01/hw01.py
[3, 1, 1, 1, 3, 1, 3, 1, 1, 1, 2]
[3, 0, 3, 2, 2, 2, 1, 1, 0, 2, 2]
[2, 1, 2, 2, 2, 2, 3, 1, 1, 1, 2]
[3, 0, 2, 2, 2, 1, 2, 1, 3, 2, 2]
[2, 3, 0, 0, 1, 0, 3, 0, 2, 2, 2]
[2, 3, 1, 1, 2, 3, 0, 1, 2, 0, 2]
[2, 0, 1, 2, 2, 2, 3, 3, 0, 3, 2]
[2, 3, 1, 2, 1, 2, 2, 2, 1, 0, 2]
[2, 0, 2, 2, 2, 2, 2, 1, 1, 0, 2]
[3, 1, 0, 2, 2, 0, 1, 1, 1, 2, 2]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
```

미로 1 저장에 사용된 메모리 크기

130 바이트

```
PS C:\Users\UserK\자료구조\assignment01> & C:/Users/UserK/AppData/Local/Programs
/Python/Python39/python.exe c:/Users/UserK/자료구조/assignment01/hw01.py
미로1을 저장하는데 사용된 메모리 크기 :  130  바이트(byte)
```

3. 저장한 미로를 출력하기

1. 문제내용

저장한 미로를 출력 함수를 이용해 화면에 표시한다. 불러온 txt 파일과 출력되는 화면이 동일해야 한다.

2. 문제해결방안

효과적으로 저장하기 위해 숫자로 표현한 2차원 리스트를 다시 규칙에 맞추어 ' ', '-', '|' ,'+' (character) 의 집합으로 표현해야 한다. 미로를 저장하기 위해 두칸씩 두줄의 정보를 한 숫자로 표현하였기에 출력할 때에도 각 숫자에 맞춰 두 칸씩 두줄을 모두 character로 표현해주어야 한다.
즉, 한 숫자에 네 개의 character가 표현되게 해주었다. 이를 코드로 나타내기 위해 각 숫자에 해당하는 홀수 줄은 up_data 의 리스트에 append를 이용해 붙여주었다. 각 숫자에 해당하는 짝수 줄은 low_data의 리스트에 append를 이용했다. 2번 문제처럼 출력을 할때에도 행과 열의 개념으로 취급해주었다. 효과적으로 저장한 2차원 리스트인 newmaze의 높이(len(newmaze))만큼 행 반복, 폭(len(newmaze[0]))만큼 열 반복하여 출력해주었다. 각 숫자에 해당하는 character는 다음과 같은 규칙을 이용해 주었다.

[규칙]
- 숫자가 0 일 때 : 항상 오른쪽과 아래쪽 선 둘 다 없는 '+'이다.
- 숫자가 3 일 때 : 항상 오른쪽과 아래쪽 선 둘 다 있는 '+'이다.

- 숫자가 1 일 때 : 짝수 줄은 항상 ' '' '이고 홀수 줄은 다음의 경우로 나뉜다.
 1. 바로 윗줄에 '|' 모양이 있는 경우 : '+''-'
 2. 바로 왼쪽칸에 '-' 모양이 있는 경우
    2.1 왼쪽 칸에 '-' 모양이 있으면서 윗 줄에 ' ' 모양인 경우 : '-''-'
    2.2 왼쪽 칸에 '-' 모양이 있으면서 윗 줄에 '|' 모양인 경우 (즉, 왼쪽칸 윗 줄에 둘 다 있을 경우) : '+''-'
 3. 왼쪽 칸에 ' ' 모양이면서 윗 줄이 ' ' 모양인 경우 : '+''-'

- 숫자가 2일 때 : 짝수 줄은 항상 '|'' '이고 홀수 줄은 다음의 경우로 나뉜다.
 1. 바로 왼쪽 칸에 '-' 모양이 있는 경우 : '+'' '
 2. 바로 윗 줄에 '|' 모양이 있는 경우
    2.1 윗 줄에 '|' 모양이 있으면서 왼쪽 칸이 ' '인 경우 : '|'' '
    2.2 윗 줄에 '|' 모양이 있으면서 왼쪽 칸에 '-'인 경우 : '+'' '
 3. 왼쪽 칸 ' ' 모양이면서 윗 줄이 ' ' 모양인 경우 : '+'' '

*홀수 줄 표현
위 규칙을 코드로 나타내기 위해 숫자가 1일때엔 대부분 '+''-' 이고 특정 경우 일때만 '-''-'로 표현되기 때문에 중첩 조건문을 사용해 왼쪽 칸이 '-' 인 경우면서 윗줄에 ' '인 경우 (즉, (newmaze의 왼쪽 열 숫자 1 or 3) and (newmaze의 위쪽 행 숫자 0 or 1))

에만 '-''-'를 출력하도록 하고 나머지는 '+''-'로 나타내도록 했다. 숫자가 2인 경우에도 대부분 '+'' ''이고 특정 경우 ((newmaze의 왼쪽 열 숫자 0 or 2) and (newmaze의 위쪽 행 숫자 2 or 3)) 에만 '|'' '를 부여하고 나머지 경우엔 '+'' '로 나타내도록 했다.

2번을 해결하면서 마지막 열과 마지막 행을 따로 처리해준 것처럼 반대로 출력할때에도 따로 처리를 해줄 열과 행이 있다. 첫 번째 열과 첫 번째 행이다. 첫 번째 행은 미로의 끝 벽이기 때문에 1일 경우 '-''-' 로 들어가도록 해주었고, 첫 번째 열 또한 미로의 끝 벽이기 때문에 2일 경우 '|'' '로 출력되도록 했다.

- 코드

```
def draw_maze(newmaze):
    character = [' ', '-', '|', '+']
    draw=[]
    for row_idx in range(len(newmaze)):
        up_data = []
        low_data = []
        for col_idx in range(len(newmaze[0])):

            if newmaze[row_idx][col_idx] == 0 :
                up_data.append(character[3] + character[0])
                low_data.append(character[0] + character[0])

            elif newmaze[row_idx][col_idx] == 1 :  #'+-'
                low_data.append(character[0] + character[0])

                if (col_idx > 0) and (row_idx > 0) and \
                ((newmaze[row_idx][col_idx-1] == 1) or (newmaze[row_idx][col_idx-1] == 3)) and \
                ((newmaze[row_idx-1][col_idx] == 0) or (newmaze[row_idx-1][col_idx] == 1)):
                    up_data.append(character[1] + character[1])
                elif row_idx == 0 :
                    up_data.append(character[1] + character[1])

                else:
                    up_data.append(character[3] + character[1])

            elif newmaze[row_idx][col_idx] == 2 :
                low_data.append(character[2] + character[0])

                if (col_idx > 0 ) and (row_idx > 0) and \
                ((newmaze[row_idx][col_idx-1] == 0 ) or (newmaze[row_idx][col_idx-1] == 2)) and \
                ((newmaze[row_idx-1][col_idx] == 2) or (newmaze[row_idx-1][col_idx] == 3)):
                    up_data.append(character[2] + character[0])
                elif col_idx == 0 :
                    up_data.append(character[2] + character[0])
                else:
                    up_data.append(character[3] + character[0])
            elif newmaze[row_idx][col_idx] == 3 :
                up_data.append(character[3] + character[1])
                low_data.append(character[2] + character[0])

            up = ''.join(up_data)
            low = ''.join(low_data)
        draw.append(up[:-1])
        draw.append(low[:-1])

    for i in range(len(draw)):
        print(draw[i])
```

3. 결과

- maze1

- maze2